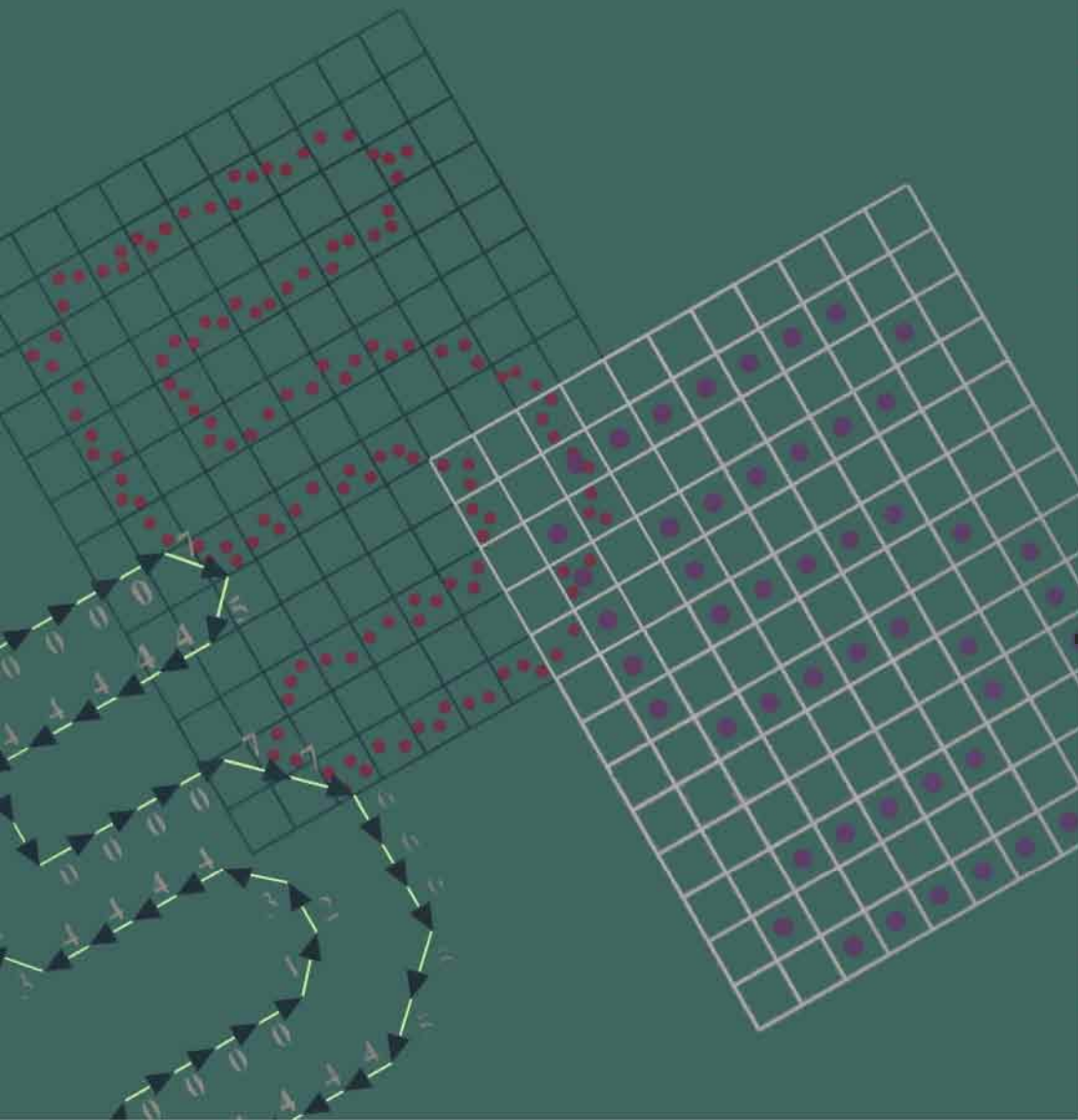
PATTERN RECOGNITION

SECOND EDITION

SERGIOS THEODORIDIS
KONSTANTINOS KOUTROUMBAS



PATTERN RECOGNITION

SECOND EDITION

PATTERN RECOGNITION
SECOND EDITION

SERGIOS THEODORIDIS

Department of Informatics and Telecommunications
University of Athens
Greece

and

KONSTANTINOS KOUTROUMBAS

Institute of Space Applications & Remote Sensing
National Observatory of Athens
Greece

ELSEVIER
ACADEMIC
PRESS

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Academic Press is an imprint of Elsevier

This book is printed on acid-free paper. ♾

Copyright 2003, Elsevier (USA).

All rights reserved.
No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher. Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.com.uk. You may also complete your request on-line via the Elsevier homepage (http://elsevier.com), by selecting "Customer Support" and then "Obtaining Permissions."

ACADEMIC PRESS
An imprint of Elsevier
525 B Street, Suite 1900, San Diego, CA 92101-4495, USA
http://www.academicpress.com

Academic Press
84 Theobald's Road, London WC1X 8RR, UK
http://www.academicpress.com

Library of Congress Control Number: 2002117797

International Standard Book Number: 0-12-685875-6

PRINTED IN THE UNITED STATES OF AMERICA
03 04 05 06 07 08 9 8 7 6 5 4 3 2

CONTENTS

Preface xiii

CHAPTER 1 INTRODUCTION 1

1.1 Is Pattern Recognition Important? 1
1.2 Features, Feature Vectors, and Classifiers 3
1.3 Supervised Versus Unsupervised Pattern Recognition 6
1.4 Outline of the Book 8

CHAPTER 2 CLASSIFIERS BASED ON BAYES DECISION THEORY 13

2.1 Introduction 13
2.2 Bayes Decision Theory 13
2.3 Discriminant Functions and Decision Surfaces 19
2.4 Bayesian Classification for Normal Distributions 20
2.5 Estimation of Unknown Probability Density Functions 27
2.5.1 Maximum Likelihood Parameter Estimation 28
2.5.2 Maximum a Posteriori Probability Estimation 31
2.5.3 Bayesian Inference 32
2.5.4 Maximum Entropy Estimation 34
2.5.5 Mixture Models 35
2.5.6 Nonparametric Estimation 39
2.6 The Nearest Neighbor Rule 44

CHAPTER 3 LINEAR CLASSIFIERS 55

3.1 Introduction 55
3.2 Linear Discriminant Functions and Decision Hyperplanes 55
3.3 The Perceptron Algorithm 57

	3.4	Least Squares Methods	65
		3.4.1 Mean Square Error Estimation	65
		3.4.2 Stochastic Approximation and the LMS Algorithm	68
		3.4.3 Sum of Error Squares Estimation	70
	3.5	Mean Square Estimation Revisited	72
		3.5.1 Mean Square Error Regression	72
		3.5.2 MSE Estimates Posterior Class Probabilities	73
		3.5.3 The Bias–Variance Dilemma	76
	3.6	Support Vector Machines	77
		3.6.1 Separable Classes	77
		3.6.2 Nonseparable Classes	82
CHAPTER 4	NONLINEAR CLASSIFIERS		93
	4.1	Introduction	93
	4.2	The XOR Problem	93
	4.3	The Two-Layer Perceptron	94
		4.3.1 Classification Capabilities of the Two-Layer Perceptron	98
	4.4	Three-Layer Perceptrons	101
	4.5	Algorithms Based on Exact Classification of the Training Set	102
	4.6	The Backpropagation Algorithm	104
	4.7	Variations on the Backpropagation Theme	112
	4.8	The Cost Function Choice	115
	4.9	Choice of the Network Size	118
	4.10	A Simulation Example	124
	4.11	Networks With Weight Sharing	126
	4.12	Generalized Linear Classifiers	127
	4.13	Capacity of the l-Dimensional Space in Linear Dichotomies	129
	4.14	Polynomial Classifiers	131
	4.15	Radial Basis Function Networks	133
	4.16	Universal Approximators	137
	4.17	Support Vector Machines: The Nonlinear Case	139
	4.18	Decision Trees	143
		4.18.1 Set of Questions	146
		4.18.2 Splitting Criterion	146
		4.18.3 Stop-Splitting Rule	147
		4.18.4 Class Assignment Rule	147
	4.19	Discussion	150

CHAPTER 5 FEATURE SELECTION 163

5.1 Introduction 163
5.2 Preprocessing 164
5.2.1 Outlier Removal 164
5.2.2 Data Normalization 165
5.2.3 Missing Data 165
5.3 Feature Selection Based on Statistical Hypothesis Testing 165
5.3.1 Hypothesis Testing Basics 166
5.3.2 Application of the t-Test in Feature Selection 171
5.4 The Receiver Operating Characteristics CROC Curve 173
5.5 Class Separability Measures 174
5.5.1 Divergence 174
5.5.2 Chernoff Bound and Bhattacharyya Distance 177
5.5.3 Scatter Matrices 179
5.6 Feature Subset Selection 181
5.6.1 Scalar Feature Selection 182
5.6.2 Feature Vector Selection 183
5.7 Optimal Feature Generation 187
5.8 Neural Networks and Feature Generation/Selection 191
5.9 A Hint on the Vapnik–Chernovenkis Learning Theory 193

CHAPTER 6 FEATURE GENERATION I: LINEAR TRANSFORMS 207

6.1 Introduction 207
6.2 Basis Vectors and Images 208
6.3 The Karhunen–Loève Transform 210
6.4 The Singular Value Decomposition 215
6.5 Independent Component Analysis 219
6.5.1 ICA Based on Second- and Fourth-Order Cumulants 221
6.5.2 ICA Based on Mutual Information 222
6.5.3 An ICA Simulation Example 226
6.6 The Discrete Fourier Transform (DFT) 226
6.6.1 One-Dimensional DFT 227
6.6.2 Two-Dimensional DFT 229
6.7 The Discrete Cosine and Sine Transforms 230
6.8 The Hadamard Transform 231
6.9 The Haar Transform 233

	6.10	The Haar Expansion Revisited	235
	6.11	Discrete Time Wavelet Transform (DTWT)	239
	6.12	The Multiresolution Interpretation	249
	6.13	Wavelet Packets	252
	6.14	A Look at Two-Dimensional Generalizations	252
	6.15	Applications	255
CHAPTER 7	FEATURE GENERATION II		269
	7.1	Introduction	269
	7.2	Regional Features	270
		7.2.1 Features for Texture Characterization	270
		7.2.2 Local Linear Transforms for Texture Feature Extraction	279
		7.2.3 Moments	281
		7.2.4 Parametric Models	286
	7.3	Features for Shape and Size Characterization	294
		7.3.1 Fourier Features	295
		7.3.2 Chain Codes	298
		7.3.3 Moment-Based Features	301
		7.3.4 Geometric Features	302
	7.4	A Glimpse at Fractals	303
		7.4.1 Self-Similarity and Fractal Dimension	303
		7.4.2 Fractional Brownian Motion	306
CHAPTER 8	TEMPLATE MATCHING		321
	8.1	Introduction	321
	8.2	Measures Based on Optimal Path Searching Techniques	322
		8.2.1 Bellman's Optimality Principle and Dynamic Programming	324
		8.2.2 The Edit Distance	325
		8.2.3 Dynamic Time Warping in Speech Recognition	329
	8.3	Measures Based on Correlations	337
	8.4	Deformable Template Models	343
CHAPTER 9	CONTEXT-DEPENDENT CLASSIFICATION		351
	9.1	Introduction	351
	9.2	The Bayes Classifier	351
	9.3	Markov Chain Models	352
	9.4	The Viterbi Algorithm	353

9.5 Channel Equalization 356
9.6 Hidden Markov Models 361
9.7 Training Markov Models via Neural Networks 373
9.8 A discussion of Markov Random Fields 375

CHAPTER 10 SYSTEM EVALUATION 385

10.1 Introduction 385
10.2 Error Counting Approach 385
10.3 Exploiting the Finite Size of the Data Set 387
10.4 A Case Study From Medical Imaging 390

CHAPTER 11 CLUSTERING: BASIC CONCEPTS 397

11.1 Introduction 397
11.1.1 Applications of Cluster Analysis 400
11.1.2 Types of Features 401
11.1.3 Definitions of Clustering 402
11.2 Proximity Measures 404
11.2.1 Definitions 404
11.2.2 Proximity Measures between Two Points 407
11.2.3 Proximity Functions between a Point and a Set 418
11.2.4 Proximity Functions between Two Sets 423

CHAPTER 12 CLUSTERING ALGORITHMS I: SEQUENTIAL ALGORITHMS 429

12.1 Introduction 429
12.1.1 Number of Possible Clusterings 429
12.2 Categories of Clustering Algorithms 431
12.3 Sequential Clustering Algorithms 433
12.3.1 Estimation of the Number of Clusters 435
12.4 A Modification of BSAS 437
12.5 A Two-Threshold Sequential Scheme 438
12.6 Refinement Stages 441
12.7 Neural Network Implementation 443
12.7.1 Description of the Architecture 443
12.7.2 Implementation of the BSAS Algorithm 444

CHAPTER 13 CLUSTERING ALGORITHMS II: HIERARCHICAL ALGORITHMS 449

13.1 Introduction 449

13.2 Agglomerative Algorithms 450
13.2.1 Definition of Some Useful Quantities 451
13.2.2 Agglomerative Algorithms Based on Matrix Theory 453
13.2.3 Monotonicity and Crossover 461
13.2.4 Implementational Issues 464
13.2.5 Agglomerative Algorithms Based on Graph Theory 464
13.2.6 Ties in the Proximity Matrix 474
13.3 The Cophenetic Matrix 476
13.4 Divisive Algorithms 478
13.5 Choice of the Best Number of Clusters 480

CHAPTER 14 CLUSTERING ALGORITHMS III: SCHEMES BASED ON FUNCTION OPTIMIZATION 489

14.1 Introduction 489
14.2 Mixture Decomposition Schemes 491
14.2.1 Compact and Hyperellipsoidal Clusters 493
14.2.2 A Geometrical Interpretation 497
14.3 Fuzzy Clustering Algorithms 500
14.3.1 Point Representatives 505
14.3.2 Quadric Surfaces as Representatives 507
14.3.3 Hyperplane Representatives 517
14.3.4 Combining Quadric and Hyperplane Representatives 519
14.3.5 A Geometrical Interpretation 521
14.3.6 Convergence Aspects of the Fuzzy Clustering Algorithms 522
14.3.7 Alternating Cluster Estimation 522
14.4 Possibilistic Clustering 522
14.4.1 The Mode-Seeking Property 526
14.4.2 An Alternative Possibilistic Scheme 529
14.5 Hard Clustering Algorithms 529
14.5.1 The Isodata or k-Means or c-Means Algorithm 531
14.6 Vector Quantization 533

CHAPTER 15 CLUSTERING ALGORITHMS IV 545

15.1 Introduction 545
15.2 Clustering Algorithms Based on Graph Theory 545
15.2.1 Minimum Spanning Tree Algorithms 546

15.2.2 Algorithms Based on Regions of Influence 549
15.2.3 Algorithms Based on Directed Trees 550
15.3 Competitive Learning Algorithms 552
15.3.1 Basic Competitive Learning Algorithm 554
15.3.2 Leaky Learning Algorithm 556
15.3.3 Conscientious Competitive Learning Algorithms 556
15.3.4 Competitive Learning–Like Algorithms Associated with Cost Functions 558
15.3.5 Self-Organizing Maps 559
15.3.6 Supervised Learning Vector Quantization 560
15.4 Branch and Bound Clustering Algorithms 561
15.5 Binary Morphology Clustering Algorithms (BMCAs) 564
15.5.1 Discretization 564
15.5.2 Morphological Operations 565
15.5.3 Determination of the Clusters in a Discrete Binary Set 568
15.5.4 Assignment of Feature Vectors to Clusters 570
15.5.5 The Algorithmic Scheme 571
15.6 Boundary Detection Algorithms 573
15.7 Valley-Seeking Clustering Algorithms 576
15.8 Clustering Via Cost Optimization (Revisited) 578
15.8.1 Simulated Annealing 579
15.8.2 Deterministic Annealing 580
15.9 Clustering Using Genetic Algorithms 582
15.10 Other Clustering Algorithms 583

CHAPTER 16 CLUSTER VALIDITY 591

16.1 Introduction 591
16.2 Hypothesis Testing Revisited 592
16.3 Hypothesis Testing in Cluster Validity 594
16.3.1 External Criteria 596
16.3.2 Internal Criteria 602
16.4 Relative Criteria 605
16.4.1 Hard Clustering 608
16.4.2 Fuzzy Clustering 614
16.5 Validity of Individual Clusters 621
16.5.1 External Criteria 621
16.5.2 Internal Criteria 622
16.6 Clustering Tendency 624
16.6.1 Tests for Spatial Randomness 628

Appendix A
Hints from Probability and Statistics 643

Appendix B
Linear Algebra Basics 655

Appendix C
Cost Function Optimization 659

Appendix D
Basic Definitions from Linear Systems Theory 677

Index 681

PREFACE

This book is the outgrowth of our teaching advanced undergraduate and graduate courses over the past 20 years. These courses have been taught to different audiences, including students in electrical and electronics engineering, computer engineering, computer science and informatics, as well as to an interdisciplinary audience of a graduate course on automation. This experience led us to make the book as self-contained as possible and to address students with different backgrounds. As prerequisitive knowledge the reader requires only basic calculus, elementary linear algebra, and some probability theory basics. A number of mathematical tools, such as probability and statistics as well as constrained optimization, needed by various chapters, are treated in four Appendices. The book is designed to serve as a text for advanced undergraduate and graduate students, and it can be used for either a one- or a two-semester course. Furthermore, it is intended to be used as a self-study and reference book for research and for the practicing scientist/engineer. This latter audience was also our second incentive for writing this book, due to the involvement of our group in a number of projects related to pattern recognition.

The philosophy of the book is to present various pattern recognition tasks in a unified way, including image analysis, speech processing, and communication applications. Despite their differences, these areas do share common features and their study can only benefit from a unified approach. Each chapter of the book starts with the basics and moves progressively to more advanced topics and reviews up-to-date techniques. A number of problems and computer exercises are given at the end of each chapter and a solutions manual is available from the publisher. Furthermore, a number of demonstrations based on MATLAB are available via the web at the book's site, http://www.di.uoa.gr/~stpatrec.

Our intention is to update the site regularly with more and/or improved versions of these demonstrations. Suggestions are always welcome. Also at this web site, a page will be available for typos, which are unavoidable, despite frequent careful reading. The authors would appreciate readers notifying them about any typos found.

This book would have not be written without the constant support and help from a number of colleagues and students throughout the years. We are especially indebted to Prof. K. Berberidis, Dr. E. Kofidis, Prof. A. Liavas, Dr. A. Rontogiannis, Dr. A. Pikrakis, Dr. Gezerlis and Dr. K. Georgoulakis. The constant support provided by Dr. I. Kopsinis from the early stages up to the final stage, with those long nights, has been invaluable. The book improved a great deal after the careful reading and the serious comments and suggestions of Prof. G. Moustakides, Prof. V. Digalakis, Prof. T. Adali, Prof. M. Zervakis, Prof. D. Cavouras, Prof. A. Böhm, Prof. G. Glentis, Prof. E. Koutsoupias, Prof. V. Zissimopoulos, Prof. A. Likas, Dr. A. Vassiliou, Dr. N. Vassilas, Dr. V. Drakopoulos, Dr. S. Hatzispyros. We are greatly indebted to these colleagues for their time and their constructive criticisms. Our collaboration and friendship with Prof. N. Kalouptsidis have been a source of constant inspiration for all these years. We are both deeply indebted to him.

Last but not least, K. Koutroumbas would like to thank Sophia for her tolerance and support and S. Theodoridis would like to thank Despina, Eva, and Eleni, his joyful and supportive "harem."

CHAPTER 1

INTRODUCTION

1.1 IS PATTERN RECOGNITION IMPORTANT?

Pattern recognition is the scientific discipline whose goal is the classification of *objects* into a number of categories or *classes*. Depending on the application, these objects can be images or signal waveforms or any type of measurements that need to be classified. We will refer to these objects using the generic term *patterns*. Pattern recognition has a long history, but before the 1960s it was mostly the output of theoretical research in the area of statistics. As with everything else, the advent of computers increased the demand for practical applications of pattern recognition, which in turn set new demands for further theoretical developments. As our society evolves from the industrial to its postindustrial phase, automation in industrial production and the need for information handling and retrieval are becoming increasingly important. This trend has pushed pattern recognition to the high edge of today's engineering applications and research. Pattern recognition is an integral part in most *machine intelligence* systems built for decision making.

Machine vision is an area in which pattern recognition is of importance. A machine vision system captures images via a camera and analyzes them to produce descriptions of what is imaged. A typical application of a machine vision system is in the manufacturing industry, either for automated visual inspection or for automation in the assembly line. For example, in inspection, manufactured objects on a moving conveyor may pass the inspection station, where the camera stands, and it has to be ascertained whether there is a defect. Thus, images have to be analyzed on line, and a pattern recognition system has to classify the objects into the "defect" or "non-defect" class. After that, an action has to be taken, such as to reject the offending parts. In an assembly line, different objects must be located and "recognized," that is, classified in one of a number of classes known *a priori*. Examples are the "screwdriver class," the "German key class," and so forth in a tools' manufacturing unit. Then a robot arm can place the objects in the right place.

Character (letter or number) recognition is another important area of pattern recognition, with major implications in automation and information handling. Optical character recognition (OCR) systems are already commercially available and more or less familiar to all of us. An OCR system has a "front end" device

consisting of a *light source*, a *scan lens*, a *document transport*, and a *detector*. At the output of the light-sensitive detector, light intensity variation is translated into "numbers" and an image array is formed. In the sequel, a series of image processing techniques are applied leading to *line* and *character segmentation*. The pattern recognition software then takes over to recognize the characters—that is, to classify each character in the correct "letter, number, punctuation" class. Storing the recognized document has a twofold advantage over storing its scanned image. First, further electronic processing, if needed, is easy via a word processor, and second, it is much more efficient to store ASCII characters than a document image. Besides the printed character recognition systems, there is a great deal of interest invested in systems that recognize handwriting. A typical commercial application of such a system is in the machine reading of bank checks. The machine must be able to recognize the amounts in figures and digits and match them. Furthermore, it could check whether the payee corresponds to the account to be credited. Even if only half of the checks are manipulated correctly by such a machine, much labor can be saved from a tedious job. Another application is in automatic mail-sorting machines for postal code identification in post offices. On-line handwriting recognition systems are another area of great commercial interest. Such systems will accompany *pen computers*, with which the entry of data will be done not via the keyboard but by writing. This complies with today's tendency to develop machines and computers with interfaces acquiring human-like skills.

Computer-aided diagnosis is another important application of pattern recognition, aiming at assisting doctors in making diagnostic decisions. The final diagnosis is, of course, made by the doctor. Computer-assisted diagnosis has been applied to and is of interest for a variety of medical data, such as X-rays, computed tomographic images, ultrasound images, electrocardiograms (ECGs), and electroencephalograms (EEGs). The need for a computer-aided diagnosis stems from the fact that medical data are often not easily interpretable, and the interpretation can depend very much on the skill of the doctor. Let us take for example *X-ray mammography* for the detection of breast cancer. Although mammography is currently the best method for detecting breast cancer, 10%–30% of women who have the disease and undergo mammography have negative mammograms. In approximately two thirds of these cases with false results the radiologist failed to detect the cancer, which was evident retrospectively. This may be due to poor image quality, eye fatigue of the radiologist, or the subtle nature of the findings. The percentage of correct classifications improves at a second reading by another radiologist. Thus, one can aim to develop a pattern recognition system in order to assist radiologists with a "second" opinion. Increasing confidence in the diagnosis based on mammograms would, in turn, decrease the number of patients with suspected breast cancer who have to undergo surgical breast biopsy, with its associated complications.

Speech recognition is another area in which a great deal of research and development effort has been invested. Speech is the most natural means by which humans

communicate and exchange information. Thus, the goal of building intelligent machines that recognize *spoken information* has been a long-standing one for scientists and engineers as well as science fiction writers. Potential applications of such machines are numerous. They can be used, for example, to improve efficiency in a manufacturing environment, to control machines in hazardous environments remotely, and to help handicapped people to control machines by talking to them. A major effort, which has already had considerable success, is to enter data into a computer via a microphone. Software, built around a pattern (spoken sounds in this case) recognition system, recognizes the spoken text and translates it into ASCII characters, which are shown on the screen and can be stored in the memory. Entering information by "talking" to a computer is twice as fast as entry by a skilled typist. Furthermore, this can enhance our ability to communicate with deaf and dumb people.

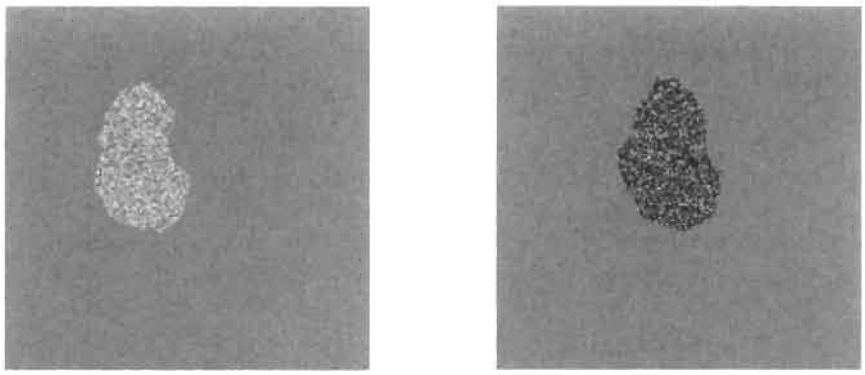
The foregoing are only four examples from a much larger number of possible applications. Typically, we refer to fingerprint identification, signature authentication, text retrieval, and face and gesture recognition. The last applications have recently attracted much research interest and investment in an attempt to facilitate human–machine interaction and further enhance the role of computers in office automation, automatic personalization of environments, and so forth. Just to provoke imagination, it is worth pointing out that the MPEG-7 standard includes provision for content-based video information retrieval from digital libraries of the type: search and find all video scenes in a digital library showing person "X" laughing. Of course, to achieve the final goals in all of these applications, pattern recognition is closely linked with other scientific disciplines, such as linguistics, computer graphics, and vision.

Having aroused the reader's curiosity about pattern recognition, we will next sketch the basic philosophy and methodological directions in which the various pattern recognition approaches have evolved and developed.

1.2 FEATURES, FEATURE VECTORS, AND CLASSIFIERS

Let us first simulate a simplified case "mimicking" a medical image classification task. Figure 1.1 shows two images, each having a distinct region inside it. The two regions are also themselves visually different. We could say that the region of Figure 1.1a results from a benign lesion, class A, and that of Figure 1.1b from a malignant one (cancer), class B. We will further assume that these are not the only patterns (images) that are available to us, but we have access to an image database with a number of patterns, some of which are known to originate from class A and some from class B.

The first step is to identify the measurable quantities that make these two regions *distinct* from each other. Figure 1.2 shows a plot of the mean value of the intensity in each region of interest versus the corresponding standard deviation around



(a) (b)

FIGURE 1.1: Examples of image regions corresponding to (a) class A and (b) class B.

this mean. Each point corresponds to a different image from the available database. It turns out that class A patterns tend to spread in a different area from class B patterns. The straight line seems to be a good candidate for separating the two classes. Let us now assume that we are given a new image with a region in it and that we do not know to which class it belongs. It is reasonable to say that we

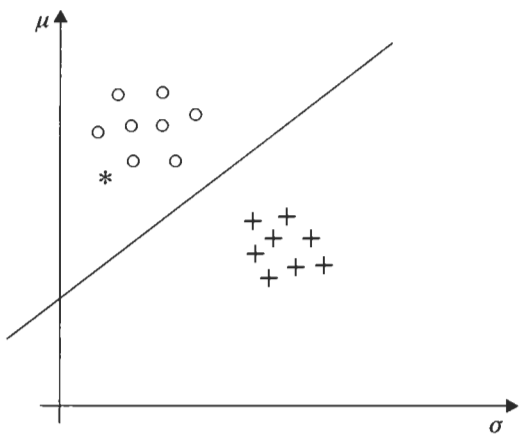


FIGURE 1.2: Plot of the mean value versus the standard deviation for a number of different images originating from class A (○) and class B (+). In this case, a straight line separates the two classes.

measure the mean intensity and standard deviation in the region of interest and we plot the corresponding point. This is shown by the asterisk (*) in Figure 1.2. Then it is sensible to assume that the unknown pattern is *more likely* to belong to class A than class B.

The preceding artificial *classification* task has outlined the rationale behind a large class of pattern recognition problems. The measurements used for the classification, the mean value and the standard deviation in this case, are known as *features*. In the more general case l features x_i, $i = 1, 2, \ldots, l$, are used and they form the *feature vector*

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_l]^T$$

where T denotes transposition. Each of the feature vectors identifies *uniquely* a single pattern (object). Throughout this book features and feature vectors will be treated as *random variables* and *vectors*, respectively. This is natural, as the measurements resulting from different patterns exhibit a random variation. This is due partly to the measurement noise of the measuring devices and partly to the distinct characteristics of each pattern. For example, in X-ray imaging large variations are expected because of the differences in physiology among individuals. This is the reason for the scattering of the points in each class shown in Figure 1.1.

The straight line in Figure 1.2 is known as the *decision* line, and it constitutes the *classifier* whose role is to divide the feature space into regions that correspond to either class A or class B. If a feature vector $\boldsymbol{x}$, corresponding to an unknown pattern, falls in the class A region, it is classified as class A, otherwise as class B. This does not necessarily mean that the decision is correct. If it is not correct, a *misclassification* has occurred. In order to draw the straight line in Figure 1.2 we exploited the fact that we knew the labels (class A or B) for each point of the figure. The patterns (feature vectors) whose true class is known and which are used for the design of the classifier are known as *training patterns (training feature vectors)*.

Having outlined the definitions and the rationale, let us point out the basic questions arising in a classification task.

- How are the features generated? In the preceding example, we used the mean and the standard deviation, because we knew how the images had been generated. In practice, this is far from obvious. It is problem dependent, and it concerns the *feature generation stage* of the design of a classification system that performs a given pattern recognition task.
- What is the best number l of features to use? This is also a very important task and it concerns the *feature selection stage* of the classification system. In practice, a larger than necessary number of feature candidates is generated and then the "best" of them is adopted.

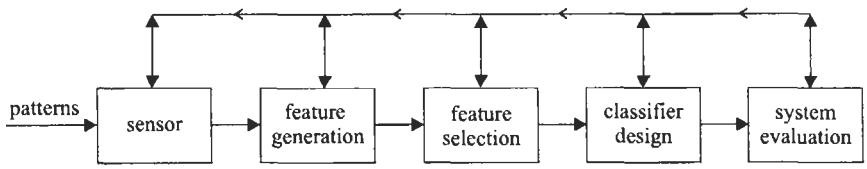


FIGURE 1.3: The basic stages involved in the design of a classification system.

- Having adopted the appropriate, for the specific task, features, how does one design the classifier? In the preceding example the straight line was drawn empirically, just to please the eye. In practice, this cannot be the case, and the line should be drawn optimally, with respect to an *optimality criterion*. Furthermore, problems for which a linear classifier (straight line or hyperplane in the l-dimensional space) can result in acceptable performance are not the rule. In general, the surfaces dividing the space in the various class regions are nonlinear. What type of nonlinearity must one adopt and what type of optimizing criterion must be used in order to locate a surface in the right place in the l-dimensional *feature space*? These questions concern the *classifier design stage*.
- Finally, once the classifier has been designed, how can one assess the performance of the designed classifier? That is, what is the *classification error rate*? This is the task of the *system evaluation stage*.

Figure 1.3 shows the various stages followed for the design of a classification system. As is apparent from the feedback arrows, these stages are not independent. On the contrary, they are interrelated and, depending on the results, one may go back to redesign earlier stages in order to improve the overall performance. Furthermore, there are some methods that combine stages, for example, the feature selection and the classifier design stage, in a common optimization task.

Although the reader has already been exposed to a number of basic problems at the heart of the design of a classification system, there are still a few things to be said.

1.3 SUPERVISED VERSUS UNSUPERVISED PATTERN RECOGNITION

In the example of Figure 1.1, we assumed that a set of training data were available, and the classifier was designed by exploiting this *a priori* known information. This is known as *supervised pattern recognition*. However, this is not always the case, and there is another type of pattern recognition tasks for which training data, of known class labels, are not available. In this type of problem, we are given a set of feature vectors $\boldsymbol{x}$ and the goal is to unravel the underlying *similarities*, and

cluster (group) "similar" vectors together. This is known as *unsupervised pattern recognition* or *clustering*. Such tasks arise in many applications in social sciences and engineering, such as remote sensing, image segmentation, and image and speech coding. Let us pick two such problems.

In *multispectral remote sensing*, the electromagnetic energy emanating from the earth's surface is measured by sensitive scanners located aboard a satellite, an aircraft, or a space station. This energy may be reflected solar energy (passive) or the reflected part of the energy transmitted from the vehicle (active) in order to "interrogate" the earth's surface. The scanners are sensitive to a number of wavelength bands of the electromagnetic radiation. Different properties of the earth's surface contribute to the reflection of the energy in the different bands. For example, in the visible–infrared range properties such as the mineral and moisture contents of soils, the sedimentation of water, and the moisture content of vegetation are the main contributors to the reflected energy. In contrast, at the thermal end of the infrared, it is the thermal capacity and thermal properties of the surface and near subsurface that contribute to the reflection. Thus, each band measures different properties of the same patch of the earth's surface. In this way, images of the earth's surface corresponding to the spatial distribution of the reflected energy in each band can be created. The task now is to exploit this information in order to identify the various ground cover types, that is, built-up land, agricultural land, forest, fire burn, water, and diseased crop. To this end, one feature vector $\boldsymbol{x}$ for each cell from the "sensed" earth's surface is formed. The elements x_i, $i = 1, 2, \ldots, l$, of the vector are the corresponding image pixel intensities in the various spectral bands. In practice, the number of spectral bands varies.

A *clustering* algorithm can be employed to reveal the groups in which feature vectors are clustered in the l-dimensional feature space. Points that correspond to the same ground cover type, such as water, are expected to cluster together and form groups. Once this is done, the analyst can identify the type of each cluster by associating a sample of points in each group with available reference ground data, that is, maps or visits. Figure 1.4 demonstrates the procedure.

Clustering is also widely used in the social sciences in order to study and correlate survey and statistical data and draw useful conclusions, *which will then lead to the right actions*. Let us again resort to a simplified example and assume that we are interested in studying whether there is any relation between a country's gross national product (GNP) and the level of people's illiteracy, on the one hand, and children's mortality rate on the other. In this case, each country is represented by a three-dimensional feature vector whose coordinates are indices measuring the quantities of interest. A clustering algorithm will then reveal a rather compact cluster corresponding to countries that exhibit low GNPs, high illiteracy levels, and high children's mortality expressed as a population percentage.

A major issue in unsupervised pattern recognition is that of defining the "similarity" between two feature vectors and choosing an appropriate measure

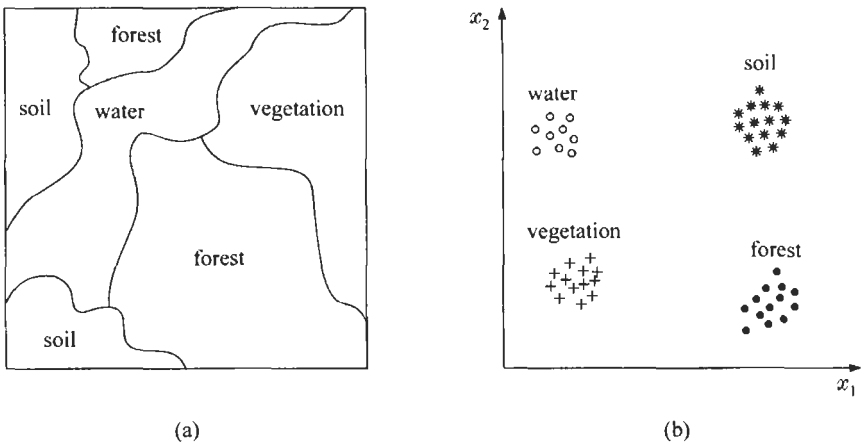


FIGURE 1.4: (a) An illustration of various types of ground cover and (b) clustering of the respective features for multispectral imaging using two bands.

for it. Another issue of importance is choosing an algorithmic scheme that will cluster (group) the vectors on the basis of the adopted similarity measure. In general, different algorithmic schemes may lead to different results, which the expert has to interpret.

1.4 OUTLINE OF THE BOOK

Chapters 2–10 deal with supervised pattern recognition and Chapters 11–16 deal with the unsupervised case. The goal of each chapter is to start with the basics, definitions and approaches, and move progressively to more advanced issues and recent techniques. To what extent the various topics covered in the book will be presented in a first course on pattern recognition depends very much on the course's focus, on the students' background, and, of course, on the lecturer. In the following outline of the chapters, we give our view and the topics that we cover in a first course on pattern recognition. No doubt, other views do exist and may be better suited to different audiences. At the end of each chapter, a number of problems and computer exercises are provided.

Chapter 2 is focused on Bayesian classification and techniques for estimating unknown probability density functions. In a first course on pattern recognition, the sections related to Bayesian inference, the maximum entropy, and the expectation maximization (EM) algorithm are omitted. Special focus is put on the Bayesian classification, the minimum distance (Euclidean and Mahalanobis), and the nearest neighbor classifiers.

Chapter 3 deals with the design of linear classifiers. The sections dealing with the probability estimation property of the mean square solution as well as the bias variance dilemma are only briefly mentioned in our first course. The basic philosophy underlying the support vector machines can also be explained, although a deeper treatment requires mathematical tools (summarized in Appendix C) that most of the students are not familiar during a first course class. On the contrary, emphasis is put on the linear separability issue, the perceptron algorithm, and the mean square and least squares solutions. After all, these topics have a much broader horizon and applicability.

Chapter 4 deals with the design of nonlinear classifiers. The section dealing with exact classification is bypassed in a first course. The proof of the backpropagation algorithm is usually very boring for most of the students and we bypass its details. A description of its rationale is given and the students experiment with it using MATLAB. The issues related to cost functions are bypassed. Pruning is discussed with an emphasis on generalization issues. Emphasis is also given to Cover's theorem and radial basis function (RBF) networks. The nonlinear support vector machines and decision trees are only briefly touched via a discussion on the basic philosophy behind their rationale.

Chapter 5 deals with the feature selection stage, and we have made an effort to present most of the well-known techniques. In a first course we put emphasis on the t-test. This is because hypothesis testing also has a broad horizon, and at the same time it is easy for the students to apply it in computer exercises. Then, depending on time constraints, divergence, Bhattacharrya distance, and scattered matrices are presented and commented on, although their more detailed treatment is for a more advanced course.

Chapter 6 deals with the feature generation stage using orthogonal transforms. The Karhunen–Loève transform and singular value decomposition are introduced. ICA is bypassed in a first course. Then the DFT, DCT, DST, Hadamard, and Haar transforms are defined. The rest of the chapter focuses on the discrete time wavelet transform. The incentive is to give all the necessary information so that a newcomer in the wavelet field can grasp the basics and be able to develop software, based on filter banks, in order to generate features. This chapter is bypassed in a first course.

Chapter 7 deals with feature generation focused on image classification. The sections concerning local linear transforms, moments, parametric models, and fractals are not covered in a first course. Emphasis is placed on first- and second-order statistics features as well as the run length method. The chain code for shape description is also taught. Computer exercises are then offered to generate these features and use them for classification for some case studies. In a one-semester course there is no time to cover more topics.

Chapter 8 deals with template matching. Dynamic programming (DP) and the Viterbi algorithm are presented and then applied to speech recognition. In a two-semester course, emphasis is given to the DP and the Viterbi algorithm.

The edit distance seems to be a good case for the students to grasp the basics. Correlation matching is taught and the basic philosophy behind deformable template matching can also be presented.

Chapter 9 deals with context-dependent classification. Hidden Markov models are introduced and applied to communications and speech recognition. This chapter is bypassed in a first course.

Chapter 10 deals with system evaluation. The various error rate estimation techniques are discussed and a case study with real data is treated. The leave-one-out method and the resubstitution methods are emphasized in the second semester and students practice with computer exercises.

Chapter 11 deals with the basic concepts of clustering. It focuses on definitions as well as on the major stages involved in a clustering task. The various types of data encountered in clustering applications are reviewed and the most commonly used proximity measures are provided. In a first course, only the most widely used proximity measures are covered (e.g., l_p norms, inner product, Hamming distance).

Chapter 12 deals with sequential clustering algorithms. These include some of the simplest clustering schemes and they are well suited for a first course to introduce students to the basics of clustering and allow them to experiment with the computer. The sections related to estimation of the number of clusters and neural network implementations are bypassed.

Chapter 13 deals with hierarchical clustering algorithms. In a first course, only the general agglomerative scheme is considered with an emphasis on single link and complete link algorithms, based on matrix theory. Agglomerative algorithms based on graph theory concepts as well as the divisive schemes are bypassed.

Chapter 14 deals with clustering algorithms based on cost function optimization, using tools from differential calculus. Hard clustering and fuzzy and possibilistic schemes are considered, based on various types of cluster representatives, including point representatives, hyperplane representatives, and shell-shaped representatives. In a first course, most of these algorithms are bypassed, and emphasis is given to the isodata algorithm.

Chapter 15 features a high degree of modularity. It deals with clustering algorithms based on different ideas, which cannot be grouped under a single philosophy. Competitive learning, branch and bound, simulated annealing, and genetic algorithms are some of the schemes treated in this chapter. These are bypassed in a first course.

Chapter 16 deals with the clustering validity stage of a clustering procedure. It contains rather advanced concepts and is omitted in a first course. Emphasis is given to the definitions of internal, external, and relative criteria and the random hypotheses used in each case. Indices, adopted in the framework of external and internal criteria, are presented, and examples are provided showing the use of these indices.

Syntactic pattern recognition methods are not treated in this book. Syntactic pattern recognition methods differ in philosophy from the methods discussed in this book and, in general, are applicable to different types of problems. In syntactic pattern recognition, the structure of the patterns is of paramount importance and pattern recognition is performed on the basis of a set of pattern *primitives*, a set of rules in the form of a *grammar*, and a recognizer called *automaton*. Thus, we were faced with a dilemma: either to increase the size of the book substantially, or to provide a short overview (which, however, exists in a number of other books), or to omit it. The last option seemed to be the most sensible choice.

CHAPTER 2

CLASSIFIERS BASED ON BAYES DECISION THEORY

2.1 INTRODUCTION

This is the first chapter, out of three, dealing with the design of the classifier in a pattern recognition system. The approach to be followed builds upon probabilistic arguments stemming from the statistical nature of the generated features. As has already been pointed out in the previous introductory chapter, this is due to the statistical variation of the patterns as well as to the noise in the measuring sensors. Adopting this reasoning as our kickoff point, we will design classifiers that classify an unknown pattern in the most probable of the classes. Thus, our task now becomes to define what "most probable" means.

Given a classification task of M classes, $\omega_1, \omega_2, \ldots, \omega_M$, and an unknown pattern, which is represented by a feature vector $\boldsymbol{x}$, we form the M conditional probabilities $P(\omega_i|\boldsymbol{x}), i = 1, 2, \ldots, M$. Sometimes, these are also referred to as *a posteriori probabilities*. In words, each of them represents the probability that the unknown pattern belongs to the respective class ω_i, given that the corresponding feature vector takes the value $\boldsymbol{x}$. Who could then argue that these conditional probabilities are not sensible choices to quantify the term "most probable"? Indeed, the classifiers to be considered in this chapter compute either the maximum of these M values or, equivalently, the maximum of an appropriately defined function of them. The unknown pattern is then assigned to the class corresponding to this maximum.

The first task we are faced with is the computation of the conditional probabilities. The Bayes rule will once more prove its usefulness! In the following a major effort in this chapter will be devoted to techniques for estimating probability density functions (pdf's), based on the available experimental evidence, that is, the feature vectors corresponding to the patterns of the training set.

2.2 BAYES DECISION THEORY

We will initially focus on the two-class case. Let ω_1, ω_2 be the two classes in which our patterns belong. In the sequel, we assume that the *a priori probabilities*

$P(\omega_1)$, $P(\omega_2)$ are known. This is a very reasonable assumption, because even if they are not known, they can easily be estimated from the available training feature vectors. Indeed, if N is the total number of available training patterns, and N_1, N_2 of them belong to ω_1 and ω_2, respectively, then $P(\omega_1) \approx N_1/N$ and $P(\omega_2) \approx N_2/N$.

The other statistical quantities assumed to be known are the class-conditional probability density functions $p(\boldsymbol{x}|\omega_i), i = 1, 2$, describing the distribution of the feature vectors in each of the classes. If these are not known, they can also be estimated from the available training data, as we will discuss later on in this chapter. The pdf $p(\boldsymbol{x}|\omega_i)$ is sometimes referred to as the *likelihood function of* ω_i *with respect to* $\boldsymbol{x}$. Here we should stress the fact that an implicit assumption has been made. That is, the feature vectors can take any value in the l-dimensional feature space. In the case that feature vectors can take only discrete values, density functions $p(\boldsymbol{x}|\omega_i)$ become probabilities and will be denoted by $P(\boldsymbol{x}|\omega_i)$.

We now have all the ingredients to compute our conditional probabilities, as stated in the introduction. To this end, let us recall from our probability course basics the *Bayes rule* (Appendix A)

$$P(\omega_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_i)P(\omega_i)}{p(\boldsymbol{x})} \tag{2.1}$$

where $p(\boldsymbol{x})$ is the pdf of $\boldsymbol{x}$ and for which we have (Appendix A)

$$p(\boldsymbol{x}) = \sum_{i=1}^{2} p(\boldsymbol{x}|\omega_i)P(\omega_i) \tag{2.2}$$

The *Bayes classification rule* can now be stated as

$$\begin{aligned} &\text{If} \quad P(\omega_1|\boldsymbol{x}) > P(\omega_2|\boldsymbol{x}), \quad \boldsymbol{x} \text{ is classified to } \omega_1 \\ &\text{If} \quad P(\omega_1|\boldsymbol{x}) < P(\omega_2|\boldsymbol{x}), \quad \boldsymbol{x} \text{ is classified to } \omega_2 \end{aligned} \tag{2.3}$$

The case of equality is detrimental and the pattern can be assigned to either of the two classes. Using (2.1), the decision can equivalently be based on the inequalities

$$p(\boldsymbol{x}|\omega_1)P(\omega_1) \gtrless p(\boldsymbol{x}|\omega_2)P(\omega_2) \tag{2.4}$$

$p(\boldsymbol{x})$ is not taken into account, because it is the same for all classes and it does not affect the decision. Furthermore, if the *a priori* probabilities are equal, that is, $P(\omega_1) = P(\omega_2) = 1/2$, Eq. (2.4) becomes

$$p(\boldsymbol{x}|\omega_1) \gtrless p(\boldsymbol{x}|\omega_2) \tag{2.5}$$

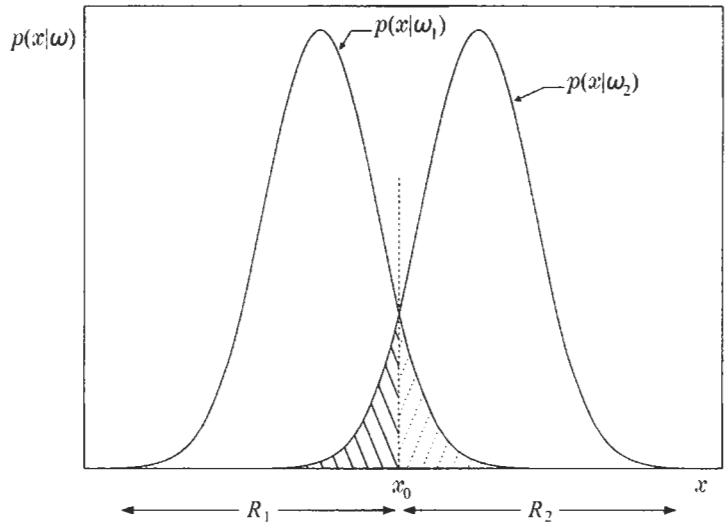


FIGURE 2.1: Example of the two regions R_1 and R_2 formed by the Bayesian classifier for the case of two equiprobable classes.

Thus, the search for the maximum now rests on the values of the conditional pdf's evaluated at $\boldsymbol{x}$. Figure 2.1 presents an example of two equiprobable classes and shows the variations of $p(x|\omega_i), i = 1, 2$, as functions of x for the simple case of a single feature ($l = 1$). The dotted line at x_0 is a threshold partitioning the feature space into two regions, R_1 and R_2. According to the Bayes decision rule, for all values of x in R_1 the classifier decides ω_1 and for all values in R_2 it decides ω_2. However, it is obvious from the figure that decision errors are unavoidable. Indeed, there is a finite probability for an x to lie in the R_2 region and at the same time to belong in class ω_1. Then our decision is in error. The same is true for points originating from class ω_2. It does not take much thought to see that the total probability, P_e, of committing a decision error is given by

$$2P_e = \int_{-\infty}^{x_0} p(x|\omega_2)\, dx + \int_{x_0}^{+\infty} p(x|\omega_1)\, dx \tag{2.6}$$

which is equal to the total shaded area under the curves in Figure 2.1. We have now touched on a very important issue. Our starting point to arrive at the Bayes classification rule was rather empirical, via our interpretation of the term "most probable." We will now see that this classification test, though simple in its formulation, has a much more sound mathematical interpretation.

Minimizing the Classification Error Probability

We will show that *the Bayesian classifier is optimal with respect to minimizing the classification error probability*. Indeed, the reader can easily verify, as an exercise, that moving the threshold away from x_0, in Figure 2.1, always increases the corresponding shaded area under the curves. Let us now proceed with a more formal proof.

Proof. Let R_1 be the region of the feature space in which we decide in favor of ω_1 and R_2 be the corresponding region for ω_2. Then an error is made if $\boldsymbol{x} \in R_1$ although it belongs to ω_2 or if $\boldsymbol{x} \in R_2$ although it belongs to ω_1. That is,

$$P_e = P(\boldsymbol{x} \in R_2, \omega_1) + P(\boldsymbol{x} \in R_1, \omega_2) \tag{2.7}$$

where $P(\cdot, \cdot)$ is the joint probability of two events. Recalling, once more, our probability basics (Appendix A), this becomes

$$\begin{aligned} P_e &= P(\boldsymbol{x} \in R_2|\omega_1)P(\omega_1) + P(\boldsymbol{x} \in R_1|\omega_2)P(\omega_2) \\ &= P(\omega_1)\int_{R_2} p(\boldsymbol{x}|\omega_1)\, d\boldsymbol{x} + P(\omega_2)\int_{R_1} p(\boldsymbol{x}|\omega_2)\, d\boldsymbol{x} \end{aligned} \tag{2.8}$$

or using the Bayes rule

$$P_e = \int_{R_2} P(\omega_1|\boldsymbol{x})p(\boldsymbol{x})\, d\boldsymbol{x} + \int_{R_1} P(\omega_2|\boldsymbol{x})p(\boldsymbol{x})\, d\boldsymbol{x} \tag{2.9}$$

It is now easy to see that the error is minimized if *the partitioning regions* R_1 and R_2 *of the feature space are chosen so that*

$$\begin{aligned} R_1&: P(\omega_1|\boldsymbol{x}) > P(\omega_2|\boldsymbol{x}) \\ R_2&: P(\omega_2|\boldsymbol{x}) > P(\omega_1|\boldsymbol{x}) \end{aligned} \tag{2.10}$$

Indeed, since the union of the regions R_1, R_2 covers all the space, from the definition of a probability density function we have that

$$\int_{R_1} P(\omega_1|\boldsymbol{x})p(\boldsymbol{x})\, d\boldsymbol{x} + \int_{R_2} P(\omega_1|\boldsymbol{x})p(\boldsymbol{x})\, d\boldsymbol{x} = P(\omega_1) \tag{2.11}$$

Combining Eqs. (2.9) and (2.11), we get

$$P_e = P(\omega_1) - \int_{R_1} (P(\omega_1|\boldsymbol{x}) - P(\omega_2|\boldsymbol{x}))\, p(\boldsymbol{x})\, d\boldsymbol{x} \tag{2.12}$$

This suggests that the probability of error is minimized if R_1 is the region of space in which $P(\omega_1|\boldsymbol{x}) > P(\omega_2|\boldsymbol{x})$. Then, R_2 becomes the region where the reverse is true. ■

So far, we have dealt with the simple case of two classes. Generalizations to the multiclass case are straightforward. In a classification task with M classes, $\omega_1, \omega_2, \ldots, \omega_M$, an unknown pattern, represented by the feature vector $\boldsymbol{x}$, is assigned to class ω_i if

$$P(\omega_i|\boldsymbol{x}) > P(\omega_j|\boldsymbol{x}) \quad \forall j \neq i \tag{2.13}$$

It turns out that such a choice also minimizes the classification error probability (Problem 2.1).

Minimizing the Average Risk

The classification error probability is not always the best criterion to be adopted for minimization. This is because it assigns the same importance to all errors. However, there are cases in which some errors may have more serious implications than others. Thus, in such cases it is more appropriate to assign a penalty term to weigh each error. Let us consider an M-class problem and let R_j, $j = 1, 2, \ldots, M$, be the regions of the feature space assigned to classes ω_j, respectively. Assume now that a feature vector $\boldsymbol{x}$ that belongs to class ω_k lies in R_i, $i \neq k$. Then we misclassify this vector in ω_i and an error is committed. A penalty term λ_{ki}, known as *loss*, is associated with this wrong decision. The matrix L, which has at its (k, i) location the corresponding penalty term, is known as the *loss matrix*.[1] The *risk or loss* associated with ω_k is defined as

$$r_k = \sum_{i=1}^{M} \lambda_{ki} \int_{R_i} p(\boldsymbol{x}|\omega_k)\, d\boldsymbol{x} \tag{2.14}$$

Observe that the integral is the overall probability of a feature vector from class ω_k being classified in ω_i. This probability is weighted by λ_{ki}. Our goal now is to choose the partitioning regions R_j so that the *average risk*

$$\begin{aligned} r &= \sum_{k=1}^{M} r_k P(\omega_k) \\ &= \sum_{i=1}^{M} \int_{R_i} \left(\sum_{k=1}^{M} \lambda_{ki}\, p(\boldsymbol{x}|\omega_k) P(\omega_k) \right) d\boldsymbol{x} \end{aligned} \tag{2.15}$$

[1] The terminology comes from the general decision theory.

is minimized. This is achieved if each of the integrals is minimized, which is equivalent to selecting partitioning regions so that

$$\boldsymbol{x} \in R_i \quad \text{if} \quad l_i \equiv \sum_{k=1}^{M} \lambda_{ki}\, p(\boldsymbol{x}|\omega_k) P(\omega_k) < l_j \equiv \sum_{k=1}^{M} \lambda_{kj}\, p(\boldsymbol{x}|\omega_k) P(\omega_k) \quad \forall j \neq i \tag{2.16}$$

It is obvious that if $\lambda_{ki} = 1 - \delta_{ki}$, where δ_{ki} is *Kronecker's delta* (0 if $k \neq i$ and 1 if $k = i$), then minimizing the average risk becomes equivalent to minimizing the classification error probability.

The two-class case. For this specific case we obtain

$$\begin{aligned} l_1 &= \lambda_{11}\, p(\boldsymbol{x}|\omega_1) P(\omega_1) + \lambda_{21}\, p(\boldsymbol{x}|\omega_2) P(\omega_2) \\ l_2 &= \lambda_{12} p(\boldsymbol{x}|\omega_1) P(\omega_1) + \lambda_{22} p(\boldsymbol{x}|\omega_2) P(\omega_2) \end{aligned} \tag{2.17}$$

We assign $\boldsymbol{x}$ to ω_1 if $l_1 < l_2$, that is,

$$(\lambda_{21} - \lambda_{22}) p(\boldsymbol{x}|\omega_2) P(\omega_2) < (\lambda_{12} - \lambda_{11}) p(\boldsymbol{x}|\omega_1) P(\omega_1) \tag{2.18}$$

It is natural to assume that $\lambda_{ij} > \lambda_{ii}$ (correct decisions are penalized much less than wrong ones). Adopting this assumption, the decision rule (2.16) for the two-class case now becomes

$$\boldsymbol{x} \in \omega_1 (\omega_2) \quad \text{if} \quad l_{12} \equiv \frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} > (<) \frac{P(\omega_2)}{P(\omega_1)} \frac{\lambda_{21} - \lambda_{22}}{\lambda_{12} - \lambda_{11}} \tag{2.19}$$

The ratio l_{12} is known as the *likelihood ratio* and the preceding test as the *likelihood ratio test*. Let us now investigate Eq. (2.19) a little further and consider the case of Figure 2.1. Assume that the loss matrix is of the form

$$L = \begin{pmatrix} 0 & \lambda_{12} \\ \lambda_{21} & 0 \end{pmatrix}$$

If misclassification of patterns that come from ω_2 is considered to have serious consequences, then we must choose $\lambda_{21} > \lambda_{12}$. Thus, patterns are assigned to class ω_2 if

$$p(\boldsymbol{x}|\omega_2) > p(\boldsymbol{x}|\omega_1) \frac{\lambda_{12}}{\lambda_{21}}$$

where $P(\omega_1) = P(\omega_2) = 1/2$ has been assumed. That is, $p(\boldsymbol{x}|\omega_1)$ is multiplied by a factor less than 1 and the effect of this is to move the threshold in Figure 2.1 to the left of x_0. In other words, region R_2 is increased while R_1 is decreased. The opposite would be true if $\lambda_{21} < \lambda_{12}$.

Example 2.1. In a two-class problem with a single feature x the pdf's are Gaussians with variance $\sigma^2 = 1/2$ for both classes and mean values 0 and 1, respectively, that is,

$$p(x|\omega_1) = \frac{1}{\sqrt{\pi}} \exp(-x^2)$$
$$p(x|\omega_2) = \frac{1}{\sqrt{\pi}} \exp(-(x-1)^2)$$

If $P(\omega_1) = P(\omega_2) = 1/2$, compute the threshold value x_0 (a) for minimum error probability and (b) for minimum risk if the loss matrix is

$$L = \begin{pmatrix} 0 & 0.5 \\ 1.0 & 0 \end{pmatrix}$$

Taking into account the shape of the Gaussian function graph (Appendix A), the threshold for the minimum probability case will be

$$x_0 : \exp(-x^2) = \exp(-(x-1)^2)$$

Taking the logarithm of both sides, we end up with $x_0 = 1/2$. In the minimum risk case we get

$$x_0 : \exp(-x^2) = 2\exp(-(x-1)^2)$$

or $x_0 = (1 - \ln 2)/2 < 1/2$; that is, the threshold moves to the left of $1/2$. If the two classes are not equiprobable, then it is easily verified that if $P(\omega_1) > (<) P(\omega_2)$ the threshold moves to the right (left). That is, we expand the region in which we decide in favor of the most probable class, since it is better to make less errors for the most probable class.

2.3 DISCRIMINANT FUNCTIONS AND DECISION SURFACES

It is by now clear that minimizing either the risk or the error probability (and also some other costs; see, for example, Problem 2.6) is equivalent to partitioning the feature space into M regions, for a task with M classes. If regions R_i, R_j happen to be contiguous, then they are separated by a *decision surface* in the multidimensional feature space. For the minimum error probability case, this is described by the equation

$$P(\omega_i|\boldsymbol{x}) - P(\omega_j|\boldsymbol{x}) = 0 \tag{2.20}$$

From the one side of the surface this difference is positive and from the other it is negative. Sometimes, instead of working directly with probabilities (or risk functions), it may be more convenient, from a mathematical point of view, to work with an equivalent function of them, for example, $g_i(\boldsymbol{x}) \equiv f(P(\omega_i|\boldsymbol{x}))$,

where $f(\cdot)$ is a monotonically increasing function. $g_i(\boldsymbol{x})$ is known as a *discriminant function*. The decision test (2.13) is now stated as

$$\text{classify } \boldsymbol{x} \text{ in } \omega_i \quad \text{if} \quad g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \quad \forall j \neq i \tag{2.21}$$

The decision surfaces, separating contiguous regions, are described by

$$g_{ij}(\boldsymbol{x}) \equiv g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = 0, \quad i, j = 1, 2, \ldots, M, \quad i \neq j \tag{2.22}$$

So far, we have approached the classification problem via Bayesian probabilistic arguments and the goal was to minimize the classification error probability or the risk. However, as we will soon see, not all problems are well suited to such approaches. For example, in many cases the involved pdf's are complicated and their estimation is not an easy task. In such cases it may be preferable to compute decision surfaces *directly by means of alternative costs*, and this will be our focus in Chapters 3 and 4. Such approaches give rise to discriminant functions and decision surfaces, which are entities with no (necessary) relation to Bayesian classification, and they are, in general, suboptimal with respect to Bayesian classifiers.

In the following we will focus on a particular family of decision surfaces associated with the Bayesian classification for the specific case of Gaussian density functions.

2.4 BAYESIAN CLASSIFICATION FOR NORMAL DISTRIBUTIONS

One of the most commonly encountered probability density functions in practice is the Gaussian or normal density function. The major reasons for its popularity are its computational tractability and the fact that it models adequately a large number of cases. Assume now that the likelihood functions of ω_i with respect to $\boldsymbol{x}$ in the l-dimensional feature space follow the general multivariate normal density (Appendix A)

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{(2\pi)^{l/2}|\sum_i|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \sum_i^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i)\right), \quad i = 1, \ldots, M \tag{2.23}$$

where $\boldsymbol{\mu}_i = E[\boldsymbol{x}]$ is the mean value of the ω_i class and Σ_i the $l \times l$ *covariance matrix* (Appendix A) defined as

$$\Sigma_i = E[(\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^T] \tag{2.24}$$

$|\Sigma_i|$ denotes the determinant of Σ_i and $E[\cdot]$ the mean (or average or expected) value of a random variable. Sometimes, the symbol $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is used to denote a

Gaussian pdf with mean value $\boldsymbol{\mu}$ and covariance Σ. Our goal, now, is to design the Bayesian classifier. Because of the exponential form of the involved densities, it is preferable to work with the following discriminant functions, which involve the (monotonic) logarithmic function $\ln(\cdot)$:

$$g_i(\boldsymbol{x}) = \ln(p(\boldsymbol{x}|\omega_i)P(\omega_i)) = \ln p(\boldsymbol{x}|\omega_i) + \ln P(\omega_i) \tag{2.25}$$

or

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) + \ln P(\omega_i) + c_i \tag{2.26}$$

where c_i is a constant equal to $-(l/2)\ln 2\pi - (1/2)\ln|\Sigma_i|$. Expanding, we obtain

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^T\Sigma_i^{-1}\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T\Sigma_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\boldsymbol{\mu}_i^T\Sigma_i^{-1}\boldsymbol{\mu}_i + \frac{1}{2}\boldsymbol{\mu}_i^T\Sigma_i^{-1}\boldsymbol{x} + \ln P(\omega_i) + c_i \tag{2.27}$$

In general, this is a nonlinear quadratic form. Take, for example, the case of $l = 2$ and assume that

$$\Sigma_i = \begin{pmatrix} \sigma_i^2 & 0 \\ 0 & \sigma_i^2 \end{pmatrix}$$

Then (2.27) becomes

$$g_i(\boldsymbol{x}) = -\frac{1}{2\sigma_i^2}(x_1^2 + x_2^2) + \frac{1}{\sigma_i^2}(\mu_{i1}x_1 + \mu_{i2}x_2) - \frac{1}{2\sigma_i^2}(\mu_{i1}^2 + \mu_{i2}^2) + \ln P(\omega_i) + c_i \tag{2.28}$$

and obviously the associated decision curves $g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = 0$ are *quadrics* (i.e., ellipsoids, parabolas, hyperbolas, pairs of lines). That is, in such cases, the Bayesian classifier is a *quadric classifier*, in the sense that the partition of the feature space is performed via quadric decision surfaces. For $l > 2$ the decision surfaces are *hyperquadrics*. Figure 2.2a and 2.2b show the decision curves corresponding to $P(\omega_1) = P(\omega_2)$, $\boldsymbol{\mu}_1 = [0, 0]^T$ and $\boldsymbol{\mu}_2 = [1, 0]^T$. The covariance matrices for the two classes are

$$\Sigma_1 = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.15 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 0.2 & 0.0 \\ 0.0 & 0.25 \end{bmatrix}$$

for the case of Figure 2.2a and

$$\Sigma_1 = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.15 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 0.15 & 0.0 \\ 0.0 & 0.1 \end{bmatrix}$$

for the case of Figure 2.2b.

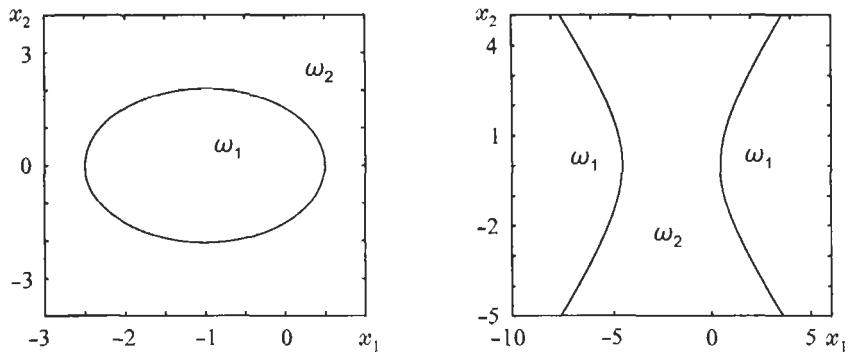


FIGURE 2.2: Quadric decision curves.

Decision Hyperplanes

The only quadratic contribution in (2.27) comes from the term $\boldsymbol{x}^T \Sigma_i^{-1} \boldsymbol{x}$. If we now assume that the covariance matrix is the same in all classes, that is, $\Sigma_i = \Sigma$, the quadratic term will be the same in all discriminant functions. Hence, it does not enter into the comparisons for computing the maximum and it cancels out in the decision surface equations. The same is true for the constants c_i. Thus, they can be omitted and we may redefine $g_i(\boldsymbol{x})$ as

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} \tag{2.29}$$

where

$$\boldsymbol{w}_i = \Sigma^{-1} \boldsymbol{\mu}_i \tag{2.30}$$

and

$$w_{i0} = \ln P(\omega_i) - \frac{1}{2} \boldsymbol{\mu}_i^T \Sigma^{-1} \boldsymbol{\mu}_i \tag{2.31}$$

Hence $g_i(\boldsymbol{x})$ is a *linear function* of $\boldsymbol{x}$ and the respective decision surfaces are *hyperplanes*. Let us investigate this a bit more.

- *Diagonal covariance matrix with equal elements*: Assume that the individual features, constituting the feature vector, are *mutually uncorrelated and of the same variance* ($E[(x_i - \mu_i)(x_j - \mu_j)] = \sigma^2 \delta_{ij}$). Then, as discussed in

Appendix A, $\Sigma = \sigma^2 I$, where I is the l-dimensional identity matrix, and (2.29) becomes

$$g_i(\boldsymbol{x}) = \frac{1}{\sigma^2}\boldsymbol{\mu}_i^T \boldsymbol{x} + w_{i0} \tag{2.32}$$

Thus, the corresponding decision hyperplanes can now be written as (verify it)

$$g_{ij}(\boldsymbol{x}) \equiv g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = \boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{x}_0) = 0 \tag{2.33}$$

where

$$\boldsymbol{w} = \boldsymbol{\mu}_i - \boldsymbol{\mu}_j \tag{2.34}$$

and

$$\boldsymbol{x}_0 = \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) - \sigma^2 \ln\left(\frac{P(\omega_i)}{P(\omega_j)}\right)\frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2} \tag{2.35}$$

where $\|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_l^2}$ is the Euclidean norm of $\boldsymbol{x}$. Thus, the decision surface is a *hyperplane* passing through the point $\boldsymbol{x}_0$. Obviously, if $P(\omega_i) = P(\omega_j)$, then $\boldsymbol{x}_0 = \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)$, and the hyperplane passes through the mean of $\boldsymbol{\mu}_i, \boldsymbol{\mu}_j$.

The geometry is illustrated in Figure 2.3 for the two-dimensional case. We observe that the decision hyperplane (straight line) is orthogonal to $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j$. Indeed, for any point $\boldsymbol{x}$ lying on the decision hyperplane, the vector $\boldsymbol{x} - \boldsymbol{x}_0$ also lies on the hyperplane and

$$g_{ij}(\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{x}_0) = (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T(\boldsymbol{x} - \boldsymbol{x}_0) = 0$$

That is, $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j$ is orthogonal to the decision hyperplane. Another point to be stressed is that the hyperplane is located closer to $\boldsymbol{\mu}_i$ ($\boldsymbol{\mu}_j$) if $P(\omega_i) < P(\omega_j)$ ($P(\omega_i) > P(\omega_j)$). Furthermore, if σ^2 is small with respect to $\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|$, the location of the hyperplane is rather insensitive to the values of $P(\omega_i)$, $P(\omega_j)$. This is expected, because small variance indicates that the random vectors are clustered within a small radius around their mean values. Thus a small shift of the decision hyperplane has a small effect on the result.

Figure 2.4 illustrates this. For each class, the circles around the means indicate regions where samples have a high probability, say 98%, of being found. The case of Figure 2.4a corresponds to small variance and that of Figure 2.4b to large variance. No doubt the location of the decision hyperplane in Figure 2.4b is much more critical than that in Figure 2.4a.

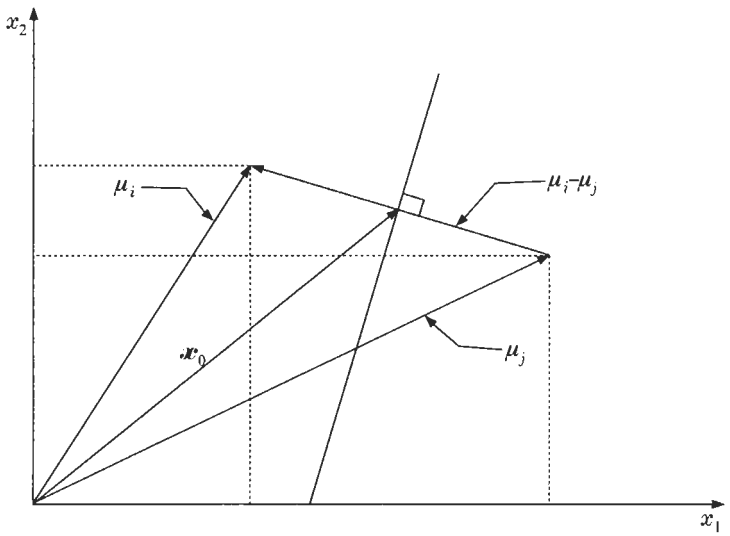


FIGURE 2.3: Decision line for two classes and normally distributed vectors with $\Sigma = \sigma^2 I$.

- *Nondiagonal covariance matrix*: Following algebraic arguments similar to those used before, we end up with hyperplanes described by

$$g_{ij}(\boldsymbol{x}) = \boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{x}_0) = 0 \tag{2.36}$$

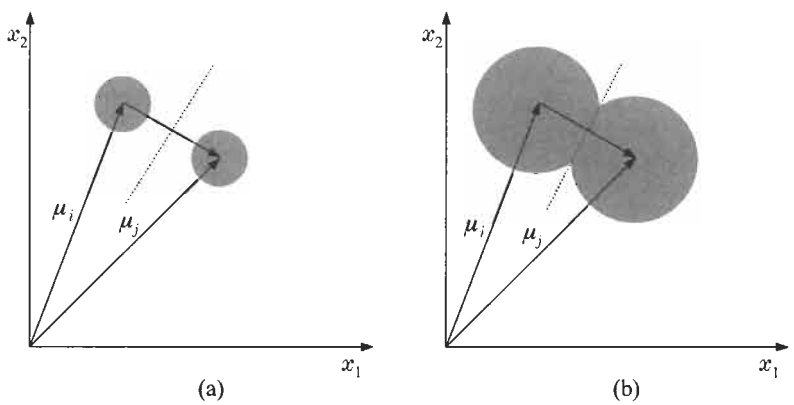


FIGURE 2.4: Decision line (a) for compact and (b) for noncompact classes.

where

$$\boldsymbol{w} = \Sigma^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) \tag{2.37}$$

and

$$\boldsymbol{x}_0 = \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) - \ln\left(\frac{P(\omega_i)}{P(\omega_j)}\right)\frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2_{\Sigma^{-1}}} \tag{2.38}$$

where $\|\boldsymbol{x}\|_{\Sigma^{-1}} \equiv (\boldsymbol{x}^T\Sigma^{-1}\boldsymbol{x})^{1/2}$ is the so-called Σ^{-1} norm of $\boldsymbol{x}$. The comments made before for the case of the diagonal covariance matrix are still valid, with one exception. *The decision hyperplane is no longer orthogonal to the vector $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j$ but to its linear transformation $\Sigma^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$.*

Minimum Distance Classifiers

We will now view this from a slightly different angle. Assuming equiprobable classes with the same covariance matrix, $g_i(\boldsymbol{x})$ in (2.26) is simplified to

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) \tag{2.39}$$

where constants have been neglected.

- $\Sigma = \sigma^2 I$: In this case maximum $g_i(\boldsymbol{x})$ implies minimum

$$\textit{Euclidean distance:} \quad d_\epsilon = \|\boldsymbol{x} - \boldsymbol{\mu}_i\| \tag{2.40}$$

Thus, feature vectors are assigned to classes according to their Euclidean distance from the respective mean points. Can you verify that this result ties in with the geometry of the hyperplanes discussed before?

Figure 2.5a shows curves of equal distance $d_\epsilon = c$ from the mean points of each class. They are obviously circles of radius c (hyperspheres in the general case).

- Nondiagonal Σ: For this case maximizing $g_i(\boldsymbol{x})$ is equivalent to minimizing the Σ^{-1} norm, known as the

$$\textit{Mahalanobis distance:} \quad d_m = \left((\boldsymbol{x} - \boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i)\right)^{1/2} \tag{2.41}$$

In this case the constant distance $d_m = c$ curves are ellipses (hyperellipses). Indeed, the covariance matrix is symmetric and, as discussed in Appendix B, it can always be diagonalized by a unitary transform

$$\Sigma = \Phi\Lambda\Phi^T \tag{2.42}$$

where $\Phi^T = \Phi^{-1}$ and Λ is the diagonal matrix whose elements are the eigenvalues of Σ.Φ has as its columns the corresponding (orthonormal)

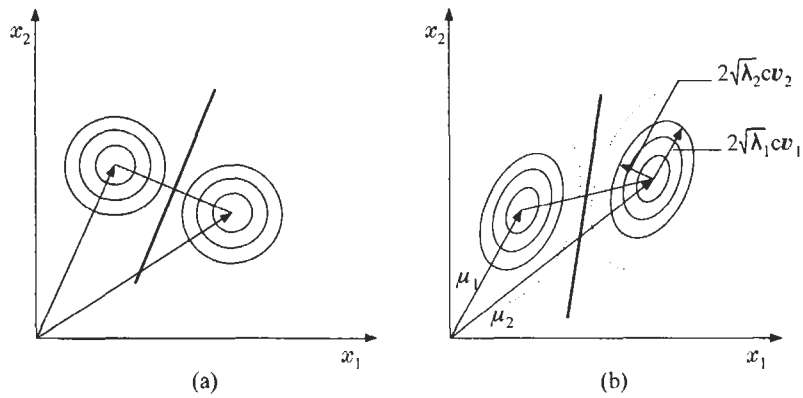


FIGURE 2.5: Curves of (a) equal Euclidean distance and (b) equal Mahalanobis distance from the mean points of each class.

eigenvectors of Σ

$$\Phi = [v_1, v_2, \ldots, v_l] \tag{2.43}$$

Combining (2.41) and (2.42), we obtain

$$(x - \mu_i)^T \Phi \Lambda^{-1} \Phi^T (x - \mu_i) = c^2 \tag{2.44}$$

Define $x' = \Phi^T x$. The coordinates of x' are equal to $v_k^T x$, $k = 1, 2, \ldots, l$, that is, the projections of x onto the eigenvectors. In other words, they are the coordinates of x with respect to a new coordinate system whose axes are determined by v_k, $k = 1, 2, \ldots, l$. Equation (2.44) can now be written as

$$\frac{(x'_1 - \mu'_{i1})^2}{\lambda_1} + \cdots + \frac{(x'_l - \mu'_{il})^2}{\lambda_l} = c^2 \tag{2.45}$$

This is the equation of a hyperellipsoid in the new coordinate system. Figure 2.5b shows the $l = 2$ case. The center of mass of the ellipse is at μ_i, and the principal axes are aligned with the corresponding eigenvectors and have lengths $2\sqrt{\lambda_k}c$, respectively. Thus, *all points having the same Mahalanobis distance from a specific point are located on an ellipse.*

Example 2.2. In a two-class, two-dimensional classification task the feature vectors are generated by two normal distributions sharing the same covariance matrix

$$\Sigma = \begin{bmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{bmatrix}$$

and the mean vectors are $\mu_1 = [0, 0]^T$, $\mu_2 = [3, 3]^T$, respectively.

(a) Classify the vector $[1.0, 2.2]^T$ according to the Bayesian classifier.

It suffices to compute the Mahalanobis distance of $[1.0, 2.2]^T$ from the two mean vectors. Thus,

$$d_m^2(\boldsymbol{\mu}_1, \boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{\mu}_1)^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_1)$$
$$= [1.0, 2.2] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.2 \end{bmatrix} = 2.952$$

Similarly,

$$d_m^2(\boldsymbol{\mu}_2, \boldsymbol{x}) = [-2.0, -0.8] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} -2.0 \\ -0.8 \end{bmatrix} = 3.672 \tag{2.46}$$

Thus, the vector is assigned to the class with mean vector $[0, 0]^T$. *Notice that the given vector* $[1.0, 2.2]^T$ *is closer to* $[3, 3]^T$ *with respect to the Euclidean distance.*

(b) Compute the principal axes of the ellipse centered at $[0, 0]^T$ that corresponds to a constant Mahalanobis distance $d_m = \sqrt{2.952}$ from the center.

To this end, we first calculate the eigenvalues of Σ.

$$\det\left(\begin{bmatrix} 1.1 - \lambda & 0.3 \\ 0.3 & 1.9 - \lambda \end{bmatrix}\right) = \lambda^2 - 3\lambda + 2 = 0$$

or $\lambda_1 = 1$ and $\lambda_2 = 2$. To compute the eigenvectors we substitute these values into the equation

$$(\Sigma - \lambda I)\boldsymbol{v} = \boldsymbol{0}$$

and we obtain the unit norm eigenvectors

$$\boldsymbol{v}_1 = \begin{bmatrix} \frac{3}{\sqrt{10}} \\ -\frac{1}{\sqrt{10}} \end{bmatrix}, \qquad \boldsymbol{v}_2 = \begin{bmatrix} \frac{1}{\sqrt{10}} \\ \frac{3}{\sqrt{10}} \end{bmatrix}$$

It can easily be seen that they are mutually orthogonal. The principal axes of the ellipse are parallel to $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ and have lengths 3.436 and 4.859, respectively.

2.5 ESTIMATION OF UNKNOWN PROBABILITY DENSITY FUNCTIONS

So far, we have assumed that the probability density functions are known. However, this is not the most common case. In many problems, the underlying pdf has to be estimated from the available data. There are various ways to approach the problem. Sometimes we may know the type of the pdf (e.g., Gaussian, Rayleigh) but we do not know certain parameters, such as the mean value or the variance. In contrast, in other cases we may not have information about the type of the pdf but we may know certain statistical parameters, such as the mean value and

the variance. Depending on the available information, different approaches can be adopted. This will be our focus in the next subsections.

2.5.1 Maximum Likelihood Parameter Estimation

Let us consider an M-class problem with feature vectors distributed according to $p(\boldsymbol{x}|\omega_i), i = 1, 2, \ldots, M$. We assume that these likelihood functions are given in a *parametric* form and that the corresponding parameters form the vectors $\boldsymbol{\theta}_i$ which are unknown. To show the dependence on $\boldsymbol{\theta}_i$ we write $p(\boldsymbol{x}|\omega_i; \boldsymbol{\theta}_i)$. Our goal is to estimate the unknown parameters using a set of known feature vectors in each class. If we further assume that data from one class do not affect the parameter estimation of the others, we can formulate the problem independent of classes and simplify our notation. At the end, one has to solve one such problem for each class independently.

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ be random samples drawn from pdf $p(\boldsymbol{x}; \boldsymbol{\theta})$. We form the joint pdf $p(X; \boldsymbol{\theta})$, where $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ is the set of the samples. Assuming *statistical independence* between the different samples we have

$$p(X; \boldsymbol{\theta}) \equiv p(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N; \boldsymbol{\theta}) = \prod_{k=1}^{N} p(\boldsymbol{x}_k; \boldsymbol{\theta}) \tag{2.47}$$

This is a function of $\boldsymbol{\theta}$ and it is also known as the likelihood function of $\boldsymbol{\theta}$ with respect to X. *The maximum likelihood (ML) method estimates $\boldsymbol{\theta}$ so that the likelihood function takes its maximum value*, that is,

$$\hat{\boldsymbol{\theta}}_{ML} = arg \max_{\boldsymbol{\theta}} \prod_{k=1}^{N} p(\boldsymbol{x}_k; \boldsymbol{\theta}) \tag{2.48}$$

A necessary condition that $\hat{\boldsymbol{\theta}}_{ML}$ must satisfy in order to be a maximum is the gradient of the likelihood function with respect to $\boldsymbol{\theta}$ to be zero, that is

$$\frac{\partial \prod_{k=1}^{N} p(\boldsymbol{x}_k; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{0} \tag{2.49}$$

Because of the monotonicity of the logarithmic function we define the *loglikelihood function* as

$$L(\boldsymbol{\theta}) \equiv \ln \prod_{k=1}^{N} p(\boldsymbol{x}_k; \boldsymbol{\theta}) \tag{2.50}$$

and (2.49) is equivalent to

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{N} \frac{\partial \ln p(\boldsymbol{x}_k; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{N} \frac{1}{p(\boldsymbol{x}_k; \boldsymbol{\theta})} \frac{\partial p(\boldsymbol{x}_k; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{0} \tag{2.51}$$

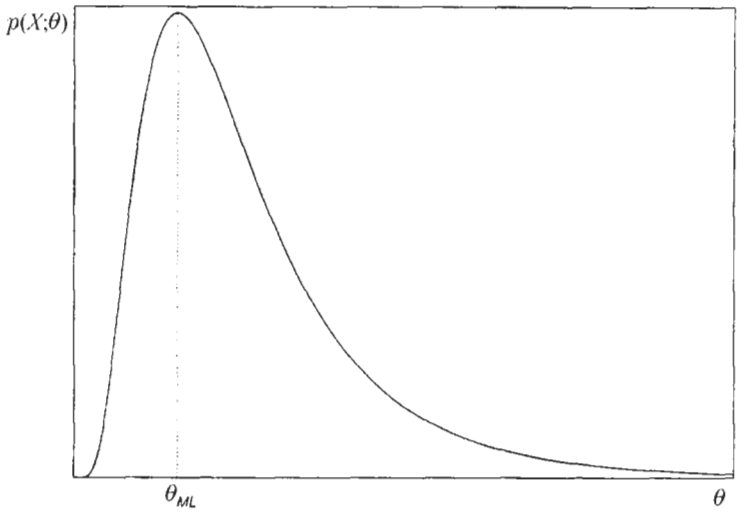


FIGURE 2.6: Maximum likelihood estimator.

Figure 2.6 illustrates the method for the single unknown parameter case. The ML estimate corresponds to the peak of the (log) likelihood function.

Maximum likelihood estimation has some very desirable properties. If $\boldsymbol{\theta}_0$ is the true value of the unknown parameter in $p(\boldsymbol{x}; \boldsymbol{\theta})$, it can be shown that under generally valid conditions the following are true [Papo 91]

- The ML estimate is *asymptotically unbiased*, which by definition means that

$$\lim_{N \to \infty} E[\hat{\boldsymbol{\theta}}_{ML}] = \boldsymbol{\theta}_0 \tag{2.52}$$

  Alternatively, we say that the estimate *converges in the mean* to the true value. The meaning of this is as follows. The estimate $\hat{\boldsymbol{\theta}}_{ML}$ is itself a random vector, because for different sample sets X different estimates will result. An estimate is called *unbiased* if its mean is the true value of the unknown parameter. In the ML case this is true only asymptotically ($N \to \infty$).
- The ML estimate is *asymptotically consistent*, that is, it satisfies

$$\lim_{N \to \infty} \text{prob}\{\|\hat{\boldsymbol{\theta}}_{ML} - \boldsymbol{\theta}_0\| \leq \epsilon\} = 1 \tag{2.53}$$

  where ϵ is arbitrarily small. Alternatively, we say that the estimate converges *in probability*. In other words, for large N it is highly probable that the resulting estimate will be arbitrarily close to the true value. A stronger

condition for consistency is also true:

$$\lim_{N\to\infty} E[\|\hat{\boldsymbol{\theta}}_{ML} - \boldsymbol{\theta}_0\|^2] = 0 \tag{2.54}$$

In such cases we say that the estimate converges in the *mean square*. In words, for large N, the variance of the ML estimates tends to zero.

Consistency is very important for an estimator, because it may be unbiased but the resulting estimates exhibit large variations around the mean. In such cases we have little confidence in the result from a single set X.

- The ML estimate is asymptotically *efficient*; that is, it achieves the Cramer–Rao lower bound (Appendix A). This is the lowest value of variance, which *any* estimate can achieve.
- The pdf of the ML estimate as $N \to \infty$ approaches the Gaussian distribution with mean $\boldsymbol{\theta}_0$ [Cram 46]. This property is an offspring of (a) the central limit theorem (Appendix A) and (b) the fact that the ML estimate is related to the *sum* of random variables, that is, $\partial \ln(p(\boldsymbol{x}_k; \boldsymbol{\theta}))/\partial\boldsymbol{\theta}$ (Problem 2.16).

In summary, the ML estimator is unbiased, is normally distributed, and has the minimum possible variance. However, all these nice properties are valid *only* for large values of N.

Example 2.3. Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ be vectors stemmed from a normal distribution with known covariance matrix and unknown mean, that is,

$$p(\boldsymbol{x}_k; \boldsymbol{\mu}) = \frac{1}{(2\pi)^{l/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}_k - \boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}_k - \boldsymbol{\mu})\right)$$

For N available samples we have

$$L(\boldsymbol{\mu}) \equiv \ln \prod_{k=1}^{N} p(\boldsymbol{x}_k; \boldsymbol{\mu}) = -\frac{N}{2}\ln((2\pi)^l|\Sigma|) - \frac{1}{2}\sum_{k=1}^{N}(\boldsymbol{x}_k - \boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}_k - \boldsymbol{\mu}) \tag{2.55}$$

Taking the gradient with respect to $\boldsymbol{\mu}$, we obtain

$$\frac{\partial L(\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} \equiv \begin{bmatrix} \frac{\partial L}{\partial \mu_1} \\ \frac{\partial L}{\partial \mu_2} \\ \vdots \\ \frac{\partial L}{\partial \mu_l} \end{bmatrix} = \sum_{k=1}^{N} \Sigma^{-1}(\boldsymbol{x}_k - \boldsymbol{\mu}) = \boldsymbol{0} \tag{2.56}$$

or

$$\hat{\boldsymbol{\mu}}_{ML} = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{x}_k \tag{2.57}$$

That is, the ML estimate of the mean, for Gaussian densities, is the sample mean. However, this very "natural approximation" is not necessarily ML optimal for non-Gaussian density functions.

2.5.2 Maximum A Posteriori Probability Estimation

For the derivation of the maximum likelihood estimate we considered $\boldsymbol{\theta}$ as an unknown parameter. In this subsection we will consider it as a random vector, and we will estimate its value on the condition that samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ have occurred. Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$. Our starting point is $p(\boldsymbol{\theta}|X)$. From our familiar Bayes theorem we have

$$p(\boldsymbol{\theta})p(X|\boldsymbol{\theta}) = p(X)p(\boldsymbol{\theta}|X) \tag{2.58}$$

or

$$p(\boldsymbol{\theta}|X) = \frac{p(\boldsymbol{\theta})p(X|\boldsymbol{\theta})}{p(X)} \tag{2.59}$$

The *maximum a posteriori probability* (MAP) estimate $\hat{\boldsymbol{\theta}}_{MAP}$ is defined at the point where $p(\boldsymbol{\theta}|X)$ becomes maximum,

$$\hat{\boldsymbol{\theta}}_{MAP}: \frac{\partial}{\partial \boldsymbol{\theta}} p(\boldsymbol{\theta}|X) = \mathbf{0} \quad \text{or} \quad \frac{\partial}{\partial \boldsymbol{\theta}}(p(\boldsymbol{\theta})p(X|\boldsymbol{\theta})) = \mathbf{0} \tag{2.60}$$

The difference between the ML and the MAP estimates lies in the involvement of $p(\boldsymbol{\theta})$ in the latter case. If we assume that this obeys the uniform distribution, that is, is constant for all $\boldsymbol{\theta}$, both estimates yield identical results. This is also approximately true if $p(\boldsymbol{\theta})$ exhibits small variation. However, in the general case, the two methods yield different results. Figures 2.7a and 2.7b illustrate the two cases.

Example 2.4. Let us assume that in the previous example 2.3 the unknown mean vector $\boldsymbol{\mu}$ is known to be normally distributed as

$$p(\boldsymbol{\mu}) = \frac{1}{(2\pi)^{l/2}\sigma_\mu^l} \exp\left(-\frac{1}{2}\frac{\|\boldsymbol{\mu} - \boldsymbol{\mu}_0\|^2}{\sigma_\mu^2}\right)$$

The MAP estimate is given by the solution of

$$\frac{\partial}{\partial \boldsymbol{\mu}} \ln\left(\prod_{k=1}^{N} p(\boldsymbol{x}_k|\boldsymbol{\mu})p(\boldsymbol{\mu})\right) = \mathbf{0}$$

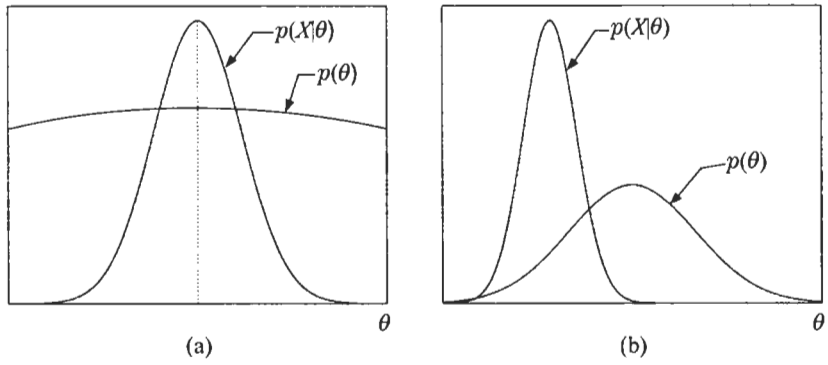


FIGURE 2.7: ML and MAP estimates of θ will be approximately the same in (a) and different in (b).

or, for $\Sigma = \sigma^2 I$,

$$\sum_{k=1}^{N} \frac{1}{\sigma^2}(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}}) - \frac{1}{\sigma_\mu^2}(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}_0) = \mathbf{0} \Rightarrow$$

$$\hat{\boldsymbol{\mu}}_{MAP} = \frac{\boldsymbol{\mu}_0 + \frac{\sigma_\mu^2}{\sigma^2}\sum_{k=1}^{N} \boldsymbol{x}_k}{1 + \frac{\sigma_\mu^2}{\sigma^2} N}$$

We observe that if $\frac{\sigma_\mu^2}{\sigma^2} \gg 1$, that is, the variance σ_μ^2 is very large and the corresponding Gaussian is very wide with little variation over the range of interest, then

$$\hat{\boldsymbol{\mu}}_{MAP} \approx \hat{\boldsymbol{\mu}}_{ML} = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{x}_k$$

Furthermore, observe that this is also the case for $N \to \infty$, irrespective of the values of the variances. Thus, the MAP estimate tends asymptotically to the ML one. This is a more general result. For large values of N, the likelihood term $\prod_{k=1}^{N} p(\boldsymbol{x}_k|\boldsymbol{\mu})$ becomes sharply peaked around the true value (of the unknown parameter) and is the term that basically determines where the maximum occurs. This can be better understood by mobilizing the properties of the ML estimate given before.

2.5.3 Bayesian Inference

Both methods considered in the preceding subsections compute a specific estimate of the unknown parameter vector $\boldsymbol{\theta}$. In the current method a different path is adopted. Given the set X of the N training vectors and the *a priori* information

about the pdf $p(\boldsymbol{\theta})$, the goal is to compute the conditional pdf $p(\boldsymbol{x}|X)$. After all, this is what we actually need to know. To this end, and making use of known identities from our statistics basics, we have the following set of relations at our disposal:

$$p(\boldsymbol{x}|X) = \int p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|X)\, d\boldsymbol{\theta} \tag{2.61}$$

with

$$p(\boldsymbol{\theta}|X) = \frac{p(X|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(X)} = \frac{p(X|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(X|\boldsymbol{\theta})p(\boldsymbol{\theta})\, d\boldsymbol{\theta}} \tag{2.62}$$

$$p(X|\boldsymbol{\theta}) = \prod_{k=1}^{N} p(\boldsymbol{x}_k|\boldsymbol{\theta}) \tag{2.63}$$

Once more, Eq. (2.63) presupposes statistical independence among the training samples. No doubt, a major drawback of this procedure is its complexity. Analytical solutions are feasible only in special cases.

Remarks

- If $p(\boldsymbol{\theta}|X)$ in Eq. (2.62) is sharply peaked at a $\hat{\boldsymbol{\theta}}$ and we treat it as a delta function, Eq. (2.61) becomes $p(\boldsymbol{x}|X) \approx p(\boldsymbol{x}|\hat{\boldsymbol{\theta}})$; that is, the parameter estimate is approximately equal to the MAP estimate. This happens, for example, if $p(X|\boldsymbol{\theta})$ is concentrated around a sharp peak and $p(\boldsymbol{\theta})$ is broad enough around this peak. Then the resulting estimate approximates the ML one.
- Further insight into the method can be gained by focusing on the following simple example. Let $p(x|\mu)$ be a univariate Gaussian $\mathcal{N}(\mu, \sigma^2)$ with unknown parameter the mean, which is also assumed to follow a Gaussian $\mathcal{N}(\mu_0, \sigma_0^2)$. It is a matter of simple algebra (Problem 2.22) to show that, given a number N of samples, $p(\mu|X)$ turns out to be also Gaussian, with mean

$$\mu_N = \frac{N\sigma_0^2\bar{x} + \sigma^2\mu_0}{N\sigma_0^2 + \sigma^2} \tag{2.64}$$

and variance

$$\sigma_N^2 = \frac{\sigma^2\sigma_0^2}{N\sigma_0^2 + \sigma^2} \tag{2.65}$$

where $\bar{x} = \frac{1}{N}\sum_{k=1}^{N} x_k$. Letting N vary from 1 to ∞, we generate a sequence of Gaussians $\mathcal{N}(\mu_N, \sigma_N^2)$, whose mean values move away from μ_0 and tend, in the limit, to the sample mean $\bar{x}$. Furthermore, their variance

keeps decreasing at the rate σ^2/N for large N. Hence, for large values of N, $p(\mu|X)$ becomes sharply peaked around $\bar{x}$. Thus, recalling what we have said before, the Bayesian inference estimator tends to the ML one (Problem 2.22). This is a more general property valid for most of the pdf's used in practice and it is a consequence of the convergence of the likelihood term $\prod_{k=1}^{N} p(\boldsymbol{x}_k|\boldsymbol{\theta})$ to a delta function as N tends to $+\infty$. *Thus, all three methods considered so far result, asymptotically, in the same estimate. However, the results are different for small numbers N of training samples.*

2.5.4 Maximum Entropy Estimation

The concept of *entropy* is known from Shannon's information theory. It is a measure of the uncertainty concerning an event and, from another viewpoint, a measure of randomness of the messages (feature vectors in our case) occurring at the output of a system. If $p(\boldsymbol{x})$ is the density function, the associated entropy H is given by

$$H = -\int_{\boldsymbol{x}} p(\boldsymbol{x}) \ln p(\boldsymbol{x})\, d\boldsymbol{x} \tag{2.66}$$

Assume now that $p(\boldsymbol{x})$ is unknown but we know a number of related constraints (mean value, variance, etc.). The *maximum entropy* estimate of the unknown pdf is the one that maximizes the entropy, subject to the given constraints. According to the principle of maximum entropy, stated by Jaynes [Jayn 82], such an estimate corresponds to the distribution that exhibits the highest possible randomness, subject to the available constraints.

Example 2.5. The random variable x is nonzero for $x_1 \leq x \leq x_2$ and zero otherwise. Compute the maximum entropy estimate of its pdf.

We have to maximize (2.66) subject to the constraint

$$\int_{x_1}^{x_2} p(x)\, dx = 1 \tag{2.67}$$

Using Langrange multipliers (Appendix C), this is equivalent to maximizing

$$H_L = -\int_{x_1}^{x_2} p(x)(\ln p(x) - \lambda)\, dx \tag{2.68}$$

Taking the derivative with respect to $p(x)$, we obtain

$$\frac{\partial H_L}{\partial p(x)} = -\int_{x_1}^{x_2} \{(\ln p(x) - \lambda) + 1\}\, dx \tag{2.69}$$

Equating to zero, we obtain

$$\hat{p}(x) = \exp(\lambda - 1) \tag{2.70}$$

To compute λ, we substitute this into the constraint equation (2.67) and we get $\exp(\lambda - 1) = \frac{1}{x_2 - x_1}$. Thus

$$\hat{p}(x) = \begin{cases} \frac{1}{x_2 - x_1} & \text{if } x_1 \leq x \leq x_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.71}$$

That is, the maximum entropy estimate of the unknown pdf is the uniform distribution. This is within the maximum entropy spirit. Since we have imposed no other constraint but the obvious one, the resulting estimate is the one that maximizes randomness and all points are equally probable. It turns out that if the mean value and the variance are given as the second and third constraints, the resulting maximum entropy estimate of the pdf, for $-\infty < x < +\infty$, is the Gaussian (Problem 2.25).

2.5.5 Mixture Models

An alternative way to model an unknown $p(\boldsymbol{x})$ is via a linear combination of density functions in the form of

$$p(\boldsymbol{x}) = \sum_{j=1}^{J} p(\boldsymbol{x}|j) P_j \tag{2.72}$$

where

$$\sum_{j=1}^{J} P_j = 1, \qquad \int_{\boldsymbol{x}} p(\boldsymbol{x}|j)\, d\boldsymbol{x} = 1 \tag{2.73}$$

In other words, it is assumed that there are J distributions contributing to the formation of $p(\boldsymbol{x})$. Thus, this modeling implicitly assumes that each point $\boldsymbol{x}$ may be "drawn" from any of the J model distributions with probability P_j, $j = 1, 2, \ldots, J$. It can be shown that this modeling can approximate arbitrarily closely any continuous density function for a sufficient number of *mixtures* J and appropriate model parameters. The first step of the procedure involves the choice of the set of density components $p(\boldsymbol{x}|j)$ in parametric form, that is, $p(\boldsymbol{x}|j; \boldsymbol{\theta})$, and then the computation of the unknown parameters, $\boldsymbol{\theta}$ and P_j, $j = 1, 2, \ldots, J$, based on the set of the available training samples $\boldsymbol{x}_k$. There are various ways to achieve this. A typical maximum likelihood formulation, maximizing the likelihood function $\prod_k p(\boldsymbol{x}_k; \boldsymbol{\theta}, P_1, P_2, \ldots, P_J)$ with respect to $\boldsymbol{\theta}$ and the P_j's, is a first thought. The difficulty here arises from the fact that the unknown parameters enter

the maximization task in a *nonlinear fashion*; thus, nonlinear optimization iterative techniques have to be adopted (Appendix C). A review of related techniques is given in [Redn 84]. The source of this complication is the lack of information concerning the labels of the available training samples, that is, the specific mixture from which each sample is contributed. This is the issue that makes the current problem different from the ML case treated in Section 2.5.1. There, the class labels were known and this led to a *separate* ML problem for each of the classes. In the same way, if the mixture labels were known, we could collect all data from the same mixture and carry out J separate ML tasks. The missing label information makes our current problem a typical task with an *incomplete data set.*

In the sequel, we will focus on the so-called EM algorithm, which has attracted a great deal of interest over the past few years in a wide range of applications involving tasks with incomplete data sets.

The Expectation Maximization (EM) Algorithm

This algorithm is ideally suited for cases in which the available data set is incomplete. Let us first state the problem in more general terms and then apply it to our specific task. Let us denote by $\boldsymbol{y}$ the *complete data* samples, with $\boldsymbol{y} \in Y \subseteq \mathcal{R}^m$, and let the corresponding pdf be $p_{\boldsymbol{y}}(\boldsymbol{y}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is an unknown parameter vector. The samples $\boldsymbol{y}$ however *cannot be directly observed.* What we observe instead are samples $\boldsymbol{x} = g(\boldsymbol{y}) \in X_{ob} \subseteq \mathcal{R}^l, l < m$. We denote the corresponding pdf $p_{\boldsymbol{x}}(\boldsymbol{x}; \boldsymbol{\theta})$. This is a *many-to-one mapping.* Let $Y(\boldsymbol{x}) \subseteq Y$ be the subset of all the $\boldsymbol{y}$'s corresponding to a specific $\boldsymbol{x}$. Then the pdf of the incomplete data is given by

$$p_{\boldsymbol{x}}(\boldsymbol{x}; \boldsymbol{\theta}) = \int_{Y(\boldsymbol{x})} p_{\boldsymbol{y}}(\boldsymbol{y}; \boldsymbol{\theta})\, d\boldsymbol{y} \tag{2.74}$$

As we already know, the maximum likelihood estimate of $\boldsymbol{\theta}$ is given by

$$\hat{\boldsymbol{\theta}}_{ML}: \sum_k \frac{\partial \ln(p_{\boldsymbol{y}}(\boldsymbol{y}_k; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} = \mathbf{0} \tag{2.75}$$

However, the $\boldsymbol{y}$'s are not available. So, the EM algorithm maximizes the *expectation* of the loglikelihood function, *conditioned on the observed samples and the current iteration estimate of* $\boldsymbol{\theta}$. The two steps of the algorithm are:

- *E-step*: At the $(t+1)$th step of the iteration, where $\boldsymbol{\theta}(t)$ is available, compute the expected value of

$$Q(\boldsymbol{\theta}; \boldsymbol{\theta}(t)) \equiv E\left[\sum_k \ln(p_{\boldsymbol{y}}(\boldsymbol{y}_k; \boldsymbol{\theta} | X; \boldsymbol{\theta}(t))\right] \tag{2.76}$$

  This is the so-called *expectation step* of the algorithm.

- *M-step*: Compute the next $(t+1)$th estimate of $\boldsymbol{\theta}$ by maximizing $Q(\boldsymbol{\theta}; \boldsymbol{\theta}(t))$, that is,

$$\boldsymbol{\theta}(t+1): \frac{\partial Q(\boldsymbol{\theta}; \boldsymbol{\theta}(t))}{\partial \boldsymbol{\theta}} = \mathbf{0} \tag{2.77}$$

  This is the *maximization step*, where, obviously, differentiability has been assumed.

To apply the EM algorithm, we start from an initial estimate $\boldsymbol{\theta}(0)$ and iterations are terminated if $\|\boldsymbol{\theta}(t+1) - \boldsymbol{\theta}(t)\| \leq \epsilon$ for an appropriately chosen vector norm and ϵ.

Remarks

- It can be shown that the successive estimates $\boldsymbol{\theta}(t)$ never decrease the likelihood function. The likelihood function keeps increasing until a maximum (local or global) is reached and the EM algorithm converges. The convergence proof can be found in the seminal paper [Demp 77] and further discussions in [Wu 83, Boyl 83]. Theoretical results as well as practical experimentation confirm that the convergence is slower than the quadratic convergence of Newton-type searching algorithms (Appendix C), although near the optimum a speedup may be possible. However, the great advantage of the algorithm is that its convergence is smooth and is not vulnerable to instabilities. Furthermore, it is computationally more attractive than Newton-like methods, which require the computation of the Hessian matrix. The keen reader may obtain more information on the EM algorithm and some of its applications from [McLa 88, Titt 85, Moon 96].

Application to the Mixture Modeling Problem

In this case, the complete data set consists of the joint events $(\boldsymbol{x}_k, j_k)$, $k = 1, 2, \ldots, N$, and j_k takes integer values in the interval $[1, J]$, and it denotes the mixture from which $\boldsymbol{x}_k$ is generated. Employing our familiar rule, we obtain

$$p(\boldsymbol{x}_k, j_k; \boldsymbol{\theta}) = p(\boldsymbol{x}_k | j_k; \boldsymbol{\theta}) P_{j_k} \tag{2.78}$$

Assuming mutual independence among samples of the data set, the loglikelihood function becomes

$$L(\boldsymbol{\theta}) = \sum_{k=1}^{N} \ln\left(p(\boldsymbol{x}_k | j_k; \boldsymbol{\theta}) P_{j_k}\right) \tag{2.79}$$

Let $\boldsymbol{P} = [P_1, P_2, \ldots, P_J]^T$. In the current framework, the unknown parameter vector is $\boldsymbol{\Theta}^T = [\boldsymbol{\theta}^T, \boldsymbol{P}^T]^T$. Taking the expectation over the *unobserved data*, conditioned on the training samples and the current estimates, $\boldsymbol{\Theta}(t)$, of the unknown

parameters, we have

$$\text{E-step: } Q(\Theta; \Theta(t)) = E\left[\sum_{k=1}^{N} \ln(p(\boldsymbol{x}_k|j_k; \boldsymbol{\theta})P_{j_k})\right]$$

$$= \sum_{k=1}^{N} E[\ln(p(\boldsymbol{x}_k|j_k; \boldsymbol{\theta})P_{j_k})] \tag{2.80}$$

$$= \sum_{k=1}^{N} \sum_{j_k=1}^{J} P(j_k|\boldsymbol{x}_k; \Theta(t)) \ln(p(\boldsymbol{x}_k|j_k; \boldsymbol{\theta})P_{j_k}) \tag{2.81}$$

The notation can now be simplified by dropping the index k from j_k. This is because, for each k, we sum up over all possible J values of j_k and these are the same for all k. We will demonstrate the algorithm for the case of Gaussian mixtures with diagonal covariance matrices of the form $\Sigma_j = \sigma_j^2 I$, that is,

$$p(\boldsymbol{x}_k|j; \boldsymbol{\theta}) = \frac{1}{(2\pi\sigma_j^2)^{l/2}} \exp\left(-\frac{\|\boldsymbol{x}_k - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \tag{2.82}$$

Assume that besides the prior probabilities, P_j, the respective mean values $\boldsymbol{\mu}_j$ as well as the variances σ_j^2, $j = 1, 2, \ldots, J$, of the Gaussians are also unknown. Thus, $\boldsymbol{\theta}$ is a $J(l+1)$-dimensional vector. Combining Eqs. (2.81) and (2.82) and omitting constants, we get

E-step:

$$Q(\Theta; \Theta(t)) = \sum_{k=1}^{N} \sum_{j=1}^{J} P(j|\boldsymbol{x}_k; \Theta(t)) \left(-\frac{l}{2} \ln \sigma_j^2 - \frac{1}{2\sigma_j^2} \|\boldsymbol{x}_k - \boldsymbol{\mu}_j\|^2 + \ln P_j\right) \tag{2.83}$$

M-step: Maximizing the above with respect to $\boldsymbol{\mu}_j$, σ_j^2, and P_j results in (Problem 2.26)

$$\boldsymbol{\mu}_j(t+1) = \frac{\sum_{k=1}^{N} P(j|\boldsymbol{x}_k; \Theta(t))\boldsymbol{x}_k}{\sum_{k=1}^{N} P(j|\boldsymbol{x}_k; \Theta(t))} \tag{2.84}$$

$$\sigma_j^2(t+1) = \frac{\sum_{k=1}^{N} P(j|\boldsymbol{x}_k; \Theta(t))\|\boldsymbol{x}_k - \boldsymbol{\mu}_j(t+1)\|^2}{l \sum_{k=1}^{N} P(j|\boldsymbol{x}_k; \Theta(t))} \tag{2.85}$$

$$P_j(t+1) = \frac{1}{N}\sum_{k=1}^{N} P(j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t)) \tag{2.86}$$

For the iterations to be complete we need only to compute $P(j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))$. This is easily obtained from

$$P(j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t)) = \frac{p(\boldsymbol{x}_k|j; \boldsymbol{\theta}(t))P_j(t)}{p(\boldsymbol{x}_k; \boldsymbol{\Theta}(t))} \tag{2.87}$$

$$p(\boldsymbol{x}_k; \boldsymbol{\Theta}(t)) = \sum_{j=1}^{J} p(\boldsymbol{x}_k|j; \boldsymbol{\theta}(t))P_j(t) \tag{2.88}$$

Equations (2.84)–(2.88) constitute the EM algorithm for the estimation of the unknown parameters of the Gaussian mixtures in (2.72).

2.5.6 Nonparametric Estimation

So far in our discussion a pdf parametric modeling was incorporated, in one way or another, and the associated unknown parameters were estimated. In the current subsection we will deal with nonparametric techniques. These are basically variations of the *histogram* approximation of an unknown pdf, which is familiar to us from our statistics basics. Let us take for example the simple one-dimensional case. Figure 2.8 shows two examples of a pdf and its approximation by the histogram method. That is, the x-axis (one-dimensional space) is first divided into successive bins of length h. Then the probability of a sample x being located in a bin is estimated for each of the bins. If N is the total number of samples and k_N

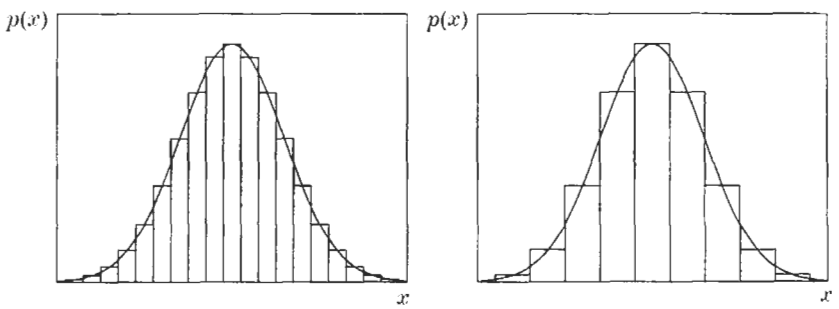


FIGURE 2.8: Probability density function approximation by the histogram method with (a) small and (b) large size intervals.

of these are located inside a bin, the corresponding probability is approximated by the *frequency ratio*

$$P \approx k_N / N \tag{2.89}$$

This approximation converges to the true P as $N \to \infty$ (Problem 2.27). The corresponding pdf value is assumed constant throughout the bin and is approximated by

$$\hat{p}(x) \equiv \hat{p}(\hat{x}) \approx \frac{1}{h} \frac{k_N}{N}, \qquad |x - \hat{x}| \leq \frac{h}{2} \tag{2.90}$$

where $\hat{x}$ is the midpoint of the bin. This determines the amplitude of the histogram curve over the bin. This is a reasonable approximation for continuous $p(x)$ and small enough h so that the assumption of constant $p(x)$ in the bin is sensible. It can be shown that $\hat{p}(x)$ converges to the true value $p(x)$ as $N \to \infty$ provided:

- $h_N \to 0$
- $k_N \to \infty$
- $\frac{k_N}{N} \to 0$

where h_N is used to show the dependence on N. These conditions can be understood from simple reasoning, without having to resort to mathematical details. The first has already been discussed. The other two show the way that k_N must grow to guarantee convergence. Indeed, at all points where $p(x) \neq 0$ fixing the size h_N, however small, the probability P of points occurring in this bin is finite. Hence, $k_N \approx PN$ and k_N tends to infinity as N grows to infinity. On the other hand, as the size h_N of the bin tends to zero, the corresponding probability also goes to zero, justifying the last condition. In practice, the number N of data points is finite. The preceding conditions indicate the way that the various parameters must be chosen. N must be "large enough," h_N "small enough," and the number of points falling in each bin "large enough" too. How small and how large depend on the type of the pdf function and the degree of approximation one is satisfied with. Two popular approaches used in practice are described next.

Parzen Windows. In the multidimensional case, instead of bins of size h, the l-dimensional space is divided into hypercubes with length of side h and volume h^l. Let $\boldsymbol{x}_i, i = 1, 2, \ldots, N$, be the available feature vectors. Define the function $\phi(\boldsymbol{x})$ so that

$$\phi(\boldsymbol{x}_i) = \begin{cases} 1 & \text{for } |x_{ij}| \leq 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{2.91}$$

where $x_{ij}, j = 1, \ldots, l$, are the components of $\boldsymbol{x}_i$. In words, the function is equal to 1 for all points inside the unit side hypercube centered at the origin and 0

outside it. Then (2.90) can be "rephrased" as

$$\hat{p}(\boldsymbol{x}) = \frac{1}{h^l}\left(\frac{1}{N}\sum_{i=1}^{N}\phi\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}}{h}\right)\right) \tag{2.92}$$

The interpretation of this is straightforward. We consider a hypercube with length of side h centered at $\boldsymbol{x}$, the point where the pdf is to be estimated. The summation equals k_N, that is, the number of points falling inside this hypercube. Then the pdf estimate results from dividing k_N by N and the respective hypercube volume h^l. However, viewing Eq. (2.92) from a slightly different perspective, we see that we try to approximate a continuous function $p(\boldsymbol{x})$ via an expansion in terms of discontinuous step functions $\phi(\cdot)$. Thus, the resulting estimate will suffer from this "ancestor's sin." This led Parzen [Parz 62] to generalize (2.92) by using smooth functions in the place of $\phi(\cdot)$. It can be shown that, provided

$$\phi(\boldsymbol{x}) \geq 0 \quad \text{and} \tag{2.93}$$

$$\int_{\boldsymbol{x}} \phi(\boldsymbol{x})\, d\boldsymbol{x} = 1 \tag{2.94}$$

the resulting estimate is a legitimate pdf. Such smooth functions are known as *kernels* or *potential functions* or *Parzen windows*. Typical examples are exponentials, Gaussians $\mathcal{N}(\mathbf{0}, I)$, and so forth. In the sequel, we will examine the limiting behavior of the approximation. To this end, let us take the mean value of (2.92)

$$\begin{aligned} E[\hat{p}(\boldsymbol{x})] &= \frac{1}{h^l}\left(\frac{1}{N}\sum_{i=1}^{N} E\left[\phi\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}}{h}\right)\right]\right) \\ &\equiv \int_{\boldsymbol{x}'} \frac{1}{h^l}\phi\left(\frac{\boldsymbol{x}' - \boldsymbol{x}}{h}\right) p(\boldsymbol{x}')\, d\boldsymbol{x}' \end{aligned} \tag{2.95}$$

Thus, the mean value is a *smoothed version* of the true pdf $p(\boldsymbol{x})$. However as $h \to 0$ the function $\frac{1}{h^l}\phi(\frac{\boldsymbol{x}'-\boldsymbol{x}}{h})$ tends to the delta function $\delta(\boldsymbol{x}' - \boldsymbol{x})$. Indeed, its amplitude goes to infinity, its width tends to zero, and its integral from (2.94) remains equal to one. Thus, in this limiting case and for well-behaved continuous pdf's, $\hat{p}(\boldsymbol{x})$ is an unbiased estimate of $p(\boldsymbol{x})$. *Note that this is independent of the size N of the data set.* Concerning the variance of the estimate, the following remarks are valid:

- For fixed N, the smaller the h the higher the variance, and this is indicated by the noisy appearance of the resulting pdf estimate, for example, Figures 2.9a and 2.10a. This is because $p(\boldsymbol{x})$ is approximated by a finite sum of δ-like spiky functions, centered at the training sample points. Thus, as one

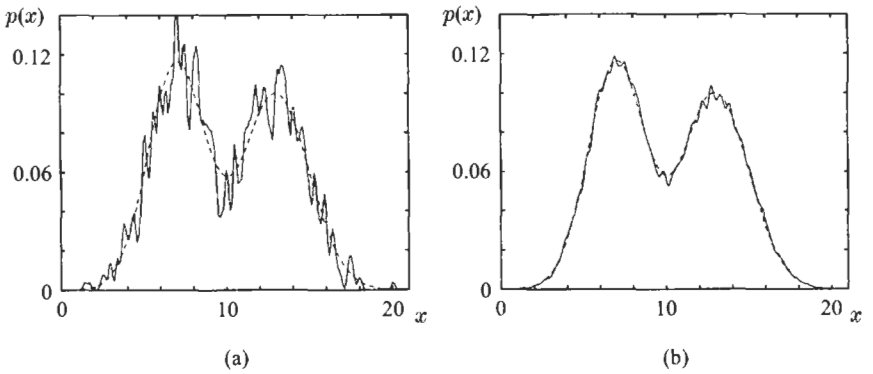


FIGURE 2.9: Approximation of a pdf (dotted line) via Parzen windows, using Gaussian kernels with (a) $h = 0.1$ and 1,000 training samples and (b) $h = 0.1$ and 20,000 samples. Observe the influence of the number of samples on the smoothness of the resulting estimate.

moves $\boldsymbol{x}$ in space the response of $\hat{p}(\boldsymbol{x})$ will be very high near the training points and it will decrease very rapidly as one moves away, leading to this noiselike appearance.

- For a fixed h, the variance decreases as the number of sample points N tends to infinity. This is because the space becomes dense in points, and the spiky functions are closely located, Figure 2.9(b). Furthermore, for a large enough number of samples, the smaller the h the better the accuracy of the resulting estimate, for example, Figures 2.9(b) and 2.10(b).
- It can be shown, for example, [Parz 62, Fuku 90] that, under some mild conditions imposed on $\phi(\cdot)$, which are valid for most density functions, if h tends to zero but in such a way that $hN \rightarrow \infty$, the resulting estimate is both unbiased and asymptotically consistent.

Remarks

- In practice, where only a finite number of samples is possible, a compromise between h and N must be made. The choice of suitable values for h is crucial and several approaches have been proposed in the literature, for example, [Wand 95]. A straightforward way is to start with an initial estimate of h and then modify it iteratively to minimize the resulting misclassification error. The latter can be estimated by appropriate manipulation of the training set. For example, the set can be split into two subsets, one for training and one for testing. We will say more on this in Chapter 10.

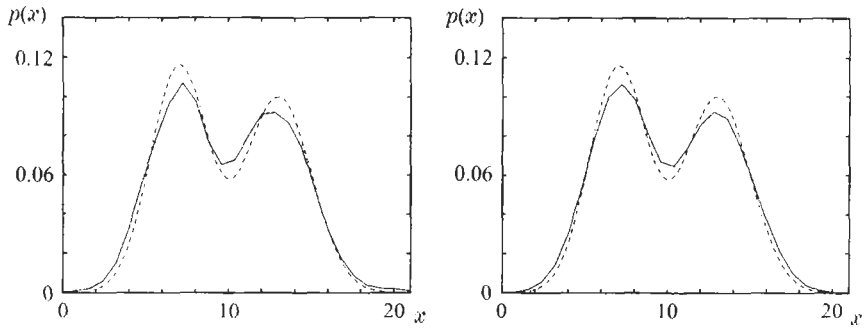


FIGURE 2.10: Approximation of a pdf (dotted line) via Parzen windows, using Gaussian kernels with (a) $h = 0.8$ and 1,000 training samples and (b) $h = 0.8$ and 20,000 samples. Observe that in this case, increasing the number of samples has little influence on the smoothness as well as the approximation accuracy of the resulting estimate.

- Usually, a large N is necessary for acceptable performance. This number grows exponentially with the dimensionality l. If a one-dimensional interval needs to be densely filled with, say, N equidistant points, the corresponding two-dimensional square will need N^2, the three-dimensional cube N^3, and so on. We usually refer to this as the *curse of dimensionality*. The large number of data points puts a high burden on complexity requirements.

Application to classification: On the reception of a feature vector $\boldsymbol{x}$ the likelihood test in (2.19) becomes

$$\text{assign } \boldsymbol{x} \text{ to } \omega_1(\omega_2) \quad \text{if} \quad l_{12} \approx \left(\frac{\frac{1}{N_1 h^l} \sum_{i=1}^{N_1} \phi\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}}{h}\right)}{\frac{1}{N_2 h^l} \sum_{i=1}^{N_2} \phi\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}}{h}\right)} \right) > (<) \frac{P(\omega_2)}{P(\omega_1)} \frac{\lambda_{21} - \lambda_{22}}{\lambda_{12} - \lambda_{11}} \tag{2.96}$$

where N_1, N_2 are the training vectors in class ω_1, ω_2, respectively. For large N_1, N_2 this computation is a very demanding job, in both processing time and memory requirements.

***k* Nearest Neighbor Density Estimation.** In the Parzen estimation of the pdf in (2.92), the volume around the points $\boldsymbol{x}$ was considered fixed (h^l) and the number of points k_N, falling inside the volume, was left to vary randomly from point to point. Here we will reverse the roles. The number of points $k_N = k$ will be fixed

and the size of the volume around $\boldsymbol{x}$ will be adjusted each time, to include k points. Thus, *in low-density areas the volume will be large and in high-density areas it will be small.* We can also consider more general types of regions, besides the hypercube. The estimator can now be written as

$$\hat{p}(\boldsymbol{x}) = \frac{k}{NV(\boldsymbol{x})} \tag{2.97}$$

where the dependence of the volume $V(\boldsymbol{x})$ on $\boldsymbol{x}$ is explicitly shown. Again it can be shown [Fuku 90] that asymptotically ($\lim k = +\infty$, $\lim N = +\infty$, $\lim(k/N) = 0$) this is an unbiased and consistent estimate of the true pdf and it is known as the *k Nearest Neighbor (kNN) density estimate.* Results concerning the finite k, N case have also been derived; see [Fuku 90, Butu 93]. A selection of seminal papers concerning NN classification techniques can be found in [Dasa 91].

From a practical point of view, at the reception of an unknown feature vector $\boldsymbol{x}$, we compute its distance d, for example, Euclidean, from *all* the training vectors of the various classes, for example, ω_1, ω_2. Let r_1 be the radius of the hypersphere, centered at $\boldsymbol{x}$, that contains k points from ω_1 and r_2 the corresponding radius of the hypersphere containing k points from class ω_2 (k may not necessarily be the same for all classes). If we denote by V_1, V_2 the respective hypersphere volumes, the likelihood ratio test becomes

$$\text{assign } \boldsymbol{x} \text{ to } \omega_1(\omega_2) \quad \text{if } l_{12} \approx \frac{kN_2V_2}{kN_1V_1} > (<) \frac{P(\omega_2)}{P(\omega_1)} \frac{\lambda_{21} - \lambda_{22}}{\lambda_{12} - \lambda_{11}}$$

$$\frac{V_2}{V_1} > (<) \frac{N_1}{N_2} \frac{P(\omega_2)}{P(\omega_1)} \frac{\lambda_{21} - \lambda_{22}}{\lambda_{12} - \lambda_{11}} \tag{2.98}$$

If the Mahalanobis distance is alternatively adopted, we will have hyperellipsoids in the place of the hyperspheres.

2.6 THE NEAREST NEIGHBOR RULE

A variation of the kNN density estimation technique results in a *suboptimal,* yet popular in practice, nonlinear classifier. Although this does not fall in the Bayesian framework, it fits nicely at this point. In a way, this section could be considered as a bridge with Chapter 4. The algorithm for the so-called *nearest neighbor rule* is summarized as follows. Given an unknown feature vector $\boldsymbol{x}$ and a distance measure, then:

- Out of the N training vectors, identify the k nearest neighbors, *irrespective* of class label. k is chosen to be odd for a two class problem, and in general not to be a multiple of the number of classes M.

- Out of these k samples, identify the number of vectors, k_i, that belong to class ω_i, $i = 1, 2, \ldots, M$. Obviously, $\sum_i k_i = k$.
- Assign $\boldsymbol{x}$ to the class ω_i with the maximum number k_i of samples.

Various distance measures can be used, including the Euclidean and Mahalanobis distance. In [Hast 96] an effective metric is suggested exploiting the local information at each point. The simplest version of the algorithm is for $k = 1$, known as the *Nearest Neighbor* (NN) rule. In other words, a feature vector $\boldsymbol{x}$ is assigned to the class of its nearest neighbor! Provided that the number of training samples is large enough, this simple rule exhibits good performance. This is also substantiated by theoretical findings. It can be shown [Duda 73, Devr 96] that, as $N \to \infty$, the classification error probability, for the NN rule, P_{NN}, is bounded by

$$P_B \leq P_{NN} \leq P_B\left(2 - \frac{M}{M-1}P_B\right) \leq 2P_B \tag{2.99}$$

where P_B is the optimal Bayesian error. Thus, the error committed by the NN classifier is (*asymptotically*) at most twice that of the optimal classifier. The asymptotic performance of the kNN is better than that of the NN, and a number of interesting bounds have been derived. For example, for the two-class case it can be shown, for example, [Devr 96] that

$$P_B \leq P_{kNN} \leq P_B + \frac{1}{\sqrt{ke}} \quad \text{or} \quad P_B \leq P_{kNN} \leq P_B + \sqrt{\frac{2P_{NN}}{k}} \tag{2.100}$$

Both of these suggest that as $k \to \infty$ the performance of the kNN tends to the optimal one. Furthermore, for small values of Bayesian errors, the following approximations are valid [Devr 96]:

$$P_{NN} \approx 2P_B \tag{2.101}$$

$$P_{3NN} \approx P_B + 3(P_B)^2 \tag{2.102}$$

Thus, for large N and small Bayesian errors, we expect the 3NN classifier to give performance almost identical to that of the Bayesian classifier. As an example, let us say that the error probability of the Bayesian classifier is of the order of 1%; then the error resulting from a 3NN classifier will be of the order of 1.03%! The approximation improves for higher values of k. A little thought can provide justification for this without too much mathematics. Under the assumption of large N, the radius of the hypersphere (Euclidean distance) centered at $\boldsymbol{x}$ and containing its k nearest neighbors tends to zero [Devr 96]. This is natural, because for very large N we expect the space to be densely filled with samples. Thus, the k (a very small portion of N) neighbors of $\boldsymbol{x}$ will be located very close to it, and the conditional class probabilities, at all points inside the hypersphere around $\boldsymbol{x}$, will be approximately equal

to $P(\omega_i|\boldsymbol{x})$ (assuming continuity). Furthermore, for large k (yet an infinitesimally small fraction of N), the majority of the points in the region will belong to the class corresponding to the maximum conditional probability. Thus, the kNN rule tends to the Bayesian classifier. Of course, all these are true asymptotically. In the finite sample case there are even counterexamples (Problem 2.29) where the kNN results in higher error probabilities than the NN. However, in conclusion, it can be stated that the nearest neighbor techniques are among the serious candidates to be adopted as classifiers in a number of applications. A comparative study of the various statistical classifiers considered in this chapter as well as others, can be found in [Aebe 94].

Remarks

- A serious drawback associated with (k)NN techniques is the complexity in search of the nearest neighbor(s) among the N available training samples. Brute-force searching amounts to operations proportional to kN ($O(kN)$).[2] The problem becomes particularly severe in high-dimensional feature spaces. To reduce the computational burden a number of efficient searching schemes have been suggested; see, for example, [Fuku 75, Dasa 91, Brod 90, Djou 97, Nene 97, Hatt 00, Kris 00]. Attempts to reduce the number N of feature vectors, in an optimal way, have also appeared in the literature (e.g., [Yan 93, Huan 02] and the references therein).
- When the $k = 1$ nearest neighbor rule is used, the training feature vectors $\boldsymbol{x}_i, i = 1, 2, \ldots, N$, define a partition of the l-dimensional space into N regions, R_i. Each of these regions is defined by

$$R_i = \{\boldsymbol{x}: d(\boldsymbol{x}, \boldsymbol{x}_i) < d(\boldsymbol{x}, \boldsymbol{x}_j), i \neq j\} \tag{2.103}$$

  that is, R_i contains all points in space that are closer to $\boldsymbol{x}_i$ than any other point of the training set, with respect to the distance d. This partition of the feature space is known as *Voronoi tessellation*. Figure 2.11 is an example of the resulting *Voronoi tessellation* for the case of $l = 2$ and the Euclidean distance.

Problems

2.1 Show that in a multiclass classification task, the Bayes decision rule minimizes the error probability.
Hint: It is easier to work with the probability of correct decision.

[2] $O(n)$ denotes order of n calculations.

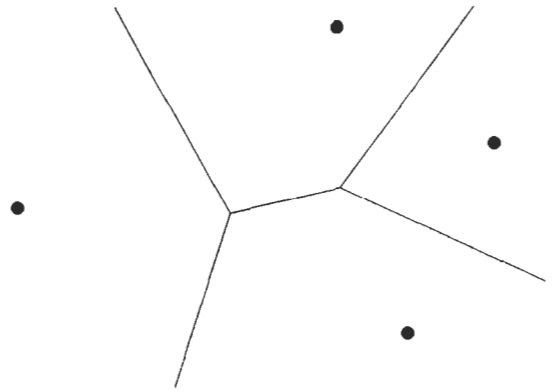


FIGURE 2.11: An example of Voronoi tessellation in the two-dimensional space and for Euclidean distance.

2.2 In a two-class one-dimensional problem the pdf's are the Gaussians $\mathcal{N}(0, \sigma^2)$ and $\mathcal{N}(1, \sigma^2)$ for the two classes, respectively. Show that the threshold x_0 minimizing the average risk is equal to

$$x_0 = 1/2 - \sigma^2 \ln \frac{\lambda_{21} P(\omega_2)}{\lambda_{12} P(\omega_1)}$$

where $\lambda_{11} = \lambda_{22} = 0$ has been assumed.

2.3 Consider a two equiprobable classes problem with a loss matrix L. Show that if ϵ_1 is the probability of error corresponding to feature vectors from class ω_1 and ϵ_2 for those from class ω_2, then the average risk r is given by

$$r = P(\omega_1)\lambda_{11} + P(\omega_2)\lambda_{22} + P(\omega_1)(\lambda_{12} - \lambda_{11})\epsilon_1 + P(\omega_2)(\lambda_{21} - \lambda_{22})\epsilon_2$$

2.4 Show that in a multiclass problem with M classes the probability of classification error for the optimum classifier is bounded by

$$P_e \leq \frac{M-1}{M}$$

Hint: Show first that for each $\boldsymbol{x}$ the maximum of $P(\omega_i|\boldsymbol{x})$, $i = 1, 2, \ldots, M$, is greater than or equal to $1/M$. Equality holds if all $P(\omega_i|\boldsymbol{x})$ are equal.

2.5 Consider a two (equiprobable) class one-dimensional problem with samples distributed according to the Rayleigh pdf in each class, that is,

$$p(x|\omega_i) = \begin{cases} \frac{x}{\sigma_i^2} \exp\left(\frac{-x^2}{2\sigma_i^2}\right) & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Compute the decision boundary point $g(x) = 0$.

2.6 In a two-class classification task we constrain the error probability for one of the classes to be fixed, that is, $\epsilon_1 = \epsilon$. Then show that minimizing the error probability of the other class results in the likelihood test

$$\text{decide } \boldsymbol{x} \text{ in } \omega_1 \text{ if } \frac{P(\omega_1|\boldsymbol{x})}{P(\omega_2|\boldsymbol{x})} > \theta$$

where θ is chosen so that the constraint is fulfilled. This is known as the *Neyman–Pearson test* and it is similar to the Bayesian minimum risk rule.

Hint: Use a Langrange multiplier to show that this problem is equivalent to minimizing the quantity

$$q = \theta(\epsilon_1 - \epsilon) + \epsilon_2$$

2.7 In a three-class two-dimensional problem the feature vectors in each class are normally distributed with covariance matrix

$$\Sigma = \begin{bmatrix} 1.2 & 0.4 \\ 0.4 & 1.8 \end{bmatrix}$$

The mean vectors for each class are $[0.1, 0.1]^T$, $[2.1, 1.9]^T$, $[-1.5, 2.0]^T$. Assuming that the classes are equiprobable, (a) classify the feature vector $[1.6, 1.5]^T$ according to the Bayes minimum error probability classifier; (b) draw the curves of equal Mahalanobis distance from $[2.1, 1.9]^T$.

2.8 In a two-class three-dimensional classification problem, the feature vectors in each class are normally distributed with covariance matrix

$$\Sigma = \begin{bmatrix} 0.3 & 0.1 & 0.1 \\ 0.1 & 0.3 & -0.1 \\ 0.1 & -0.1 & 0.3 \end{bmatrix}$$

The respective mean vectors are $[0, 0, 0]^T$ and $[0.5, 0.5, 0.5]^T$. Derive the corresponding linear discriminant functions and the equation describing the decision surface.

2.9 In a two equiprobable class classification problem the feature vectors in each class are normally distributed with covariance matrix Σ and the corresponding mean vectors are $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$. Show that for the Bayesian minimum error classifier, the error probability is given by

$$P_B = \int_{(1/2)d_m}^{+\infty} \frac{1}{\sqrt{2\pi}} \exp(-z^2/2)\, dz$$

where d_m is the Mahalanobis distance between the mean vectors. Observe that this is a decreasing function of d_m.

Hint: Compute the loglikelihood ratio $u = \ln p(\boldsymbol{x}|\omega_1) - \ln p(\boldsymbol{x}|\omega_2)$. Observe that u is also a random variable normally distributed as $\mathcal{N}((1/2)d_m^2, d_m^2)$ if $\boldsymbol{x} \in \omega_1$ and as $\mathcal{N}(-(1/2)d_m^2, d_m^2)$ if $\boldsymbol{x} \in \omega_2$. Use this information to compute the error probability.

2.10 Show that in the case in which the feature vectors follow Gaussian pdf's, the likelihood ratio test in (2.19)

$$\boldsymbol{x} \in \omega_1 (\omega_2) \quad \text{if} \quad l_{12} \equiv \frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} > (<)\theta$$

is equivalent to

$$d_m^2(\boldsymbol{\mu}_1, \boldsymbol{x}|\Sigma_1) - d_m^2(\boldsymbol{\mu}_2, \boldsymbol{x}|\Sigma_2) + \ln \frac{|\Sigma_1|}{|\Sigma_2|} < (>) - 2\ln\theta$$

where $d_m(\boldsymbol{\mu}_i, \boldsymbol{x}|\Sigma_i)$ is the Mahalanobis distance between $\boldsymbol{\mu}_i$ and $\boldsymbol{x}$ with respect to the Σ_i^{-1} norm.

2.11 If $\Sigma_1 = \Sigma_2 = \Sigma$, show that the criterion of the previous problem becomes

$$(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} \boldsymbol{x} > (<) \Theta$$

where

$$\Theta = \ln\theta + 1/2(\|\boldsymbol{\mu}_1\|_{\Sigma^{-1}} - \|\boldsymbol{\mu}_2\|_{\Sigma^{-1}})$$

2.12 Consider a two-class two-dimensional classification task, where the feature vectors in each of the classes ω_1, ω_2 are distributed according to

$$p(\boldsymbol{x}|\omega_1) = \frac{1}{\left(\sqrt{2\pi\sigma_1^2}\right)^2} \exp\left(-\frac{1}{2\sigma_1^2}(\boldsymbol{x} - \boldsymbol{\mu}_1)^T(\boldsymbol{x} - \boldsymbol{\mu}_1)\right)$$

$$p(\boldsymbol{x}|\omega_2) = \frac{1}{\left(\sqrt{2\pi\sigma_2^2}\right)^2} \exp\left(-\frac{1}{2\sigma_2^2}(\boldsymbol{x} - \boldsymbol{\mu}_2)^T(\boldsymbol{x} - \boldsymbol{\mu}_2)\right)$$

with

$$\boldsymbol{\mu}_1 = [1.\ 1]^T, \qquad \boldsymbol{\mu}_2 = [1.5,\ 1.5]^T, \qquad \sigma_1^2 = \sigma_2^2 = 0.2$$

Assume that $P(\omega_1) = P(\omega_2)$ and design a Bayesian classifier

(a) that minimizes the error probability

(b) that minimizes the average risk with loss matrix

$$\Lambda = \begin{bmatrix} 0 & 1 \\ 0.5 & 0 \end{bmatrix}$$

Using a pseudorandom number generator, produce 100 feature vectors from each class, according to the preceding pdf's. Use the classifiers designed to classify the generated vectors. What is the percentage error for each case? Repeat the experiments for $\boldsymbol{\mu}_2 = [3.0.\ 3.0]^T$.

2.13 Repeat the preceding experiment if the feature vectors are distributed according to

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\Big)$$

with

$$\Sigma = \begin{bmatrix} 1.01 & 0.2 \\ 0.2 & 1.01 \end{bmatrix}$$

and $\boldsymbol{\mu}_1 = [1, 1]^T$, $\boldsymbol{\mu}_2 = [1.5, 1.5]^T$.

Hint: To generate the vectors, recall from [Papo 91, page 144] that a linear transformation of Gaussian random vectors also results in Gaussian vectors. Note also that

$$\begin{bmatrix} 1.01 & 0.2 \\ 0.2 & 1.01 \end{bmatrix} = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}$$

2.14 Consider a two-class problem with normally distributed vectors with the same Σ in both classes. Show that the decision hyperplane at the point $\boldsymbol{x}_0$, Eq. (2.38), is tangent to the constant Mahalanobis distance hyperellipsoids.

Hint: (a) Compute the gradient of Mahalanobis distance with respect to $\boldsymbol{x}$. (b) Recall from vector analysis that $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is normal to the tangent of the surface $f(\boldsymbol{x}) =$ constant.

2.15 Consider a two-class one-dimensional problem with $p(x|\omega_1)$ being $\mathcal{N}(\mu, \sigma^2)$ and $p(x|\omega_2)$ a uniform distribution between a and b. Show that the Bayesian error probability is bounded by $G(\frac{b-\mu}{\sigma}) - G(\frac{a-\mu}{\sigma})$, where $G(x) \equiv P(y \leq x)$ and y is $\mathcal{N}(0, 1)$.

2.16 Show that the mean value of the random vector $\frac{\partial \ln(p(\boldsymbol{x};\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}}$ is zero.

2.17 In a heads or tails coin-tossing experiment the probability of occurrence of a head (1) is q and that of a tail (0) is $1-q$. Let x_i, $i = 1, 2, \ldots, N$, be the resulting experimental outcomes, $x_i \in \{0, 1\}$. Show that the ML estimate of q is

$$q_{ML} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Hint: The likelihood function is

$$P(X; q) = \prod_{i=1}^{N} q^{x_i}(1-q)^{(1-x_i)}$$

Then show that the ML results from the solution of the equation

$$q^{\sum_i x_i}(1-q)^{(N-\sum_i x_i)}\left(\frac{\sum_i x_i}{q} - \frac{N-\sum_i x_i}{1-q}\right) = 0$$

2.18 The random variable x is normally distributed $\mathcal{N}(\mu, \sigma^2)$, where μ is considered unknown. Given N measurements of the variable, compute the Cramer–Rao bound $-E\left[\frac{\partial^2 L(\mu)}{\partial^2 \mu}\right]$ (Appendix A). Compare the bound with the variance of the resulting ML estimate of μ. Repeat this if the unknown parameter is the variance σ^2. Comment on the results.

2.19 Show that if the likelihood function is Gaussian with unknowns the mean $\boldsymbol{\mu}$ as well as the covariance matrix Σ, then the ML estimates are given by

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{x}_k$$

$$\hat{\Sigma} = \frac{1}{N}\sum_{k=1}^{N} (\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})^{\mathrm{T}}$$

2.20 The random variable x follows the Erlang pdf

$$p(x; \theta) = \theta^2 x \exp(-\theta x) u(x)$$

where $u(x)$ is the unit step function,

$$u(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Show that the maximum likelihood estimate of θ, given N measurements, $x_1, \ldots, x_N$, of x, is

$$\hat{\theta}_{ML} = \frac{2N}{\sum_{k=1}^{N} x_k}$$

2.21 In the ML estimation, the zero of the derivative of the log pdf derivative was computed. Using a multivariate Gaussian pdf, show that this corresponds to a maximum and not to a minimum.

2.22 Show relations (2.64) and (2.65). Then show that $p(x|X)$ is also normal with mean μ_N and variance $\sigma^2 + \sigma_N^2$. Comment on the result.

2.23 The random variable x is normally distributed as $\mathcal{N}(\mu, \sigma^2)$, with μ being the unknown parameter described by the Rayleigh pdf

$$p(\mu) = \frac{\mu \exp(-\mu^2/2\sigma_\mu^2)}{\sigma_\mu^2}$$

Show that the maximum a posteriori probability estimate of μ is given by

$$\hat{\mu}_{MAP} = \frac{Z}{2R}\left(1 + \sqrt{1 + \frac{4R}{Z^2}}\right)$$

where

$$Z = \frac{1}{\sigma^2}\sum_{k=1}^{N} x_k, \qquad R = \frac{N}{\sigma^2} + \frac{1}{\sigma_\mu^2}$$

2.24 Show that for the lognormal distribution

$$p(x) = \frac{1}{\sigma x\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \theta)^2}{2\sigma^2}\right), \quad x > 0$$

the ML estimate is given by

$$\hat{\theta}_{ML} = \frac{1}{N}\sum_{k=1}^{N} \ln x_k$$

2.25 Show that if the mean value and the variance of a random variable are known, that is,

$$\mu = \int_{-\infty}^{+\infty} x p(x)\, dx, \qquad \sigma^2 = \int_{-\infty}^{+\infty} (x-\mu)^2 p(x)\, dx$$

the maximum entropy estimate of the pdf is the Gaussian $\mathcal{N}(\mu, \sigma^2)$.

2.26 Show Eqs. (2.84), (2.85), and (2.86).

Hint: For the latter, note that the probabilities add to one; thus a Langrangian multiplier must be used.

2.27 Let P be the probability of a random point x being located in a certain interval h. Given N of these points, the probability of having k of them inside h is given by the binomial distribution

$$\text{prob}\{k\} = \frac{N!}{k!(N-k)!} P^k (1-P)^{N-k}$$

Show that $E[k/N] = P$ and that the variance around the mean is $\sigma^2 = E[(k/N - P)^2] = P(1-P)/N$. That is, the probability estimator $P = k/N$ is unbiased and asymptotically consistent.

2.28 Consider three Gaussian pdf's, $\mathcal{N}(1.0, 0.1)$, $\mathcal{N}(3.0, 0.1)$, $\mathcal{N}(2.0, 0.2)$. Generate 500 samples according to the following rule. The first two samples are generated from the second Gaussian, the third sample from the first one, and the fourth sample from the last Gaussian. This rule repeats until all 500 samples have been generated. The pdf underlying the random samples is modeled as a mixture

$$\sum_{i=1}^{3} \mathcal{N}(\mu_i, \sigma_i^2) P_i$$

Use the EM algorithm and the generated samples to estimate the unknown parameters μ_i, σ_i^2, P_i.

2.29 Consider two classes ω_1, ω_2 in the two-dimensional space. The data from class ω_1 are uniformly distributed inside a circle of radius r. The data of class ω_2 are also uniformly distributed inside another circle of radius r. The distance between the centers of the circles is greater than $4r$. Let N be the number of the available training samples. Show that the probability of error of the NN classifier is always smaller than that of the kNN, for any $k \geq 3$.

2.30 Generate 50 feature vectors for each of the two classes of Problem 2.12, and use them as training points. In the sequel, generate 100 vectors from each class and classify them according to the NN and 3NN rules. Compute the classification error percentages.

2.31 The pdf of a random variable is given by

$$p(x) = \begin{cases} \frac{1}{2} & \text{for } 0 < x < 2 \\ 0 & \text{otherwise} \end{cases}$$

Use the Parzen window method to approximate it using as the kernel function the Gaussian $\mathcal{N}(0, 1)$. Choose the smoothing parameter to be (a) $h = 0.05$ and (b) $h = 0.2$. For each case, plot the approximation based on $N = 32$, $N = 256$, and $N = 5000$ points, which are generated from a pseudorandom generator according to $p(x)$.

2.32 Repeat the preceding problem by generating $N = 5000$ points and using k nearest neighbor estimation with $k = 32$, 64 and 256, respectively.

References

[Aebe 94] Aeberhard S., Coomans D., Devel O. "Comparative analysis of statistical pattern recognition methods in high dimensional setting," *Pattern Recognition*, Vol. 27(8), pp. 1065–1077, 1994.

[Boyl 83] Boyles R.A. "On the convergence of the EM algorithm," *J. Royal Statistical Society B*, Vol. 45(1), pp. 47–55, 1983.

[Brod 90] Broder A. "Strategies for efficient incremental nearest neighbor search," *Pattern Recognition*, Vol. 23, pp. 171–178, 1990.

[Butu 93] Buturovic L.J. "Improving k-nearest neighbor density and error estimates," *Pattern Recognition*, Vol. 26(4), pp. 611–616, 1993.

[Cram 46] Cramer H. *Mathematical Methods of Statistics*, Princeton University Press, 1941.

[Dasa 91] Dasarasthy B. *Nearest Neighbor Pattern Classification Techniques*, IEEE Computer Society Press, 1991.

[Demp 77] Dempster A.P., Laird N.M., Rubin D.B. "Maximum likelihood from incomplete data via the EM algorithm," *J. Royal Statistical Society*, Vol. 39(1), pp. 1–38, 1977.

[Devr 96] Devroye L., Gyorfi L., Lugosi G. *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag, 1996.

[Djou 97] Djouadi A., Bouktache E. "A fast algorithm for the nearest neighbor classifier," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(3), pp. 277–282, 1997.

[Duda 73] Duda R., Hart P.E. *Pattern Classification and Scene Analysis*, John Wiley, 1973.

[Fuku 75] Fukunaga F., Narendra P.M. "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Transactions on Computers*, Vol. 24, pp. 750–753, 1975.

[Fuku 90] Fukunaga F. *Introduction to Statistical Pattern Recognition*, Academic Press, 2nd ed., 1990.

[Hast 96] Hastie T., Tibshirani R. "Discriminant adaptive nearest neighbor classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18(6), pp. 607–616, 1996.

[Hatt 00] Hattori K., Takahashi M. "A new edited k-nearest neighbor rule in the pattern classification problem,"*Pattern Recognition*, Vol. 33, pp. 521–528, 2000.

[Huan 02] Huang Y.S., Chiang C.C., Shieh J.W., Grimson E. "Prototype optimization for nearest-neighbor classification," *Pattern Recognition*, Vol. 35, pp. 1237–1245, 2002.

[Jayn 82] Jaynes E.T. "On the rationale of the maximum entropy methods," *Proceedings of the IEEE*, Vol. 70(9), pp. 939–952, 1982.

[Kris 00] Krishna K., Thathachar M.A.L., Ramakrishnan K.R. "Voronoi networks and their probability of misclassification,"*IEEE Transactions on Neural Networks*, Vol. 11(6), pp. 1361–1372, 2000.

[McLa 88] McLachlan G.J., Basford K.A. *Mixture Models: Inference and Applications to Clustering*, Marcel Dekker, 1988.

[Moon 96] Moon T. "The expectation maximization algorithm," *Signal Processing Magazine*, Vol. 13(6), pp. 47–60, 1996.

[Nene 97] Nene S.A., Nayar S.K. "A simple algorithm for nearest neighbor search in high dimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(9), pp. 989–1003, 1997.

[Papo 91] Papoulis A. *Probability Random Variables and Stochastic Processes*, 3rd ed., McGraw-Hill 1991.

[Parz 62] Parzen E. "On the estimation of a probability density function and mode," *Ann. Math. Stat.* Vol. 33, pp. 1065–1076, 1962.

[Redn 84] Redner R.A., Walker H.F. "Mixture densities, maximum likelihood and the EM algorithm," *SIAM Review*, Vol. 26(2), pp. 195–239, 1984.

[Titt 85] Titterlington D.M., Smith A.F.M., Makov U.A. *Statistical Analysis of Finite Mixture Distributions*, John Wiley, 1985.

[Yan 93] Yan H. "Prototype optimization for nearest neighbor classifiers using a two layer perceptron," *Pattern Recognition*, Vol. 26(2), pp. 317–324, 1993.

[Wand 95] Wand M., Jones M. *Kernel Smoothing*, Chapman & Hall, London, 1995.

[Wu 83] Wu C. "On the convergence properties of the EM algorithm," *Annals of Statistics*, Vol. 11(1), pp. 95–103, 1983.

CHAPTER 3

LINEAR CLASSIFIERS

3.1 INTRODUCTION

Our major concern in Chapter 2 was to design classifiers based on probability density or probability functions. In some cases, we saw that the resulting classifiers were equivalent to a set of linear discriminant functions. In this chapter we will focus on the design of linear classifiers, *irrespective of the underlying distributions describing the training data*. The major advantage of linear classifiers is their simplicity and computational attractiveness. The chapter starts with the assumption that *all* feature vectors from the available classes can be classified correctly using a linear classifier, and we will develop techniques for the computation of the corresponding linear functions. In the sequel we will focus on a more general problem, in which a linear classifier cannot classify correctly all vectors, yet we will seek ways to design an *optimal linear* classifier by adopting an appropriate optimality criterion.

3.2 LINEAR DISCRIMINANT FUNCTIONS AND DECISION HYPERPLANES

Let us once more focus on the two-class case and consider linear discriminant functions. Then the respective decision hypersurface in the l-dimensional feature space is a hyperplane, that is

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = 0 \tag{3.1}$$

where $\boldsymbol{w} = [w_1, w_2, \ldots, w_l]^T$ is known as the *weight vector* and w_0 as the *threshold*. If $\boldsymbol{x}_1, \boldsymbol{x}_2$ are two points on the decision hyperplane, then the following is valid

$$\begin{aligned} 0 = \boldsymbol{w}^T\boldsymbol{x}_1 + w_0 = \boldsymbol{w}^T\boldsymbol{x}_2 + w_0 \quad &\Rightarrow \\ \boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0 \quad & \end{aligned} \tag{3.2}$$

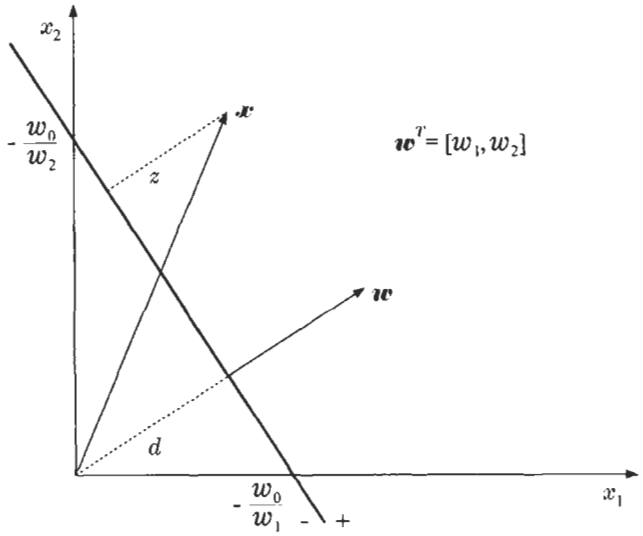


FIGURE 3.1: Geometry for the decision line. On one side of the line it is $g(\boldsymbol{x}) > 0(+)$ and on the other $g(\boldsymbol{x}) < 0\ (-)$.

Since the difference vector $\boldsymbol{x}_1 - \boldsymbol{x}_2$ obviously lies on the decision hyperplane (for any $\boldsymbol{x}_1, \boldsymbol{x}_2$), it is apparent from Eq. (3.2) that the vector $\boldsymbol{w}$ is *orthogonal* to the decision hyperplane.

Figure 3.1 shows the corresponding geometry (for $w_1 > 0, w_2 > 0, w_0 < 0$). Recalling our high school math, it is easy to see that the quantities entering in the figure are given by

$$d = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}} \tag{3.3}$$

and

$$z = \frac{|g(\boldsymbol{x})|}{\sqrt{w_1^2 + w_2^2}} \tag{3.4}$$

In other words, $|g(\boldsymbol{x})|$ is a measure of the Euclidean distance of the point $\boldsymbol{x}$ from the decision hyperplane. On one side of the plane $g(\boldsymbol{x})$ takes positive values and on the other negative. In the special case that $w_0 = 0$ the hyperplane passes through the origin.

3.3 THE PERCEPTRON ALGORITHM

Our major concern now is to compute the unknown parameters w_i, $i = 0, \ldots, l$, defining the decision hyperplane. In this section we assume that the two classes ω_1, ω_2 are *linearly separable*. In other words we assume that *there exists* a hyperplane, defined by $\boldsymbol{w}^{*T}\boldsymbol{x} = 0$, such that

$$\begin{aligned} \boldsymbol{w}^{*T}\boldsymbol{x} &> 0 \quad \forall \boldsymbol{x} \in \omega_1 \\ \boldsymbol{w}^{*T}\boldsymbol{x} &< 0 \quad \forall \boldsymbol{x} \in \omega_2 \end{aligned} \tag{3.5}$$

The formulation above also covers the case of a hyperplane not crossing the origin, that is, $\boldsymbol{w}^{*T}\boldsymbol{x} + w_0^* = 0$, since this can be brought into the previous formulation by defining the extended $(l+1)$-dimensional vectors $\boldsymbol{x}' \equiv [\boldsymbol{x}^T, 1]^T$, $\boldsymbol{w}' \equiv [\boldsymbol{w}^{*T}, w_0^*]^T$. Then $\boldsymbol{w}^{*T}\boldsymbol{x} + w_0^* = \boldsymbol{w}'^T\boldsymbol{x}'$.

We will approach the problem as a typical optimization task (Appendix C). Thus we need to adopt (a) an appropriate cost function and (b) an algorithmic scheme to optimize it. To this end, we choose the *perceptron cost* defined as

$$J(\boldsymbol{w}) = \sum_{\boldsymbol{x} \in Y} (\delta_x \boldsymbol{w}^T \boldsymbol{x}) \tag{3.6}$$

where Y is the subset of the training vectors, which are misclassified by the hyperplane defined by the weight vector $\boldsymbol{w}$. The variable δ_x is chosen so that $\delta_x = -1$ if $\boldsymbol{x} \in \omega_1$ and $\delta_x = +1$ if $\boldsymbol{x} \in \omega_2$. Obviously, the sum in (3.6) is always positive and it becomes zero when Y becomes the empty set, that is, if there are not misclassified vectors $\boldsymbol{x}$. Indeed, if $\boldsymbol{x} \in \omega_1$ and it is misclassified, then $\boldsymbol{w}^T\boldsymbol{x} < 0$ and $\delta_x < 0$, and the product is positive. The result is the same for vectors originating from class ω_2. When the cost function takes its minimum value, 0, a solution has been obtained, since all training feature vectors are correctly classified.

The perceptron cost function in (3.6) is *continuous and piecewise linear*. Indeed, if we change the weight vector smoothly, the cost $J(\boldsymbol{w})$ changes linearly until the point at which there is a change in the number of misclassified vectors (Problem 3.1). At these points the gradient is not defined and the gradient function is discontinuous.

To derive the algorithm for the iterative minimization of the cost function, we will adopt an iterative scheme in the spirit of the *gradient descent* method (Appendix C), that is,

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \rho_t \left. \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} \right|_{\boldsymbol{w}=\boldsymbol{w}(t)} \tag{3.7}$$

where $\boldsymbol{w}(t)$ is the weight vector estimate at the tth iteration step, and ρ_t is a sequence of positive real numbers. However, we must be careful here. This is not

defined at the points of discontinuity. From the definition in (3.6), and at the points where this is valid, we get

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x} \tag{3.8}$$

Substituting (3.8) into (3.7) we obtain

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \rho_t \sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x} \tag{3.9}$$

The algorithm is known as the *perceptron algorithm* and is quite simple in its structure. *Note that Eq. (3.9) is defined at all points.* The algorithm is initialized from an arbitrary weight vector $\boldsymbol{w}(0)$, and the correction vector $\sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x}$ is formed using the misclassified features. The weight vector is then corrected according to the preceding rule. This is repeated until the algorithm converges to a solution, that is, all features are correctly classified.

Figure 3.2 provides a geometric interpretation of the algorithm. It has been assumed that at step t there is only one misclassified sample, $\boldsymbol{x}$, and $\rho_t = 1$. The perceptron algorithm corrects the weight vector in the direction of $\boldsymbol{x}$. Its effect is to turn the corresponding hyperplane so that $\boldsymbol{x}$ is classified in the correct class ω_1.

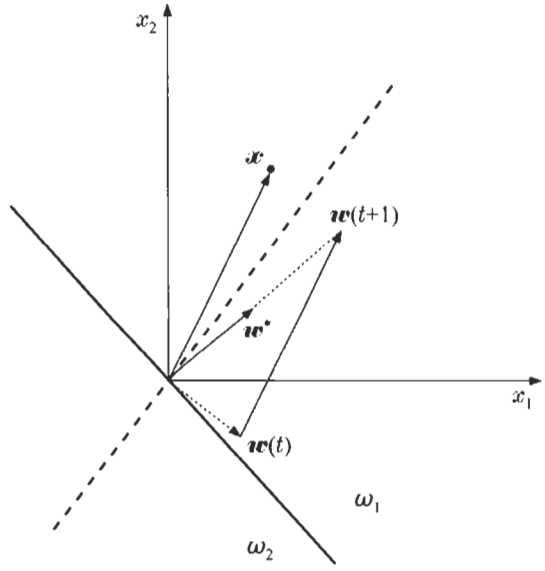


FIGURE 3.2: Geometric interpretation of the perceptron algorithm.

Note that in order to achieve this, it may take more than one iteration step, depending on the value(s) of ρ_t. No doubt, this sequence is critical for the convergence. We will now show that the perceptron algorithm converges to a solution in a *finite number of iteration steps*, provided that the sequence ρ_t is properly chosen. The solution is not unique, because there are more than one hyperplanes separating two linearly separable classes. The convergence proof is necessary because the algorithm is not a true gradient descent algorithm and the general tools for the convergence of gradient descent schemes cannot be applied.

Proof of the Perceptron Algorithm Convergence

Let α be a positive real number and $\boldsymbol{w}^*$ a solution. Then from (3.9) we have

$$\boldsymbol{w}(t+1) - \alpha \boldsymbol{w}^* = \boldsymbol{w}(t) - \alpha \boldsymbol{w}^* - \rho_t \sum_{\boldsymbol{x} \in Y} \delta_x \boldsymbol{x} \tag{3.10}$$

Squaring the Euclidean norm of both sides results in

$$\begin{aligned} \|\boldsymbol{w}(t+1) - \alpha \boldsymbol{w}^*\|^2 = \|\boldsymbol{w}(t) - \alpha \boldsymbol{w}^*\|^2 + \rho_t^2 \|\sum_{\boldsymbol{x} \in Y} \delta_x \boldsymbol{x}\|^2 \\ - 2\rho_t \sum_{\boldsymbol{x} \in Y} \delta_x (\boldsymbol{w}(t) - \alpha \boldsymbol{w}^*)^T \boldsymbol{x} \end{aligned} \tag{3.11}$$

But $-\sum_{\boldsymbol{x} \in Y} \delta_x \boldsymbol{w}^T(t) \boldsymbol{x} < 0$. Hence

$$\begin{aligned} \|\boldsymbol{w}(t+1) - \alpha \boldsymbol{w}^*\|^2 \leq \|\boldsymbol{w}(t) - \alpha \boldsymbol{w}^*\|^2 + \rho_t^2 \|\sum_{\boldsymbol{x} \in Y} \delta_x \boldsymbol{x}\|^2 \\ + 2\rho_t \alpha \sum_{\boldsymbol{x} \in Y} \delta_x \boldsymbol{w}^{*T} \boldsymbol{x} \end{aligned} \tag{3.12}$$

Define

$$\beta^2 = \max_{\tilde{Y} \subseteq \omega_1 \cup \omega_2} \|\sum_{\boldsymbol{x} \in \tilde{Y}} \delta_x \boldsymbol{x}\|^2 \tag{3.13}$$

That is, β^2 is the maximum value that the involved vector norm can take by considering *all* possible (nonempty) subsets of the available training feature vectors. Similarly, let

$$\gamma = \max_{\tilde{Y} \subseteq \omega_1 \cup \omega_2} \sum_{\boldsymbol{x} \in \tilde{Y}} \delta_x \boldsymbol{w}^{*T} \boldsymbol{x} \tag{3.14}$$

Recall that the summation in this equation is negative; thus, its maximum value over all possible subsets of $\boldsymbol{x}$'s will also be a negative number. Hence, (3.12) can

now be written as

$$\|\boldsymbol{w}(t+1) - \alpha \boldsymbol{w}^*\|^2 \leq \|\boldsymbol{w}(t) - \alpha \boldsymbol{w}^*\|^2 + \rho_t^2 \beta^2 - 2\rho_t \alpha |\gamma| \tag{3.15}$$

Choose $\alpha = \frac{\beta^2}{2|\gamma|}$ and apply (3.15) successively for steps $t, t-1, \ldots, 0$. Then

$$\|\boldsymbol{w}(t+1) - \alpha \boldsymbol{w}^*\|^2 \leq \|\boldsymbol{w}(0) - \alpha \boldsymbol{w}^*\|^2 + \beta^2 \left(\sum_{k=0}^{t} \rho_k^2 - \sum_{k=0}^{t} \rho_k \right) \tag{3.16}$$

If the sequence ρ_t is chosen to satisfy the following two conditions:

$$\lim_{t \to \infty} \sum_{k=0}^{t} \rho_k = \infty \tag{3.17}$$

$$\lim_{t \to \infty} \sum_{k=0}^{t} \rho_k^2 < \infty \tag{3.18}$$

then there will be a constant t_0 such that the right-hand side of (3.16) becomes nonpositive. Thus

$$0 \leq \|\boldsymbol{w}(t_0 + 1) - \alpha \boldsymbol{w}^*\| \leq 0 \tag{3.19}$$

or

$$\boldsymbol{w}(t_0 + 1) = \alpha \boldsymbol{w}^* \tag{3.20}$$

That is, the algorithm converges to a solution in a finite number of steps. An example of a sequence satisfying conditions (3.17), (3.18) is $\rho_t = c/t$, where c is a constant. In other words, the corrections become increasingly small. What these conditions basically state is that ρ_t should vanish as $t \to \infty$ [Eq. (3.18)] but on the other hand should not go to zero very fast [Eq. (3.17)]. Following arguments similar to those used before, it is easy to show that the algorithm also converges for constant $\rho_t = \rho$, provided ρ is properly bounded (Problem 3.2). In practice, the proper choice of the sequence ρ_t is vital for the convergence speed of the algorithm.

Variants of the Perceptron Algorithm

The algorithm we have presented is just one form of a number of variants that have been proposed for the training of a linear classifier in the case of linearly separable classes. We will now state another simpler and also popular form. The N training vectors enter the algorithm cyclically, one after the other. If the algorithm has not converged after the presentation of all the samples once, then the procedure

keeps repeating until convergence is achieved, that is, when all training samples have been classified correctly. Let $\boldsymbol{w}(t)$ be the weight vector estimate and $\boldsymbol{x}_{(t)}$ the corresponding feature vector, presented at the tth iteration step. The algorithm is stated as follows:

$$\begin{aligned}
&\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{x}_{(t)} \in \omega_1 \text{ and } \boldsymbol{w}^T(t)\boldsymbol{x}_{(t)} \leq 0 \\
&\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{x}_{(t)} \in \omega_2 \text{ and } \boldsymbol{w}^T(t)\boldsymbol{x}_{(t)} \geq 0 \\
&\boldsymbol{w}(t+1) = \boldsymbol{w}(t) \quad \text{otherwise}
\end{aligned} \tag{3.21}$$

In other words, if the current training sample is classified correctly, no action is taken. Otherwise, if the sample is misclassified, the weight vector is corrected by adding (subtracting) an amount proportional to $\boldsymbol{x}_{(t)}$. The algorithm belongs to a more general algorithmic family known as *reward and punishment* schemes. If the classification is correct, the reward is that no action is taken. If the current vector is misclassified, the punishment is the cost of correction. It can be shown that this form of the perceptron algorithm also converges in a finite number of iteration steps (Problem 3.3).

The perceptron algorithm was originally proposed by Rosenblatt in the late 1950s. The algorithm was developed for training the *perceptron*, the basic unit used for modeling neurons of the brain. This was considered central in developing powerful models for machine learning [Rose 58, Min 88].

The Perceptron

Once the perceptron algorithm has converged to a weight vector $\boldsymbol{w}$ and a threshold w_0, our next goal is the classification of an unknown feature vector to either of the two classes. Classification is achieved via the simple rule

$$\begin{aligned}
&\text{If} \quad \boldsymbol{w}^T\boldsymbol{x} + w_0 > 0 \quad \text{assign } \boldsymbol{x} \text{ to } \omega_1 \\
&\text{If} \quad \boldsymbol{w}^T\boldsymbol{x} + w_0 < 0 \quad \text{assign } \boldsymbol{x} \text{ to } \omega_2
\end{aligned} \tag{3.22}$$

A basic network unit that implements the operation is shown in Figure 3.3(a).

The elements of the feature vector $x_1, x_2, \ldots, x_l$ are applied to the *input nodes* of the network. Then each one is multiplied by the corresponding weights $w_i, i = 1, 2, \ldots, l$. These are known as *synaptic weights* or simply *synapses*. The products are summed up together with the *threshold* value w_0. The result then goes through a nonlinear device, which implements the so-called *activation function*. A common choice is a hard limiter; that is, $f(\cdot)$ is the step function [$f(x) = -1$ if $x < 0$ and $f(x) = 1$ if $x > 0$]. The corresponding feature vector is classified in one of the classes depending on the sign of the output. Besides $+1$ and -1, other values (class labels) for the hard limiter are also possible. Another popular choice is 1 and 0 and it is achieved by choosing the two levels of the step function appropriately.

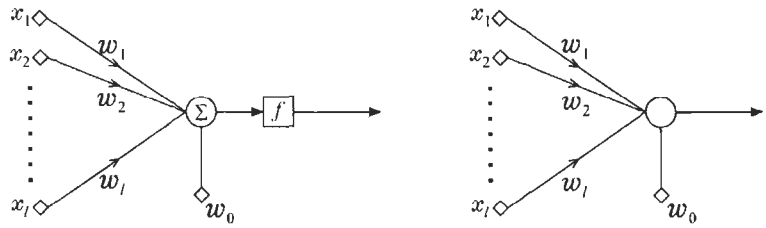


FIGURE 3.3: The basic perceptron model.

This basic network is known as a *perceptron* or *neuron*. Perceptrons are simple examples of the so-called *learning machines*, that is, structures whose free parameters are updated by a *learning algorithm*, such as the perceptron algorithm, in order to "learn" a specific task, based on a set of training data. Later on we will use the perceptron as the basic building element for more complex learning networks. Figure 3.3b is a simplified graph of the neuron where the summer and nonlinear device have been merged for notational simplification. Sometimes a neuron with a hard limiter device is referred to as a McCulloch–Pitts neuron. Other types of neurons will be considered in Chapter 4.

Example 3.1. Figure 3.4 shows the dashed line

$$x_1 + x_2 - 0.5 = 0$$

corresponding to the weight vector $[1, 1, -0.5]^T$, which has been computed from the latest iteration step of the perceptron algorithm (3.9), with $\rho_t = \rho = 0.7$. The line classifies correctly all the vectors except $[0.4, 0.05]^T$ and $[-0.20, 0.75]^T$. According to the algorithm, the next weight vector will be

$$\boldsymbol{w}(t+1) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix} - 0.7(-1) \begin{bmatrix} 0.4 \\ 0.05 \\ 1 \end{bmatrix} - 0.7(+1) \begin{bmatrix} -0.2 \\ 0.75 \\ 1 \end{bmatrix}$$

or

$$\boldsymbol{w}(t+1) = \begin{bmatrix} 1.42 \\ 0.51 \\ -0.5 \end{bmatrix}$$

The resulting new (solid) line $1.42x_1 + 0.51x_2 - 0.5 = 0$ classifies all vectors correctly and the algorithm is terminated.

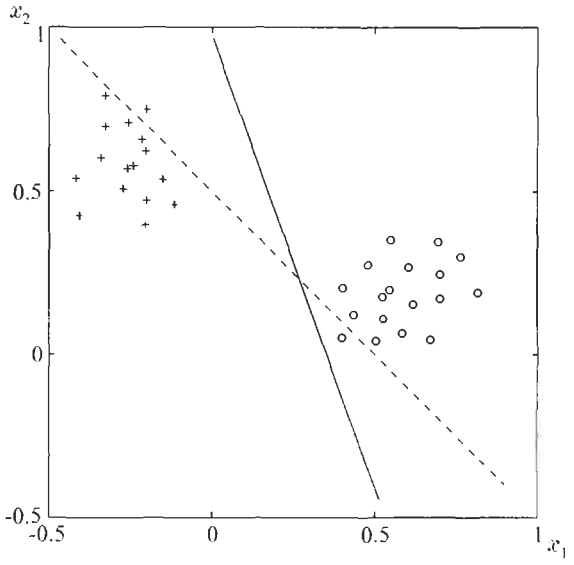


FIGURE 3.4: An example of the perceptron algorithm.

The Pocket Algorithm

A basic requirement for the convergence of the perceptron algorithm is the linear separability of the classes. If this is not true, as is usually the case in practice, the perceptron algorithm does not converge. A variant of the perceptron algorithm was suggested in [Gal 90] that converges to an optimal solution even if the linear separability condition is not fulfilled. The algorithm is known as the *pocket* algorithm and consists of the following two steps

- Initialize the weight vector $\boldsymbol{w}(0)$ randomly. Define a stored (in the pocket!) vector $\boldsymbol{w}_s$. Set a history counter h_s of the $\boldsymbol{w}_s$ to zero.
- At the tth iteration step compute the update $\boldsymbol{w}(t+1)$, according to the perceptron rule. Use the updated weight vector to test the number h of training vectors that are classified correctly. If $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(t+1)$ and h_s with h. Continue the iterations.

It can be shown that this algorithm converges with probability one to the optimal solution, that is, the one that produces the minimum number of misclassifications [Gal 90, Muse 97]. Other related algorithms that find reasonably good solutions when the classes are not linearly separable are the *thermal perceptron algorithm*

[Frea 92], the *loss minimization algorithm* [Hryc 92] and the barycentric correction procedure [Poul 95].

Kesler's Construction

So far we have dealt with the two-class case. The generalization to an M-class task is straightforward. A linear discriminant function $\boldsymbol{w}_i, i = 1, 2, \ldots, M$, is defined for each of the classes. A feature vector $\boldsymbol{x}$ (in the $(l + 1)$-dimensional space to account for the threshold) is classified in class ω_i if

$$\boldsymbol{w}_i^T \boldsymbol{x} > \boldsymbol{w}_j^T \boldsymbol{x}, \quad \forall j \neq i \tag{3.23}$$

This condition leads to the so-called Kesler's construction. For each of the training vectors from class $\omega_i, i=1, 2, \ldots, M$, we construct $M-1$ vectors $\boldsymbol{x}_{ij} = [\boldsymbol{0}^T, \boldsymbol{0}^T, \ldots, \boldsymbol{x}^T, \ldots, -\boldsymbol{x}^T, \ldots, \boldsymbol{0}^T]^T$ of dimension $(l+1)M \times 1$. That is, they are block vectors having zeros everywhere except at the ith and jth block positions, where they have $\boldsymbol{x}$ and $-\boldsymbol{x}$ respectively, for $j \neq i$. We also construct the block vector $\boldsymbol{w} = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_M^T]^T$. If $\boldsymbol{x} \in \omega_i$, this imposes the requirement that $\boldsymbol{w}^T \boldsymbol{x}_{ij} > 0, \forall j = 1, 2, \ldots, M, j \neq i$. The task now is to design a linear classifier, in the extended $(l + 1)M$-dimensional space, so that each of the $(M - 1)N$ training vectors lies in its positive side. The perceptron algorithm will have no difficulty in solving this problem for us, provided that such a solution is possible, that is, if all the training vectors can be correctly classified using a set of linear discriminant functions.

Example 3.2. Let us consider a three class problem in the two dimensional space. The training vectors for each of the classes are the following

$$\omega_1: [1, 1]^T, [2, 2]^T, [2, 1]^T$$
$$\omega_2: [1, -1]^T, [1, -2]^T, [2, -2]^T$$
$$\omega_3: [-1, 1]^T, [-1, 2]^T, [-2, 1]^T$$

This is obviously a linearly separable problem, since the vectors of different classes lie in different quadrants.

To compute the linear discriminant functions we first extend the vectors to the 3-dimensional space and then we use Kesler's construction. For example

For $[1, 1]^T$ we get $[1, 1, 1, -1, -1, -1, 0, 0, 0]^T$ and
$[1, 1, 1, 0, 0, 0, -1, -1, -1]^T$

For $[1, -2]^T$ we get $[-1, 2, -1, 1, -2, 1, 0, 0, 0]^T$ and
$[0, 0, 0, 1, -2, 1, -1, 2, -1]^T$

For $[-2, 1]^T$ we get $[2, -1, -1, 0, 0, 0, -2, 1, 1]^T$ and
$[0, 0, 0, 2, -1, -1, -2, 1, 1]^T$

Similarly we obtain the other twelve vectors. To obtain the corresponding weight vectors

$$\boldsymbol{w}_1 = [w_{11}, w_{12}, w_{10}]^T$$
$$\boldsymbol{w}_2 = [w_{21}, w_{22}, w_{20}]^T$$
$$\boldsymbol{w}_3 = [w_{31}, w_{32}, w_{30}]^T$$

we can run the perceptron algorithm by requiring $\boldsymbol{w}^T\boldsymbol{x} > 0$, $\boldsymbol{w} = [\boldsymbol{w}_1^T, \boldsymbol{w}_2^T, \boldsymbol{w}_3^T]^T$, for each of the eighteen 9-dimensional vectors. That is, we require all the vectors to lie on the same side of the decision hyperplane. The initial vector of the algorithm $\boldsymbol{w}(0)$ is computed using the uniform pseudorandom sequence generator in [0, 1]. The learning sequence ρ_t was chosen to be constant and equal to 0.5. The algorithm converges after four iterations and gives

$$\boldsymbol{w}_1 = [5.13, 3.60, 1.00]^T$$
$$\boldsymbol{w}_2 = [-0.05, -3.16, -0.41]^T$$
$$\boldsymbol{w}_3 = [-3.84, 1.28, 0.69]^T$$

3.4 LEAST SQUARES METHODS

As we have already pointed out, the attractiveness of linear classifiers lies in their simplicity. Thus, in many cases, although we know that the classes are not linearly separable, we still wish to adopt a linear classifier, despite the fact that this will lead to *suboptimal* performance from the classification error probability point of view. The goal now is to compute the corresponding weight vector under a suitable optimality criterion. The least squares methods are familiar to us, in one way or another, from our early college courses. Let us then build upon them.

3.4.1 Mean Square Error Estimation

Let us once more focus on the two-class problem. In the previous section we saw that the perceptron output was ± 1, depending on the class ownership of $\boldsymbol{x}$. Since the classes were linearly separable, these outputs were correct for all the training feature vectors, after, of course, the perceptron algorithm's convergence. In this section we will attempt to design a linear classifier so that its *desired* output is again ± 1, depending on the class ownership of the input vector. However, we will have to live with errors; that is, the true output will not always be equal to the desired one. Given a vector $\boldsymbol{x}$, the output of the classifier will be $\boldsymbol{w}^T\boldsymbol{x}$ (thresholds can be accommodated by vector extensions). The desired output will be denoted as $y(\boldsymbol{x}) \equiv y = \pm 1$. The weight vector will be computed so as to minimize the

mean square error (MSE) between the desired and true outputs, that is,

$$J(\boldsymbol{w}) = E[|y - \boldsymbol{x}^T \boldsymbol{w}|^2] \tag{3.24}$$

$$\hat{\boldsymbol{w}} = arg \min_{\boldsymbol{w}} J(\boldsymbol{w}) \tag{3.25}$$

The reader can easily check that $J(\boldsymbol{w})$ is equal to

$$J(\boldsymbol{w}) = P(\omega_1) \int (1 - \boldsymbol{x}^T \boldsymbol{w})^2 p(\boldsymbol{x}|\omega_1)\, d\boldsymbol{x} + P(\omega_2) \int (1 + \boldsymbol{x}^T \boldsymbol{w})^2 p(\boldsymbol{x}|\omega_2)\, d\boldsymbol{x} \tag{3.26}$$

Minimizing (3.24) easily results in

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 2E[\boldsymbol{x}(y - \boldsymbol{x}^T \boldsymbol{w})] = \boldsymbol{0} \tag{3.27}$$

Then

$$\hat{\boldsymbol{w}} = R_x^{-1} E[\boldsymbol{x} y] \tag{3.28}$$

where

$$R_x \equiv E[\boldsymbol{x}\boldsymbol{x}^T] = \begin{bmatrix} E[x_1 x_1] & \cdots & E[x_1 x_l] \\ E[x_2 x_1] & \cdots & E[x_2 x_l] \\ \vdots & \vdots & \vdots \\ E[x_l x_1] & \cdots & E[x_l x_l] \end{bmatrix} \tag{3.29}$$

is known as the *correlation* or autocorrelation matrix and is equal to the covariance matrix, introduced in the previous chapter, if the respective mean values are zero. The vector

$$E[\boldsymbol{x} y] = E\left[\begin{bmatrix} x_1 y \\ \vdots \\ x_l y \end{bmatrix}\right] \tag{3.30}$$

is known as the *cross-correlation* between the desired output and the (input) feature vectors. Thus, the mean square optimal weight vector results as the solution of a *linear set of equations*, provided, of course, that the correlation matrix is invertible.

It is interesting to point out that there is a geometrical interpretation of this solution. Random variables can be considered as points in a vector space. It is straightforward to see that the expectation operation $E[xy]$ between two random variables satisfies the properties of the inner product. Indeed, $E[x^2] \geq 0$, $E[xy] = E[yx]$, $E[x(c_1 y + c_2 z)] = c_1 E[xy] + c_2 E[xz]$. In such a vector space $\boldsymbol{w}^T \boldsymbol{x} = w_1 x_1 + \cdots + w_l x_l$ is a linear combination of vectors and thus it lies in the subspace defined by the x_i's.

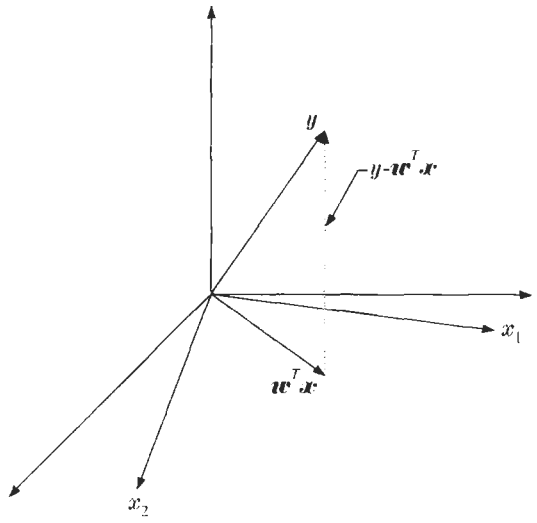


FIGURE 3.5: Interpretation of the MSE estimate as an orthogonal projection on the input vector elements' subspace.

This is illustrated by an example in Figure 3.5. Then, if we want to approximate y by this linear combination, the resulting error is $y-\boldsymbol{w}^T\boldsymbol{x}$. Equation (3.27) states that the minimum mean square error solution results if the error is orthogonal to each x_i, thus it is *orthogonal* to the vector subspace spanned by x_i, $i = 1, 2, \ldots, l$—in other words, if y is approximated by its *orthogonal projection* on the subspace (Figure 3.5). Equation (3.27) is also known as the *orthogonality condition*.

Multiclass Generalization

In the multiclass case the task is to design the M linear discriminant functions $g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x}$ according to the MSE criterion. The corresponding desired output responses (i.e., class labels) are chosen so that $y_i = 1$ if $\boldsymbol{x} \in \omega_i$ and $y_i = 0$ otherwise. This is in agreement with the two-class case. Indeed, for such a choice and if $M = 2$, the design of the decision hyperplane $\boldsymbol{w}^T\boldsymbol{x} \equiv (\boldsymbol{w}_1 - \boldsymbol{w}_2)^T\boldsymbol{x}$ corresponds to ± 1 desired responses, depending on the respective class ownership.

Let us now define $\boldsymbol{y}^T = [y_1, \ldots, y_M]$, for a given vector $\boldsymbol{x}$, and $W = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_M]$. That is, matrix W has as columns the weight vectors $\boldsymbol{w}_i$. The MSE criterion in (3.25) can now be generalized to minimize the norm of the error vector $\boldsymbol{y} - W^T\boldsymbol{x}$, that is,

$$\hat{W} = \arg\min_W E[\|\boldsymbol{y} - W^T\boldsymbol{x}\|^2] = \arg\min_W E\left[\sum_{i=1}^{M}\left(y_i - \boldsymbol{w}_i^T\boldsymbol{x}\right)^2\right] \tag{3.31}$$

This is equivalent to M MSE independent minimization problems of the (3.25) type, with scalar desired responses. *In other words, in order to design the MSE optimal linear discriminant functions, it suffices to design each one of them so that its desired output is* 1 *for vectors belonging to the corresponding class and* 0 *for all the others.*

3.4.2 Stochastic Approximation and the LMS Algorithm

The solution of (3.28) requires the computation of the correlation matrix and cross-correlation vector. This presupposes knowledge of the underlying distributions, which in general are not known. After all, if they were known, why not then use Bayesian classifiers? Thus, our major goal now becomes to see if it is possible to solve (3.27) without having this statistical information available. The answer has been provided by Robbins and Monro [Robb 51] in the more general context of stochastic approximation theory. Consider an equation of the form $E[F(\boldsymbol{x}_k, \boldsymbol{w})] = 0$, where $\boldsymbol{x}_k$, $k = 1, 2, \ldots$, is a sequence of random vectors from the same distribution, $F(\cdot, \cdot)$ a function, and $\boldsymbol{w}$ the vector of the unknown parameters. Then adopt the iterative scheme

$$\hat{\boldsymbol{w}}(k) = \hat{\boldsymbol{w}}(k-1) + \rho_k F(\boldsymbol{x}_k, \hat{\boldsymbol{w}}(k-1)) \tag{3.32}$$

In other words, the place of the *mean value* (which cannot be computed due to lack of information) is taken by the samples of the random variables resulting from the experiments. It turns out that under mild conditions the iterative scheme converges in probability to the solution $\boldsymbol{w}$ of the original equation, provided that the sequence ρ_k satisfies the two conditions

$$\sum_{k=1}^{\infty} \rho_k \to \infty \tag{3.33}$$

$$\sum_{k=1}^{\infty} \rho_k^2 < \infty \tag{3.34}$$

and which implies that

$$\rho_k \to 0 \tag{3.35}$$

That is,

$$\lim_{k \to \infty} \text{prob}\{\hat{\boldsymbol{w}}(k) = \boldsymbol{w}\} = 1 \tag{3.36}$$

The stronger, in the mean square sense, convergence is also true

$$\lim_{k \to \infty} E[\|\hat{\boldsymbol{w}}(k) - \boldsymbol{w}\|^2] = 0 \tag{3.37}$$

Conditions (3.33), (3.34) have already been met before and guarantee that the corrections of the estimates in the iterations tend to zero. Thus, for large values of k (in theory at infinity) iterations freeze. However, this must not happen too early (first condition) to make sure that the iterations do not stop away from the solution. The second condition guarantees that the accumulated noise, due to the stochastic nature of the variables, remains finite and the algorithm can cope with it [Fuku 90]. The proof is beyond the scope of the present text. However, we will demonstrate its validity via an example.

Let us consider the simple equation $E[x_k - w] = 0$. For $\rho_k = 1/k$ the iteration becomes

$$\hat{w}(k) = \hat{w}(k-1) + \frac{1}{k}[x_k - \hat{w}(k-1)] = \frac{(k-1)}{k}\hat{w}(k-1) + \frac{1}{k}x_k$$

For large values of k it is easy to see that

$$\hat{w}(k) = \frac{1}{k}\sum_{r=1}^{k} x_r$$

That is, the solution is the sample mean of the measurements. Most natural!

Let us now return to our original problem and apply the iteration to solve (3.27). Then (3.32) becomes

$$\hat{\boldsymbol{w}}(k) = \hat{\boldsymbol{w}}(k-1) + \rho_k \boldsymbol{x}_k\left(y_k - \boldsymbol{x}_k^T \hat{\boldsymbol{w}}(k-1)\right) \tag{3.38}$$

where $(y_k, \boldsymbol{x}_k)$ are the desired output (± 1)–input training sample pairs, successively presented to the algorithm. The algorithm is known as the least mean squares (LMS) or Widrow–Hoff algorithm, after those who suggested it in the early 1960s [Widr 60, Widr 90]. The algorithm converges asymptotically to the MSE solution.

A number of variants of the LMS algorithm have been suggested and used. The interested reader may consult, for example, [Hayk 96, Kalou 93]. A common variant is to use a constant ρ in the place of ρ_k. However, in this case the algorithm does not converge to the MSE solution. It can be shown, for example, [Hayk 96] that if $0 < \rho < 2/\text{trace}\{R_x\}$ then

$$E[\hat{\boldsymbol{w}}(k)] \rightarrow \boldsymbol{w}_{MSE} \quad \text{and} \quad E[\|\hat{\boldsymbol{w}}(k) - \boldsymbol{w}_{MSE}\|^2] \rightarrow \text{constant} \tag{3.39}$$

where $\boldsymbol{w}_{MSE}$ denotes the MSE optimal estimate and trace$\{\cdot\}$ the trace of the matrix. That is, the mean value of the LMS estimate is equal to the MSE solution and also the corresponding variance remains finite. It turns out that the smaller the ρ, the smaller the variance around the desired MSE solution. However, the smaller the ρ, the slower the convergence of the LMS algorithm. The reason for using constant ρ in place of a vanishing sequence is to keep the algorithm "alert" to track variations when the statistics is not stationary but is *slowly* varying, that is, when the underlying distributions are time dependent.

Remarks

- Observe that in the case of the LMS, the parameters' update iteration step, k, coincides with the index of the current input sample $\boldsymbol{x}_k$. In case k is a time index, LMS is a time-*adaptive* scheme, which adapts to the solution as successive samples become available to the system.
- Observe that equation (3.38) can be seen as the training algorithm of a *linear neuron*, that is, a neuron without the nonlinear activation function. This type of training, which neglects the nonlinearity *during training* and applies the desired response just after the adder of the linear combiner part of the neuron (Figure 3.3a), was used by Widrow and Hoff. The resulting neuron architecture is known as *adaline* (adaptive linear element). After training and once the weights have been fixed, the model is the same as in Figure 3.3, with the hard limiter following the linear combiner. In other words, the adaline is a neuron which is trained according to the LMS instead of the perceptron algorithm.

3.4.3 Sum of Error Squares Estimation

A criterion closely related to the MSE is the *sum of error squares* criterion defined as

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{w}\right)^2 \equiv \sum_{i=1}^{N} e_i^2 \tag{3.40}$$

In other words, the errors between the desired output of the classifier (± 1 in the two class case) and the true output are summed up over all the available training feature vectors, instead of averaging them out. In this way we overcome the need for explicit knowledge of the underlying pdf's. Minimizing (3.40) with respect to $\boldsymbol{w}$ results in

$$\sum_{i=1}^{N} \boldsymbol{x}_i \left(y_i - \boldsymbol{x}_i^T \hat{\boldsymbol{w}}\right) = 0 \Rightarrow \left(\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T\right) \hat{\boldsymbol{w}} = \sum_{i=1}^{N} (\boldsymbol{x}_i y_i) \tag{3.41}$$

For the sake of mathematical formulation let us define

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.42}$$

That is, X is an $N \times l$ matrix whose rows are the available training feature vectors, and $\boldsymbol{y}$ is a vector consisting of the corresponding desired responses. Then $\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T = X^T X$ and also $\sum_{i=1}^{N} \boldsymbol{x}_i y_i = X^T \boldsymbol{y}$. Hence, (3.41) can now be

written as

$$(X^T X)\hat{\boldsymbol{w}} = X^T \boldsymbol{y} \Rightarrow \hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y} \tag{3.43}$$

Thus, the optimal weight vector is again provided as the solution of a linear set of equations. Matrix $X^T X$ is known as the *sample correlation matrix*. Matrix $X^+ \equiv (X^T X)^{-1} X^T$ is known as the *pseudoinverse* of X, and it is meaningful only if $X^T X$ is invertible, that is, X is of rank l. X^+ is a generalization of the inverse of an invertible square matrix. Indeed, if X is an $l \times l$ square and invertible matrix, then it is straightforward to see that $X^+ = X^{-1}$. In such a case the estimated weight vector is the solution of the linear system $X\hat{\boldsymbol{w}} = \boldsymbol{y}$. If, however, there are more equations than unknowns, $N > l$, as is the usual case in pattern recognition, there is not, in general, a solution. The solution obtained by the pseudoinverse is the vector that minimizes the sum of error squares. It is easy to show that (under mild assumptions) the sum of error squares tends to the MSE solution for large values of N (Problem 3.8).

Remarks

- So far we have restricted the desired output values to be ± 1. Of course, this is not necessary. All we actually need is a desired response positive for ω_1 and negative for ω_2. Thus, in the place of ± 1 in the $\boldsymbol{y}$ vector we could have any positive (negative) values. Obviously, all we have said so far is still applicable. However, the interesting aspect of this generalization would be to compute these desired values in an optimal way, to obtain a better solution. The Ho–Kashyap algorithm is such a scheme solving for both the optimal $\boldsymbol{w}$ and optimal desired values y_i. The interested reader may consult [Ho 65, Tou 74].
- Generalization to the multiclass case follows the same concept as that introduced for the MSE cost, and it is easily shown that it reduces to M equivalent problems of scalar desired responses, one for each discriminant function (Problem 3.10).

Example 3.3. Class ω_1 consists of the two-dimensional vectors $[0.2, 0.7]^T$, $[0.3, 0.3]^T$, $[0.4, 0.5]^T$, $[0.6, 0.5]^T$, $[0.1, 0.4]^T$ and class ω_2 of $[0.4, 0.6]^T$, $[0.6, 0.2]^T$, $[0.7, 0.4]^T$, $[0.8, 0.6]^T$, $[0.7, 0.5]^T$ Design the sum of error squares optimal linear classifier $w_1 x_1 + w_2 x_2 + w_0 = 0$.

We first extend the given vectors by using 1 as their third dimension and form the 10×3 matrix X, which has as rows the transposes of these vectors. The resulting sample correlation 3×3 matrix $X^T X$ is equal to

$$X^T X = \begin{bmatrix} 2.8 & 2.24 & 4.8 \\ 2.24 & 2.41 & 4.7 \\ 4.8 & 4.7 & 10 \end{bmatrix}$$

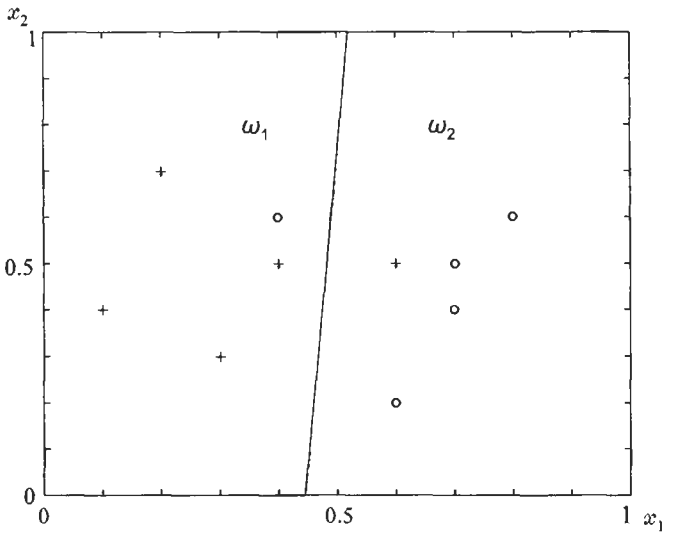


FIGURE 3.6: Least sum of error squares linear classifier.

The corresponding y consists of five 1's and then five -1's and

$$X^T y = \begin{bmatrix} -1.6 \\ 0.1 \\ 0.0 \end{bmatrix}$$

Solving the corresponding set of equations results in $[w_1, w_2, w_0] = [-3.218, 0.241, 1.431]$. Figure 3.6 shows the resulting geometry.

3.5 MEAN SQUARE ESTIMATION REVISITED

3.5.1 Mean Square Error Regression

In this subsection we will approach the MSE task from a slightly different perspective and in a more general framework.

Let $\boldsymbol{y}, \boldsymbol{x}$ be two random vectors of dimensions $M \times 1$ and $l \times 1$, respectively, and assume that they are described by the joint pdf $p(\boldsymbol{y}, \boldsymbol{x})$. The task of interest is to estimate the value of $\boldsymbol{y}$, given the value of $\boldsymbol{x}$, obtained from an experiment. No doubt the classification task falls under this more general formulation. For example, when we are given a feature vector $\boldsymbol{x}$, our goal is to estimate the value of the class label y, which is ± 1 in the two-class case.

The mean square estimate $\hat{\boldsymbol{y}}$ of the random vector $\boldsymbol{y}$, given the value $\boldsymbol{x}$, is defined as

$$\hat{\boldsymbol{y}} = arg \min_{\tilde{\boldsymbol{y}}} E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] \tag{3.44}$$

Note that the mean value here is with respect to the conditional pdf $p(\boldsymbol{y}|\boldsymbol{x})$. We will show that the optimal estimate is the mean value of $\boldsymbol{y}$, that is,

$$\hat{\boldsymbol{y}} = E[\boldsymbol{y}|\boldsymbol{x}] \equiv \int_{-\infty}^{\infty} \boldsymbol{y} p(\boldsymbol{y}|\boldsymbol{x})\, d\boldsymbol{y} \tag{3.45}$$

Proof. Let $\tilde{\boldsymbol{y}}$ be another estimate. It will be shown that it results in higher mean square error. Indeed,

$$\begin{aligned} E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] &= E[\|\boldsymbol{y} - \hat{\boldsymbol{y}} + \hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}\|^2] = E[\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2] \\ &\quad + E[\|\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}\|^2] + 2E[(\boldsymbol{y} - \hat{\boldsymbol{y}})^T(\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}})] \end{aligned} \tag{3.46}$$

where the dependence on $\boldsymbol{x}$ has been omitted for notational convenience. Note now that $\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}$ is a constant. Thus,

$$E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] \geq E[\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2] + 2E[(\boldsymbol{y} - \hat{\boldsymbol{y}})^T](\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}) \tag{3.47}$$

and from the definition of $\hat{\boldsymbol{y}} = E[\boldsymbol{y}]$ it follows that

$$E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] \geq E[\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2] \tag{3.48}$$

■

Remark

- This is a very elegant result. Given a measured value of $\boldsymbol{x}$, the best (in the MSE sense) estimate of $\boldsymbol{y}$ is given by the function $\boldsymbol{y}(\boldsymbol{x}) \equiv E[\boldsymbol{y}|\boldsymbol{x}]$. In general this is a *nonlinear* vector-valued function of $\boldsymbol{x}$ (i.e., $\boldsymbol{g}(\cdot) \equiv [g_1(\cdot), \ldots, g_M(\cdot)]^T$), and it is known as the *regression* of $\boldsymbol{y}$ *conditioned* on $\boldsymbol{x}$. It can be shown (Problem 3.11) that if $(\boldsymbol{y}, \boldsymbol{x})$ are jointly Gaussian, then the MSE optimal regressor is a linear function.

3.5.2 MSE Estimates Posterior Class Probabilities

In the beginning of the chapter we promised to "emancipate" ourselves from the Bayesian classification. However, the nice surprise is that a Bayesian flavor still remains, although in a disguised form. Let us reveal it—it can only be beneficial.

We will consider the multiclass case. Given $\boldsymbol{x}$, we want to estimate its class label. Let $g_i(\boldsymbol{x})$ be the discriminant functions to be designed. The cost function in Equation (3.31) now becomes

$$J = E\left[\sum_{i=1}^{M}(g_i(\boldsymbol{x}) - y_i)^2\right] \equiv E[\|\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{y}\|^2] \tag{3.49}$$

where the vector $\boldsymbol{y}$ consists of zeros and a single 1 at the appropriate place. Note that each $g_i(\boldsymbol{x})$ depends *only* on $\boldsymbol{x}$, whereas the y_i's depend on the class ω_j to which $\boldsymbol{x}$ belongs. Let $p(\boldsymbol{x}, \omega_i)$ be the joint probability density of the feature vector belonging to the ith class. Then (3.49) is written as

$$J = \int_{-\infty}^{+\infty} \sum_{j=1}^{M} \left\{\sum_{i=1}^{M}(g_i(\boldsymbol{x}) - y_i)^2\right\} p(\boldsymbol{x}, \omega_j)\, d\boldsymbol{x} \tag{3.50}$$

Taking into account that $p(\boldsymbol{x}, \omega_j) = P(\omega_j|\boldsymbol{x})p(\boldsymbol{x})$, (3.50) becomes

$$\begin{aligned} J &= \int_{-\infty}^{\infty} \left\{\sum_{j=1}^{M}\sum_{i=1}^{M}(g_i(\boldsymbol{x}) - y_i)^2\, P(\omega_j|\boldsymbol{x})\right\} p(\boldsymbol{x})\, d\boldsymbol{x} \\ &= E\left[\sum_{j=1}^{M}\sum_{i=1}^{M}(g_i(\boldsymbol{x}) - y_i)^2\, P(\omega_j|\boldsymbol{x})\right] \end{aligned} \tag{3.51}$$

where the mean is taken with respect to $\boldsymbol{x}$. Expanding this, we get

$$J = E\left[\sum_{j=1}^{M}\sum_{i=1}^{M}\left(g_i^2(\boldsymbol{x})P(\omega_j|\boldsymbol{x}) - 2g_i(\boldsymbol{x})y_i\,P(\omega_j|\boldsymbol{x}) + y_i^2 P(\omega_j|\boldsymbol{x})\right)\right] \tag{3.52}$$

Exploiting the fact that $g_i(\boldsymbol{x})$ is a function of $\boldsymbol{x}$ only and $\sum_{j=1}^{M} P(\omega_j|\boldsymbol{x}) = 1$, (3.52) becomes

$$\begin{aligned} J &= E\left[\sum_{i=1}^{M}\left(g_i^2(\boldsymbol{x}) - 2g_i(\boldsymbol{x})\sum_{j=1}^{M} y_i\,P(\omega_j|\boldsymbol{x}) + \sum_{j=1}^{M} y_i^2 P(\omega_j|\boldsymbol{x})\right)\right] \\ &= E\left[\sum_{i=1}^{M}\left(g_i^2(\boldsymbol{x}) - 2g_i(\boldsymbol{x})E[y_i|\boldsymbol{x}] + E[y_i^2|\boldsymbol{x}]\right)\right] \end{aligned} \tag{3.53}$$

where $E[y_i|\boldsymbol{x}]$ and $E[y_i^2|\boldsymbol{x}]$ are the respective mean values conditioned on $\boldsymbol{x}$. Adding and subtracting $(E[y_i|\boldsymbol{x}])^2$, Eq. (3.53) becomes

$$J = E\left[\sum_{i=1}^{M}\left(g_i(\boldsymbol{x}) - E[y_i|\boldsymbol{x}]\right)^2\right] + E\left[\sum_{i=1}^{M}\left(E[y_i^2|\boldsymbol{x}] - (E[y_i|\boldsymbol{x}])^2\right)\right] \tag{3.54}$$

The second term in (3.54) does not depend on the functions $g_i(\boldsymbol{x})$, $i = 1, 2, \ldots, M$. Thus, minimization of J with respect to (the parameters of) $g_i(\cdot)$ affects only the first of the two terms. Let us concentrate and look at it more carefully. Each of the M summands involves two terms: the unknown discriminant function $g_i(\cdot)$ and the conditional mean of the corresponding desired response. Let us now write $g_i(\cdot) = g_i(\cdot; \boldsymbol{w}_i)$, to state explicitly that the functions are defined in terms of a set of parameters, to be determined optimally during training. Minimizing J with respect to $\boldsymbol{w}_i$, $i = 1, 2, \ldots, M$, results in *the mean square estimates of the unknown parameters, $\hat{\boldsymbol{w}}_i$, so that the discriminant functions approximate optimally the corresponding conditional means — that is, the regressions of y_i conditioned on $\boldsymbol{x}$*. Moreover, for the M-class problem and the preceding definitions we have

$$E[y_i|\boldsymbol{x}] \equiv \sum_{j=1}^{M} y_i P(\omega_j|\boldsymbol{x}) \tag{3.55}$$

However $y_i = 1(0)$ if $\boldsymbol{x} \in \omega_i (\boldsymbol{x} \in \omega_j, j \neq i)$. Hence

$$g_i(\boldsymbol{x}, \hat{\boldsymbol{w}}_i) \text{ is the MSE estimate of } P(\omega_i|\boldsymbol{x}) \tag{3.56}$$

This is an important result. Training the discriminant functions g_i with desired outputs 1 or 0 in the MSE sense, Eq. (3.49) is equivalent to obtaining the MSE estimates of the class posterior probabilities, without using any statistical information or pdf modeling! It suffices to say that these estimates may in turn be used for Bayesian classification. An important issue here is to assess *how good the resulting estimates are. It all depends on how well the adopted functions $g_i(\cdot; \boldsymbol{w}_i)$ can model the desired (in general) nonlinear functions $P(\omega_i|\boldsymbol{x})$*. If, for example, we adopt linear models, as was the case in Eq. (3.31), and $P(\omega_i|\boldsymbol{x})$ is highly nonlinear, the resulting MSE optimal approximation will be a bad one. Our focus in the next chapter will be on developing modeling techniques for nonlinear functions.

Finally, it must be emphasized that the *conclusion above is an implication of the cost function itself and not of the specific model function used.* The latter plays its part when the approximation accuracy issue comes into the scene. MSE cost is just one of the costs that have this important property. Other cost functions share this property too, see for example, [Rich 91, Bish 95, Pear 90, Cid 99].

3.5.3 The Bias–Variance Dilemma

So far we have touched on some very important issues concerning the interpretation of the output of an optimally designed classifier. We saw that a classifier can be viewed as a *learning machine* realizing a set of functions $g(\boldsymbol{x})$, which attempt to estimate the corresponding class label y and make a decision based on these estimates. In practice, the functions $g(\cdot)$ are estimated using a *finite* training data set $\mathcal{D} = \{(y_i, \boldsymbol{x}_i), i = 1, 2, \ldots, N\}$ and a suitable methodology (e.g., sum of error squares, LMS, maximum likelihood). To emphasize the explicit dependence on $\mathcal{D}$ we write $g(\boldsymbol{x}; \mathcal{D})$. This subsection is focused on the capabilities of $g(\boldsymbol{x}; \mathcal{D})$ to approximate the MSE optimal regressor $E[y|\boldsymbol{x}]$ and on how this is affected by the data size N.

The key factor here is the dependence of the approximation on $\mathcal{D}$. The approximation may be very good for a specific training data set but very bad for another. The effectiveness of an estimator can be evaluated by computing its mean square deviation from the desired optimal value. This is achieved by averaging over all possible sets $\mathcal{D}$ of size N, that is,

$$E_{\mathcal{D}}\left[(g(\boldsymbol{x}; \mathcal{D}) - E[y|\boldsymbol{x}])^2\right] \tag{3.57}$$

If we add and subtract $E_{\mathcal{D}}[g(\boldsymbol{x}; \mathcal{D})]$ and follow a procedure similar to that in the proof of (3.45), we easily obtain

$$\begin{aligned} E_{\mathcal{D}}\left[(g(\boldsymbol{x}; \mathcal{D}) - E[y|\boldsymbol{x}])^2\right] &= (E_{\mathcal{D}}[g(\boldsymbol{x}; \mathcal{D})] - E[y|\boldsymbol{x}])^2 \\ &\quad + E_{\mathcal{D}}\left[(g(\boldsymbol{x}; \mathcal{D}) - E_{\mathcal{D}}[g(\boldsymbol{x}; \mathcal{D})])^2\right] \end{aligned} \tag{3.58}$$

The first term is the contribution of the *bias* and the second that of the *variance*. In other words, even if the estimator is unbiased, it can still result in a large mean square error due to a large variance term. For a finite data set, it turns out that there is a trade-off between these two terms. Increasing the bias decreases the variance and vice versa. This is known as the *bias–variance dilemma*. This behavior is reasonable. The problem at hand is similar to that of a curve fitting through a given data set. If, for example, the adopted model is complex (many parameters involved) with respect to the number N, the model will fit the idiosyncrasies of the specific data set. Thus, it will result in low bias but will yield high variance, as we change from one data set to another. The major issue now is to seek ways to make both bias and variance low at the same time. It turns out that this may be possible only asymptotically, as the number N grows to infinity. Moreover, N has to grow in such a way as *to allow more complex models, g, to be fitted (which reduces bias) and at the same time to ensure low variance*. However, in practice N is finite and one should aim at the best compromise. If, on the other hand, some *a priori*

knowledge is available, this must be exploited in the form of constraints that the classifier has to satisfy. This can lead to lower values of both the variance and the bias, compared with a more general type of classifier. This is natural, because one takes advantage of the available information and helps the optimization process. A simple and excellent treatment of the topic can be found in [Gema 92]. As for ourselves, this was only the beginning. We will come to the finite data set issue and its implications many times throughout this book and from different points of view.

3.6 SUPPORT VECTOR MACHINES

3.6.1 Separable Classes

In this section an alternative rationale for designing linear classifiers will be adopted. We will start with the two-class linearly separable task and then we will extend the method to more general cases where data are not separable.

Let $\boldsymbol{x}_i, i = 1, 2, \ldots, N$, be the feature vectors of the training set, X. These belong to either of two classes, ω_1, ω_2, which are assumed to be linearly separable. The goal, once more, is to design a hyperplane

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = 0 \tag{3.59}$$

that classifies correctly all the training vectors. As we have already discussed in Section 3.3, such a hyperplane is not unique. The perceptron algorithm may converge to any one of the possible solutions. Having gained in experience, this time we will be more demanding. Figure 3.7 illustrates the classification task with two possible hyperplane[1] solutions. Both hyperplanes do the job for the training set. However, which one of the two would any sensible engineer choose as the classifier for operation in practice, where data outside the training set will be fed to it? No doubt the answer is: the full-line one. The reason is that this hyperplane leaves more "room" on either side, so that data in both classes can move a bit more freely, with less risk of causing an error. Thus such a hyperplane can be trusted more, when it is faced with the challenge of operating with unknown data. Here we have touched a very important issue in the classifier design stage. It is known as the *generalization performance of the classifier*. This refers to the capability of the classifier, designed using the training data set, to operate satisfactorily with data outside this set. We will come to this issue over and over again.

After the above brief discussion, we are ready to accept that a very sensible choice for the hyperplane classifier would be the one that leaves the maximum margin from both classes. Later on, at the end of Chapter 5, we will see that this

[1] We will refer to lines as hyperplanes to cover the general case.

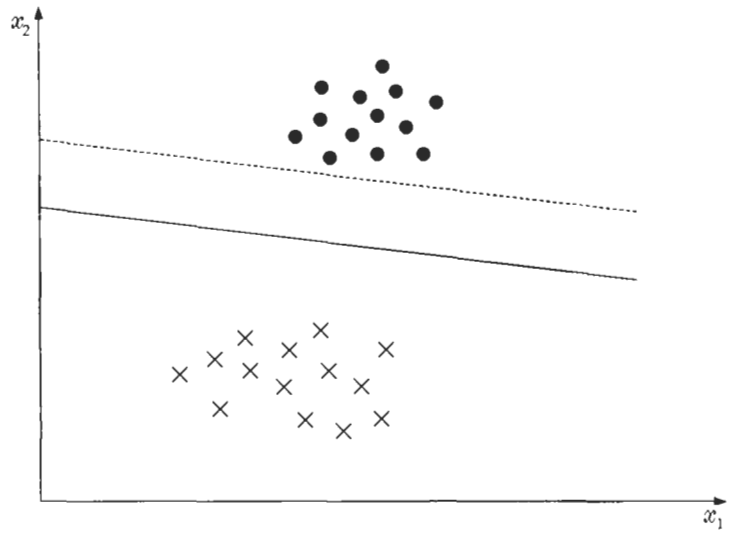


FIGURE 3.7: An example of a linearly separable two class problem with two possible linear classifiers.

sensible choice has a deeper justification, springing from the elegant mathematical formulation that Vapnik and Chernovenkis have offered to us.

Let us now quantify the term "margin" that a hyperplane leaves from both classes. Every hyperplane is characterized by its direction (determined by $\boldsymbol{w}$) and its exact position in space (determined by w_0). Since we want to give no preference to either of the classes, then it is reasonable for each direction to select that hyperplane which has the same distance from the respective nearest points in ω_1 and ω_2. This is illustrated in Figure 3.8. The hyperplanes shown with dark lines are the selected ones from the infinite set in the respective direction. The margin for direction "1" is $2z_1$ and the margin for direction "2" is $2z_2$. *Our goal is to search for the direction that gives the maximum possible margin.* However, each hyperplane is determined within a scaling factor. We will free ourselves from it, by appropriate scaling of all the candidate hyperplanes. Recall from Section 3.2 that the distance of a point from a hyperplane is given by

$$z = \frac{|g(\boldsymbol{x})|}{\|\boldsymbol{w}\|}$$

We can now scale $\boldsymbol{w}$, w_0 so that the value of $g(\boldsymbol{x})$, at the nearest points in ω_1, ω_2 (circled in Figure 3.8), is equal to 1 for ω_1 and, thus, equal to -1 for ω_2. This is

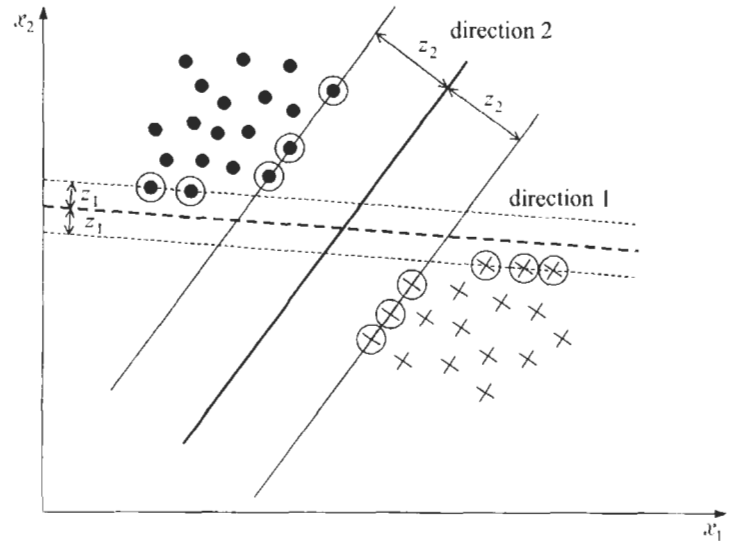


FIGURE 3.8: The margin for direction 2 is larger than the margin for direction 1.

equivalent with

1. Having a margin of $\frac{1}{\|\boldsymbol{w}\|} + \frac{1}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$
2. Requiring that

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 1, \qquad \forall \boldsymbol{x} \in \omega_1$$
$$\boldsymbol{w}^T\boldsymbol{x} + w_0 \leq -1, \qquad \forall \boldsymbol{x} \in \omega_2$$

We have now reached the point where mathematics will take over. For each $\boldsymbol{x}_i$, we denote the corresponding class indicator by y_i ($+1$ for ω_1, -1 for ω_2.) Our task can now be summarized as: Compute the parameters $\boldsymbol{w}$, w_0 of the hyperplane so that to:

$$\text{minimize} \quad J(\boldsymbol{w}) \equiv \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{3.60}$$

$$\text{subject to} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) \geq 1, \quad i = 1, 2, \ldots, N \tag{3.61}$$

Obviously, minimizing the norm makes the margin maximum. This is a nonlinear (quadratic) optimization task subject to a set of linear inequality constraints. The Karush–Kuhn–Tucker (KKT) conditions (Appendix C) that the minimizer of

(3.60), (3.61) has to satisfy are

$$\frac{\partial}{\partial \boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = \mathbf{0} \tag{3.62}$$

$$\frac{\partial}{\partial w_0} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = 0 \tag{3.63}$$

$$\lambda_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.64}$$

$$\lambda_i [y_i (\boldsymbol{w}^T \boldsymbol{x}_i + w_0) - 1] = 0, \quad i = 1, 2, \ldots, N \tag{3.65}$$

where $\boldsymbol{\lambda}$ is the vector of the Langrange multipliers, λ_i, and $\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda})$ is the Langrangian function defined as

$$\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} - \sum_{i=1}^{N} \lambda_i [y_i (\boldsymbol{w}^T \boldsymbol{x}_i + w_0) - 1] \tag{3.66}$$

Combining (3.66) with (3.62) and (3.63) results in

$$\boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i \tag{3.67}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.68}$$

Remarks

- The Langrange multipliers can be either zero or positive (Appendix C.) Thus, the vector parameter $\boldsymbol{w}$ of the optimal solution is a linear combination of $N_s \leq N$ feature vectors which are associated with $\lambda_i \neq 0$. That is,

$$\boldsymbol{w} = \sum_{i=1}^{N_s} \lambda_i y_i \boldsymbol{x}_i \tag{3.69}$$

  These are known as *support vectors* and the optimum hyperplane classifier as a *support vector machine* (SVM). As it is pointed out in Appendix C, a nonzero Langrange multiplier corresponds to a so called active constraint. Hence, as the set of constraints in (3.65) suggest for $\lambda_i \neq 0$, *the support vectors lie on either of the two hyperplanes*, i.e.,

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = \pm 1 \tag{3.70}$$

  that is, they are the training vectors that are closest to the linear classifier, and they constitute the *critical elements of the training set*. Feature vectors

corresponding to $\lambda_i = 0$ can either lie outside the "class separation band," defined as the region between the two hyperplanes given in (3.70), or they can also lie on one of these hyperplanes (degenerate case, Appendix C.) The resulting hyperplane classifier is insensitive to the number and position of such feature vectors, provided they do not cross the class separation band.

- Although $\boldsymbol{w}$ is explicitly given, w_0 can be implicitly obtained by any of the (*complementary slackness*) conditions (3.65), satisfying strict complementarity (i.e., $\lambda_i \neq 0$, Appendix C.) In practice, w_0 is computed as an average value obtained using all conditions of this type.
- The cost function in (3.60) is a strict convex one (Appendix C), a property that is guaranteed by the fact that the corresponding Hessian matrix is positive definite [Flet 87]. Furthermore, the inequality constraints consist of linear functions. As discussed in Appendix C, these two conditions guarantee that any local minimum is also global and unique. This is most welcome. *The optimal hyperplane classifier of a support vector machine is unique.*

Having stated all these very interesting properties of the optimal hyperplane of a support vector machine, the next step is the computation of the involved parameters. From a computational point of view this is not always an easy task and a number of algorithms exist, e.g., [Baza 79]. We will move to a path which is suggested to us by the special nature of our optimization task, given in (3.60) and (3.61). It belongs to the *convex programming* family of problems, since the cost function is convex and the set of constraints are linear and define a convex set of feasible solutions. As we discuss in Appendix C, such problems can be solved by considering the so called *Langrangian duality* and the problem can be stated equivalently by its Wolfe dual representation form, i.e.,

$$\text{maximize} \quad \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) \tag{3.71}$$

$$\text{subject to} \quad \boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i \tag{3.72}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.73}$$

$$\boldsymbol{\lambda} \geq \mathbf{0} \tag{3.74}$$

The two equality constraints are the result of equating to zero the gradient of the Langrangian, with respect to $\boldsymbol{w}$, w_0. We have already gained something. The training feature vectors enter into the problem via equality constraints and not inequality ones, which can be easier to handle. Substituting (3.72) and (3.73) into (3.71) and

after a bit of algebra we end up with the equivalent optimization task

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right) \tag{3.75}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.76}$$

$$\boldsymbol{\lambda} \geq \mathbf{0} \tag{3.77}$$

Once the optimal Langrangian multipliers have been computed, by maximizing (3.75), the optimal hyperplane is obtained via (3.72), and w_0 via the complementary slackness conditions, as before.

Remarks

- Besides the more attractive setting of the involved constraints in (3.75), (3.76), there is another important reason that makes this formulation popular. The training vectors enter into the game in pairs, in the form of inner products. This is most interesting. *The cost function does not depend explicitly on the dimensionality of the input space*! This property allows for efficient generalizations in the case of nonlinearly separable classes. We will return to this at the end of Chapter 4.
- Although the resulting optimal hyperplane is unique, there is no guarantee about the uniqueness of the associated Langrange multipliers λ_i. In words, the expansion of $\boldsymbol{w}$ in terms of support vectors in (3.72) may not be unique, although the final result is unique (Example 3.4).

3.6.2 Nonseparable Classes

In the case where the classes are not separable, the above setup is not valid any more. Figure 3.9 illustrates the case. The two classes are not separable. Any attempt to draw a hyperplane will never end up with a class separation band with no data points inside it, as was the case in the linearly separable task. Recall that the margin is defined as the distance between the pair of parallel hyperplanes described by

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = \pm 1$$

The training feature vectors now belong to one of the following three categories:

- Vectors that fall outside the band and are correctly classified. These vectors comply with the constraints in (3.61).

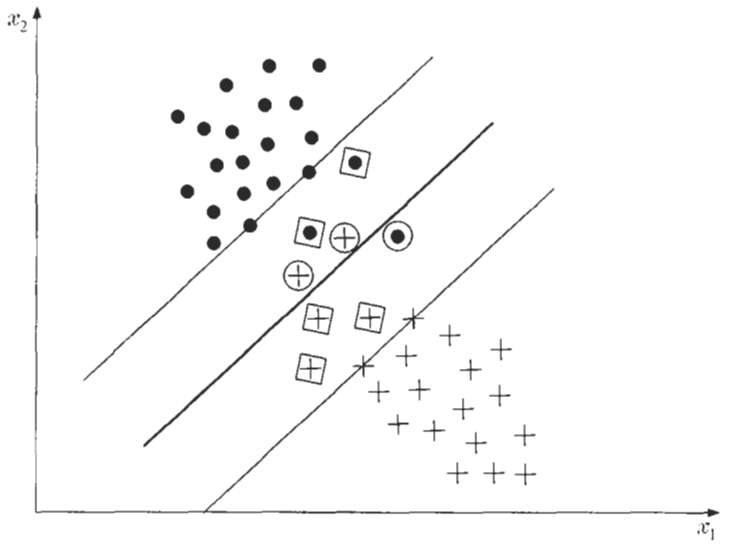


FIGURE 3.9: In the nonseparable class case, points fall inside the class separation band.

- Vectors falling inside the band and which are correctly classified. These are the points placed in squares in Figure 3.9 and they satisfy the inequality

$$0 \leq y_i(\boldsymbol{w}^T\boldsymbol{x} + w_0) < 1$$

- Vectors that are misclassified. They are enclosed by circles and obey the inequality

$$y_i(\boldsymbol{w}^T\boldsymbol{x} + w_0) < 0$$

All three cases can be treated under a single type of constraints by introducing a new set of variables, namely

$$y_i[\boldsymbol{w}^T\boldsymbol{x} + w_0] \geq 1 - \xi_i \tag{3.78}$$

The first category of data correspond to $\xi_i = 0$, the second to $0 < \xi_i \leq 1$, and the third to $\xi_i > 1$. The variables ξ_i are known as *slack variables*. The optimizing task becomes more involved, yet it falls under the same rationale as before. The goal now is to make the margin as large as possible but at the same time to keep the number of points with $\xi > 0$ as small as possible. In mathematical terms, this is

equivalent to adopting to minimize the cost function

$$J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i) \tag{3.79}$$

where $\boldsymbol{\xi}$ is the vector of the parameters ξ_i and

$$I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases} \tag{3.80}$$

The parameter C is a positive constant that controls the relative influence of the two competing terms. However, optimization of the above is difficult since it involves a discontinuous function $I(\cdot)$. As it is common in such cases, we choose to optimize a closely related cost function, and the goal becomes

$$\text{minimize} \quad J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i \tag{3.81}$$

$$\text{subject to} \quad y_i[\boldsymbol{w}^T\boldsymbol{x}_i + w_0] \geq 1 - \xi_i, \quad i = 1, 2, \ldots, N \tag{3.82}$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.83}$$

The problem is again a convex programming one, and the corresponding Langrangian is given by

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i \\ - \sum_{i=1}^{N}\lambda_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - 1 + \xi_i] \end{aligned} \tag{3.84}$$

The corresponding Karush–Kuhn–Tucker conditions are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{0} \quad \text{or} \quad \boldsymbol{w} = \sum_{i=1}^{N}\lambda_i y_i \boldsymbol{x}_i \tag{3.85}$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = 0 \quad \text{or} \quad \sum_{i=1}^{N}\lambda_i y_i = 0 \tag{3.86}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \quad \text{or} \quad C - \mu_i - \lambda_i = 0, \quad i = 1, 2, \ldots, N \tag{3.87}$$

$$\lambda_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - 1 + \xi_i] = 0, \quad i = 1, 2, \ldots, N \tag{3.88}$$

$$\mu_i \xi_i = 0, \quad i = 1, 2, \ldots, N \tag{3.89}$$

$$\mu_i \geq 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.90}$$

The associated Wolfe dual representation now becomes

$$\begin{aligned} \text{maximize} \quad & \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}) \\ \text{subject to} \quad & \boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i \\ & \sum_{i=1}^{N} \lambda_i y_i = 0 \\ & C - \mu_i - \lambda_i = 0, \quad i = 1, 2, \ldots, N \\ & \lambda_i \geq 0, \ \mu_i \geq 0, \quad i = 1, 2, \ldots, N \end{aligned}$$

Substituting the above equality constraints into the Langrangian we end up with

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right) \tag{3.91}$$

$$\text{subject to} \quad 0 \leq \lambda_i \leq C, \quad i = 1, 2, \ldots, N \tag{3.92}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.93}$$

Remarks

- The only difference with the previously considered case of linearly separable classes is in the first of the two constraints, where the Langrange multipliers need to be bounded above by C. The linearly separable case corresponds to $C \to \infty$. The slack variables, ξ_i, and their associated Langrange multipliers, μ_i, do not enter into the problem explicitly. Their presence is indirectly reflected through C.
- In all our discussions, so far, we have been involved with the two-class classification task. In an M-class problem, a straightforward extension can be to look at it as a set of M two-class problems. For each one of the classes, we seek to design an optimal discriminant function, $g_i(\boldsymbol{x})$, $i = 1, 2, \ldots, M$, so that $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$, $\forall j \neq i$, if $\boldsymbol{x} \in \omega_i$. Adopting the SVM methodology, we can design the discriminant functions so that $g_i(\boldsymbol{x}) = 0$ to be the optimal hyperplane separating class ω_i from all the others, assuming of course that this is possible. Thus, the resulting linear function will give $g_i(\boldsymbol{x}) > 0$ for

$\boldsymbol{x} \in \omega_i$ and $g_i(\boldsymbol{x}) < 0$ otherwise. Classification is then achieved according to the following rule:

$$\text{assign} \quad \boldsymbol{x} \text{ in } \omega_i \quad \text{if } i = arg \max_k \{g_k(\boldsymbol{x})\}$$

This technique, however, may lead to indeterminate regions, where more than one $g_i(\boldsymbol{x})$ is positive (Problem 3.15).
Another approach is to extent the two class SVM mathematical formulation to the M-class problem, see, for example, [Vapn 98].

Example 3.4. Consider the two-class classification task that consists of the following points:

$$\omega_1: [1, 1]^T, [1, -1]^T$$
$$\omega_2: [-1, 1]^T, [-1, -1]$$

Using the SVM approach, we will demonstrate that the optimal separating hyperplane (line) is $x_1 = 0$ and that this is obtained via different sets of Langrange multipliers.

The points lie on the corners of a square, as shown in Figure 3.10. The simple geometry of the problem allows for a straightforward computation of the SVM linear classifier. Indeed, a careful observation of Figure 3.10 suggests that the optimal line

$$g(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + w_0 = 0$$

is obtained for $w_2 = w_0 = 0$ and $w_1 = 1$, i.e.,

$$g(\boldsymbol{x}) = x_1 = 0$$

Hence for this case, all four points become support vectors and the margin of the separating line from both classes is equal to 1. For any other direction, e.g., $g_1(\boldsymbol{x}) = 0$, the margin is smaller. It must be pointed out that, the same solution is obtained if one solves the associated KKT conditions (Problem 3.16.)

Let us now consider the mathematical formulation of our problem. The linear inequality constraints are

$$w_1 + w_2 + w_0 - 1 \geq 0$$
$$w_1 - w_2 + w_0 - 1 \geq 0$$
$$w_1 - w_2 - w_0 - 1 \geq 0$$
$$w_1 + w_2 - w_0 - 1 \geq 0$$

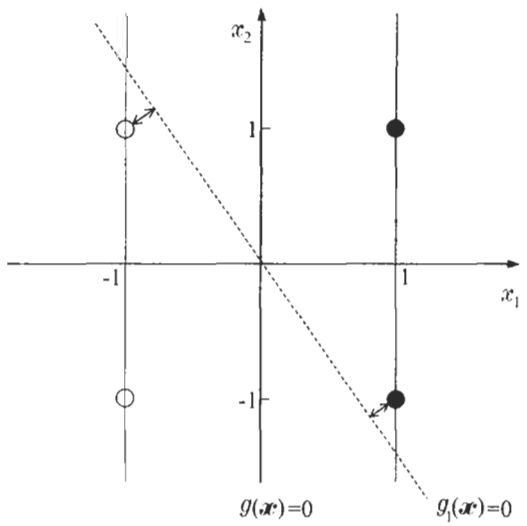


FIGURE 3.10: In this example all four points are support vectors. The margin associated with $g_1(\boldsymbol{x}) = 0$ is smaller compared to the margin defined by the optimal $g(\boldsymbol{x}) = 0$.

and the associated Langrangian function becomes

$$\begin{aligned}\mathcal{L}(w_2, w_1, w_0, \boldsymbol{\lambda}) = \frac{w_1^2 + w_2^2}{2} &- \lambda_1(w_1 + w_2 + w_0 - 1) \\ &- \lambda_2(w_1 - w_2 + w_0 - 1) \\ &- \lambda_3(w_1 - w_2 - w_0 - 1) \\ &- \lambda_4(w_1 + w_2 - w_0 - 1)\end{aligned}$$

The KKT conditions are given by

$$\frac{\partial \mathcal{L}}{\partial w_1} = 0 \;\Rightarrow\; w_1 = \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 \tag{3.94}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = 0 \;\Rightarrow\; w_2 = \lambda_1 + \lambda_4 - \lambda_2 - \lambda_3 \tag{3.95}$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = 0 \;\Rightarrow\; \lambda_1 + \lambda_2 - \lambda_3 - \lambda_4 = 0 \tag{3.96}$$

$$\lambda_1(w_1 + w_2 + w_0 - 1) = 0 \tag{3.97}$$

$$\lambda_2(w_1 - w_2 + w_0 - 1) = 0 \tag{3.98}$$

$$\lambda_3(w_1 - w_2 - w_0 - 1) = 0 \tag{3.99}$$

$$\lambda_4(w_1 + w_2 - w_0 - 1) = 0 \tag{3.100}$$

$$\lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0 \tag{3.101}$$

Since we know that the solution for $\boldsymbol{w}, w_0$ is unique, we can substitute the solution $w_1 = 1, w_2 = w_0 = 0$ into the above equations. Then we are left with a linear system of three equations with four unknowns, i.e.,

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \tag{3.102}$$

$$\lambda_1 + \lambda_4 - \lambda_2 - \lambda_3 = 0 \tag{3.103}$$

$$\lambda_1 + \lambda_2 - \lambda_3 - \lambda_4 = 0 \tag{3.104}$$

which has, obviously, more than one solution. However, all of them lead to the unique optimal separating line.

Problems

3.1 Explain why the perceptron cost function is a *continuous* piecewise linear function.

3.2 Show that if $\rho_k = \rho$ in the perceptron algorithm, the algorithm converges after $k_0 = \frac{\|\boldsymbol{w}(0) - \alpha \boldsymbol{w}^*\|}{\beta^2 \rho(2-\rho)}$ steps, where $\alpha = \frac{\beta^2}{|\gamma|}$ and $\rho < 2$.

3.3 Show that the reward and punishment form of the perceptron algorithm converges in a finite number of iteration steps.

3.4 Consider a case in which class ω_1 consists of the two feature vectors $[0, 0]^T$ and $[0, 1]^T$ and class ω_2 of $[1, 0]^T$ and $[1, 1]^T$. Use the perceptron algorithm in its reward and punishment form, with $\rho = 1$ and $\boldsymbol{w}(0) = [0, 0]^T$, to design the line separating the two classes.

3.5 Consider the two-class task of Problem 2.12 of the previous chapter with

$$\boldsymbol{\mu}_1^T = [1, 1], \quad \boldsymbol{\mu}_2^T = [0,0], \quad \sigma_1^2 = \sigma_2^2 = 0.2$$

Produce 50 vectors from each class. To guarantee linear separability of the classes, disregard vectors with $x_1 + x_2 < 1$ for the $[1, 1]$ class and vectors with $x_1 + x_2 > 1$ for the $[0, 0]$ class. In the sequel use these vectors to design a linear classifier using the perceptron algorithm of (3.21). After convergence, draw the corresponding decision line.

3.6 Consider once more the classification task of Problem 2.12. Produce 100 samples for each of the classes. Use these data to design a linear classifier via the LMS algorithm. Once all samples have been presented to the algorithm, draw the corresponding hyperplane to which the algorithm has converged. Use $\rho_k = \rho = 0.01$.

3.7 Show, using Kesler's construction, that the tth iteration step of the reward and punishement form of the perceptron algorithm (3.21), for an $\boldsymbol{x}_{(t)} \in \omega_i$, becomes

$$\begin{aligned}
&\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{w}_i^T(t)\boldsymbol{x}_{(t)} \leq \boldsymbol{w}_j^T(t)\boldsymbol{x}_{(t)},\ j \neq i \\
&\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) - \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{w}_i^T(t)\boldsymbol{x}_{(t)} \leq \boldsymbol{w}_j^T(t)\boldsymbol{x}_{(t)},\ j \neq i \\
&\boldsymbol{w}_k(t+1) = \boldsymbol{w}_k(t), \quad \forall k \neq j \text{ and } k \neq i
\end{aligned}$$

3.8 Show that the sum of error squares optimal weight vector tends asymptotically to the MSE solution.

3.9 Repeat Problem 3.6 and design the classifier using the sum of error squares criterion.

3.10 Show that the design of an M class linear, sum of error squares optimal, classifier reduces to M equivalent ones, with scalar desired responses.

3.11 Show that, if x, y are jointly Gaussian, the regression of y on x is given by

$$E[y|x] = \frac{\alpha\sigma_y x}{\sigma_x} + \mu_y - \frac{\alpha\sigma_y \mu_x}{\sigma_x}, \quad \text{where } \Sigma = \begin{bmatrix} \sigma_x^2 & \alpha\sigma_x\sigma_y \\ \alpha\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} \tag{3.105}$$

3.12 Let an M class classifier be given in the form of parameterized functions $g(\boldsymbol{x}; \boldsymbol{w}_k)$. The goal is to estimate the parameters $\boldsymbol{w}_k$ so that the outputs of the classifier give desired response values, depending on the class of $\boldsymbol{x}$. Assume that as $\boldsymbol{x}$ varies randomly in each class, the classifier outputs vary around the corresponding desired response values, according to a Gaussian distribution of known variance, assumed to be the same for all outputs. Show that in this case the sum of error squares criterion and the ML estimation result in identical estimates.

Hint: Take N training data samples of known class labels. For each of them form $y_i = g(\boldsymbol{x}_i; \boldsymbol{w}_k) - d_k^i$, where d_k^i is the desired response for the kth class of the ith sample. The y_i's are normally distributed with zero mean and variance σ^2. Form the likelihood function using the y_i's.

3.13 In a two-class problem the Bayes optimal decision surface is given by $g(\boldsymbol{x}) = P(\omega_1|\boldsymbol{x}) - P(\omega_2|\boldsymbol{x}) = 0$. Show that if we train a decision surface $f(\boldsymbol{x}; \boldsymbol{w})$ in the MSE so as to give $+1(-1)$ for the two classes, respectively, this is equivalent to approximating $g(\cdot)$ in terms of $f(\cdot; \boldsymbol{w})$, in the MSE optimal sense.

3.14 Consider a two-class classification task with jointly Gaussian distributed feature vectors and with the same variance Σ in both classes. Design the linear MSE classifier and show that in this case the Bayesian classifier (Problem 2.11) and the resulting MSE one differ only in the threshold value. For simplicity, consider equiprobable classes.

Hint: To compute the MSE hyperplane $\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0$, increase the dimension of $\boldsymbol{x}$ by one and show that the solution is provided by

$$\begin{bmatrix} R & E[\boldsymbol{x}] \\ E[\boldsymbol{x}]^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ w_0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ 0 \end{bmatrix}$$

Then relate R with Σ and show that the MSE classifier takes the form

$$(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} \left(\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) \right) \geq 0.$$

3.15 In an M class classification task the classes can be linearly separated. Design M hyperplanes, so that hyperplane $g_i(\boldsymbol{x}) = 0$ leaves class ω_i on its positive side and the rest of the classes on its negative side. Demonstrate via an example, e.g., $M = 3$, that the partition of the space using this rule creates indeterminate regions (where no training data exist) for which more than one $g_i(\boldsymbol{x})$ is positive or all of them are negative.

3.16 Obtain the optimal line for the task of Example 3.4, via the KKT conditions. Restrict the search for the optimum among the lines crossing the origin.

References

[Baza 79] Bazaraa M.S., Shetty C.M. *Nonlinear Programming*, John Wiley & Sons, 1979.

[Bish 95] Bishop C. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[Cid 99] Cid-Sueiro J., Arribas J.I., Urban-Munoz S., Figueras-Vidal A.R. "Cost functions to estimate a-posteriori probabilities in multiclass problems," *IEEE Transactions on Neural Networks*, Vol. 10(3), pp. 645–656, 1999.

[Flet 87] Fletcher R. *Practical Methods of Optimization*, 2nd edition, John Wiley & Sons, 1987.

[Frea 92] Frean M., "A thermal perceptron learning rule," *Neural Computation*, Vol. 4, pp. 946–957, 1992.

[Fuku 90] Fukunaga K. *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press, 1990.

[Gal 90] Gallant S.I. "Perceptron based learning algorithms," *IEEE Transactions on Neural Networks*, Vol. 1(2), pp. 179–191, 1990.

[Gema 92] Geman S., Bienenstock E., Doursat R. "Neural networks and the bias/variance dilemma," *Neural Computation*, Vol. 4, pp. 1–58, 1992.

[Hayk 96] Haykin S. *Adaptive Filter Theory*, 3rd ed., Prentice Hall, 1996.

[Ho 65] Ho Y.H., Kashyap R.L. "An algorithm for linear inequalities and its applications," *IEEE Transactions on Electronic Computers*, Vol. 14(5), pp. 683–688, 1965.

[Hryc 92] Hrycej T., *Modular learning in neural networks*, New York: Wiley, 1992.

[Kalou 93] Kalouptsidis N., Theodoridis S. *Adaptive System Identification and Signal Processing Algorithms*, Prentice Hall, 1993.

[Min 88] Minsky M.L., Papert S.A. *Perceptrons*, expanded edition, MIT Press, Cambridge, MA, 1988.

[Muse 97] Muselli M. "On convergence properties of pocket algorithm," *IEEE Transactions on Neural Networks*, Vol. 8(3), pp. 623–629, 1997.

[Papo 91] Papoulis A. *Probability, Random Variables and Stochastic Processes*, 3rd ed., McGraw-Hill, 1991.

[Pear 90] Pearlmutter B., Hampshire J. "Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function," *Proceedings Connectionists Models Summer School*, San Diego, CA:Morgan Kauffman, 1990.

[Poul 95] Poulard H., "Barycentric correction procedure: A fast method of learning threshold units," *Proc. WCNN '95*, Vol. 1, Washington, D.C., pp. 710–713, July, 1995.

[Rich 91] Richard M.D., Lippmann R.P. "Neural network classifiers estimate Bayesian a posteriori probabilities," *Neural Computation*, Vol. 3, pp. 461–483, 1991.

[Robb 51] Robbins H., Monro S. "A stochastic approximation method," *Annals of Mathematical Statistics*, Vol. 22, pp. 400–407, 1951.

[Rose 58] Rosenblatt F. "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, Vol. 65, pp. 386–408, 1958.

[Tou 74] Tou J., Gonzalez R.C. *Pattern Recognition Principles*, Addison-Wesley, 1974.

[Vapn 98] Vapnik V.N. *Statistical Learning Theory*, John Wiley & Sons, 1998.

[Widr 60] Widrow B., Hoff M.E., Jr. "Adaptive switching circuits," *IRE WESCON Convention Record*, pp. 96–104, 1960.

[Widr 90] Widrow B., Lehr M.A. "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, Vol. 78(9), pp. 1415–1442, 1990.

CHAPTER 4

NONLINEAR CLASSIFIERS

4.1 INTRODUCTION

In the previous chapter we dealt with the design of linear classifiers described by linear discriminant functions (hyperplanes) $g(\boldsymbol{x})$. In the simple two-class case we saw that the perceptron algorithm computes the weights of the linear function $g(\boldsymbol{x})$, provided that the classes are linearly separable. For non–linearly separable classes linear classifiers were optimally designed, for example, by minimizing the squared error. In this chapter we will deal with problems that are not linearly separable and for which the design of a linear classifier, even in an optimal way, does not lead to satisfactory performance. The design of nonlinear classifiers emerges now as an unescapable necessity.

4.2 THE XOR PROBLEM

To seek nonlinearly separable problems one does not need to go into complicated situations. The well-known *Exclusive OR (XOR)* Boolean function is a typical example of such a problem. Boolean functions can be interpreted as classification tasks. Indeed, depending on the values of the input binary data $\boldsymbol{x} = [x_1, x_2, \ldots, x_l]^T$, the output is either 0 or 1, and $\boldsymbol{x}$ is classified into one of the two classes $A(1)$ or $B(0)$. The corresponding truth table for the XOR operation is shown in Table 4.1.

Figure 4.1 shows the position of the classes in space. It is apparent from this figure that no single straight line exists that separates the two classes. In contrast, the other two Boolean functions, AND and OR, are linearly separable. The corresponding truth tables for the AND and OR operations are given in Table 4.2 and the respective class positions in the two-dimensional space are shown in Figure 4.2a and 4.2b. Figure 4.3 shows a perceptron, introduced in the previous chapter, with synaptic weights computed so as to realize an OR gate (verify).

Our major concern now is first to tackle the XOR problem and then to extend the procedure to more general cases of nonlinearly separable classes. Our kickoff point will be geometry.

Table 4.1: Truth table for the XOR problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

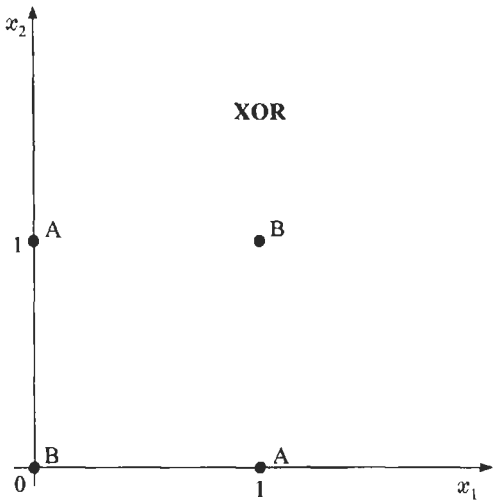


FIGURE 4.1: Classes A and B for the XOR problem.

4.3 THE TWO-LAYER PERCEPTRON

To separate the two classes A and B in Figure 4.1, a first thought that comes into mind is to draw two, instead of one, straight lines.

Figure 4.4 shows two such possible lines, $g_1(\boldsymbol{x}) = g_2(\boldsymbol{x}) = 0$, as well as the regions in space for which $g_1(\boldsymbol{x}) \gtrless 0$, $g_2(\boldsymbol{x}) \gtrless 0$. The classes can now be separated. Class A is to the right $(+)$ of $g_1(\boldsymbol{x})$ and to the left $(-)$ of $g_2(\boldsymbol{x})$. The region corresponding to class B lies either to the left or to the right of both lines. What we have really done is to attack the problem in *two* successive phases. During the first phase we calculate the position of a feature vector $\boldsymbol{x}$ with respect to *each* of the two decision lines. In the second phase we combine the results of the previous phase and we find the position of $\boldsymbol{x}$ with respect to *both* lines, that is, outside or inside the shaded area. We will now view this from a slightly different perspective, which will subsequently lead us easily to generalizations.

Table 4.2: Truth table for AND and OR problems

x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A

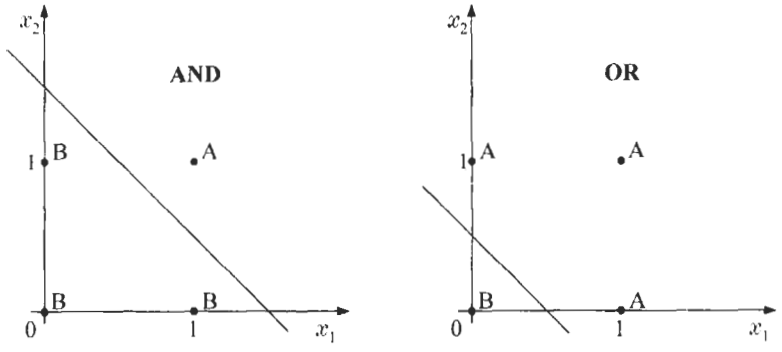


FIGURE 4.2: Classes A and B for the AND and OR problems.

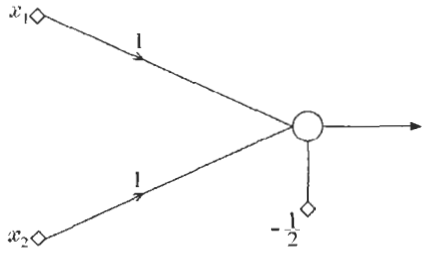


FIGURE 4.3: A perceptron realizing an OR gate.

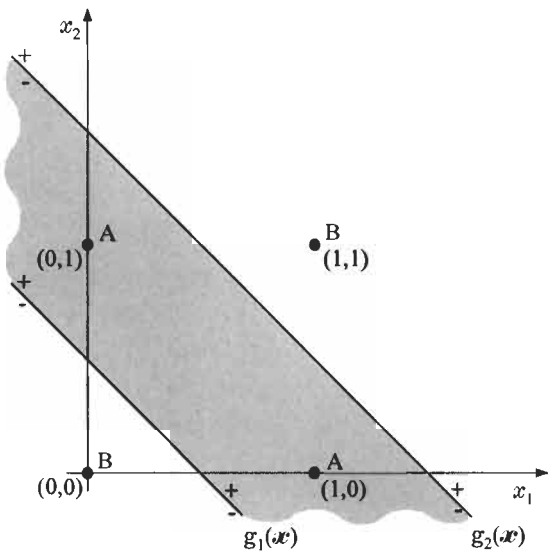


FIGURE 4.4: Decision lines realized by a two-layer perceptron for the XOR problem.

Table 4.3: Truth table for the two computation phases of the XOR problem

1st Phase				
x_1	x_2	y_1	y_2	2nd Phase
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

Realization of the two decision lines (hyperplanes), $g_1(\cdot)$ and $g_2(\cdot)$, during the first phase of computations is achieved with the adoption of two perceptrons with inputs x_1, x_2 and appropriate synaptic weights. The corresponding outputs are $y_i = f(g_i(\boldsymbol{x})), i = 1, 2$, where the activation function $f(\cdot)$ is the step function with levels 0 and 1. Table 4.3 summarizes the y_i values for all possible combinations of the inputs. These are nothing else than the relative positions of the input vector $\boldsymbol{x}$ with respect to each of the two lines. From another point of view, the computations during the first phase *perform a mapping* of the input vector $\boldsymbol{x}$ to a new one $\boldsymbol{y} = [y_1, y_2]^T$. The decision during the second phase is now based on

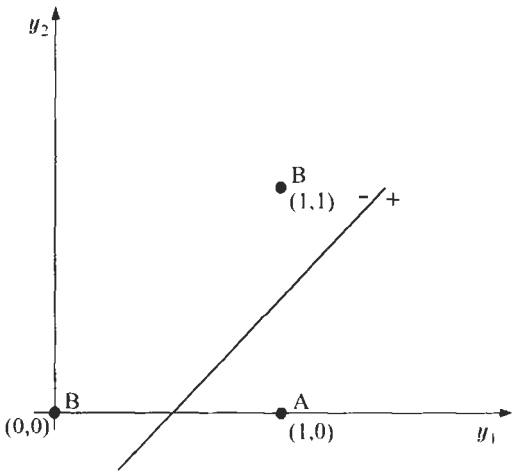


FIGURE 4.5: Decision line formed by the neuron of the second layer for the XOR problem.

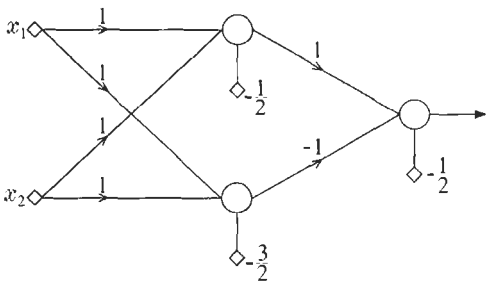


FIGURE 4.6: A two-layer perceptron solving the XOR problem.

the transformed data; that is, our goal is now to separate $[y_1, y_2] = [0, 0]$ and $[y_1, y_2] = [1, 1]$, which correspond to class B vectors, from the $[y_1, y_2] = [1, 0]$, which corresponds to class A vectors. As is apparent from Figure 4.5 this is easily achieved by drawing a third line $g(\boldsymbol{y})$, which can be realized via a third neuron. *In other words, the mapping of the first phase transforms the nonlinearly separable problem to a linearly separable one.* We will return to this important issue later on. Figure 4.6 gives a possible realization of these steps. Each of the three lines is realized via a neuron with appropriate synaptic weights. The resulting *multilayer* architecture can be considered as a generalization of the perceptron, and it is

known as a *two-layer perceptron* or a *two-layer feedforward*[1] *neural network*. The two neurons (nodes) of the first layer perform computations of the first phase and they constitute the so-called *hidden layer*. The single neuron of the second layer performs the computations of the final phase and constitutes the *output layer*. In Figure 4.6 the *input layer* corresponds to the (non-processing) nodes where input data are applied. Thus, the number of input layer nodes equals the dimension of the input space. Note that at the input layer nodes no processing takes place. The lines that are realized by the two-layer perceptron of the figure are

$$g_1(\boldsymbol{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

$$g_2(\boldsymbol{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

$$g(\boldsymbol{y}) = y_1 - y_2 - \frac{1}{2} = 0$$

The multilayer perceptron architecture of Figure 4.6 can be generalized to l-dimensional input vectors and to more than two (one) neurons in the hidden (output) layer. We will now turn our attention to the investigation of the class discriminatory capabilities of such networks for more complicated nonlinear classification tasks.

4.3.1 Classification Capabilities of the Two-Layer Perceptron

A careful look at the two-layer perceptron of Figure 4.6 reveals that the action of the neurons of the hidden layer is actually a mapping of the input space $\boldsymbol{x}$ onto the vertices of a square of unit side length in the two-dimensional space (Figure 4.5).

For the more general case, we will consider input vectors in the l-dimensional space, that is, $\boldsymbol{x} \in \mathcal{R}^l$, and p neurons in the hidden layer (Figure 4.7). For the time being we will keep one output neuron, although this can also be easily generalized to many. Again employing the step activation function, the mapping of the input space, performed by the hidden layer, is now onto the vertices of the hypercube of unit side length in the p-dimensional space, denoted by H_p. This is defined as

$$H_p = \{[y_1, \ldots, y_p]^T \in \mathcal{R}^p, y_i \in [0, 1], 1 \leq i \leq p\}$$

The vertices of the hypercube are all the points $[y_1, \ldots, y_p]^T$ of H_p with $y_i \in \{0, 1\}$, $1 \leq i \leq p$.

[1] To distinguish it from other related structures where feedback paths from the output back to the input exist.

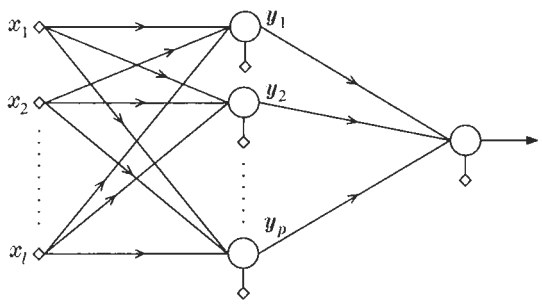


FIGURE 4.7: A two-layer perceptron.

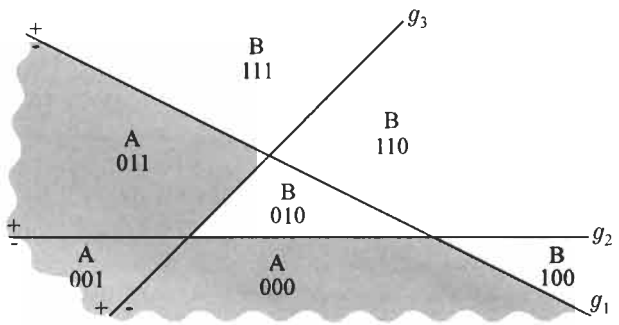


FIGURE 4.8: Polyhedra formed by the neurons of the first hidden layer of a multilayer perceptron.

The mapping of the input space onto the vertices of the hypercube is achieved via the creation of p hyperplanes. Each of the hyperplanes is created by a neuron in the hidden layer, and the output of each neuron is 0 or 1, depending on the relevant position of the input vector with respect to the corresponding hyperplane. Figure 4.8 is an example of three intersecting hyperplanes (three neurons) in the two-dimensional space. Each region defined by the intersections of these hyperplanes corresponds to a vertex of the unit three-dimensional hypercube, depending on its position with respect to each of these hyperplanes. The ith dimension of the vertex shows the position of the region with respect to the g_i hyperplane. For example, the 001 vertex corresponds to the region that is in the (–) side of g_1, in the (–) side of g_2, and in the (+) side of g_3. Thus, the conclusion we reach is that *the first layer of neurons divides the input l-dimensional space into polyhedra,*[2]

[2]A polyhedron or polyhedral set is the finite intersection of closed half-spaces of $\mathcal{R}^l$, which are defined by a number of hyperplanes.

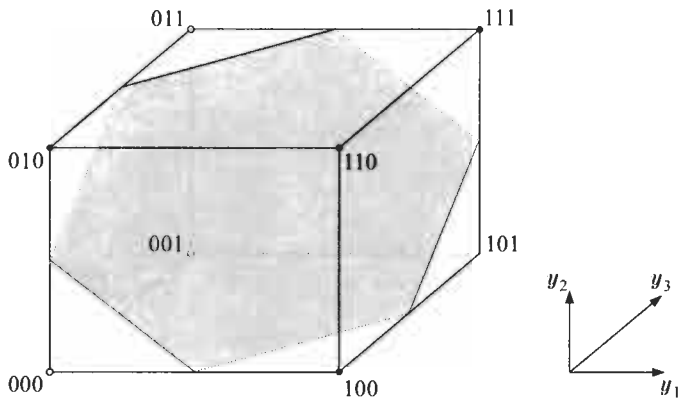


FIGURE 4.9: The neurons of the first hidden layer map an input vector onto one of the vertices of a unit (hyper)cube. The output neuron realizes a (hyper)plane to separate vertices according to their class label.

which are formed by hyperplane intersections. All vectors located within one of these polyhedral regions are mapped onto a specific vertex of the unit H_p hypercube. The output neuron subsequently realizes another hyperplane, which separates the hypercube into two parts, having some of its vertices on one and some on the other side. This neuron provides the multilayer perceptron with the potential to classify vectors into *classes consisting of unions of the polyhedral regions.* Let us consider, for example, that class A consists of the union of the regions mapped onto vertices 000, 001, 011 and class B consists of the rest (Figure 4.8). Figure 4.9 shows the H_3 unit (hyper)cube and a (hyper)plane that separates the space $\mathcal{R}^3$ into two regions with the (class A) vertices 000, 001, 011 on one side and (class B) vertices 010, 100, 110, 111 on the other. This is the $-y_1 - y_2 + y_3 + 0.5 = 0$ plane, which is realized by the output neuron. With such a configuration all vectors from class A result in an output of 1(+) and all vectors from class B in 0(–). On the other hand, if class A consists of the union 000 $\cup$ 111 $\cup$ 110 and class B of the rest, it is not possible to construct a single plane that separates class A from class B vertices. Thus, we can conclude that *a two-layer perceptron can separate classes each consisting of unions of polyhedral regions but not any union of such regions.* It all depends on the relative positions of the vertices of H_p, where the classes are mapped, and whether these are linearly separable or not. Before we proceed further to see ways to overcome this shortcoming, it should be pointed out that vertex 101 of the cube does not correspond to any of the polyhedral regions. Such vertices are said to correspond to *virtual polyhedra*, and they do not influence the classification task.

4.4 THREE-LAYER PERCEPTRONS

The inability of the two-layer perceptrons to separate classes resulting from *any* union of polyhedral regions springs from the fact that the output neuron can realize only a single hyperplane. This is the same situation the basic perceptron was confronted with when dealing with the XOR problem. The difficulty was overcome by constructing two lines instead of one. A similar escape path will be adopted here.

Figure 4.10 shows a three-layer perceptron architecture with two layers of hidden neurons and one output layer. We will show, constructively, that such an architecture can *separate classes resulting from any union of polyhedral regions*. Indeed, let us assume that all regions of interest are formed by intersections of p l-dimensional half-spaces defined by the p hyperplanes. These are realized by the p neurons of the first hidden layer, which also perform the mapping of the input space onto the vertices of the H_p hypercube of unit side length. In the sequel let us assume that class A consists of the union of J of the resulting polyhedra and class B of the rest. We then use J neurons in the second hidden layer. Each of these neurons realizes a hyperplane in the p-dimensional space. The synaptic weights for each of the second-layer neurons are chosen so that the realized hyperplane leaves only one of the H_p vertices on one side and *all* the rest on the other. For each neuron a different vertex is isolated, that is, one of the J A class vertices. In other words, each time an input vector from class A enters the network, one of the J neurons of the second layer results in a 1 and the remaining $J-1$ give 0. In contrast, for class B vectors all neurons in the second layer output a 0. Classification is now a straightforward task. Choose the output layer neuron to realize an OR gate. Its output will be 1 for class A and 0 for class B vectors. The proof is now complete.

The number of neurons in the second hidden layer can be reduced by exploiting the geometry that results from each specific problem—for example, whenever two of the J vertices are located in a way that makes them separable from the rest, using

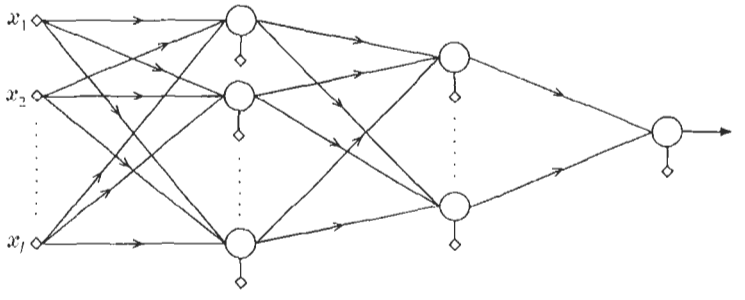


FIGURE 4.10: Architecture of a multilayer perceptron with two hidden layers of neurons and a single output neuron.

a single hyperplane. Finally, the multilayer structure can be generalized to more than two classes. To this end, the output layer neurons are increased in number, realizing one OR gate for each class. Thus, one of them results in 1 every time a vector from the respective class enters the network, and all the others give 0. The number of second-layer neurons is also affected (why?).

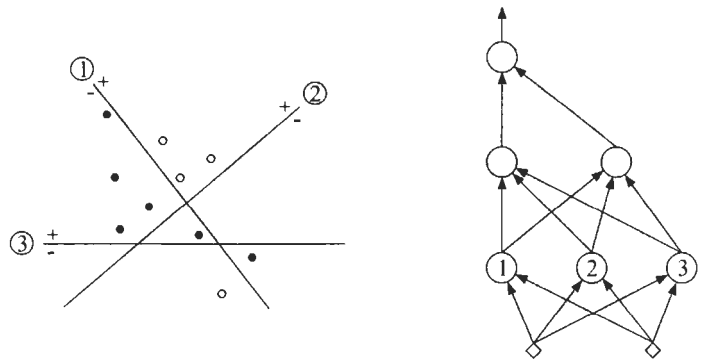
In summary, we can say that *the neurons of the first layer form the hyperplanes, those of the second layer form the regions, and finally the neurons of the output layer form the classes.*

So far, we have focused on the potential capabilities of a three-layer perceptron to separate any union of polyhedral regions. To assume that in practice we know the regions where the data are located and we can compute the respective hyperplane equations analytically is no doubt wishful thinking, for this is as yet an unrealizable goal. All we know in practice are points within these regions. As was the case with the perceptron, one has to resort to learning algorithms that learn the synaptic weights from the available training data vectors. There are two major directions in which we will focus on our attention. In one of them the network is constructed in a way that classifies correctly *all* the available training data, by building it as a succession of linear classifiers. The other direction relieves itself of the correct classification constraint and computes the synaptic weights so as to minimize a preselected cost function.

4.5 ALGORITHMS BASED ON EXACT CLASSIFICATION OF THE TRAINING SET

The starting point of these techniques is a small architecture (usually unable to solve the problem at hand), which is successively augmented until the correct classification of all N feature vectors of the training set X is achieved. Different algorithms follow different ways to augment their architectures. Thus, some algorithms expand their architectures in terms of the number of layers [Meza 89, Frea 90], whereas others use one or two hidden layers and expand them in terms of the number of their nodes (neurons) [Kout 94, Bose 96]. Moreover, some of these algorithms [Frea 90] allow connections between nodes of nonsuccessive layers. Others allow connections between nodes of the same layer [Refe 91]. A general principle adopted by most of these techniques is the decomposition of the problem into smaller problems that are easier to handle. For each smaller problem, a single node is employed. Its parameters are determined either iteratively using appropriate learning algorithms, such as the pocket algorithm or the LMS algorithm (Chapter 3), or directly via analytical computations. From the way these algorithms build the network, they are sometimes referred to as *constructive techniques.*

The *tiling algorithm* [Meza 89] constructs architectures with many (usually more than three) layers. We describe the algorithm for the two-class (A and B) case.



(a) (b)

FIGURE 4.11: Decision lines and the corresponding architecture resulting from the tiling algorithm. The open (filled) circles correspond to class A (B).

The algorithm starts with a single node, $n(X)$, in the first layer, which is called the *master unit* of this layer.

This node is trained using the pocket algorithm (Chapter 3) and, after the completion of the training, it divides the training data set X into two subsets X^+ and X^- (line 1 in Figure 4.11). If X^+ (X^-) contains feature vectors from both classes, we introduce an additional node, $n(X^+)$ ($n(X^-)$), which is called *ancillary unit*. This node is trained using only the feature vectors in X^+ (X^-) (line 2). If one of the X^{++}, X^{+-} (X^{-+}, X^{--}) produced by neuron $n(X^+)$ ($n(X^-)$) contains vectors from both classes, additional ancillary nodes are added. This procedure stops after a finite number of steps, since the number of vectors a newly added (ancillary) unit has to discriminate decreases at each step. Thus, the first layer consists of a single master unit and, in general, more than one ancillary units. It is easy to show that in this way we succeed so that no two vectors from *different* classes give the same first-layer outputs.

Let $X_1 = \{\boldsymbol{y} : \boldsymbol{y} = f_1(\boldsymbol{x}), \boldsymbol{x} \in X\}$, where f_1 is the mapping implemented by the first layer. Applying the procedure just described to the set X_1 of the transformed $\boldsymbol{y}$ samples, we construct the second layer of the architecture and so on. In [Meza 89] it is shown that proper choice of the weights between two adjacent layers ensures that each newly added master unit classifies correctly all the vectors that are correctly classified by the master unit of the previous layer, plus at least one more vector. Thus, the tiling algorithm produces an architecture that classifies correctly all patterns of X in a finite number of steps.

An interesting observation is that all but the first layer treat binary vectors. This reminds us of the unit hypercube of the previous section. Mobilizing the

same arguments as before, we can show that this algorithm may lead to correct classification architectures having three layers of nodes at the most.

Another family of constructive algorithms builds on the idea of the nearest neighbor classification rule, discussed in Chapter 2. The neurons of the first layer implement the hyperplanes bisecting the line segments joining the training feature vectors [Murp 90]. The second layer forms the regions, using an appropriate number of neurons that implement AND gates, and the classes are formed via the neurons of the last layer, which implement OR gates. The major drawback of this technique is the large number of neurons involved. Techniques that reduce this number have also been proposed ([Kout 94, Bose 96]).

4.6 THE BACKPROPAGATION ALGORITHM

The other direction we will follow to design a multilayer perceptron is to fix the architecture and compute its synaptic parameters so as to minimize an appropriate cost function of its output. This is by far the most popular approach, which not only overcomes the drawback of the resulting large networks of the previous section but also makes these networks powerful tools for a number of other applications, beyond pattern recognition. However, such an approach is soon confronted with a serious difficulty. This is the discontinuity of the step (activation) function, prohibiting differentiation with respect to the unknown parameters (synaptic weights). Differentiation enters into the scene as a result of the cost function minimization procedure. In the sequel we will see how this difficulty can be overcome.

The multilayer perceptron architectures we have considered so far have been developed around the McCulloch–Pitts neuron, employing as the activation function the step function

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

A popular family of continuous differentiable functions, which approximate the step function, is the family of *sigmoid functions*. A typical representative is the *logistic function*

$$f(x) = \frac{1}{1 + \exp(-ax)} \tag{4.1}$$

where a is a slope parameter.

Figure 4.12 shows the sigmoid function for different values of a, along with the step function. Sometimes a variation of the logistic function is employed that is antisymmetric with respect to the origin, that is, $f(-x) = -f(x)$. It is

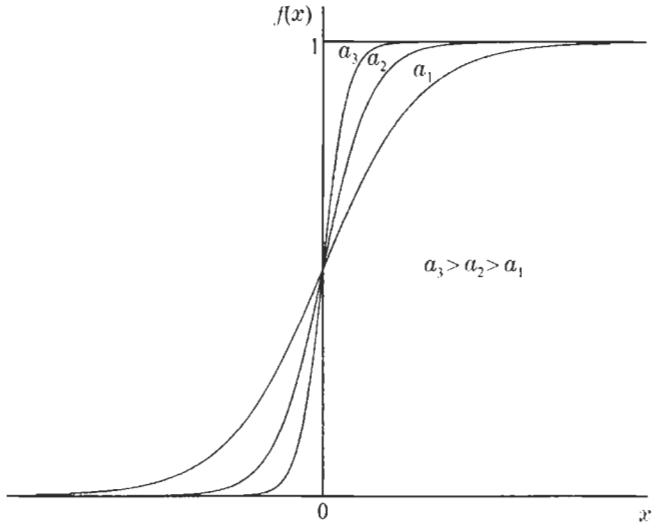


FIGURE 4.12: The logistic function.

defined as

$$f(x) = \frac{2}{1 + \exp(-ax)} - 1 \tag{4.2}$$

It varies between 1 and -1, and it belongs to the family of hyperbolic tangent functions,

$$f(x) = c\frac{1 - \exp(-ax)}{1 + \exp(-ax)} = c\tanh\left(\frac{ax}{2}\right) \tag{4.3}$$

All these functions are also known as *squashing functions* since their output is limited in a finite range of values. In the sequel, we will adopt multilayer neural architectures like the one in Figure 4.10, and we will assume that the activation functions are of the form given in (4.1)–(4.3). Our goal is to derive an iterative training algorithm that computes the synaptic weights of the network so that an appropriately chosen cost function is minimized. Before going into the derivation of such a scheme, an important point must be clarified. From the moment we move away from the step function, all we have said before about mapping the input vectors onto the vertices of a unit hypercube is no longer valid. It is now the cost function that takes on the burden for correct classification.

For the sake of generalization, let us assume that the network consists of a fixed number of L layers of neurons, with k_0 nodes in the input layer and k_r neurons

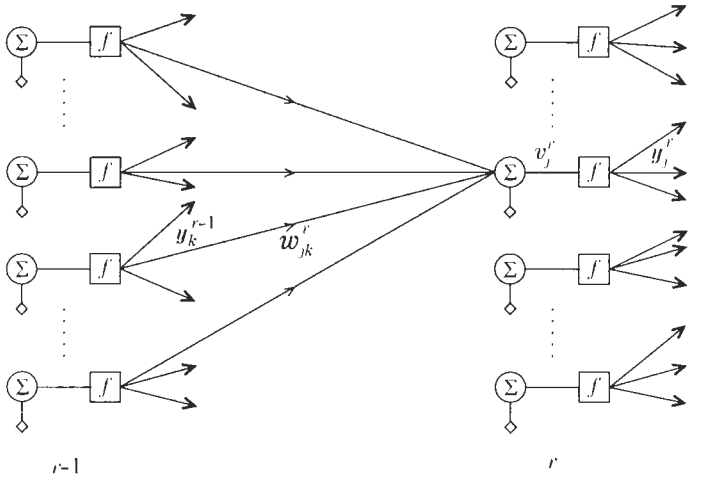


FIGURE 4.13: Definition of variables involved in the backpropagation algorithm.

in the rth layer, for $r = 1, 2, \ldots, L$. Obviously, k_0 equals l. All the neurons employ the same sigmoid activation function. As was the case in Section 3.3, we assume that there are N training pairs available $(\boldsymbol{y}(i), \boldsymbol{x}(i)), i = 1, 2, \ldots, N$.[3] Because we have now assumed k_L output neurons, the output is no longer a scalar but a k_L-dimensional vector, $\boldsymbol{y}(i) = [y_1(i), \ldots, y_{k_L}(i)]^T$. The input (feature) vectors are k_0-dimensional vectors, $\boldsymbol{x}(i) = [x_1(i), \ldots, x_{k_0}(i)]^T$. During training, when vector $\boldsymbol{x}(i)$ is applied to the input, the output of the network will be $\hat{\boldsymbol{y}}(i)$, which is different from the desired value, $\boldsymbol{y}(i)$. *The synaptic weights are computed such that an appropriate (for each problem) cost function J, which is dependent on the values $\boldsymbol{y}(i)$ and $\hat{\boldsymbol{y}}(i)$, $i = 1, 2, \ldots, N$, is minimized.* It is obvious that J depends, through $\hat{\boldsymbol{y}}(i)$, on the weights and that this is a nonlinear dependence, due to the nature of the network itself. Thus, minimization of the cost function can be achieved via iterative techniques. In this section we will adopt the gradient descent scheme (Appendix C), which is the most widely used approach. Let $\boldsymbol{w}_j^r$ be the weight vector (including the threshold) of the jth neuron in the rth layer, which is a vector of dimension $k_{r-1}+1$ and is defined as (Figure 4.13) $\boldsymbol{w}_j^r = [w_{j0}^r, w_{j1}^r, \ldots, w_{jk_{r-1}}^r]^T$. The basic iteration step will be of the form

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r$$

[3] In contrast to other chapters, we use i in parentheses and not as an index. This is because, for the needs of the chapter, the latter notation can become very cumbersome.

with

$$\Delta \boldsymbol{w}_j^r = -\mu \frac{\partial J}{\partial \boldsymbol{w}_j^r} \tag{4.4}$$

where $\boldsymbol{w}_j^r$(old) is the current estimate of the unknown weights and $\Delta \boldsymbol{w}_j^r$ the corresponding correction to obtain the next estimate $\boldsymbol{w}_j^r$(new).

In Figure 4.13 υ_j^r is the weighted summation of the inputs to the jth neuron of the rth layer and y_j^r the corresponding output after the activation function. In the sequel we will focus our attention on cost functions of the form

$$J = \sum_{i=1}^{N} \mathcal{E}(i) \tag{4.5}$$

where $\mathcal{E}$ is an appropriately defined function depending on $\hat{\boldsymbol{y}}(i)$ and $\boldsymbol{y}(i)$, $i = 1, 2, \ldots, N$. In other words, J is expressed as a sum of the N values that function $\mathcal{E}$ takes for each of the training pairs $(\boldsymbol{y}(i), \boldsymbol{x}(i))$. For example, we can choose $\mathcal{E}(i)$ as the sum of squared errors in the output neurons

$$\mathcal{E}(i) = \frac{1}{2} \sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2} \sum_{m=1}^{k_L} (y_m(i) - \hat{y}_m(i))^2, \quad i = 1, 2, \ldots, N \tag{4.6}$$

For the computation of the correction term in (4.4) the gradient of the cost function J with respect to the weights is required and, consequently, the evaluation of $\partial \mathcal{E}(i)/\partial \boldsymbol{w}_j^r$.

Computation of the Gradients

Let $y_k^{r-1}(i)$ be the output of the kth neuron, $k = 1, 2, \ldots, k_{r-1}$, in the $(r-1)$th layer for the ith training pair and w_{jk}^r the current estimate of the corresponding weight leading to the jth neuron in the rth layer, with $j = 1, 2, \ldots, k_r$ (Figure 4.13). Thus, the argument of the activation function $f(\cdot)$ of the latter neuron will be

$$\upsilon_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) + w_{jo}^r \equiv \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) \tag{4.7}$$

where by definition $y_0^r(i) \equiv +1$, $\forall r, i$; so as to include the thresholds in the weights. For the output layer, we have $r = L$, $y_k^r(i) = \hat{y}_k(i)$, $k = 1, 2, \ldots, k_L$, that is, the outputs of the neural network, and for $r = 1$, $y_k^{r-1}(i) = x_k(i)$, $k = 1, 2, \ldots, k_0$, that is, the network inputs.

As is apparent from (4.7), the dependence of $\mathcal{E}(i)$ on $\boldsymbol{w}_j^r$ passes through $v_j^r(i)$. By the chain rule in differentiation, we have

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \frac{\partial v_j^r(i)}{\partial \boldsymbol{w}_j^r} \tag{4.8}$$

From (4.7) we obtain

$$\frac{\partial}{\partial \boldsymbol{w}_j^r} v_j^r(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w_{j0}^r} v_j^r(i) \\ \vdots \\ \frac{\partial}{\partial w_{jk_{r-1}}^r} v_j^r(i) \end{bmatrix} = \boldsymbol{y}^{r-1}(i) \tag{4.9}$$

where

$$\boldsymbol{y}^{r-1}(i) = \begin{bmatrix} +1 \\ y_1^{r-1}(i) \\ \vdots \\ y_{k_{r-1}}^{r-1}(i) \end{bmatrix} \tag{4.10}$$

Let us define

$$\frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \equiv \delta_j^r(i) \tag{4.11}$$

Then (4.4) becomes

$$\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i) \tag{4.12}$$

Relation (4.12) is general for *any* differentiable cost function of the form (4.5). In the sequel we will compute $\delta_j^r(i)$ for the special case of least squares (4.6). *The procedure is similar for alternative cost function choices.*

Computation of $\delta_j^r(i)$ for the Cost Function in (4.6)

The computations start from $r = L$ and *propagate backward* for $r = L - 1$, $L - 2, \ldots, 1$. This is why the algorithm that will be derived is known as *the backpropagation algorithm.*

(i) $r = L$

$$\delta_j^L(i) = \frac{\partial \mathcal{E}(i)}{\partial v_j^L(i)} \tag{4.13}$$

$$\mathcal{E}(i) \equiv \frac{1}{2}\sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2}\sum_{m=1}^{k_L}(f(v_m^L(i)) - y_m(i))^2 \tag{4.14}$$

Hence

$$\delta_j^L(i) = e_j(i) f'(v_j^L(i)) \tag{4.15}$$

where f' is the derivative of $f(\cdot)$. In the last layer, the dependence of $\mathcal{E}(i)$ on $v_j^L(i)$ is explicit and the computation of the derivative is straightforward. This is not true, however, for the hidden layers, where the computations of the derivatives need more elaboration.

(ii) $r < L$. Due to the successive dependence among the layers, the value of $v_j^{r-1}(i)$ influences all $v_k^r(i)$, $k = 1, 2, .., k_r$, of the next layer. Employing the chain rule in differentiation once more, we obtain

$$\frac{\partial \mathcal{E}(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \frac{\partial \mathcal{E}(i)}{\partial v_k^r(i)} \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} \tag{4.16}$$

and from the respective definition (4.11)

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} \tag{4.17}$$

But

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = \frac{\partial \left[\sum_{m=0}^{k_{r-1}} w_{km}^r y_m^{r-1}(i)\right]}{\partial v_j^{r-1}(i)} \tag{4.18}$$

with

$$y_m^{r-1}(i) = f(v_m^{r-1}(i)) \tag{4.19}$$

Hence,

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = w_{kj}^r f'(v_j^{r-1}(i)) \tag{4.20}$$

From (4.20) and (4.17) the following results:

$$\delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r\right] f'(v_j^{r-1}(i)) \tag{4.21}$$

and for uniformity with (4.15)

$$\delta_j^{r-1}(i) = e_j^{r-1}(i) f'(v_j^{r-1}(i)) \tag{4.22}$$

where

$$e_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r \tag{4.23}$$

Relations (4.15), (4.22), (4.23) constitute the iterations leading to the computation of $\delta_j^r(i), r = 1, 2, \ldots, L, j = 1, 2, \ldots, k_r$. The only quantity that is not yet computed is $f'(\cdot)$. For the function in (4.1) we have

$$f'(x) = \alpha f(x)(1 - f(x))$$

The algorithm has now been derived. The algorithmic scheme was first presented in [Werb 74] in a more general formulation.

The Backpropagation Algorithm

- *Initialization:* Initialize all the weights with small random values from a pseudorandom sequence generator.
- *1. Forward computations:* For each of the training feature vectors $\boldsymbol{x}(i), i = 1, 2, \ldots, N$, compute all the $v_j^r(i)$, $y_j^r(i) = f(v_j^r(i))$, $j = 1, 2, \ldots, k_r$, $r = 1, 2, \ldots, L$, from (4.7). Compute the cost function for the current estimate of weights from (4.5) and (4.14).
- *2. Backward computations:* For each $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, k_L$ compute $\delta_j^L(i)$ from (4.15) and in the sequel compute $\delta_j^{r-1}(i)$ from (4.22) and (4.23) for $r = L, L-1, \ldots, 2$, and $j = 1, 2, \ldots, k_r$
- *Update the weights:* For $r = 1, 2, \ldots, L$ and $j = 1, 2, \ldots, k_r$

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r$$

$$\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i)$$

Remarks

- A number of criteria have been suggested for terminating the iterations. In [Kram 89] it is suggested that we terminate the iterations either when the cost function J becomes smaller than a certain threshold or when its gradient with respect to the weights becomes small. Of course, the latter has a direct effect on the rate of change of the weights between successive iteration steps.
- As with all the algorithms that spring from the gradient descent method, the convergence speed of the backpropagation scheme depends on the value of the learning constant μ. Its value must be sufficiently small to guarantee convergence but not too small, because the convergence speed becomes very slow. The best choice of μ depends very much on the problem and the cost function shape in the weight space. Broad minima yield small gradients; thus large values of μ lead to faster convergence. On the other hand, for steep and narrow minima small values of μ are required to avoid overshooting the minimum. As we will soon see, scenarios with adaptive μ are also possible.
- The cost function minimization for a multilayer perceptron is a nonlinear minimization task. Thus, the existence of local minima in the corresponding cost function surface is an expected reality. Hence, the backpropagation algorithm runs the risk of being trapped in a local minimum. If the local minimum is deep enough, this may still be a good solution. However, in cases in which this is not true, getting stuck in such a minimum is an undesirable situation and the algorithm should be reinitialized from a different set of initial conditions.
- The algorithm described in this section updates the weights once *all* the training (input–desired output) pairs have appeared in the network. This mode of operation is known as the *batch mode*. A variation of this approach is to update the weights for each of the training pairs. This is known as the *pattern* or *on-line mode*. This is analogous to the LMS, where, according to the Robbins–Monro approach, the instantaneous value of the gradient is computed instead of its mean. In the backpropagation case, the sum of $\delta_j^r(i)$ over all i is substituted with *each* of them. The algorithm in its pattern mode of operation then becomes

$$\boldsymbol{w}_j^r(i+1) = \boldsymbol{w}_j^r(i) - \mu\delta_j^r(i)\mathbf{y}^{r-1}(i)$$

  Compared with the pattern mode, the batch mode is an inherent averaging process. This leads to a better estimate of the gradient, thus to more well-behaved convergence. On the other hand, the pattern mode presents a higher degree of randomness during training. This may help the algorithm to avoid being trapped in a local minimum. In [Siet 91] it is suggested that the beneficial effects that randomness may have on training can be further emphasized by adding a (small) white noise sequence in the training data.

Another commonly used practice focuses on the way the training data are presented in the network. *During training, the available training vectors are used in the update equation more than once* until the algorithm converges. One complete presentation of all N training pairs constitutes an *epoch*. As successive epochs are applied, it is good practice from the convergence point of view to randomize the order of presentation of the training pairs. Randomization can again help the pattern mode algorithm to jump out of regions around local minima, when this occurs. However, the final choice between the batch and pattern modes of operation depends on the specific problem [Hert 91, page 119].

- Once training of the network has been achieved, the values to which the synapses and thresholds have converged are frozen and the network is ready for classification. This is a much easier task than training. An unknown feature vector is presented in the input and is classified in the class that is indicated by the output of the network. The computations performed by the neurons are of the multiply–add type followed by a nonlinearity. This has led to various hardware implementations ranging from optical to VLSI chip design. Furthermore, neural networks have a natural built-in parallelism and computations in each layer can be performed in parallel. These distinct characteristics of neural networks have led to the development of special *neurocomputers*, and a number of those are already commercially available; see, for example, [Koli 97].

4.7 VARIATIONS ON THE BACKPROPAGATION THEME

Both versions of the backpropagation scheme, the batch and the pattern modes, inherit the disadvantage of all methods built upon the gradient descent approach: *their convergence to the cost function minimum is slow*. Appendix C discusses the fact that this trait becomes more prominent if the eigenvalues of the corresponding Hessian matrix exhibit large spread. In such cases, the change of the cost function gradient between successive iteration steps is not smooth but oscillatory, leading to slow convergence. One way to overcome this problem is to use a *momentum term* that smoothes out the oscillatory behavior and speeds up the convergence. The backpropagation algorithm with momentum term takes the form

$$\Delta \boldsymbol{w}_j^r(\text{new}) = \alpha \Delta \boldsymbol{w}_j^r(\text{old}) - \mu \sum_{i}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i) \tag{4.24}$$

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r(\text{new}) \tag{4.25}$$

Compared with (4.4), we see that the correction vector $\Delta \boldsymbol{w}_j^r$ depends not only on the gradient term but also on its value in the previous iteration step. The constant

α is called the *momentum factor* and in practice is chosen between 0.1 and 0.8. To see the effect of the momentum factor, let us look at the correction term for a number of successive iteration steps. At the tth iteration step we have

$$\Delta \boldsymbol{w}_j^r(t) = \alpha \Delta \boldsymbol{w}_j^r(t-1) - \mu \boldsymbol{g}(t) \tag{4.26}$$

where the last term denotes the gradient. For a total of T successive iteration steps we obtain

$$\Delta \boldsymbol{w}_j^r(T) = -\mu \sum_{t=0}^{T-1} \alpha^t \boldsymbol{g}(T-t) + \alpha^T \Delta \boldsymbol{w}_j^r(0) \tag{4.27}$$

Since $\alpha < 1$, the last term goes close to zero after a few iteration steps and the smoothing (averaging) effect of the momentum term becomes apparent. Let us now assume that the algorithm is at a low-curvature point of the cost function surface in the weight space. We can then assume that the gradient is approximately constant over a number of iteration steps. Applying this, we can write that

$$\Delta \boldsymbol{w}_j^r(T) \simeq -\mu(1 + \alpha + \alpha^2 + \alpha^3 + \cdots)\boldsymbol{g} = -\frac{\mu}{1-\alpha}\boldsymbol{g}$$

In other words, in such cases the effect of the momentum term is to effectively increase the learning constant. In practice, improvements in converging speed by a factor of 2 or even more have been reported [Silv 90].

A heuristic variation of this is to use an adaptive value for the learning factor μ, depending on the cost function values at successive iteration steps. A possible procedure is the following: Let $J(t)$ be the value of the cost at the tth iteration step. If $J(t) < J(t-1)$, then increase the learning rate by a factor of r_i. If, on the other hand, the new value of the cost is larger than the old one by a factor c, then decrease the learning rate by a factor of r_d. Otherwise use the same value. In summary

$$\begin{aligned}
\frac{J(t)}{J(t-1)} &< 1, \quad \mu(t) = r_i \mu(t-1) \\
\frac{J(t)}{J(t-1)} &> c, \quad \mu(t) = r_d \mu(t-1) \\
1 \leq \frac{J(t)}{J(t-1)} &\leq c, \quad \mu(t) = \mu(t-1)
\end{aligned}$$

Typical values of the parameters which are adopted in practice are $r_i = 1.05$, $r_d = 0.7$, $c = 1.04$. For iteration steps where the cost increases, it may be advantageous not only to decrease the learning rate but also to set the momentum term equal to 0. Others suggest not to perform the update of the weighting at this step.

Another strategy for updating the learning factor μ is followed in the so called *delta-delta* rule and in its modification *delta-bar-delta* rule [Jaco 88]. The idea here is to use a different learning factor for each weight and to increase the particular learning factor if the gradient of the cost function with respect to the corresponding weight has the same sign on two successive iteration steps. Conversely, if the sign changes this is an indication of a possible oscillation and the learning factor should be reduced. A number of alternative techniques for speeding up convergence have also been suggested. In [Cich 93] a more extensive review of such techniques is provided.

The other option for faster convergence is to free ourselves from the gradient descent rationale and to adopt alternative searching schemes, usually at the expense of increased complexity. A number of such algorithmic techniques have appeared in the related literature. For example, [Kram 89, Barn 92, Joha 92] present algorithmic schemes based on the conjugate gradient algorithm, [Batt 92, Rico 88, Barn 92, Watr 88] provide schemes of the Newton family, [Palm 91, Sing 89] propose algorithms based on the Kalman filtering approach, and [Bish 95] a scheme based on the Levenberg–Marquardt algorithm. In many of these algorithms, elements of the Hessian matrix need to be computed, that is, the second derivatives of the cost function with respect to the weights

$$\frac{\partial^2 J}{\partial w_{jk}^q \partial w_{nm}^r}$$

The computations of the Hessian matrix are performed by adopting, once more, the backpropagation concept (see also Problems 4.12, 4.13). More on these issues can be found in [Hayk 99, Zura 92].

A popular scheme that is loosely based on Newton's method is the *quickprop* scheme [Fahl 90]. It is a heuristic method and treats the weights as if they were quasi-independent. It then approximates the error surface, as a function of each weight, by a quadratic polynomial. If this has its minimum at a sensible value, the latter is used as the new weight for the iterations, otherwise a number of heuristics are used. A usual form of the algorithm, for the weights in the various layers, is

$$\Delta w_{ij}(t) = \begin{cases} \alpha_{ij}(t)\Delta w_{ij}(t-1), & \text{if } \Delta w_{ij}(t-1) \neq 0 \\ \mu \frac{\partial J}{\partial w_{ij}}, & \text{if } \Delta w_{ij}(t-1) = 0 \end{cases} \tag{4.28}$$

where

$$\alpha_{ij}(t) = \min \left\{ \frac{\frac{\partial J}{\partial w_{ij}}(t)}{\frac{\partial J}{\partial w_{ij}}(t-1) - \frac{\partial J}{\partial w_{ij}}(t)}, \alpha_{max} \right\} \tag{4.29}$$

with typical values of the involved variables being $0.01 \leq \mu \leq 0.6, \alpha_{max} \approx 1.75$ [Cich 93]. An algorithm similar in spirit to quickprop has been proposed in [Ried 93]. It is reported that it is as fast as quickprop and it requires less adjustment of the parameters to be stable.

4.8 THE COST FUNCTION CHOICE

It will not come as a surprise that the least squares cost function in (4.6) is not the unique choice available to the user. Depending on the specific problem, other cost functions can lead to better results. Let us look, for example, at the least squares cost function more carefully. Since all errors in the output nodes are first squared and summed up, large error values influence the learning process much more than the small errors. Thus, if the dynamic ranges of the desired outputs are not *all* of the same order, the least squares criterion will result in weights that have "learned" via a process of unfair provision of information. Furthermore, in [Witt 88] it is shown that for a class of problems, the gradient descent algorithm with the squared error criterion can be trapped in a local minimum and fail to find a solution, although (at least) one exists. In the current context a solution is assumed to be a classifier that classifies correctly all training samples. In contrast, it is shown that there is an alternative class of functions, satisfying certain criteria, which guarantee that the gradient descent algorithm converges to such a solution, provided that one exists. This class of cost functions is known as *well-formed functions*. We will now present a cost function of this type, which is well suited for pattern recognition tasks.

The multilayer network performs a nonlinear mapping of the input vectors $\boldsymbol{x}$ to the output values $\hat{y}_k = \phi_k(\boldsymbol{x}; \boldsymbol{w})$ for each of the output nodes $k = 1, 2, \ldots k_L$, where the dependence of the mapping on the values of the weights is explicitly shown. In Chapter 3 we have seen that, if we adopt the least squares cost function and the desired outputs y_k are binary (belong to or not in class ω_k), then for the optimal values of the weights $\boldsymbol{w}^*$ the corresponding output of the network, $\hat{y}_k$, is *the least squares optimal estimate of the posterior probability* $P(\omega_k|\boldsymbol{x})$ (the question of how good or bad this estimate is will be of interest to us soon). At this point we will adopt this probabilistic interpretation of the real outputs $\hat{y}_k$ as the basis on which our cost function will be built. Let us assume that the desired output values, y_k, are independent binary random variables and that $\hat{y}_k$ are the respective posterior probabilities that these random variables are 1 [Hint 90, Baum 88].

The *cross-entropy* cost function is then defined by

$$J = -\sum_{i=1}^{N}\sum_{k=1}^{k_L}(y_k(i)\ln\hat{y}_k(i) + (1 - y_k(i))\ln(1 - \hat{y}_k(i))) \tag{4.30}$$

J takes its minimum value when $y_k(i) = \hat{y}_k(i)$ and for binary desired response values the minimum is zero. There are various interpretations of this cost function [Hint 90, Baum 88, Gish 90, Rich 91]. Let us consider, for example, the output vector $\boldsymbol{y}(i)$ when $\boldsymbol{x}(i)$ appears at the input. This consists of a 1 at the true class node and zero elsewhere. If we take into account that the probability of the kth node to be 1(0) is $\hat{y}_k(i)(1 - \hat{y}_k(i))$ and by considering nodes *independently*, then

$$p(\boldsymbol{y}) = \prod_{k=1}^{k_L} (\hat{y}_k)^{y_k} (1 - \hat{y}_k)^{1-y_k} \tag{4.31}$$

where the dependence on i has been suppressed for notational convenience. Then it is straightforward to check that J results from the negative loglikelihood of the training sample pairs. If $y_k(i)$ were true probabilities in (0, 1) then subtracting the minimum value from J (4.30) becomes

$$J = -\sum_{i=1}^{N} \sum_{k=1}^{k_L} \left(y_k(i) \ln \frac{\hat{y}_k(i)}{y_k(i)} + (1 - y_k(i)) \ln \frac{1 - \hat{y}_k(i)}{1 - y_k(i)} \right) \tag{4.32}$$

For binary valued y_ks the above is still valid if we use the limiting value $0 \ln 0 = 0$.

It is not difficult to show (Problem 4.5) that the *cross entropy cost function depends on the relative errors and not on the absolute errors, as its least squares counterpart, thus it gives the same weight to small and large values*. Furthermore, it has been shown that it satisfies the conditions of the well-formed functions [Adal 97]. Finally, it can be shown that adopting the cross entropy cost function and binary values for the desired responses, *the outputs $\hat{y}_k$ corresponding to the optimal weights $\boldsymbol{w}^*$ are indeed estimates of $P(\omega_k|\boldsymbol{x})$*, as in the least squares case [Hamp 90].

A major advantage of the cross entropy cost function is that it diverges if one of the outputs converges to the wrong extreme, hence the gradient descent reacts fast. On the other hand, the squared error cost function approaches a constant in this case, and the gradient descent on the LS will wander on a plateau, even though the error may not be small. This advantage of the cross entropy cost function is demonstrated in the channel equalization context in [Adal 97].

A different cost function results if we treat $\hat{y}_k(i)$ and $y_k(i)$ as the true and desired probabilities, respectively. Then a measure of their similarity is given by the cross-entropy function (Appendix A)

$$J = -\sum_{i=1}^{N} \sum_{k=1}^{k_L} y_k(i) \ln \frac{\hat{y}_k(i)}{y_k(i)} \tag{4.33}$$

This is also valid for binary target values (using the limiting form). However, although we have interpreted the outputs as probabilities, there is no guarantee that they sum up to unity. This can be imposed onto the network by adopting an alternative activation function for the output nodes. In [Brid 90] the so-called *softmax* activation function was suggested, given by

$$\hat{y}_k = \frac{\exp(v_k^L)}{\sum_{k'} \exp(v_{k'}^L)} \tag{4.34}$$

This guarantees that the outputs lie in the interval [0, 1] and that they sum up to unity (note that in contrast to (4.32) the output probabilities are not considered independently). It is easy to show, Problem 4.7, that in this case the quantity δ_j^L required by the backpropagation is equal to $\hat{y}_k - y_k$.

Besides the cross entropy cost function in (4.33) a number of alternative cost functions has been proposed. For example, in [Kara 92] a generalization of the quadratic error cost function is utilized with the aim of speeding up convergence. Another direction is to minimize the classification error, which after all is the major goal in pattern recognition. A number of techniques have been suggested with this philosophy [Nede 93, Juan 92, Pado 95], which is known as *discriminative learning*. The basic potential advantage of discriminative learning is that essentially it tries to move the decision surfaces so as to reduce classification error. To achieve this goal, it puts more emphasis on the largest of the class a posteriori probability estimates. In contrast, the squared error cost function, for example, assigns the same importance to all posterior probability estimates. In other words, it tries to learn more than what is necessary for classification, which may limit its performance for a fixed size network. Most of the discriminative learning techniques use a smoothed version of the classification error, so as to be able to apply differentiation in association with gradient descent approaches. This, of course, has the danger that the minimization procedure will be trapped in a local minimum. In [Mill 96] a *deterministic annealing* procedure is employed to train the networks, with an enhanced potential to avoid local minima (see Chapter 15).

The final choice of the cost function depends on the specific problem under consideration. However, as is pointed out in [Rich 91], in a number of practical situations the use of alternative, to least squares, cost functions did not necessarily lead to substantial performance improvements.

A Bayesian Framework for Network Training

All the cost functions considered so far aim at computing a single set of optimal values for the unknown parameters of the network. An alternative rationale is to look at the probability distribution function of the unknown weights, $\boldsymbol{w}$, in the weight space. The idea behind this approach stems from the Bayesian inference

technique used for the estimation of an unknown parametric pdf, as we discussed in Chapter 2. The basic steps followed for this type of network training, known as *Bayesian learning*, are (e.g., [Mack 92a]):

- Assume a model for the prior distribution $p(\boldsymbol{w})$ of the weights. This must be rather broad in shape, to provide equal chance to a rather large range of values.
- Let $Y = \{\boldsymbol{y}(i), i = 1, 2, \ldots, N\}$ be the set of the desired output training vectors for a given input data set $X = \{\boldsymbol{x}(i), i = 1, 2, \ldots N\}$. Assume a model for the likelihood function $p(Y|\boldsymbol{w})$, for example, Gaussian.[4] This basically models the error distribution between the true and desired output values, and it is the stage where the input training data come into the scene.
- Using Bayes' theorem, we obtain

$$p(\boldsymbol{w}|Y) = \frac{p(Y|\boldsymbol{w})p(\boldsymbol{w})}{p(Y)} \tag{4.35}$$

  where $p(Y) = \int p(Y|\boldsymbol{w})p(\boldsymbol{w})\, d\boldsymbol{w}$. The resulting posterior pdf will be more sharply shaped around a value $\boldsymbol{w}_0$, since it has learned from the available training data.
- Interpreting the true outputs of a network, $\hat{y}_k = \phi_k(\boldsymbol{x}; \boldsymbol{w})$, as the respective class probabilities, conditioned on the input $\boldsymbol{x}$ and the weight vector $\boldsymbol{w}$, the conditional class probability is computed by averaging over all $\boldsymbol{w}$ [Mack 92b]:

$$P(\omega_k|\boldsymbol{x}; Y) = \int \phi_k(\boldsymbol{x}; \boldsymbol{w})p(\boldsymbol{w}|Y)\, d\boldsymbol{w} \tag{4.36}$$

The major computational cost associated with this type of technique is due to the required integration in the multidimensional space. This is not an easy task, and various practical implementations have been suggested in the literature. Further discussion of these issues is beyond the scope of this book. A good introduction to Bayesian learning, including a discussion of related practical implementations, is provided in [Bish 95].

4.9 CHOICE OF THE NETWORK SIZE

In the previous sections, we assumed the number of layers and neurons for each layer to be known and fixed. How one determines the appropriate number of layers and neurons was not of interest to us. This task will become our major focus now.

[4] Strictly speaking we should write $P(Y|\boldsymbol{w}, X)$. However all probabilities and pdf's are conditioned on X and we omit it for notational convenience.

One answer to the problem could be to choose the size of the network large enough and leave the training to decide about the weights. A little thought reveals that such an approach is rather naive. Besides the associated computational complexity problems, there is a major reason why the size of the network should be kept as small as possible. This is imposed by the generalization capabilities that the network must possess. As has already been pointed out in Section 3.6, the term generalization refers to the capability of the multilayer neural network (and of any classifier) to classify correctly feature vectors that were not presented to it during the training phase—that is, the capability of a network to decide upon data unknown to it, based on what it has learned from the training set. Taking for granted the finite (and in many cases small) number N of training pairs, the number of free parameters (synaptic weights) to be estimated should be *(a) large enough to learn what makes "similar" the feature vectors within each class and at the same time what makes one class different from the other and (b) small enough, with respect to N, so as not to be able to learn the underlying differences among the data of the same class.* When the number of free parameters is large, the network tends to adapt to the particular details of the specific training data set. This is known as *overfitting* and leads to poor generalization performance, when the network is called to operate on feature vectors unknown to it. In conclusion, the network should have the smallest possible size to adjust its weights to the largest regularities in the data and ignore the smaller ones, which might also be the result of noisy measurements. Some theoretical touches concerning the generalization aspects of a classifier will be presented in Chapter 5, when we discuss the Vapnik–Chernovenkis dimension. The bias–variance dilemma, discussed in Chapter 3, is another side of the same problem.

Adaptation of the free parameters to the peculiarities of the specific training set may also occur as the result of *overtraining* (e.g., see [Chau 90]). Let us assume that we can afford the luxury of having a large set of training data. We divide this set into two subsets, one for training and one for test. The latter is known as the *validation* or *test set*. Figure 4.14 shows the trend of two curves of the output error as a function of iteration steps. One corresponds to the training set, and we observe that the error keeps decreasing as the weights converge. The other corresponds to the error of the validation set. Initially the error decreases, but at some later stage it starts increasing. This is because the weights, computed from the training set, adapt to the idiosyncrasies of the specific training set, thus affecting the generalization performance of the network. This behavior could be used in practice to determine the point where the learning process iterations must terminate. This is the point where the two curves start departing. However, this methodology assumes the existence of a large number of data sets, which is not usually the case in practice.

Besides generalization, other performance factors also demand to keep the size of a network as small as possible. Small networks are computationally faster and cheaper to build. Furthermore, their performance is easier to understand, which is important in some critical applications.

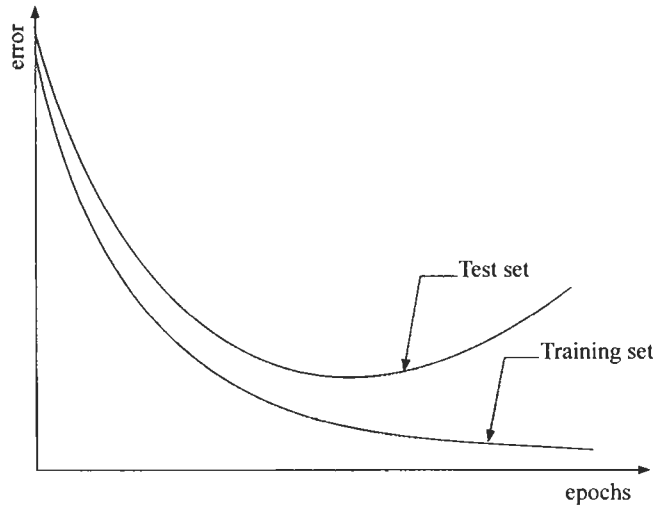


FIGURE 4.14: Trend of the output error versus the number of epochs illustrating overtraining of the training set.

In this section we focus on methods that select the appropriate number of free parameters, under certain criteria and for a given dimension of the input vector space. The latter is very important, because the input data dimension is no doubt related to the number of free parameters to be used, thus it also affects generalization properties. We will come to issues related to input space dimension reduction in Chapter 5.

The most widely used approaches to selecting the size of a multilayer network come under one of the following categories:

- *Analytical methods.* This category employs algebraic or statistical techniques to determine the number of its free parameters.
- *Pruning techniques.* A large network is initially chosen for training and then the number of free parameters is successively reduced, according to a preselected rule.
- *Constructive techniques.* A small network is originally selected and neurons are successively added, based on an appropriately adopted learning rule.

Algebraic Estimation of the Number of Free Parameters

We have already discussed in Section 4.3.1 the capabilities of a multilayer perceptron, with one hidden layer and units of the McCulloch–Pitts type, to divide the input l-dimensional space into a number of polyhedral regions. These are the

result of intersections of hyperplanes formed by the neurons. In [Mirc 89] it is shown that in the l-dimensional space a multilayer perceptron of a single hidden layer with K neurons can form *a maximum* of M polyhedral regions with M given by

$$M = \sum_{m=0}^{l} \binom{K}{m}, \text{ where } \binom{K}{m} = 0, \quad \text{for } K < m \tag{4.37}$$

and

$$\binom{K}{m} \equiv \frac{K!}{m!(K-m)!}$$

For example, if $l = 2$, and $K = 2$, this results in $M = 4$; thus, the XOR problem, with $M = 3(<4)$, can be solved with two neurons. The disadvantage of this method is that it is static and does not take into consideration the cost function used as well as the training procedure.

Pruning Techniques

These techniques start training a sufficiently large network and then they remove, in a stepwise procedure, the free parameters that have little influence on the cost function. There are two major methodological directions:

Methods based on parameter sensitivity calculations. Let us take for example the technique suggested in [Lecu 90]. Using a Taylor series expansion, the variation imposed on the cost function by parameter perturbations is

$$\delta J = \sum_i g_i \delta w_i + \frac{1}{2} \sum_i h_{ii} \delta w_i^2 + \frac{1}{2} \sum_{\substack{i,j \\ i \neq j}} h_{ij} \delta w_i \delta w_j$$

$$+ \text{higher order terms}$$

where

$$g_i = \frac{\partial J}{\partial w_i}, \qquad h_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

and i, j runs over all the weights. The derivatives can be computed via the back-propagation methodology (Problem 4.13). In practice, the computation of the derivatives is performed after some initial period of training. This allows us to adopt the assumption that a point near a minimum has been reached and the first

derivatives can be set equal to zero. A further computational simplification is to assume that the Hessian matrix is diagonal. Under these assumptions, the cost function sensitivity is approximately given by

$$\delta J = \frac{1}{2} \sum_i h_{ii} \delta w_i^2 \tag{4.38}$$

and the contribution of each parameter is determined by the *saliency* value s_i, given approximately by

$$s_i = \frac{h_{ii} w_i^2}{2} \tag{4.39}$$

where we assume that a weight of value w_i is changed to zero. Pruning is now achieved in an iterative fashion according to the following steps:

- The network is trained using the backpropagation algorithm for a number of iteration steps so that its cost function is reduced to a sufficient percentage.
- For the current weight estimates, the respective saliency values are computed and weights with small salencies are removed.
- The training process is continued with the remaining weights and the process is repeated after some iteration steps. The process is stopped when a chosen stopping criterion is met.

In [Hass 93] the full Hessian matrix has been employed for the pruning procedure. It should be stressed that although the backpropagation concept is present in this technique, the learning procedure is distinctly different from the backpropagation training algorithm of Section 4.6. *There, the number of free parameters was fixed throughout the training. In contrast, the philosophy here is exactly the opposite.*

Methods based on cost function regularization. These methods achieve the reduction of the originally large size of the network by including *penalty terms* in the cost function. The cost function now has the form

$$J = \sum_{i=1}^{N} \mathcal{E}(i) + \alpha \mathcal{E}_p(\boldsymbol{w}) \tag{4.40}$$

The first term is the performance cost function, and it is chosen according to what we have already discussed (e.g., least squares, cross entropy). The second depends on the weight vector, and it is chosen to *favor small values for the weights*. The constant α is the so-called *regularization parameter*, and it controls the relative

significance of these two terms. A popular form for the penalty term is

$$\mathcal{E}_p(\boldsymbol{w}) = \sum_{k=1}^{K} h(w_k^2) \tag{4.41}$$

with K being the total number of weights in the network and $h(\cdot)$ an appropriately chosen differentiable function. According to such a choice, weights that do not contribute significantly in the formation of the network output do not affect much the first term of the cost function. Hence, the existence of the penalty term drives them to small values. Thus, pruning is achieved. In practice, a threshold is preselected and weights are compared against it after a number of iteration steps. Weights that become smaller than it are removed, and the process is continued. This type of pruning is known as *weight elimination*. Function $h(\cdot)$ can take various forms. For example, in [Wein 90] the following is suggested:

$$h(w_k^2) = \frac{w_k^2}{w_0^2 + w_k^2} \tag{4.42}$$

where w_0 is a preselected parameter close to unity. Closer observation of this penalty term reveals that it goes to zero very fast for values $w_k < w_0$; thus such weights become insignificant. In contrast, the penalty term tends to unity for $w_k > w_0$.

A variation of (4.41) is to include in the regularized cost function another penalty term that favors small values of y_k^r, that is, small neuron outputs. Such techniques lead to removal of insignificant neurons as well as weights. A summary and discussion of various pruning techniques can be found in [Refe 91, Russ 93].

Constructive Techniques

In Section 4.5 we have already discussed such techniques for training neural networks. However, the activation function was the unit step function and also the emphasis was put on classifying *correctly all* input training data and not on the generalization properties of the resulting network. In [Fahl 90] an alternative constructive technique for training neural networks, with a single hidden layer and sigmoid activation functions, was proposed, known as *cascade correlation*. The network starts with input and output units only. Hidden neurons are added one by one and are connected to the network with two types of weights. The first type connects the new unit with the input nodes as well as the outputs of previously added hidden neurons. Each time a new hidden neuron is added in the network, these weights are trained so as to maximize the correlation between the new unit's output and the residual error signal in the network outputs prior to the addition of the new unit. Once a neuron is added, these weights are computed once and

then they remain fixed. The second type of synaptic weights connect the newly added neuron with the output nodes. These weights are not fixed and are trained adaptively, each time a new neuron is installed, in order to minimize a sum of squares error cost function. The procedure stops when the performance of the network meets the prespecified goals. A discussion on constructive techniques with an emphasis on pattern recognition can be found in [Pare 00].

4.10 A SIMULATION EXAMPLE

In this section, the capability of a multilayer perceptron to classify nonlinearly separable classes is demonstrated. The classification task consists of two distinct classes, each being the union of four regions in the two-dimensional space. Each region consists of normally distributed random vectors with statistically independent components and each with variance $\sigma^2 = 0.08$. The mean values are different for each of the regions. Specifically, the regions of the class denoted by "o" (see Figure 4.15) are formed around the mean vectors

$$[0.4, 0.9]^T, [2.0, 1.8]^T, [2.3, 2.3]^T, [2.6, 1.8]^T$$

and those of the class denoted by "+" around the values

$$[1.5, 1.0]^T, [1.9, 1.0]^T, [1.5, 3.0]^T, [3.3, 2.6]^T$$

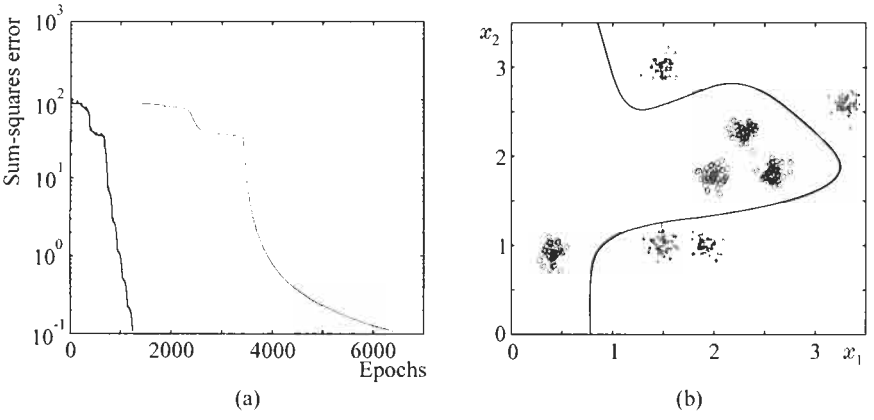


FIGURE 4.15: (a) Error convergence curves for the adaptive momentum and the momentum algorithms. Note that the adaptive momentum leads to faster convergence. (b) The decision curve formed by the multilayer perceptron.

A total of 400 training vectors were generated, 50 from each distribution. A multilayer perceptron with three neurons in the first and two neurons in the second hidden layer were used, with a single output neuron. The activation function was the logistic one with $a = 1$ and the desired outputs 1 and 0, respectively, for the two classes. Two different algorithms were used for the training, namely the momentum and the adaptive momentum. After some experimentation the algorithmic parameters employed were (a) for the momentum $\mu = 0.05$, $\alpha = 0.85$ and (b) for the adaptive momentum $\mu = 0.01$, $\alpha = 0.85$, $r_i = 1.05$, $c = 1.05$, $r_d = 0.7$. The weights were initialized by a uniform pseudorandom distribution between 0 and 1. Figure 4.15a shows the respective output error convergence curves for the two algorithms as a function of the number of epochs (each epoch consisting of the 400 training feature vectors). The respective curves can be considered typical and the adaptive momentum algorithm leads to faster convergence. Both curves correspond to the batch mode of operation. Figure 4.15b shows the resulting decision surface using the weights estimated from the adaptive momentum training. Once the weights of the network have been estimated, the decision surface can easily be drawn. To this end, a two-dimensional grid is constructed over the area of interest, and the points of the grid are given as inputs to the network, row by row. The decision surface is formed by the points where the output of the network changes from 0 to 1 or vice versa.

A second experiment was conducted in order to demonstrate the effect of the pruning. Figure 4.16 shows the resulting decision surfaces separating the samples of the two classes, denoted by "+" and "o" respectively. Figure 4.16a corresponds to a multilayer perceptron (MLP) with two hidden layers and 20 neurons in each of them, amounting to a total of 480 weights. Training was performed via the

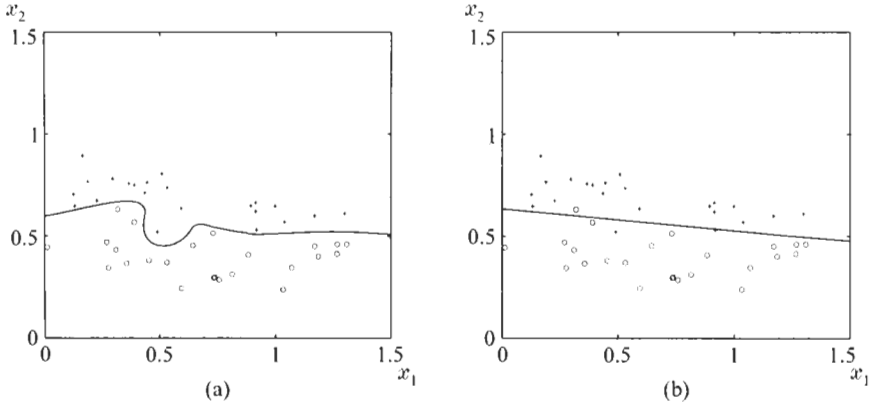


FIGURE 4.16: Decision curve (a) before pruning and (b) after pruning.

backpropagation algorithm. The overfitting nature of the resulting curve is readily observed. Figure 4.16b corresponds to the same MLP trained with a pruning algorithm. Specifically, the method based on parameter sensitivity was used, testing the saliency values of the weights every 100 epochs and removing weights with saliency value below a chosen threshold. Finally, only 25 of the 480 weights were left, and the curve is simplified to a straight line.

4.11 NETWORKS WITH WEIGHT SHARING

One major issue encountered in many pattern recognition applications is that of transformation invariance. This means that the pattern recognition system should classify correctly, independent of transformations performed on the input space, such as translation, rotation, and scaling. For example, the character "5" should "look the same" to an OCR system, irrespective of its position, orientation, and size. There is a number of ways to approach this problem. One is to choose appropriate feature vectors, which are invariant under such transformations. This will be one of our major goals in Chapter 7. Another way is to make the classifier responsible for it in the form of *built-in constraints. Weight sharing is such a constraint, which forces certain connections in the network to have the same weights.*

One type of networks in which the concept of weight sharing has been adopted is the so-called *higher order network.* These are multilayer perceptrons with activation functions acting on nonlinear, instead of linear, combinations of the input parameters. The outputs of the neurons are now of the form

$$f(y) = f\left(w_0 + \sum_i w_i x_i + \sum_{jk} w_{jk} x_j x_k\right)$$

This can be generalized to include higher order products. Let us now assume that the inputs to the network originate from a two-dimensional grid (image). Each point of the grid corresponds to a specific x_i and each pair (x_i, x_j) to a line segment. Invariance to translation is built in by constraining the weights $w_{jk} = w_{rs}$, whenever the respective line segments, defined by the points (x_j, x_k) and (x_r, x_s), are of the same gradient. Invariance to rotation can be built in by sharing weights corresponding to segments of equal length. Of course, all these are subject to inaccuracies caused by the resolution coarseness of the grid. Higher order networks can accommodate more complex transformations [Kana 92, Pera 92, Delo 94]. Because of the weight sharing, the number of free parameters for optimization is substantially reduced. However, it must be pointed out that, so far, such networks have not been widely used in practice. A special type of network called *circular backpropagation model*

results if

$$f(v) = f\left(w_0 + \sum_i w_i x_i + w_s \sum_i x_i^2\right)$$

The increase of the number of parameters is now small, and in [Ride 97] is claimed that the involvement of the nonlinear term offers the network increased representation power without affecting its generalization capabilities.

Besides the higher order networks, weight sharing has been used to impose invariance on first-order networks used for specific applications [Fuku 82, Rume 86, Fuku 92, Lecu 89]. The last, for example, is a system for handwritten zip code recognition. It is a hierarchical structure with three hidden layers and inputs the gray levels of the image pixels. Nodes in the first two layers form groups of two-dimensional arrays known as *feature maps*. Each node in a given map receives inputs from a specific window area of the previous layer, known as the *receptive field*. Translation invariance is imposed by forcing corresponding nodes in the same map, looking at different receptive fields, to share weights. Thus, if an object moves from one input receptive field to the other, the network responds in the same way.

4.12 GENERALIZED LINEAR CLASSIFIERS

In Section 4.3, dealing with the nonlinearly separable XOR problem, we saw that the neurons of the hidden layer performed a mapping that transformed the problem to a linearly separable one. The actual mapping was

$$x \longrightarrow y$$

with

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} f(g_1(x)) \\ f(g_2(x)) \end{bmatrix} \tag{4.43}$$

where $f(\cdot)$ is the activation function and $g_i(x), i = 1, 2$, the linear combination of the inputs performed by each neuron. This will be our kickoff point for this section.

Let us consider our feature vectors to be in the l-dimensional space $\mathcal{R}^l$ and assume that they belong to either of the two classes A, B, which are nonlinearly separable. Let $f_1(\cdot), f_2(\cdot), \ldots, f_k(\cdot)$ be nonlinear (in the general case) functions

$$f_i : \mathcal{R}^l \rightarrow \mathcal{R}, \quad i = 1, 2, \ldots, k$$

which define the mapping $\boldsymbol{x} \in \mathcal{R}^l \rightarrow \boldsymbol{y} \in \mathcal{R}^k$

$$\boldsymbol{y} \equiv \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_k(\boldsymbol{x}) \end{bmatrix} \tag{4.44}$$

Our goal now is to investigate whether there is an appropriate value for k and functions f_i so that classes A, B are linearly separable in the k-dimensional space of the vectors $\boldsymbol{y}$. In other words, we investigate whether there exists a k-dimensional space where we can construct a hyperplane $\boldsymbol{w} \in \mathcal{R}^k$ so that

$$w_0 + \boldsymbol{w}^T \boldsymbol{y} > 0, \quad \boldsymbol{x} \in A \tag{4.45}$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{y} < 0, \quad \boldsymbol{x} \in B \tag{4.46}$$

Assuming that in the original space the two classes were separable by a (nonlinear) hypersurface $g(\boldsymbol{x}) = 0$, relations (4.45), (4.46) are basically equivalent to approximating the nonlinear $g(\boldsymbol{x})$ as a linear combination of $f_i(\boldsymbol{x})$, that is,

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{k} w_i f_i(\boldsymbol{x}) \tag{4.47}$$

This is a typical problem of function approximation in terms of a preselected class of *interpolation functions* $f_i(\cdot)$. This is a well-studied task in numerical analysis, and a number of different interpolating functions have been proposed (exponential, polynomial, Tchebyshev, etc.). In the next sections we will focus on two such classes of functions, which have been widely used in pattern recognition.

Once the functions f_i have been selected, the problem becomes a typical design of a linear classifier, that is, to estimate the weights w_i in the k-dimensional space. This justifies the term *generalized linear classification*. Figure 4.17 shows the corresponding block diagram. The first layer of computations performs the mapping to the $\boldsymbol{y}$ space; the second layer performs the computation of the decision hyperplane. In other words, (4.47) corresponds to a *two-layer network* where the nodes of the hidden layer have different activation functions, $f_i(\cdot), i = 1, 2, \ldots, k$. For an M-class problem we need to design M such weight vectors $\boldsymbol{w}_r, r = 1, 2, \ldots, M$, one for each class, and select the rth class according to the maximum output $\boldsymbol{w}_r^T \boldsymbol{y} + w_{r0}$.

In the sequel, we will first try to justify our expectations, that by going to a higher dimensional space the classification task may be transformed into a linear one and then study popular alternatives for the choice of functions $f_i(\cdot)$.

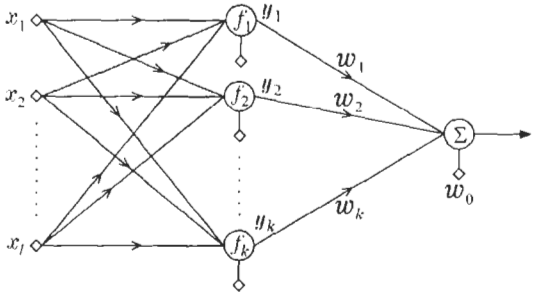


FIGURE 4.17: Generalized linear classifier.

4.13 CAPACITY OF THE l-DIMENSIONAL SPACE IN LINEAR DICHOTOMIES

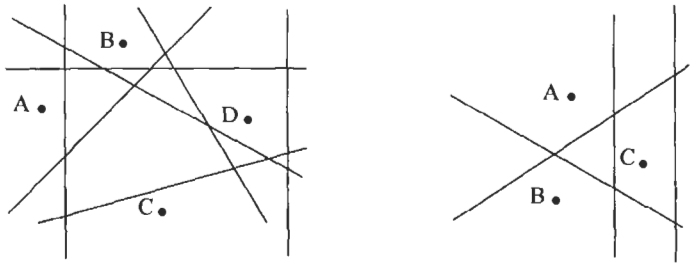
Let us consider N points in the l-dimensional space. We will say that these points are *in general position* or *well distributed* if there is no subset of $l + 1$ of them that lie on an $(l - 1)$-dimensional hyperplane. Such a definition excludes detrimental cases, such as in the two-dimensional space having three points on a straight line (a one-dimensional hyperplane). The number $O(N, l)$ of groupings that can be formed by $(l - 1)$-dimensional hyperplanes to separate the N points in two classes, taking all possible combinations, is given by ([Cove 65] and Problem 4.18):

$$O(N, l) = 2\sum_{i=0}^{l} \binom{N-1}{i} \tag{4.48}$$

where

$$\binom{N-1}{i} = \frac{(N-1)!}{(N-1-i)!i!} \tag{4.49}$$

Each of these two class groupings is also known as a (linear) *dichotomy*. From the properties of the binomial coefficients, it turns out that for $N \leq l + 1$, $O(N, l) = 2^N$. Figure 4.18 shows two examples of such hyperplanes resulting in $O(4, 2) = 14$ and $O(3, 2) = 8$ two-class groupings, respectively. The seven lines of Figure 4.18a form the following groupings. [(ABCD)], [A,(BCD)], [B,(ACD)], [C,(ABD)], [D,(ABC)], [(AB),(CD)], [(AC),(BD)]. Each grouping corresponds to two possibilities. For example, (ABCD) can belong to either class ω_1 or ω_2. Thus, the total number of combinations of assigning four points in the



(a) (b)

FIGURE 4.18: Number of linear dichotomies (a) for four and (b) for three points.

2-dimensional space in two *linearly separable classes is* 14. This number is obviously smaller than the total number of combinations of assigning N points in two classes, which is known to be 2^N. This is because the latter also involves nonlinearly separable combinations. In the case of our example, this is 16, which arises from the two extra possibilities of the grouping [(AD), (BC)]. We are now ready to write the probability (percentage) of grouping N points in the l-dimensional space in two *linearly* separable classes [Cove 65]. This is given by:

$$P_N^l = \frac{O(N,l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^{l} \binom{N-1}{i} & N > l+1 \\ 1 & N \leq l+1 \end{cases} \tag{4.50}$$

A practical way to study the dependence of P_N^l on N and l is to assume that $N = r(l+1)$ and investigate the probability for various values of r. The curve in Figure 4.19 shows the probability of having linearly separable classes for various values of l. It is readily observed that there are two regions, one to the left of $r = 2$, i.e., $N = 2(l+1)$ and one to the right. Furthermore, all curves go through the point $(P_N^l, r) = (1/2, 2)$, due to the fact that $O(2l+2, l) = 2^{2l+1}$ (Problem 4.19). The transition from one region to the other becomes sharper as $l \to \infty$. Thus, for large values of l and if $N < 2(l+1)$ the probability of any two groups of the N points being linearly separable approaches unity. The opposite is true if $N > 2(l+1)$. In practice, where we cannot afford the luxury of very large values of N and l, our findings guarantee that *if we are given N points, then mapping into a higher dimensional space increases the probability of locating them in linearly separable two-class groupings.*

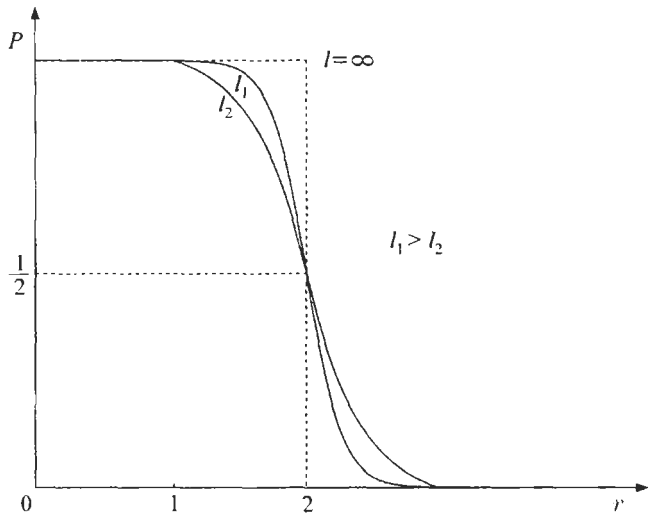


FIGURE 4.19: Probability of linearly separable groupings of $N = r(l + 1)$ points in the l-dimensional space.

4.14 POLYNOMIAL CLASSIFIERS

In this section we will focus on one of the most popular classes of interpolation functions $f_i(\boldsymbol{x})$ in (4.47). Function $g(\boldsymbol{x})$ is approximated in terms of up to order r polynomials of the $\boldsymbol{x}$ components, for large enough r. For the special case of $r = 2$ we have

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{l} w_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^{l} w_{im} x_i x_m + \sum_{i=1}^{l} w_{ii} x_i^2 \tag{4.51}$$

If $\boldsymbol{x} = [x_1, x_2]^T$, then the general form of $\boldsymbol{y}$ will be

$$\boldsymbol{y} = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

and

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{y} + w_0$$
$$\boldsymbol{w}^T = [w_1, w_2, w_{12}, w_{11}, w_{22}]$$

The number of free parameters determines the new dimension k. The generalization of (4.51) for rth-order polynomials is straightforward, and it will contain

products of the form $x_1^{p_1} x_2^{p_2} \cdots x_l^{p_l}$ where $p_1 + p_2 + \cdots + p_l \leq r$. For an rth-order polynomial and l-dimensional $\boldsymbol{x}$ it can be shown that

$$k = \frac{(l+r)!}{r!l!}$$

For $l = 10$ and $r = 10$ we obtain $k = 184{,}756$ (!!). That is, even for medium-size values of the network order and the input space dimensionality the number of free parameters gets very high.

Let us consider, for example, our familiar nonlinearly separable XOR problem. Define

$$\boldsymbol{y} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix} \tag{4.52}$$

The input vectors are mapped onto the vertices of a three-dimensional unit (hyper) cube, as shown in Figure 4.20a ((00) $\rightarrow$ (000), (11) $\rightarrow$ (111), (10) $\rightarrow$ (100), (01) $\rightarrow$ (010)). These vertices are separable by the plane

$$y_1 + y_2 - 2y_3 - \frac{1}{4} = 0$$

The plane in the three-dimensional space is equivalent to the decision function

$$g(\boldsymbol{x}) = -\frac{1}{4} + x_1 + x_2 - 2x_1 x_2 \begin{cases} > 0 & \boldsymbol{x} \in A \\ < 0 & \boldsymbol{x} \in B \end{cases}$$

in the original two-dimensional space, which is shown in Figure 4.20b.

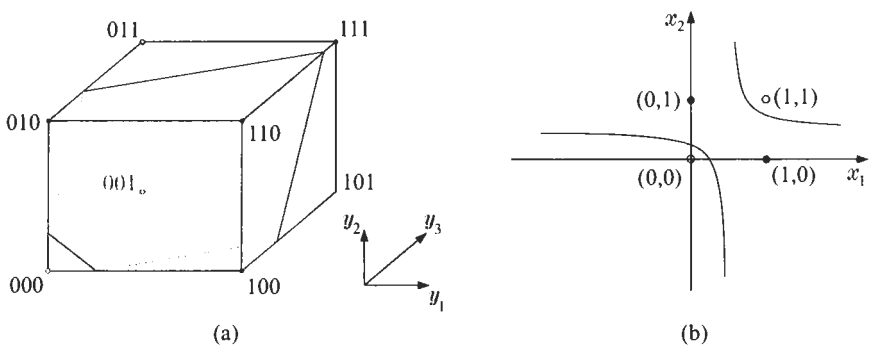


FIGURE 4.20: The XOR classification task, via the polynomial generalized linear classifier. (a) Decision plane in the three-dimensional space and (b) decision curves in the original two-dimensional space.

4.15 RADIAL BASIS FUNCTION NETWORKS

The interpolation functions (kernels) that will be considered in this section are of the general form

$$f(\|\boldsymbol{x} - \boldsymbol{c}_i\|)$$

That is, the argument of the function is the Euclidean distance of the input vector $\boldsymbol{x}$ from a center $\boldsymbol{c}_i$, which justifies the name *radial basis function (RBF)*. Function f can take various forms, for example,

$$f(\boldsymbol{x}) = \exp\left(-\frac{1}{2\sigma_i^2}\|\boldsymbol{x} - \boldsymbol{c}_i\|^2\right) \tag{4.53}$$

$$f(\boldsymbol{x}) = \frac{\sigma^2}{\sigma^2 + \|\boldsymbol{x} - \boldsymbol{c}_i\|^2} \tag{4.54}$$

The Gaussian form is more widely used. For a large enough value of k, it can be shown that the function $g(\boldsymbol{x})$ is sufficiently approximated by [Broo 88, Mood 89]

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{k} w_i \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{c}_i)^T(\boldsymbol{x} - \boldsymbol{c}_i)}{2\sigma_i^2}\right) \tag{4.55}$$

That is, the approximation is achieved via a summation of RBFs, where each is located on a different point in the space. One can easily observe the close relation that exists between this and the Parzen approximation method for the probability density functions of Chapter 2. Note, however, that there the number of the kernels was chosen to be equal to the number of training points $k = N$. In contrast, in (4.55) $k \ll N$. Besides the gains in computational complexity, this reduction in the number of kernels is beneficial for the generalization capabilities of the resulting approximation model.

Coming back to Figure 4.17, we can interpret (4.55) as the output of a network with *one* hidden layer of RBF activation functions (e.g., (4.53), (4.54)) and a *linear* output node. As has already been said in Section 4.12, for an M-class problem there will be M linear output nodes. At this point, it is important to stress one basic difference between RBF networks and multilayer perceptrons. In the latter, the inputs to the activation functions, of the first hidden layer, are linear combinations of the input feature parameters ($\sum_j w_j x_j$). That is, the output of each neuron is the same for all $\{\boldsymbol{x}: \sum_j w_j x_j = c\}$, where c is a constant. *Hence, the output is the same for all points on a hyperplane.* In contrast, in the RBF networks the output of each RBF node, $f_i(\cdot)$, is the same for all points having *the same Euclidean distance from the respective center* $\boldsymbol{c}_i$ and decreases exponentially (for Gaussians) with the distance. *In other words, the activation responses of the nodes*

are of a local nature in the RBF and of a global nature in the multilayer perceptron networks. This intrinsic difference has important repercussions for both the convergence speed and the generalization performance. In general, multilayer perceptrons learn slower than their RBF counterparts. In contrast, multilayer perceptrons exhibit improved generalization properties, especially for regions that are not represented sufficiently in the training set [Lane 91]. Simulation results in [Hart 90] show that, in order to achieve performance similar to that of multilayer perceptrons, an RBF network should be of much higher order. This is due to the locality of the RBF activation functions, which makes it necessary to use a large number of centers to fill in the space in which $g(\boldsymbol{x})$ is defined, and this number exhibits an exponential dependence on the dimension of the input space (curse of dimensionality) [Hart 90].

Let us now come back to our XOR problem and adopt an RBF network to perform the mapping to a linearly separable class problem. Choose $k = 2$, the centers $\boldsymbol{c}_1 = [1, 1]^T$, $\boldsymbol{c}_2 = [0, 0]^T$, and $f(\boldsymbol{x}) = \exp(-\|\boldsymbol{x} - \boldsymbol{c}_i\|^2)$. The corresponding $\boldsymbol{y}$ resulting from the mapping is

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}) = \begin{bmatrix} \exp(-\|\boldsymbol{x} - \boldsymbol{c}_1\|^2) \\ \exp(-\|\boldsymbol{x} - \boldsymbol{c}_2\|^2) \end{bmatrix}$$

Hence $(0, 0) \rightarrow (0.135, 1)$, $(1, 1) \rightarrow (1, 0.135)$, $(1, 0) \rightarrow (0.368, 0.368)$, $(0, 1) \rightarrow (0.368, 0.368)$. Figure 4.21a shows the resulting class position after the mapping in the $\boldsymbol{y}$ space. Obviously, the two classes are now linearly separable

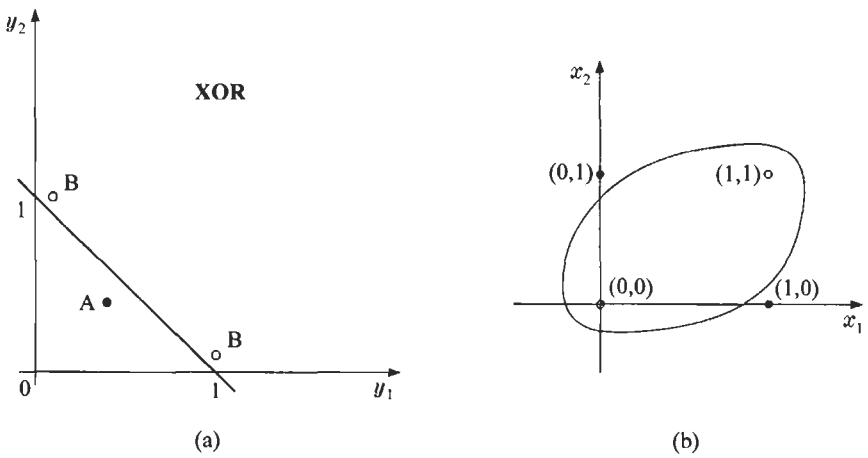


FIGURE 4.21: Decision curves formed by an RBF generalized linear classifier for the XOR task (a) in the transformed space and (b) in the original space.

and the straight line

$$g(\boldsymbol{y}) = y_1 + y_2 - 1 = 0$$

is a possible solution. Figure 4.21b shows the equivalent decision curve,

$$g(\boldsymbol{x}) = \exp(-\|\boldsymbol{x} - \boldsymbol{c}_1\|^2) + \exp(-\|\boldsymbol{x} - \boldsymbol{c}_2\|^2) - 1 = 0$$

in the input vector space. In our example we selected the centers $\boldsymbol{c}_1$, $\boldsymbol{c}_2$ as $[0, 0]^T$ and $[1, 1]^T$. The question now is, why these specific ones? *This is an important issue for RBF networks.* Some basic directions on how to tackle this problem are given in the following.

Fixed Centers

Although there exist some cases in which the nature of the problem suggests a specific choice for the centers [Theod 95], in the general case these centers can be selected randomly from the training set. Provided that the training set is distributed in a representative manner over all the feature vector space, this seems to be a reasonable way to choose the centers. Having now selected k centers for the RBF functions, the problem has become a typical linear one in the k-dimensional space of the vectors $\boldsymbol{y}$,

$$\boldsymbol{y} = \begin{bmatrix} \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{c}_1\|^2}{2\sigma_1^2}\right) \\ \vdots \\ \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{c}_k\|^2}{2\sigma_k^2}\right) \end{bmatrix}$$

where the variances are also considered to be known, and

$$g(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T \boldsymbol{y}$$

All methods described in Chapter 3 can now be recalled to estimate w_0 and $\boldsymbol{w}$.

Training of the Centers

If the centers are not preselected, they have to be estimated during the training phase along with the weights w_i and the variances σ_i^2, if the latter are also considered unknown. Let N be the number of input–desired output training pairs $(\boldsymbol{x}(j), y(j), j = 1, \ldots, N)$. We select an appropriate cost function of

the output error

$$J = \sum_{j=1}^{N} \phi(e(j))$$

where $\phi(\cdot)$ is a differentiable function (e.g., the square of its argument) of the error

$$e(j) = y(j) - g(\boldsymbol{x}(j))$$

The estimation of the weights w_i, the centers $\boldsymbol{c}_i$, and the variances σ_i^2 becomes a typical task of a nonlinear optimization process. For example, if we adopt the gradient descent approach, the following algorithm results:

$$w_i(t+1) = w_i(t) - \mu_1 \left.\frac{\partial J}{\partial w_i}\right|_t, \quad i = 0, 1, \ldots, k \tag{4.56}$$

$$\boldsymbol{c}_i(t+1) = \boldsymbol{c}_i(t) - \mu_2 \left.\frac{\partial J}{\partial \boldsymbol{c}_i}\right|_t, \quad i = 1, 2, \ldots, k \tag{4.57}$$

$$\sigma_i(t+1) = \sigma_i(t) - \mu_3 \left.\frac{\partial J}{\partial \sigma_i}\right|_t, \quad i = 1, 2, \ldots, k \tag{4.58}$$

where t is the current iteration step. The computational complexity of such a scheme is prohibitive for a number of practical situations. To overcome this drawback, alternative techniques have been suggested.

One way is to choose the centers in a manner representative of the way data are distributed in space. This can be achieved by unraveling the clustering properties of the data and choosing a representative for each cluster as the corresponding center [Mood 89]. This is a typical problem of *unsupervised learning*, and algorithms discussed in the relevant chapters later in the book can be employed. The unknown weights, w_i, are then learned via a supervised scheme (i.e., gradient descent algorithm) to minimize the output error. Thus, such schemes use a combination of supervised and unsupervised learning procedures.

An alternative strategy is described in [Chen 91]. A large number of candidate centers is initially chosen from the training vector set. Then, a forward linear regression technique is employed, such as orthogonal least squares, which leads to a parsimonious set of centers. This technique also provides a way to estimate the order of the model k. A recursive form of the method, which can lead in computational savings, is given in [Gomm 00].

Recently, another method has been proposed based on support vector machines. The idea behind this methodology is to look at the RBF network as a mapping machine, through the kernels, into a high-dimensional space. Then we design a hyperplane classifier using the vectors that are closest to the decision boundary.

These are the support vectors and correspond to the centers of the input space. The training consists of a quadratic programming problem and guarantees a global optimum [Scho 97]. The nice feature of this algorithm is that it automatically computes all the unknown parameters including the number of centers. We will return to it later in this chapter.

In [Plat 91] an approach similar in spirit to the constructive techniques, discussed for the multilayer perceptrons, has been suggested. The idea is to start training the RBF network with few nodes (initially one) and keep growing the network by allocating new ones, based on the "novelty" in the feature vectors that arrive sequentially. The novelty of each training input-desired output pair is determined by two conditions: a) the input vector to be very far (according to a threshold) from all already existing centers *and* b) the corresponding output error (using the RBF network trained up to this point) greater than another predetermined threshold. If both conditions are satisfied then the new input vector is assigned as the new center. If not, the input-desired output pair is used to update the parameters of the network according to the adopted training algorithm, for example the gradient descent scheme. A variant of this scheme that allows removal of previously assigned centers has also been suggested in [Ying 98]. This is basically a combination of the constructive and pruning philosophies. The procedure suggested in [Kara 97] also moves along the same direction. However, the assignment of the new centers is based on a procedure of progressive splitting (according to a splitting criterion) of the feature space using clustering or learning vector quantization techniques (Chapter 14). The representatives of the regions are then assigned as the centers of the RBF's. As was the case with the aforementioned techniques, network growing and training is performed concurrently.

A number of other techniques have also been suggested. For a review see, for example, [Hush 93]. A comparison of RBF networks with different center selection strategies versus multilayer perceptrons in the context of speech recognition is given in [Wett 92]. Reviews involving RBF networks and related applications are given in [Hayk 96, Mulg 96].

4.16 UNIVERSAL APPROXIMATORS

In this section we provide the basic guidelines concerning the approximation properties of the nonlinear functions used throughout this chapter, that is, sigmoid, polynomial, and radial basis functions. The theorems that are stated justify the use of the corresponding networks as decision surface approximators as well as probability function approximators, depending on how we look at the classifier.

In (4.51) the polynomial expansion was used to approximate the nonlinear function $g(\boldsymbol{x})$. This choice for the approximation functions is justified by the Weierstrass theorem.

Theorem. *Let $g(\boldsymbol{x})$ be a continuous function defined in a compact (closed) subset $S \subset \mathcal{R}^l$, and $\epsilon > 0$. Then there are an integer $r = r(\epsilon)$ and a polynomial function $\phi(\boldsymbol{x})$ of order r so that*

$$|g(\boldsymbol{x}) - \phi(\boldsymbol{x})| < \epsilon, \quad \forall \boldsymbol{x} \in S$$

In other words, function $g(\boldsymbol{x})$ can be approximated arbitrarily closely for sufficiently large r. A major problem associated with polynomial expansions is that good approximations are usually achieved for large values of r. That is, the convergence to $g(\boldsymbol{x})$ is slow. In [Barr 93] it is shown that the approximation error is reduced according to an $O(\frac{1}{r^{2/l}})$ rule, where $O(\cdot)$ denotes order of magnitude. *Thus, the error decreases more slowly with increasing dimension l of the input space*, and large values of r are necessary for a given approximation error. However, large values of r, besides the computational complexity and generalization issues (due to the large number of free parameters required), also lead to poor numerical accuracy behavior in the computations, because of the large number of products involved. On the other hand, the polynomial expansion can be used effectively for piecewise approximation, where smaller r's can be adopted.

The slow decrease of the approximation error with respect to the system order and the input space dimension is common to all expansions of the form (4.47) with fixed basis functions $f_i(\cdot)$. The scenario becomes different if data adaptive functions are chosen, as is the case with the multilayer perceptrons. In the latter, the argument in the activation functions is $f(\boldsymbol{w}^T\boldsymbol{x})$, with $\boldsymbol{w}$ computed in an optimal fashion from the available data.

Let us now consider a two-layer perceptron with one hidden layer, having k nodes with activation functions $f(\cdot)$ and an output node with *linear* activation. The output of the network is then given by

$$\phi(\boldsymbol{x}) = \sum_{j=1}^{k} w_j^o f(\boldsymbol{w}_j^{hT}\boldsymbol{x}) + w_o^o \tag{4.59}$$

where h refers to the weights, including the thresholds, of the hidden layer and o to the weights of the output layer. Provided that $f(\cdot)$ is a squashing function, the following theorem establishes the universal approximation properties of such a network [Cybe 89, Funa 89, Horn 89, Ito 91, Kalo 97].

Theorem. *Let $g(\boldsymbol{x})$ be a continuous function defined in a compact subset $S \subset \mathcal{R}^l$ and $\epsilon > 0$. Then there exist $k = k(\varepsilon)$ and a two-layer perceptron (4.59) so that*

$$|g(\boldsymbol{x}) - \phi(\boldsymbol{x})| < \epsilon, \quad \forall \boldsymbol{x} \in S$$

In [Barr 93], it is shown that, in contrast to the polynomial expansion, the approximation error decreases according to an $O(\frac{1}{k})$ rule. *In other words, the input space*

dimension does not enter explicitly into the scene and the error is inversely proportional to the system order, that is, the number of neurons. Obviously, the price we pay for it is that the optimization process is now nonlinear, with the associated disadvantage of the potential for convergence to local minima. The question that now arises is whether we gain anything by using more than one hidden layer, since a single one is sufficient for the function approximation. An answer is that using more than one layer may lead to a more efficient approximation; that is, the same accuracy is achieved with fewer neurons in the network.

The universal approximation property is also true for the class of RBF functions. For sufficiently large values of k in (4.55) the resulting expansion can approximate arbitrarily closely any continuous function in a compact subset S [Park 91, Park 93].

4.17 SUPPORT VECTOR MACHINES: THE NONLINEAR CASE

In Chapter 3, we discussed the support vector machines (SVM) as an optimal design methodology of a linear classifier. Let us now assume that there exists a mapping

$$\boldsymbol{x} \in \mathcal{R}^l \longrightarrow \boldsymbol{y} \in \mathcal{R}^k$$

from the input feature space into a k-dimensional space, where the classes can satisfactorily be separated by a hyperplane. Then, in the framework discussed in section 4.12, the SVM method can be mobilized for the design of the hyperplane classifier in the new k-dimensional space. However, there is an elegant property in the SVM methodology, that can be exploited for the development of a more general approach. This will also allow us for (implicit) mappings in infinite dimensional spaces, if required.

Recall from Chapter 3 that, in the computations involved in the Wolfe dual representation the feature vectors participate in pairs, via the inner product operation. Also, once the optimal hyperplane $(\boldsymbol{w}, w_0)$ has been computed, classification is performed according to whether the sign of

$$\begin{aligned} g(\boldsymbol{x}) &= \boldsymbol{w}^T \boldsymbol{x} + w_0 \\ &= \sum_{i=1}^{N_s} \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + w_0 \end{aligned}$$

is $+$ or $-$, where N_s is the number of support vectors. Thus, once more, only inner products enter into the scene. If the design is to take place in the new k-dimensional space, the only difference is that the involved vectors will be the k-dimensional mappings of the original input feature vectors. A naive look at it would lead to

the conclusion that now the complexity is much higher, since, usually, k is much higher than the input space dimensionality l, in order to make the classes linearly separable. However, there is a nice surprise just waiting for us. Let us start with a simple example. Assume that

$$\boldsymbol{x} \in \mathcal{R}^2 \longrightarrow \boldsymbol{y} = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

Then, it is a matter of simple algebra to show that

$$\boldsymbol{y}_i^T \boldsymbol{y}_j = \left(\boldsymbol{x}_i^T \boldsymbol{x}_j\right)^2$$

In words, the inner product of the vectors in the new (higher dimensional) space has been expressed as a function of the inner product of the corresponding vectors in the original feature space. Most interesting!

Theorem. *Mercer's Theorem. Let* $\boldsymbol{x} \in \mathcal{R}^l$ *and a mapping* $\boldsymbol{\phi}$

$$\boldsymbol{x} \longrightarrow \boldsymbol{\phi}(\boldsymbol{x}) \in H$$

where H *is a Euclidean space.*[5] *Then the inner product operation has an equivalent representation*

$$\sum_r \phi_r(\boldsymbol{x})\phi_r(\boldsymbol{z}) = K(\boldsymbol{x}, \boldsymbol{z}) \tag{4.60}$$

where $\phi_r(\boldsymbol{x})$ *is the* r*-component of the mapping* $\boldsymbol{\phi}(\boldsymbol{x})$ *of* $\boldsymbol{x}$*, and* $K(\boldsymbol{x}, \boldsymbol{z})$ *is a symmetric function satisfying the following condition*

$$\int K(\boldsymbol{x}, \boldsymbol{z})g(\boldsymbol{x})g(\boldsymbol{z})\,d\boldsymbol{x}\,d\boldsymbol{z} \geq 0 \tag{4.61}$$

for any $g(\boldsymbol{x})$, $\boldsymbol{x} \in \mathcal{R}^l$ *such that*

$$\int g(\boldsymbol{x})^2\,d\boldsymbol{x} < +\infty \tag{4.62}$$

The opposite is also true, i.e., for *any* function $K(\boldsymbol{x}, \boldsymbol{z})$ satisfying (4.61) and (4.62) there exists a space in which $K(\boldsymbol{x}, \boldsymbol{z})$ defines an inner product! Such functions are also known as *kernels*. What, however, Mercer's theorem does not disclose to us is how to find this space. That is, we do not have a general tool to construct

[5] A linear space equipped with an inner product operation

the mapping $\boldsymbol{\phi}(\cdot)$ once we know the inner product of the corresponding space. Furthermore, we lack the means to know the dimensionality of the space, which can even be infinite. For more on these issues, the mathematically inclined reader is referred to [Cour 53].

Typical examples of kernels used in pattern recognition applications are

Polynomials

$$K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z} + 1)^q, \quad q > 0 \tag{4.63}$$

Radial Basis Functions

$$K(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{\sigma^2}\right) \tag{4.64}$$

Hyperbolic Tangent

$$K(\boldsymbol{x}, \boldsymbol{z}) = \tanh\left(\beta \boldsymbol{x}^T \boldsymbol{z} + \gamma\right) \tag{4.65}$$

for appropriate values of β and γ so that Mercer's conditions are satisfied. One possibility is $\beta = 2$, $\gamma = 1$.

Once an appropriate kernel has been adopted that implicitly defines a mapping into a higher dimensional space, the Wolfe dual optimization task (Eqs. (3.91)–(3.93)) becomes

$$\max_{\boldsymbol{\lambda}} \left(\sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \tag{4.66}$$

$$\text{subject to} \quad 0 \leq \lambda_i \leq C, \quad i = 1, 2, \ldots, N \tag{4.67}$$

$$\sum_i \lambda_i y_i = 0 \tag{4.68}$$

and the resulting linear classifier is

$$\text{assign } \boldsymbol{x} \text{ in } \omega_1 (\omega_2) \text{ if } g(\boldsymbol{x}) = \sum_{i=1}^{N_s} \lambda_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + w_0 > (<)\, 0 \tag{4.69}$$

Figure 4.22 shows the corresponding architecture. This is nothing else than a special case of the generalized linear classifier of Figure 4.17. The number of nodes is determined by the number of support vectors N_s. The nodes perform the inner products between the mapping of $\boldsymbol{x}$ and the corresponding mappings of the support vectors in the high dimensional space, via the kernel operation.

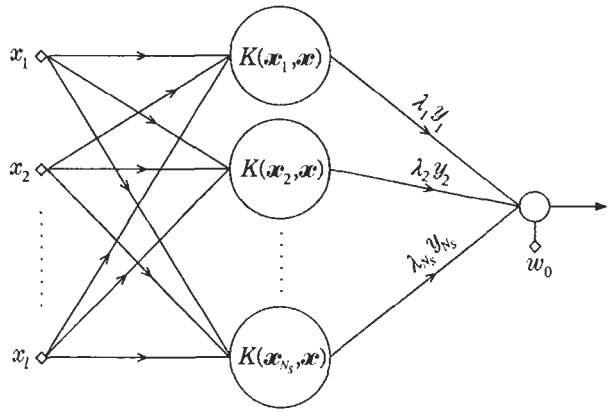


FIGURE 4.22: The SVM architecture employing kernel functions.

Remarks

- Notice that if the kernel function is the RBF, then the architecture is the same as the RBF network architecture of Figure 4.17. However, the approach followed here is different. In Section 4.15, a mapping in a k-dimensional space was first performed and the centers of the RBF functions had to be estimated. In the SVM approach, the number of nodes as well as the centers are the result of the optimization procedure.
- The hyperbolic tangent function is a sigmoid one. If it is chosen as a kernel, the resulting architecture is a special case of a two layer perceptron. Once more, the number of nodes is the result of the optimization procedure. This is important. Although the SVM architecture is the same as that of a 2-layer perceptron, the training procedure is entirely different for the two methods. The same is true for the RBF networks.
- A notable characteristic of the support vector machines is that the computational complexity is independent of the dimensionality of the kernel space, where the input feature space is mapped. Thus, the curse of dimensionality is bypassed. In other words, one designs in a high dimensional space without having to adopt explicit models using a large number of parameters, as this would be dictated by the high dimensionality of the space. This has also an influence on the generalization properties and indeed, SVM's tend to exhibit *good generalization performance*. We will return to this issue at the end of Chapter 5.
- A major limitation of support vector machines is the high computational burden required, both during training and in the test phase. For problems with a relatively small number of training data, any general purpose optimization

algorithm can be used. However, for a large number of training points (of the order of a few thousands), a special treatment is required. Training of SVM is usually performed in batch mode. For large problems this sets high demands on computer memory requirements. To attack such problems a number of procedures have been devised. Their philosophy relies on the decomposition, in one way or another, of the optimization problem into a sequence of smaller ones, e.g., [Bose 92, Osun 97, Chan 00]. Recently, an algorithm has been proposed that solves the problem iteratively, by assuming that at each iteration step only two of the Langrange multipliers are unknown and the rest are known (beginning with an initial assumption.) The procedure takes advantage of the fact that an analytical solution for the dual problem is possible for the case of two Langrange multipliers, [Matt 99, Plat 99]. Another sequential algorithm has been proposed in [Navi 01], where an iterative reweighted least squares procedure is employed and alternates weight optimization with constraint forcing. An advantage of the latter technique is that it naturally leads to on-line and adaptive implementations.

For large problems, the test phase can also be quite demanding, if the number of support vectors is excessively high. Methods that speed up computations have also been suggested, e.g., [Burg 97].

- Another major limitation of the support vector machines is that, up to now, there is no practical method for the best selection of the kernel function. This is still an unsolved, yet challenging, research issue.
- Support vector machines have been applied to a number of diverse applications, ranging from hand-written digit recognition ([Cort 95]), to object recognition ([Blan 96]), person identification ([Ben 99]), spam categorization ([Druc 99]), and channel equalization ([Seba 00]). The results from these applications indicate that SVM classifiers exhibit enhanced generalization performance, which seems to be the power of support vector machines.
- Besides support vector machines any other linear classifier that employs inner products can implicitly be executed in higher dimensional spaces by using kernels. In this way one can elegantly construct nonlinear versions of a linear algorithm. For a review on this issue see, for example, [Mull 01].

4.18 DECISION TREES

In this section we briefly review a large class of nonlinear classifiers known as *decision trees*. They are *multistage* decision systems in which classes are sequentially rejected until we reach a finally accepted class. To this end, the feature space is split into unique regions, corresponding to the classes, *in a sequential manner*.

Upon the arrival of a feature vector, the searching of the region to which the feature vector will be assigned is achieved via a sequence of decisions along a path of *nodes* of an appropriately constructed *tree*. Such schemes offer advantages when a large number of classes is involved. The most popular among the decision trees are those that split the space into hyperrectangles with sides parallel to the axes. The sequence of decisions is applied to individual features, and the questions to be answered are of the form "*is feature* $x_i \leq \alpha$?" where α is a threshold value. Such trees are known as *ordinary binary classification trees (OBCTs)*. Other types of trees are also possible that split the space into convex polyhedral cells or into pieces of spheres.

The basic idea behind an OBCT is demonstrated via the simplified example of Figure 4.23. By a successive sequential splitting of the space we have created regions corresponding to the various classes.

Figure 4.24 shows the respective binary tree with its decision nodes and leaves. Note that it is possible to reach a decision without having tested *all* the available features.

The task illustrated in Figure 4.23 is a simple one in the two-dimensional space. The thresholds used for the *binary splits* at each node of the tree in Figure 4.24 were dictated by a simple observation of the geometry of the problem. However, this is not possible in higher dimensional spaces. Furthermore, we started the queries by testing x_1 against $\frac{1}{4}$. An obvious question is why to consider x_1 first and not another feature. In the general case, in order to develop a binary decision tree,

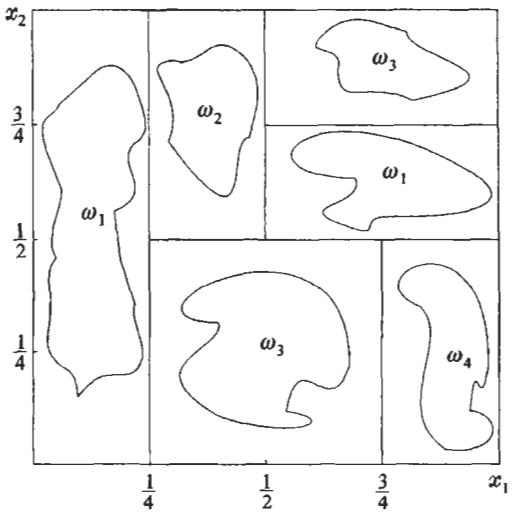


FIGURE 4.23: Decision tree partition.

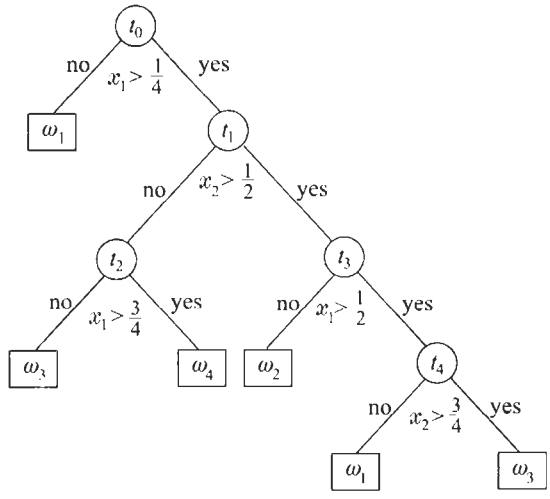


FIGURE 4.24: Decision tree classification.

the following design elements have to be considered by the designer in the training phase:

- At each node, the set of candidate questions to be asked has to be decided. Each question corresponds to a specific binary split into two *descendant* nodes. Each node, t, is associated with a specific subset X_t of the training set X. Splitting of a node is equivalent with the split of the subset X_t into two *disjoint* descendant subsets, X_{tY}, X_{tN}. The first of the two consists of the vectors in X_t that correspond to the answer "Yes" of the question and those of the second to the "No." The first (*root*) node of the tree is associated with the training set X. For every split, the following is true:

$$X_{tY} \bigcap X_{tN} = \emptyset$$
$$X_{tY} \bigcup X_{tN} = X_t$$

- A *splitting criterion* must be adopted according to which the best split from the set of candidate ones is chosen.
- A stop-splitting rule is required that controls the growth of the tree and a node is declared as a terminal one (*leaf*.)
- A rule is required that assigns each leaf to a specific class.

We are by now experienced enough to expect that there is more than one method to approach each of the above design elements.

4.18.1 Set of Questions

For the OBCT type of trees the questions are of the form "*Is* $x_k \leq \alpha$*?*" For *each* feature, every possible value of the threshold α defines a specific split of the subset X_t. Thus in theory, an infinite set of questions has to be asked if α varies in an interval $Y_\alpha \subseteq \mathcal{R}$. In practice, only a finite set of questions can be considered. For example, since the number, N, of training points in X is finite, any of the features $x_k, k = 1, \ldots, l$, can take at most $N_t \leq N$ different values, where N_t is the cardinality of the subset $X_t \subseteq X$. Thus, for feature x_k, one can use $\alpha_{kn}, n = 1, 2, \ldots, N_{tk}$ ($N_{tk} \leq N_t$), where α_{kn} are taken halfway between consecutive distinct values of x_k in the training subset X_t. The same has to be repeated for all features. Thus in such a case, the total number of candidate questions is $\sum_{k=1}^{l} N_{tk}$. However, only one of them has to be chosen to provide the binary split at the current node, t, of the tree. This is selected to be the one that leads to the best split of the associated subset X_t. The best split is decided according to a splitting criterion.

4.18.2 Splitting Criterion

Every binary split of a node, t, generates two descendant nodes. Let us denote them by t_Y and t_N according to the "Yes" or "No" answer to the single question adopted for the node t, also referred as the *ancestor* node. As we have already mentioned, the descendant nodes are associated with two new subsets, i.e., X_{tY}, X_{tN}, respectively. In order for the tree growing methodology, from the root node down to the leaves, to make sense, every split must generate subsets that are more "class homogeneous" compared to the ancestor's subset X_t. This means that the training feature vectors in each one of the new subsets show a higher preference for specific class(es), whereas data in X_t are more equally distributed among the classes. As an example, let us consider a four-class task and assume that the vectors in subset X_t are distributed among the classes with equal probability (percentage). If one splits the node so that the points that belong to ω_1, ω_2 classes form the X_{tY} subset, and the points from ω_3, ω_4 classes form the X_{tN} subset, then the new subsets are more homogeneous compared to X_t or "purer" in the decision tree terminology. The goal, therefore, is to define a measure that quantifies node impurity and split the node so that the overall impurity of the descendant nodes is optimally decreased with respect to the ancestor node's impurity.

Let $P(\omega_i|t)$ denotes the probability that a vector in the subset X_t, associated with a node t, belongs to class ω_i, $i = 1, 2, \ldots, M$. A commonly used definition of *node impurity*, denoted as $I(t)$, is given by

$$I(t) = -\sum_{i=1}^{M} P(\omega_i|t) \log_2 P(\omega_i|t) \tag{4.70}$$

where $\log_2$ is the logarithm with base 2. This is nothing else than the entropy associated with the subset X_t, known from Shannon's Information Theory. It is not difficult to show that $I(t)$ takes its maximum value if all probabilities are equal to $\frac{1}{M}$ (highest impurity) and it becomes zero (recall that $0 \log 0 = 0$) if all data belong to a single class, that is, if only one of the $P(\omega_i|t) = 1$ and all the others are zero (least impurity). In practice, probabilities are estimated by the respective percentages, $\frac{N_t^i}{N_t}$, where N_t^i is the number of points in X_t that belong to class ω_i. Assume now that performing a split, N_{tY} points are sent into the "Yes" node (X_{tY}) and N_{tN} into the "No" node (X_{tN}). The *decrease in node impurity* is defined as

$$\Delta I(t) = I(t) - \frac{N_{tY}}{N_t} I(t_Y) - \frac{N_{tN}}{N_t} I(t_N) \tag{4.71}$$

where $I(t_Y)$, $I(t_N)$ are the impurities of the t_Y and t_N nodes respectively. *The goal now becomes to adopt, from the set of candidate questions, the one that performs the split leading to the highest decrease of impurity.*

4.18.3 Stop-Splitting Rule

The natural question that now arises is when one decides to stop splitting a node and declares it as a leaf of the tree. A possibility is to adopt a threshold T and stop splitting if the maximum value of $\Delta I(t)$, over all possible splits, is less than T. Other alternatives are to stop splitting either if the cardinality of the subset X_t is small enough or if X_t is pure, in the sense that all points in it belong to a single class.

4.18.4 Class Assignment Rule

Once a node is declared to be a leaf, then it has to be given a class label. A commonly used rule is the majority rule, i.e., the leaf is labeled as ω_j where

$$j = arg \max_i P(\omega_i|t)$$

In words, we assign a leaf, t, to that class to which the majority of the vectors in X_t belong.

Having discussed the major elements needed for the growth of a decision tree, we are now ready to summarize the basic algorithmic steps for constructing a binary decision tree

- Begin with the root node, i.e., $X_t = X$
- For each new node t

* For every feature x_k, $k = 1, 2, \ldots, l$
  - For every value α_{kn}, $n = 1, 2, \ldots, N_{tk}$
    - Generate X_{tY} and X_{tN} according to the answer in the question: is $x_k(i) \leq \alpha_{kn}$, $i = 1, 2, \ldots, N_t$
    - Compute the impurity decrease
  - End
  - Choose α_{kn_0} leading to the maximum decrease w.r. to x_k
* End
* Choose x_{k_0} and associated $\alpha_{k_0 n_0}$ leading to the overall maximum decrease of impurity
* If stop-splitting rule is met declare node t as a leaf and designate it with a class label
* If not, generate two descendant nodes t_Y and t_N with associated subsets X_{tY} and X_{tN}, depending on the answer to the question: is $x_{k_0} \leq \alpha_{k_0 n_0}$

- End

Remarks

- A variety of node impurity measures can be defined. However, as it is pointed out in [Brei 84], the properties of the resulting final tree seem to be rather insensitive to the choice of the splitting criterion. Nevertheless, this is very much a problem dependent task.
- A critical factor in designing a decision tree is its size. As it was the case with the multilayer perceptrons, the size of a tree must be large enough but not too large; otherwise it tends to learn the particular details of the training set and exhibits poor generalization performance. Experience has shown that the use of a threshold value for the impurity decrease as stop-splitting rule does not lead to trees of the right size. Many times it stops tree growing either too early or too late. The most commonly used approach is to grow a tree up to a large size first and then prune nodes according to a pruning criterion. This is similar in philosophy with pruning multilayer perceptrons. A number of pruning criteria have been suggested in the literature. A commonly used criterion is to combine an estimate of the error probability with a complexity measuring term (e.g., number of terminal nodes). For more on this issue the interested reader may refer to [Brei 84, Ripl 94].
- Our discussion so far was focused on OBCT type of trees. More general partition of the feature space, via hyperplanes not parallel to the axis, is possible via questions of the type: *is* $\sum_{k=1}^{l} c_k x_k \leq \alpha$*?* This can lead to a better partition of the space. However, the training now becomes more involved; see, e.g., [Quin 93].

- Constructions of fuzzy decision trees have also been suggested, by allowing the possibility of partial membership of a feature vector in the nodes that make up the tree structure. Fuzzification is achieved by imposing a fuzzy structure over the basic skeleton of a standard decision tree; see, for example, [Suar 99] and the references therein.

Example 4.1. In a tree classification task, the set X_t, associated with node t, contains $N_t = 10$ vectors. Four of these belong to class ω_1, four to class ω_2, and two to class ω_3, in a three-class classification task. The node splitting results into two new subsets X_{tY}, with three vectors from ω_1, and one from ω_2, and X_{tN} with one vector from ω_1, three from ω_2, and two from ω_3. The goal is to compute the decrease in node impurity after splitting.

We have that

$$I(t) = -\frac{4}{10}\log_2\frac{4}{10} - \frac{4}{10}\log_2\frac{4}{10} - \frac{2}{10}\log_2\frac{2}{10} = 1.521$$

$$I(t_Y) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.815$$

$$I(t_N) = -\frac{1}{6}\log_2\frac{1}{6} - \frac{3}{6}\log_2\frac{3}{6} - \frac{2}{6}\log_2\frac{2}{6} = 1.472$$

Hence, the impurity decrease after splitting is

$$\Delta I(t) = 1.521 - \frac{4}{10}(0.815) - \frac{6}{10}(1.472) = 0.315$$

For further information and a deeper study of decision tree classifiers the interested reader may consult the seminal book [Brei 84]. A nonexhaustive sample of later contributions in the area is [Datt 85, Chou 91, Seth 90, Graj 86, Quin 93]. A recent comparative guide for a number of well-known techniques is provided in [Espo 97].

Finally, it must be stated that there are close similarities between the decision trees and the neural network classifiers. Both aim at forming complex decision boundaries in the feature space. A major difference lies in the way decisions are made. Decision trees employ a hierarchically structured decision function in a sequential fashion. In contrast, neural networks utilize a set of soft (not final) decisions in a parallel fashion.

Furthermore, their training is performed via different philosophies. However, despite their differences, it has been shown that linear tree classifiers (with a linear splitting criterion) can be adequately mapped to a multilayer perceptron structure [Seth 90, Seth 91, Park 94].

So far, from the performance point of view, comparative studies seem to give an advantage to the multilayer perceptrons with respect to the classification error, and an advantage to the decision trees with respect to the required training time [Brow 93].

4.19 DISCUSSION

The current chapter was the third one concerning the classifier design phase. Although we have not exhausted the list (as a matter of fact, a few more cases will be discussed in the chapters to follow), we feel that we have presented to the reader the most popular directions currently used for the design of a classifier.

Another trend that offers more possibilities to the designer is to *combine different classifiers* together. Thus, one can exploit their individual advantages in order to reach an overall better performance than could be achieved by using each of them separately. An important observation that justifies such an approach is the following. From the different (candidate) classifiers that we design in order to choose the one that fits our needs, one results in the best performance, that is, minimum classification error rate. However, different classifiers may fail (to classify correctly) on different patterns. That is, even the "best" classifier can fail on patterns that other classifiers succeed on. Combining classifiers aims at exploiting this complementary information that seems to reside in the various classifiers. Many interesting design issues now come onto the scene. What is the strategy one has to adopt for combining the individual outputs in order to reach the final conclusion? Should one combine the results following the product rule, the sum rule, the min rule, the max rule or the median rule? Should all classifiers be fed with the same feature vectors or must different feature vectors be selected for the different classifiers? For a review of such techniques the reader may refer to [Kitt 98, Tax 00, Mill 99, Jain 00, Witt 88, Kunc 02] and the references therein.

Boosting refers to a specific type of strategies for combining classifiers. At the heart of a boosting method lies a so called "base" classifier. This is a "weak" classifier and it suffices to perform slightly better than a random guessing. A series of classifiers is then designed iteratively, employing, each time, the base classifier but using a different subset of the training set, according to an iteratively computed distribution (or weighting over the samples of the training set). At each iteration, the computed weighting distribution gives emphasis to the "hardest" (incorrectly classified) samples. The final classifier is obtained as an appropriate weighted average (voting) of the previously hierarchically designed classifiers. It turns out that, given a sufficient number of iterations and training samples the classification error of the final combination can be arbitrarily small. This is very impressive indeed. Using a very weak classifier as the base, one can achieve an arbitrarily small error rate, by appropriate exploitation of the behavior of the training samples with respect to the sequence of the designed classifiers [Scha 98]. A deeper look at these classifiers reveals that they are basically "margin" classifiers. That is, at each iteration step they try to maximize the margin of the training samples from the decision surface, and they finally converge to a margin distribution where most examples have large margins, see, e.g., [Maso 00, Scha 98]. From this point of view there is an affinity with the support vectors machines, which also try to

maximize the margin of the training samples from the decision surface, see, for example, [Scha 98, Rats 02].

It is true that the number of available techniques is large and the user has to choose what is more appropriate for the problem at hand. There are no magic recipes. A large research effort has been focused on comparative studies of various classifiers in the context of different applications. One of the most extensive efforts was the Statlog project [Mich 94], in which a wide range of classifiers was tested using a large number of different data sets. Furthermore, research effort has been devoted to unraveling relations and affinities between the different techniques. Many of these techniques have their origin in different scientific disciplines. Therefore, until a few years ago, they were considered independently. Recently researchers have started to recognize underlying similarities among various approaches. For readers who want to dig a bit deeper into these questions, the discussions and results presented in [Chen 94, Ripl 94, Ripl 96, Spec 90, Holm 97, Josh 97, Reyn 99] will be quite enlightening.

In summary, the only tip that can be given to the designer is that all of the techniques presented in this book are still serious players in the classifier design game. The final choice depends on the specific task. *The proof of the cake is in the eating!*

Problems

4.1 We are given 10 feature vectors that originate from two classes ω_1 and ω_2 as follows

$$\omega_1: [0.1, -0.2]^T, [0.2, 0.1]^T, [-0.15, 0.2]^T, [1.1, 0.8]^T, [1.2, 1.1]^T$$
$$\omega_2: [1.1, -0.1]^T, [1.25, 0.15]^T, [0.9, 0.1]^T, [0.1, 1.2]^T, [0.2, 0.9]^T$$

Check whether these are linearly separable and, if not, design an appropriate multilayer perceptron with nodes having step function activation to classify the vectors in the two classes.

4.2 Using the computer, generate four 2-dimensional Gaussian random sequences with covariance matrices

$$\Sigma = \begin{bmatrix} 0.01 & 0.0 \\ 0.0 & 0.01 \end{bmatrix}$$

and mean values $\mu_1 = [0, 0]^T$, $\mu_2 = [1, 1]^T$, $\mu_3 = [0, 1]^T$, $\mu_4 = [1, 0]^T$. The first two form class ω_1 and the other two class ω_2. Produce 100 vectors from each distribution. Use the batch mode backpropagation algorithm of Section 4.6 to train a two-layer perceptron with two hidden neurons and one in the output. Let the activation function be the logistic one with $a = 1$. Plot the error curve as a function of iteration steps. Experiment yourselves with various values of the learning parameter μ. Once the algorithm has converged, produce 50 more vectors from each distribution and try to classify them using the weights you have obtained. What is the percentage classification error?

4.3 Draw the three lines in the two-dimensional space

$$x_1 + x_2 = 0$$
$$x_2 = \frac{1}{4}$$
$$x_1 - x_2 = 0$$

For each of the polyhedra that are formed by their intersections, determine the vertices of the cube into which they will be mapped by the first layer of a multilayer perceptron, realizing the preceding lines. Combine the regions into two classes so that (a) a two-layer network is sufficient to classify them and (b) a three-layer network is necessary. For both cases compute analytically the corresponding synaptic weights.

4.4 Show that if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two points in the l-dimensional space, the hyperplane bisecting the segment with end points $\boldsymbol{x}_1, \boldsymbol{x}_2$, leaving $\boldsymbol{x}_1$ at its positive side, is given by

$$(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \boldsymbol{x} - \frac{1}{2}\|\boldsymbol{x}_1\|^2 + \frac{1}{2}\|\boldsymbol{x}_2\|^2 = 0$$

4.5 For the cross-entropy cost function of (4.33)

- Show that its minimum value for binary desired response values is zero and it occurs when the true outputs are equal to the desired ones.
- Show that the cross-entropy cost function depends on the relative output errors.

4.6 Show that if the cost function, optimized by a multilayer perceptron, is the cross-entropy (4.33) and the activation function is the sigmoid (4.1), then the gradient $\delta_j^L(i)$ of (4.13) becomes

$$\delta_j^L(i) = a(1 - \hat{y}_j(i))y_j(i).$$

4.7 Repeat Problem 4.6 for the softmax activation function and show that $\delta_j^L(i) = \hat{y}_j(i) - y_j(i)$.

4.8 Show that for the cross-entropy cost function (4.30) the outputs of the network, corresponding to the optimal weights, approximate the conditional probabilities $P(\omega_i|\boldsymbol{x})$.

4.9 Using formula (4.37) show that if $l \geq K$ then $M = 2^K$.

4.10 Develop a program to repeat the simulation example of Section 4.10.

4.11 For the same example start with a network consisting of six neurons in the first hidden layer and nine neurons in the second. Use a pruning algorithm to reduce the size of the network.

4.12 Let the sum of error squares

$$J = \frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{k_L}(\hat{y}_m(i) - y_m(i))^2$$

be the minimized function for a multilayer perceptron. Compute the elements of the Hessian matrix

$$\frac{\partial^2 J}{\partial w_{kj}^r \partial w_{k'j'}^{r'}}$$

Show that near a minimum, this can be approximated by

$$\frac{\partial^2 J}{\partial w_{kj}^r \partial w_{k'j'}^{r'}} = \sum_{i=1}^{N}\sum_{m=1}^{k_L}\frac{\partial \hat{y}_m(i)}{\partial w_{kj}^r}\frac{\partial \hat{y}_m(i)}{\partial w_{k'j'}^{r'}}$$

Thus, the second derivatives can be approximated by products of the first-order ones. Following arguments similar to those used for the derivation of the backpropagation algorithm, show that

$$\frac{\partial \hat{y}_m(i)}{\partial w_{kj}^r} = \hat{\delta}_{jm}^r y_k^{r-1}$$

where

$$\hat{\delta}_{jm}^r = \frac{\partial \hat{y}_m(i)}{\partial v_j^r(i)}$$

Its computation takes place recursively in the backpropagation philosophy. This has been used in [Hass 93].

4.13 In Section 4.4 it was pointed out that an approximation to the Hessian matrix, which is often employed in practice, is to assume that it is diagonal. Prove that under this assumption

$$\frac{\partial^2 \mathcal{E}}{(\partial w_{kj}^r)^2}$$

is propagated via a backpropagation concept according to the formulas:

(1) $$\frac{\partial^2 \mathcal{E}}{(\partial w_{kj}^r)^2} = \frac{\partial^2 \mathcal{E}}{(\partial v_j^r)^2}(y_k^{r-1})^2$$

(2) $$\frac{\partial^2 \mathcal{E}}{(\partial v_j^L)^2} = f''(v_j^L)e_j + (f'(v_j^L))^2$$

$$(3)\quad \frac{\partial^2 \mathcal{E}}{\left(\partial v_j^{r-1}\right)^2} = \left(f'\left(v_j^{r-1}\right)\right)^2 \sum_{k=1}^{k_l} \left(w_{kj}^r\right)^2 \frac{\partial^2 \mathcal{E}}{\left(\partial v_j^r\right)^2} + f''\left(v_j^{r-1}\right) \sum_{k=1}^{k_r} w_{kj}^r \delta_k^r$$

where all off-diagonal terms of the Hessian matrix have been neglected and the dependence on i has been suppressed for notational convenience.

4.14 Derive the full Hessian matrix for a simple two-layer network with two hidden neurons and an output one. Generalize to the case of more than two hidden neurons.

4.15 Rederive the backpropagation algorithm of Section 4.6 with activation function

$$f(x) = c \tanh(bx)$$

4.16 In [Dark 91] the following scheme for adaptation of the learning parameter μ has been proposed:

$$\mu = \mu_0 \frac{1}{1 + \frac{t}{t_0}}$$

Verify that, for large enough values of t_0 (e.g., $300 \leq t_0 \leq 500$), the learning parameter is approximately constant for the early stages of training (small values of iteration step t) and decreases in inverse proportion to t for large values. The first phase is called *search phase* and the latter *converge phase*. Comment on the rationale of such a procedure.

4.17 Use a two-layer perceptron with a linear output unit to approximate the function $y(x) = 0.3 + 0.2\cos(2\pi x)$, $x \in [0, 1]$. To this end, generate a sufficient number of data points from this function for the training. Use the backpropagation algorithm in one of its forms to train the network. In the sequel produce 50 more samples, feed them into the trained network, and plot the resulting outputs. How does it compare with the original curve? Repeat the procedure for a different number of hidden units.

4.18 Show Eq. (4.48).
Hint: Show first that the following recursion is true:

$$O(N+1, l) = O(N, l) + O(N, l-1)$$

To this end, start with the N points and add an extra one. Then show that the difference in the dichotomies, as we go from N to $N+1$ points, is due to the dichotomizing hyperplanes (for the N points case) that could have been drawn via this new point.

4.19 Show that if $N = 2(l+1)$ the number of dichotomies is given by 2^{N-1}.
Hint: Use the identity that

$$\sum_{i=0}^{J} \binom{J}{i} = 2^J$$

and recall that

$$\binom{2n+1}{n-i+1} = \binom{2n+1}{n+i}$$

4.20 Repeat the experiments of Problem 4.17 using an RBF network. Select the k centers regularly spaced within $[0, 1]$. Repeat the experiments with different number of Gaussian functions and σ. Estimate the unknown weights using the least squares method.

4.21 Using your experience from the previous problem, repeat the procedure for the two-dimensional function

$$y(x_1, x_2) = 0.3 + 0.2\cos(2\pi x_1)\cos(2\pi x_2).$$

4.22 Let the mapping from the input space to a higher dimensional space be

$$x \in \mathcal{R} \longrightarrow y \equiv \boldsymbol{\phi}(x) \in \mathcal{R}^{2k+1}$$

where

$$\boldsymbol{\phi}(x) = \left[\frac{1}{\sqrt{2}}, \cos x, \cos 2x, \ldots, \cos kx, \sin x, \sin 2x, \ldots, \sin kx\right]^T$$

Then show that the corresponding inner product kernel is

$$\begin{aligned} y_i^T y_j &= K(x_i, x_j) \\ &= \frac{\sin\left(\left(k+\frac{1}{2}\right)(x_i - x_j)\right)}{2\sin\left(\frac{x_i - x_j}{2}\right)} \end{aligned}$$

References

[Adal 97] Adali T., Liu X., Sonmez K. "Conditional distribution learning with neural networks and its application to channel equalization," *IEEE Transactions on Signal Processing*, Vol. 45(4), pp. 1051–1064, 1997.

[Barn 92] Barnard E. "Optimization for training neural networks," *IEEE Transactions on Neural Networks*, Vol. 3(2), pp. 232–240, 1992.

[Barr 93] Barron A.R. "Universal approximation bounds for superposition of a sigmoid function," *IEEE Transactions on Information Theory*, Vol. 39(3), pp. 930–945, 1993.

[Batt 92] Battiti R. "First and second order methods for learning: Between steepest descent and Newton's method," *Neural Computation*, Vol. 4, pp. 141–166, 1992.

[Baum 88] Baum E.B., Wilczek F. "Supervised learning of probability distributions by neural networks," in *Neural Information Processing Systems* (Anderson D., ed.), pp. 52–61, American Institute of Physics, New York, 1988.

[Ben 99] Ben-Yakoub S., Abdeljaoued, Mayoraj E. "Fusion of face and speech data for person identity verification," *IEEE Transactions on Neural Networks*, Vol. 10(5), pp. 1065–1075, 1999.

[Bish 95] Bishop C.M. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[Blan 96] Blanz V., Schölkopf B., Bülthoff H., Burges C.,Vapnik V., Vetter T. "Comparison of view based object recognition using realistic 3D models," *Proceedings of International Conference on Artificial Neural Networks*, pp. 251–256, Berlin, 1996.

[Bose 92] Bose B.E., Guyon I.M., Vapnik, V.N. "A training algorithm for optimal margin classifiers," *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pp. 144–152, Morgan Kaufman, San Mateo, CA; 1992.

[Bose 96] Bose N.K., Liang P. *Neural Network Fundamentals with Graphs, Algorithms and Applications*, McGraw-Hill, 1996.

[Brei 84] Breiman L., Friedman J., Olshen R., Stone C., *Classification and Regression Trees*, Wadsworth International, pp. 226–239, Belmont, CA, 1984.

[Brid 90] Bridle J.S. "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation parameters," in *Neural Information Processing Systems 2* (Touretzky D.S., ed.), pp. 211–217, Morgan Kaufmann, San Mateo, CA, 1990.

[Brow 93] Brown D., Corrnble V., Pittard C.L. "A comparison of decision tree classifiers with backpropagation neural networks for multimodal classification problems," *Pattern Recognition*, Vol. 26(6), pp. 953–961, 1993.

[Broo 88] Broomhead D.S., Lowe D. "Multivariable functional interpolation and adaptive networks," *Complex Systems*, Vol. 2, pp. 321–355, 1988.

[Burg 97] Burges C.J.C., Schölkoff B. "Improving the accuracy and speed of support vectors learning machines," in *Advances in Neural Information Processing Systems 9* (Mozer M, Jordan M., Petsche T., eds.), pp. 375–381, MIT Press, Cambridge, MA, 1997.

[Chan 00] Chang, C.C., Hsu C.W., Lin C.J. "The analysis of decomposition methods for SVM," *IEEE Transactions on Neural Networks*, Vol. 11(4), pp. 1003–1008, 2000.

[Chau 90] Chauvin Y. "Generalization performance of overtrained backpropagation networks," in *Neural Networks. Proc. EURASIP Workshop* (Almeida L.B., Wellekens C.J., eds.), pp. 46–55, 1990.

[Chen 91] Chen S., Cowan C.F.N., Grant P.M. "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, Vol. 2, pp. 302–309, 1991.

[Chen 94] Cheng B., Titterington D.M. "Neural networks: A review from a statistical perspective," *Statistical Science*, Vol. 9(1), pp. 2–30, 1994.

[Chou 91] Chou P. "Optimal partitioning for classification and regression trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, pp. 340–354, 1991.

[Cich 93] Cichocki A., Unbehauen R. *Neural Networks for Optimization and Signal Processing*, John Wiley, 1993.

[Cort 95] Cortes C., Vapnik V.N. "Support vector networks," *Machine Learning*, Vol. 20, pp. 273–297, 1995.

[Cour 53] Courant R., Hilbert D. *Methods of Mathematical Physics*, Interscience, 1953.

[Cove 65] Cover T.M. "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Transactions on Electronic Computers*, Vol. 14, pp. 326–334, 1965.

[Cybe 89] Cybenko G. "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, Vol. 2, pp. 304–314, 1989.

[Dark 91] Darken C., Moody J. "Towards faster stochastic gradient search," *Advances in Neural Information Processing Systems 4*, pp. 1009–1016, Morgan Kaufmann, San Mateo, CA, 1991.

[Datt 85] Dattatreya G.R., Kanal L.N. "Decision trees in pattern recognition," in *Progress in Pattern Recognition 2* (Kanal L.N., Rosenfeld A., eds.), North Holland, 1985.

[Delo 94] Delopoulos A., Tirakis A., Kollias S. "Invariant image classification using triple correlation based neural networks," *IEEE Transactions on Neural Networks*, Vol. 5, pp. 392–408, 1994.

[Druc 99] Drucker H., Wu D., Vapnik, V.N. "Support vector machines for spam categorization," *IEEE Transactions on Neural Networks*, Vol. 10(5), pp. 1048–1055, 1999.

[Espo 97] Esposito F., Malebra D., Semeraro G. "A comparative analysis of methods for pruning decision trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(5), pp. 476–491, 1997.

[Fahl 90] Fahlman S.E., Lebiere C. "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2* (Touretzky D.S., ed.), pp. 524–532, Morgan Kaufmann, San Mateo, CA, 1990.

[Frea 90] Frean M. "The Upstart algorithm: A method for constructing and training feedforward networks," *Neural Computation*, Vol. 2(2), pp. 198–209, 1990.

[Fuku 82] Fukushima K., Miyake S., "Neogognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position," *Pattern Recognition*, Vol. 15(6), pp. 445–469, 1982.

[Fuku 92] Fukumi M., Omatu S., Takeda F., Kosaka T. "Rotational invariant neural pattern recognition system with application to coin recognition," *IEEE Transactions on Neural Networks*, Vol. 3, pp. 272–279, 1992.

[Funa 89] Funahashi K. "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, Vol. 2(3), pp. 183–192, 1989.

[Gall 90] Gallant S.I. "Perceptron-based learning algorithms," *IEEE Transactions on Neural Networks*, Vol. 1, pp. 179–191, 1990.

[Gibs 90] Gibson G.J., Cowan C.F.N. "On the decision regions of multilayer perceptrons," *Proceedings of the IEEE*, Vol. 78(10), pp. 1590–1594, 1990.

[Gish 90] Gish H. "A probabilistic approach to the understanding and training neural classifiers," in *Proceedings of the IEEE Conference on Acoustics Speech and Signal Processing*, pp. 1361–1364, April 1990.

[Gomm 00] Gomm J.B., Yu D.L. "Selecting radial basis function network centers with recursive orthogonal least squares training," *IEEE Transactions on Neural Networks*, Vol. 11(2), pp. 306–314, 2000.

[Graj 86] Grajki K.A. et al. "Classification of EEG spatial patterns with a tree structured methodology," *IEEE 3Transactions on Biomedical Engineering*, Vol. 33(12), pp. 1076–1086, 1986.

[Hamp 90] Hampshire J.B. II, Perlmutter B.A. "Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function," in *Proceedings of the 1990 Connectionist Models Summer School* (Touretzky D, et al., eds.), Morgan Kaufmann, San Mateo, CA, 1990.

[Hart 90] Hartman E.J., Keeler J.D., Kowalski J.M. "Layered neural networks with Gaussian hidden units as universal approximations," *Neural Computations*, Vol. 2(2), pp. 210–215, 1990.

[Hass 93] Hassibi B., Stork D.G., Wolff G.J. "Optimal brain surgeon and general network pruning," *Proceedings IEEE Conference on Neural Networks*, Vol. 1, pp. 293–299, San Francisco, 1993.

[Hayk 96] Haykin S. "Neural networks expand SP's horizons," *IEEE Signal Processing Magazine*, Vol. 13(2), pp. 24–49, 1996.

[Hayk 99] Haykin S. *Neural Networks: A Comprehensive Foundation*, 2nd edition, Prentice Hall, 1999.

[Hert 91] Hertz J., Krogh A., Palmer R.G. *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991.

[Hint 90] Hinton G.E. "Connectionist learning procedures," in *Machine Learning: Paradigms and Methods* (Carbonell J.G., ed.), pp. 185–234, MIT Press, Cambridge, MA, 1990.

[Holm 97] Holmstrom L., Koistinen P., Laaksonen J., Oja E. "Neural and statistical classifiers—taxonomy and two case studies," *IEEE Transactions on Neural Networks*, Vol. 8(1), pp. 5–17, 1997.

[Horn 89] Hornik K., Stinchcombe M., White H. "Multilayer feedforward networks are universal approximators," *Neural Networks*, Vol. 2(5), pp. 359–366, 1989.

[Hush 93] Hush D.R., Horne B.G. "Progress in supervised neural networks," *IEEE Signal Processing Magazine*, Vol. 10(1), pp. 8–39, 1993.

[Ito 91] Ito Y. "Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory," *Neural Networks*, Vol. 4(3), pp. 385–394, 1991.

[Jain 00] Jain A.K., Duin P.W., Mao J. "Statistical pattern recognition: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22(1), pp. 4–37, 2000.

[Jaco 88] Jacobs R.A. "Increased rates of convergence through learning rate of adaptation," *Neural Networks*, Vol. 2, pp. 359–366, 1988.

[Joha 92] Johansson E.M., Dowla F.U., Goodman D.M. "Backpropagation learning for multilayer feedforward neural networks using conjugate gradient method," *International Journal of Neural Systems*, Vol. 2(4), pp. 291–301, 1992.

[Josh 97] Joshi A., Ramakrishman N., Houstis E.N., Rice J.R. "On neurobiological, neuro-fuzzy, machine learning, and statistical pattern recognition techniques," *IEEE Transactions on Neural Networks*, Vol. 8(1), pp. 18–31, 1997.

[Juan 92] Juang B.H., Katagiri S. "Discriminative learning for minimum error classification," *IEEE Transactions on Signal Processing*, Vol. 40(12), pp. 3043–3054, 1992.

[Kalo 97] Kalouptsidis N. *Signal Processing Systems, Theory and Design*, John Wiley, 1997.

[Kana 92] Kanaoka T., Chellapa R., Yoshitaka M., Tomita S. "A higher order neural network for distortion invariant pattern recognition," *Pattern Recognition Letters*, Vol. 13, pp. 837–841, 1992.

[Kara 92] Karayiannis N.B., Venetsanopoulos A.N. "Fast learning algorithm for neural networks," *IEEE Transactions on Circuits and Systems*, Vol. 39, pp. 453–474, 1992.

[Kara 97] Karayiannis N.B., Mi G.W. "Growing radial basis neural networks. Merging supervised and unsupervised learning with network growth techniques," *IEEE Transactions on Neural Networks*, Vol. 8(6), pp. 1492–1506, 1997.

[Kitt 98] Kittler J., Hatef M., Duin R. Matas J. "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20(3) 1998.

[Koli 97] Kolinummi P., Hamalainen T., Kaski K. "Designing a digital neurocomputer," *IEEE Circuits and Systems Magazine*, Vol. 13(2), pp. 19–27, 1997.

[Kout 94] Koutroumbas K., Kalouptsidis N. "Nearest neighbor pattern classification neural networks," *Proc. of IEEE World Congress of Computational Intelligence*, pp. 2911–2915, Orlando, FL, July 1994.

[Kram 89] Kramer A.H., Sangiovanni-Vincentelli A. "Efficient parallel learning algorithms for neural networks," in *Advances in Neural Information Processing Systems I* (Touretzky D.S., ed.), pp. 40–48, Morgan Kaufmann, San Mateo, CA, 1989.

[Kunc 02] Kunchera L.I. "A theoretical study on six classifier fusion strategies," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24(2), pp. 281–286, 2002.

[Lane 91] Lane S.H., Flax M.G., Handelman D.A., Gelfand J.J. "Multilayer perceptrons with B-spline receptive field functions," in *Advances in Neural Information Processing Systems 3* (Lippmann R.P., Moody J., Touretzky D.S., eds.), pp. 684–692, Morgan Kaufmann, San Mateo, CA, 1991.

[Lecu 89] Le Cun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D. "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, Vol. 1(4), pp. 541–551, 1989.

[Lecu 90] Le Cun Y., Denker J.S., Solla S.A. "Optimal brain damage," in *Advances in Neural Information Systems 2* (Touretzky D.S., ed.), pp. 598–605, Morgan Kaufmann, San Mateo, CA, 1990.

[Lipp 87] Lippmann R.P. "An introduction to computing with neural networks," *IEEE ASSP Magazine*, Vol. 4(2), pp. 4–22, 1987.

[Mack 92a] MacKay D.J.C. "A practical Bayesian framework for backpropagation networks," *Neural Computation*, Vol. 4(3), pp. 448–472, 1992.

[Mack 92b] MacKay D.J.C. "The evidence framework applied to classification networks," *Neural Computation*, Vol. 4(5), pp. 720–736, 1992.

[Maso 00] Mason L., Baxter J., Bartleet P., Frean M. "Boosting algorithms as gradient descent," in *Neural Information Processing Systems*, Vol. 12, 2000.

[Matt 99] Mattera D., Palmieri F., Haykin S. "An explicit algorithm for training support vector machines," *IEEE Signal Processing Letters*, Vol. 6(9), pp. 243–246. 1999.

[Meza 89] Mezard M., Nadal J.P. "Learning in feedforward layered networks: The tilling algorithm," *Journal of Physics*, Vol. A 22, pp. 2191–2203, 1989.

[Mich 94] Michie D., Spiegelhalter D.J., Taylor C.C., eds. *Machine Learning, Neural, and Statistical Classification*, Ellis Horwood Ltd., London, 1994.

[Mill 99] Miller D.J., Yan L. "Critic-driven ensemble classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 47(10), pp. 2833–2844, 1999.

[Mill 96] Miller D., Rao A., Rose K., Gersho A. "A global optimization technique for statistical classifier design," *IEEE Transactions on Signal Processing*, Vol. 44(12), pp. 3108–3122, 1996.

[Mirc 89] Mirchandini G., Cao W. "On hidden nodes in neural nets," *IEEE Transactions on Circuits and Systems*, Vol. 36(5), pp. 661–664, 1989.

[Mood 89] Moody J., Darken C.J. "Fast learning in networks of locally tuned processing units," *Neural Computation*, Vol. 6(4), pp. 281–294, 1989.

[Mulg 96] Mulgrew B. "Applying radial basis functions," *IEEE Signal Processing Magazine*, Vol. 13(2), pp. 50–65, 1996.

[Mull 01] Müller R.M., Mika S., Rätsch G., Tsuda K., Schölkopf B. "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, Vol. 12(2), pp. 181–201, 2001.

[Murp 90] Murphy O.J. "Nearest neighbor pattern classification perceptrons," *Proceedings of the IEEE*, Vol. 78(10), October 1990.

[Navi 01] Navia-Vasquez A., Perez-Cuz F., Artes-RodriguezA., Figueiras-Vidal A. "Weighted least squares training of support vector classifiers leading to compact and adaptive schemes," *IEEE Transcations on Neural Networks*, Vol. 12(5), pp. 1047–1059, 2001.

[Nede 93] Nedeljkovic V. "A novel multilayer neural networks training algorithm that minimizes the probability of classification error," *IEEE Transactions on Neural Networks*, Vol. 4(4), pp. 650–659, 1993.

[Osun 97] Osuna E., Freund R., Girosi F., "An improved training algorithm for support vector machines," *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pp. 276–285, Amelia Island, FL, 1997.

[Pado 95] Pados D.A., Papantoni-Kazakos P. "New non least squares neural network learning algorithms for hypothesis testing," *IEEE Transactions on Neural Networks*, Vol. 6, pp. 596–609, 1995.

[Palm 91] Palmieri F., Datum M., Shah A., Moiseff A. "Sound localization with a neural network trained with the multiple extended Kalman algorithm," *International Joint Conference on Neural Networks*, Vol. 1, pp. 125–131, Seattle, 1991.

[Pare 00] Parekh R., Yang J., Honavar V. "Constructive neural network learning algorithms for pattern classification,"*IEEE Transactions on Neural Networks*, Vol. 11(2), pp. 436–451, 2000.

[Park 91] Park J., Sandberg I.W. "Universal approximation using radial basis function networks," *Neural Computation*, Vol. 3(2), pp. 246–257, 1991.

[Park 93] Park J., Sandberg I.W. "Approximation and radial basis function networks," *Neural Computation*, Vol. 5(2), pp. 305–316, 1993.

[Park 94] Park Y. "A comparison of neural net classifiers and linear tree classifiers: Their similaritics and differences," *Pattern Recognition*, Vol. 27(11), pp. 1493–1503, 1994.

[Pera 92] Perantonis S.J., Lisboa P.J.G. "Translation, rotation, and scale invariant pattern recognition by high-order neural networks and moment classifiers," *IEEE Transactions on Neural Networks*, Vol. 3(2), pp. 241–251, 1992.

[Plat 91] Platt J. "A resource allocating network for function interpolation," *Neural Computation*, Vol. 3, pp. 213–225, 1991.

[Plat 99] Platt J. "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Learning*, (Scholkopf B., Burges C.J.C., Smola A.J. Eds), pp. 185–208, MIT Press, 1999.

[Quin 93] Quinlan J.R. *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

[Rats 02] Rätsch G., Mika S., Schölkopf B., Müller K.R. "Constructing boosting algorithms from SVMS: an application to one class classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24(9), pp. 1184–1199, 2002.

[Reyn 99] Reyneri L. "Unification of neural and wavelet networks and fuzzy systems," *IEEE Transactions on Neural Networks*, Vol. 10(4), pp. 801–814, 1999.

[Refe 91] Refenes A., Chen L. "Analysis of methods for optimal network construction," *University College London Report*, CC30/080:DCN, 1991.

[Rich 91] Richard M., Lippmann R.P. "Neural network classifiers estimate Bayesian a posteriori probabilities," *Neural Computation*, Vol. 3, pp. 461–483, 1991.

[Rico 88] Ricotti L.P., Ragazinni S., Martinelli G. "Learning of word stress in a suboptimal second order backpropagation neural network," in *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 1, pp. 355–361, San Diego, 1988.

[Ride 97] Ridella S., Rovetta S., Zunino R. "Circular backpropagation networks for classification," *IEEE Transactions on Neural Networks*, Vol. 8(1), pp. 84–97, 1997.

[Ried 93] Riedmiller M., Brau H. "A direct adaptive method for faster backpropagation learning: the rprop algorithm," *Proceedings of the IEEE Conference on Neural Networks*, San Francisco, 1993.

[Ripl 94] Ripley B.D. "Neural networks and related methods for classification," *Journal of Royal Statistical Society*, Vol. B, 56(3), pp. 409–456, 1994.

[Ripl 96] Ripley B.D. *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.

[Rume 86] Rumelhart D.E., Hinton G.E., Williams R.J. "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructures of Cognition* (Rumelhart D.E., McClelland J.L., eds.), Vol. 1, pp. 318–362, MIT Press, 1986.

[Russ 93] Russell R. "Pruning algorithms. A survey," *IEEE Transactions on Neural Networks*, Vol. 4(5), pp. 740–747, 1993.

[Scha 98] Schapire R.E., Freund V., Bartlett P., Lee W.S. "Boosting the margin: a new explanation for the effectiveness of voting methods," *The Annals of Statistics*, Vol. 26(5), pp. 1651–1686, 1998.

[Scho 97] Schölkopf B., Sung K.-K., Burges C.J.C., Girosi F., Niyogi P., Poggio T., Vapnic V. "Comparing support vector machines with Gaussian kernels to RBF classifiers," *IEEE Transactions on Signal Processing*, Vol. 45(11), pp. 2758–2766, 1997.

[Seba 00] Sebald D.J., Bucklew J.A. "Support vector machine techniques for nonlinear equalization," *IEEE Transactions on Signal Processing*, Vol. 48(11), pp. 3217–3227, 2000.

[Seth 90] Sethi I.K. "Entropy nets: From decision trees to neural networks," *Proceedings of the IEEE*, Vol. 78, pp. 1605–1613, 1990.

[Seth 91] Sethi I.K. "Decision tree performance enhancement using an artificial neural network interpretation," in *Artificial Neural Networks and Statistical Pattern Recognition* (Sethi I., Jain A., eds.), Elsevier Science Publishers, 1991.

[Siet 91] Sietsma J., Dow R.J.F. "Creating artificial neural networks that generalize," *Neural Networks*, Vol. 4, pp. 67–79, 1991.

[Silv 90] Silva F.M., Almeida L.B. "Accelaration technique for the backpropagation algorithm," *Proceedings of the EURASIP Workshop on Neural Networks* (Almeida L.B. et al., eds.), pp. 110–119, Portugal, 1990.

[Sing 89] Singhal S., Wu L. "Training feedforward networks with the extended Kalman filter," *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, pp. 1187–1190, Glasgow, 1989.

[Spec 90] Specht D. "Probabilistic neural networks," *Neural Networks*, Vol. 3, pp. 109–118, 1990.

[Suar 99] Suarez A., Lutsko J.F. "Globally optimal fuzzy decision trees for classification and regression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21(12), pp. 1297–1311, 1999.

[Tax 00] Tax D.M.J., Breukelen M., Duin R.P.W., Kittler J. "Combining multiple classifiers by averaging or by multiplying?"*Pattern Recognition*, Vol. 33, pp. 1475–1485, 2000.

[Theod 95] Theodoridis S., Cowan C.F.N., Callender C., Lee C.M.S. "Schemes for equalization in communication channels with nonlinear impairments," *IEE Proceedings on Communications*, Vol. 61(3), pp. 268–278, 1995.

[Watr 88] Watrous R.L. "Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 2, pp. 619–627, San Diego, 1988.

[Wein 90] Weigend A.S., Rumelhart D.E., Huberman B.A. "Backpropagation, weight elimination and time series prediction," in *Proceedings of the Connectionist Models Summer School* (Touretzky D., Elman J., Sejnowski T., Hinton G., eds.), pp. 105–116, 1990.

[Werb 74] Werbos P.J. "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. Thesis, Harvard University, Cambridge, MA, 1974.

[Wett 92] Wettschereck D., Dietterich T. "Improving the performance of radial basis function networks by learning center locations," in *Advances in Neural Information Processing Systems 4* (Moody J.E., Hanson S.J., Lippmann R.P., eds.), pp. 1133–1140, Morgan Kaufmann, San Mateo, CA, 1992.

[Witt 00] Witten I., Frank E. *Data Mining: Practical Machine Learning Tools and Techniques with JAVA Implementations*, Morgan Kaufmann Publishers, 2000.

[Witt 88] Wittner B.S., Denker J.S. "Strategies for teaching layered networks classification tasks," in *Neural Information Processing Systems* (Anderson D.Z., ed.), pp. 850–859, American Institute of Physics, New York, 1988.

[Ying 98] Yingwei L., Sundararajan N., Saratihandran P. "Performance evaluation of a sequential minimal RBF neural network learning algorithm," *IEEE Transactions on Neural Networks*, Vol. 9(2), pp. 308–318, 1998.

[Zura 92] Zurada J. *Introduction to Artificial Neural Networks*, West Publishing Company, St. Paul, MN., 1992.

CHAPTER 5

FEATURE SELECTION

5.1 INTRODUCTION

In all previous chapters, we considered the features to be available prior to the design of the classifier. The goal of this chapter is to study methodologies related to the selection of these variables. As we pointed out very early in the book, a major problem associated with pattern recognition is the so-called curse of dimensionality (Section 2.5.6). The number of features at the disposal of the designer of a classification system is usually very large. As we will see in Chapter 7, this number can easily become of the order of a few dozens or even hundreds. There is more than one reason for the necessity to reduce the number of features to a sufficient minimum. Computational complexity is the obvious one. A related reason is that although two features may carry good classification information when treated separately, there is little gain if they are combined together in a feature vector, because of a high mutual correlation. Thus, complexity increases without much gain. Another major reason is that imposed by the required generalization properties of the classifier, as discussed in Section 4.9 of the previous chapter. According to the discussion there and as we will state more formally at the end of this chapter, the higher the ratio of the number of training patterns N to the number of free classifier parameters, the better the generalization properties of the resulting classifier. A large number of features is directly translated into a large number of classifier parameters (e.g., synaptic weights in a neural network, weights in a linear classifier). Thus, for a finite and usually limited number N of training patterns, keeping the number of features as small as possible is in line with our desire to design classifiers with good generalization capabilities. Furthermore, the ratio N/l enters the scene from another nearby corner. One important step in the design of a classification system is the performance evaluation stage, in which the classification error probability of the designed classifier is estimated. We not only need to design a classification system, we must also assess its performance. As is pointed out in Chapter 10, the classification error estimate improves as this ratio becomes higher. In [Fine 83] it is pointed out that in some cases ratios as high as 10 to 20 were considered necessary.

The major task of this chapter can now be summarized as follows. *Given a number of features, how can one select the most important of them so as to reduce their*

number and at the same time retain as much as possible of their class discriminatory information? The procedure is known as *feature selection* or *reduction*. It must be emphasized that this step is very crucial. If we selected features with little discrimination power, the subsequent design of a classifier would lead to poor performance. On the other hand, if information-rich features are selected, the design of the classifier can be greatly simplified. In a more quantitative description, we should aim to select features leading to *large between-class distance and small within-class variance* in the feature vector space. This means that features should take distant values in the different classes and closely located values in the same class. To this end, different scenarios will be adopted. One is to examine the features individually and discard those with little discriminatory capability. A better alternative is to examine them in combinations. Sometimes the application of a linear or nonlinear transformation to a feature vector may lead to a new one with better discriminatory properties. All these paths will be our touring directions in this chapter.

Finally, it must be pointed out that there is some confusion in the literature concerning the terminology of this stage. In some texts the term feature extraction is used, but we feel that this may be confused with the feature generation stage treated in Chapter 7. Others prefer to call it a preprocessing stage. We have kept the latter term to describe the processing performed on the features prior to their utilization. Such processing involves outlier removal, scaling of the features to safeguard comparable dynamic range of their respective values, treating missing data, and so forth.

5.2 PREPROCESSING

5.2.1 Outlier Removal

An *outlier* is defined as a point that lies very far from the mean of the corresponding random variable. This distance is measured with respect to a given threshold, usually a number of times the standard deviation. For a normally distributed random variable a distance of two times the standard deviation covers 95% of the points, and a distance of three times the standard deviation covers 99% of the points. Points with values very different from the mean value produce large errors during training and may have disastrous effects. These effects are even worse when the outliers are the result of noisy measurements. If the number of outliers is very small, they are usually discarded. However, if this is not the case and they are the result of a distribution with long tails, then the designer may have to adopt cost functions that are not very sensitive in the presence of outliers. For example, the least squares criterion is very sensitive to outliers, because large errors dominate the cost function due to the squaring of the terms. A review of related techniques that attempt to address such problems is given in [Hube 81].

5.2.2 Data Normalization

In many practical situations a designer is confronted with features whose values lie within different dynamic ranges. Thus, features with large values may have a larger influence in the cost function than features with small values, although *this does not necessarily reflect their respective significance in the design of the classifier.* The problem is overcome by normalizing the features so that their values lie within similar ranges. A straightforward technique is normalization via the respective estimates of the mean and variance. For N available data of the kth feature we have

$$\bar{x}_k = \frac{1}{N}\sum_{i=1}^{N} x_{ik}, \qquad k = 1, 2, \ldots, l$$

$$\sigma_k^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_{ik} - \bar{x}_k)^2$$

$$\hat{x}_{ik} = \frac{x_{ik} - \bar{x}_k}{\sigma_k}$$

In words, all the resulting normalized features will now have zero mean and unit variance. This is obviously a linear method. Other linear techniques limit the feature values in the range of $[0, 1]$ or $[-1, 1]$ by proper scaling. Besides the linear methods, nonlinear methods can also be employed in cases in which the data are not evenly distributed around the mean. In such cases transformations based on nonlinear (i.e., logarithmic or sigmoid) functions can be used to map data within specified intervals. The so-called softmax scaling is a popular candidate. It consists of two steps

$$y = \frac{x_{ik} - \bar{x}_k}{r\sigma_k}, \qquad \hat{x}_{ik} = \frac{1}{1 + \exp(-y)} \tag{5.1}$$

This is basically a squashing function limiting data in the range of $[0, 1]$. Using a series expansion approximation, it is not difficult to see that for small values of y this is an approximately linear function with respect to x_{ik}. The range of values of x_{ik} that correspond to the linear section depends on the standard deviation and the factor r, which is user defined. Values away from the mean are squashed exponentially.

5.2.3 Missing Data

In practice, it may happen that the number of available data is not the same for all features. If the number of training data is high enough, we can afford to discard some of them and keep a smaller number, the same for all features, in order to form the feature vectors. However, in many cases it is a luxury to drop available data. In

these cases missing data have to be predicted heuristically. An obvious thought is to replace missing values with the corresponding mean, computed from the available values of the respective features. More elaborate techniques, exploiting the statistical information about the underlying distribution, have also been proposed and used [Ghah 94, Lowe 90]. We will return to the missing data problem in Chapter 11.

5.3 FEATURE SELECTION BASED ON STATISTICAL HYPOTHESIS TESTING

A first step in feature selection is to look at each of the generated features *independently* and test their discriminatory capability for the problem at hand. Although looking at the features independently is far from optimal, this procedure helps us to discard easily recognizable "bad" choices and keeps the more elaborate techniques, which we will consider next, from unnecessary computational burden.

Let x be the random variable representing a specific feature. We will try to investigate whether the values it takes for the different classes, say ω_1, ω_2, *differ significantly*. To give an answer to this question we will formulate the problem in the context of statistical *hypothesis testing*. That is, we will try to answer which of the following hypotheses is correct:

H_1: The values of the feature differ significantly
H_0: The values of the feature do not differ significantly

H_0 is known as the *null hypothesis* and H_1 as the *alternative hypothesis*. The decision is reached on the basis of *experimental evidence* supporting the rejection or not of H_0. This is accomplished by exploiting statistical information, and obviously any decision will be taken subject to an error probability. We will approach the problem by considering the differences of the mean values corresponding to a specific feature in the various classes, and we will test whether these differences are significantly different from zero. Let us first, however, refresh our memory with some basics from the statistics related to hypothesis testing.

5.3.1 Hypothesis Testing Basics

Let x be a random variable with a probability density function, which is assumed to be known within an unknown parameter θ. As we have already seen in Chapter 2, in the case of a Gaussian this parameter may be the mean value or its variance. Our interest here lies in the following hypothesis test:

$$H_1: \quad \theta \neq \theta_0$$
$$H_0: \quad \theta = \theta_0$$

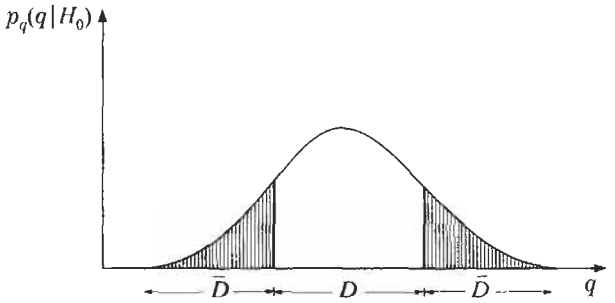


FIGURE 5.1: Acceptance and critical regions for hypothesis testing. The area of the shaded region is the probability of an erroneous decision.

The decision on this test is reached in the following context. Let x_i, $i = 1, 2, \ldots, N$, be the experimental samples of the random variable x. A function $f(\cdot, \ldots, \cdot)$ is selected, depending on the specific problem, and let $q = f(x_1, x_2, \ldots, x_N)$. The function is selected so that the probability density function of q is easily parameterized in terms of the unknown θ, that is, $p_q(q; \theta)$. Let D be the interval of q in which it has a high probability of lying *under hypothesis* H_0. Let $\bar{D}$ be its complement, that is, the interval of low probability, also under hypothesis H_0. Then, if the value of q that results from the available samples, x_i, $i = 1, 2, \ldots, N$, lies in D we will accept H_0, and if it lies in $\bar{D}$ we will reject it. D is known as the *acceptance interval* and $\bar{D}$ as the *critical interval*. The variable q is known as *test statistic*. The obvious question now refers to the probability of reaching a wrong decision. Let H_0 be true. Then the probability of an error in our decision is $P(q \in \bar{D}|H_0) \equiv \rho$. This probability is obviously the integral of $p_q(q|H_0)(p_q(q; \theta_o))$ over $\bar{D}$ (Figure 5.1). In practice, we preselect this value of ρ, which is known as the *significance level*, and we sometimes denote the corresponding critical (acceptance) interval as $\bar{D}_\rho(D_\rho)$. Let us now apply this procedure in the case in which the unknown parameter is the mean of x.

The Known Variance Case

Let x be a random variable and x_i, $i = 1, 2, \ldots, N$, the resulting experimental samples, which we will assume to be *mutually independent*. Let

$$E[x] = \mu$$

$$E[(x - \mu)^2] = \sigma^2$$

A popular estimate of μ based on the known samples is the *sample mean*

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Using a different set of N samples, a different estimate will result. Thus, $\bar{x}$ is also a random variable and it is described in terms of a probability density function $p_{\bar{x}}(\bar{x})$. The corresponding mean is

$$E[\bar{x}] = \frac{1}{N}E\left[\sum_{i=1}^{N} x_i\right] = \frac{1}{N}\sum_{i=1}^{N} E[x_i] = \mu \tag{5.2}$$

Thus, $\bar{x}$ is an *unbiased* estimate of the mean μ of x. The variance $\sigma_{\bar{x}}^2$ of $\bar{x}$ is

$$\begin{aligned} E[(\bar{x}-\mu)^2] &= E\left[\left(\frac{1}{N}\sum_{i=1}^{N} x_i - \mu\right)^2\right] \\ &= \frac{1}{N^2}\sum_{i=1}^{N} E[(x_i-\mu)^2] + \frac{1}{N^2}\sum_{i}\sum_{j\neq i} E[(x_i-\mu)(x_j-\mu)] \end{aligned}$$

The statistical independence of the samples dictates

$$E[(x_i-\mu)(x_j-\mu)] = E[x_i-\mu]E[x_j-\mu] = 0$$

Hence

$$\sigma_{\bar{x}}^2 = \frac{1}{N}\sigma^2 \tag{5.3}$$

In words, the larger the number of measurement samples, the smaller the variance of $\bar{x}$ around the true mean μ.

Having now acquired the necessary ingredients, let us assume that we are given a value $\hat{\mu}$ and we have to decide upon

$$\begin{aligned} H_1&: \quad E[x] \neq \hat{\mu} \\ H_0&: \quad E[x] = \hat{\mu} \end{aligned}$$

To this end we define the test statistic

$$q = \frac{\bar{x}-\hat{\mu}}{\sigma/\sqrt{N}} \tag{5.4}$$

Table 5.1: Acceptance intervals $[-x_\rho, x_\rho]$ corresponding to various probabilities for an $\mathcal{N}(0, 1)$ normal distribution

$1-\rho$	0.8	0.85	0.9	0.95	0.98	0.99	0.998	0.999
x_ρ	1.282	1.440	1.645	1.967	2.326	2.576	3.090	3.291

Recalling the central limit theorem from Appendix A, the probability density function of $\bar{x}$ *under* H_0 (i.e., given $\hat{\mu}$) is (approximately) the Gaussian $\mathcal{N}(\hat{\mu}, \frac{\sigma^2}{N})$

$$p_{\bar{x}}(\bar{x}) = \frac{\sqrt{N}}{\sqrt{2\pi}\sigma} \exp\left(-\frac{N(\bar{x}-\hat{\mu})^2}{\sigma^2}\right)$$

Hence, the probability density function of q under H_0 is approximately $\mathcal{N}(0, 1)$. For a significance level ρ the acceptance interval $D \equiv [-x_\rho, x_\rho]$, is chosen as the interval in which the random variable q lies with probability $1-\rho$ (ρ the probability of being in $\bar{D}$). This is readily provided from available tables.

An example for normally distributed $\mathcal{N}(0, 1)$ variables is given in Table 5.1. The decision on the test hypothesis can now be reached by the following steps.

- Given the N experimental samples of x, compute $\bar{x}$ and then q.
- Choose the significance level ρ.
- Compute from the corresponding tables for $\mathcal{N}(0, 1)$ the acceptance interval $D = [-x_\rho, x_\rho]$, corresponding to probability $1-\rho$.
- If $q \in D$ decide H_0, if not decide H_1.

Basically, all we say is that we *expect* the resulting value q to lie in the high-percentage $1-\rho$ interval. If it does not, then we decide that this is because the assumed mean value is not "correct." Of course, the assumed mean value may be correct, but it so happens that the resulting q lies in the least probable area because of the specific set of experimental samples available. In such a case our decision is erroneous, and the probability of committing such an error is ρ.

Example 5.1. Let us consider an experiment with a random variable x of $\sigma = 0.23$, and assume N to be equal to 16 and the resulting $\bar{x}$ equal to 1.35. Adopt the significance level $\rho = 0.05$. We will test if the hypothesis $\hat{\mu} = 1.4$ is true.

From Table 5.1 we have

$$\text{prob}\left\{-1.97 < \frac{\bar{x}-\hat{\mu}}{0.23/4} < 1.97\right\} = 0.95$$

$$\text{prob}\left\{-0.113 < \bar{x}-\hat{\mu} < 0.113\right\} = 0.95$$

Thus, since the value of $\hat{\mu}$, which we have assumed, is in the interval

$$1.237 = 1.35 - 0.113 < \hat{\mu} < 1.35 + 0.113 = 1.463$$

we accept it, as *there is no evidence at the 5% level that the mean value is not equal to* $\hat{\mu}$. The interval [1.237, 1463] is also known as the *confidence interval at the* $1-\rho = 0.95$ *level.*

The Unknown Variance Case

If the variance of x is not known it must be estimated. The estimate

$$\hat{\sigma}^2 \equiv \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{5.5}$$

is an unbiased estimate of the variance. Indeed,

$$\begin{aligned} E[\hat{\sigma}^2] &= \frac{1}{N-1}\sum_{i=1}^{N} E[(x_i - \bar{x})^2] \\ &= \frac{1}{N-1}\sum_{i=1}^{N} E\left[((x_i - \mu) - (\bar{x} - \mu))^2\right] \\ &= \frac{1}{N-1}\sum_{i=1}^{N}\left(\sigma^2 + \frac{\sigma^2}{N} - 2E[(x_i - \mu)(\bar{x} - \mu)]\right) \end{aligned}$$

Due to the independence of the experimental samples

$$E[(x_i - \mu)(\bar{x} - \mu)] = \frac{1}{N}E\left[(x_i - \mu)\left((x_1 - \mu) + \cdots + (x_N - \mu)\right)\right] = \frac{\sigma^2}{N}$$

Thus,

$$E[\hat{\sigma}^2] = \frac{N}{N-1}\frac{N-1}{N}\sigma^2 = \sigma^2$$

The test statistic is now defined as

$$q = \frac{\bar{x} - \mu}{\hat{\sigma}/\sqrt{N}} \tag{5.6}$$

However, this is no longer a Gaussian variable. Following Appendix A, and if we assume that x *is a Gaussian random variable*, then q is described by the so-called t-distribution with $N-1$ degrees of freedom. Table 5.2 shows the confidence interval $D = [-x_\rho, x_\rho]$ for various significance levels and degrees of freedom of the t-distribution.

Table 5.2: Interval values at various significance levels and degrees of freedom for a t-distribution

Degrees of freedom	$1-\rho$	0.9	0.95	0.975	0.99	0.995
10		1.81	2.23	2.63	3.17	3.58
11		1.79	2.20	2.59	3.10	3.50
12		1.78	2.18	2.56	3.05	3.43
13		1.77	2.16	2.53	3.01	3.37
14		1.76	2.15	2.51	2.98	3.33
15		1.75	2.13	2.49	2.95	3.29
16		1.75	2.12	2.47	2.92	3.25
17		1.74	2.11	2.46	2.90	3.22
18		1.73	2.10	2.44	2.88	3.20
19		1.73	2.09	2.43	2.86	3.17
20		1.72	2.09	2.42	2.84	3.15

Example 5.2. For the case of Example 5.1 let us assume that the estimate of the standard deviation $\hat{\sigma}$ is 0.23. Then, according to Table 5.2 for 15 degrees of freedom (N=16) and significance level $\rho = 0.025$

$$\text{prob}\left\{-2.49 < \frac{\bar{x} - \hat{\mu}}{0.23/4} < 2.49\right\} = 0.975$$

and the confidence interval for $\hat{\mu}$ at the 0.975 level is

$$1.207 < \hat{\mu} < 1.493$$

5.3.2 Application of the t-Test in Feature Selection

We will now see how all of this is specialized for the case of feature selection in a classification problem. Our major concern now will be to test, against zero, the difference $\mu_1 - \mu_2$ between the means of the values taken by a feature in two classes. Let $x_i, i = 1, 2, \ldots, N$, be the sample values of the feature in class ω_1 with mean μ_1. Correspondingly, for the other class ω_2 we have $y_i, i = 1, 2, \ldots, N$, with mean μ_2. *Let us now assume that the variance of the feature values is the same in both classes*, $\sigma_1^2 = \sigma_2^2 = \sigma^2$. To decide about the closeness of the two mean values we will test for the hypothesis

$$\begin{aligned} H_1&: \quad \Delta\mu = \mu_1 - \mu_2 \neq 0 \\ H_0&: \quad \Delta\mu = \mu_1 - \mu_2 = 0 \end{aligned} \tag{5.7}$$

To this end, let

$$z = x - y \tag{5.8}$$

where x, y denote the random variables corresponding to the values of the feature in the two classes ω_1, ω_2, respectively, for which *statistical independence* has been assumed. Obviously, $E[z] = \mu_1 - \mu_2$, and due to the independence assumption $\sigma_z^2 = 2\sigma^2$. Following arguments similar to those used before, we now have

$$\bar{z} = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i) = \bar{x} - \bar{y} \tag{5.9}$$

and for the known variance case $\bar{z}$ follows the normal $\mathcal{N}(\mu_1 - \mu_2, \frac{2\sigma^2}{N})$ distribution for large N. Thus, Table 5.1 can be used to decide about (5.7). If the variance is not known, then we choose the test statistic

$$q = \frac{(\bar{x} - \bar{y}) - (\mu_1 - \mu_2)}{s_z\sqrt{\frac{2}{N}}} \tag{5.10}$$

where

$$s_z^2 = \frac{1}{2N-2}\left(\sum_{i=1}^{N}(x_i - \bar{x})^2 + \sum_{i=1}^{N}(y_i - \bar{y})^2\right)$$

It can be shown that $\frac{s_z^2(2N-2)}{\sigma^2}$ follows a chi-square distribution with $2N-2$ degrees of freedom (Appendix A and Problem 5.1). As is pointed out in Appendix A, if x, y are *normally distributed* variables of the same variance σ^2, then the random variable q turns out to follow the t-distribution with $2N-2$ degrees of freedom. Thus, Table 5.2 has to be adopted for the test. When the available number of samples is not the same in all classes, a slight modification is required (Problem 5.2). Furthermore, in practice the variances may not be the same in the two classes. Sometimes this becomes the object of another hypothesis test, concerning the ratio $F = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2}$, to check whether it is close to unity. It can be shown that F, being the ratio of two chi-square distributed variables, follows the so-called F-distribution and the related tables should be used [Fras 58]. Finally, if the Gaussian assumption about x is not a valid one, other criteria can be used to check the equality hypothesis of the means, such as the Kruskal–Wallis statistic [Walp 78, Fine 83].

Example 5.3. The sample measurements of a feature in two classes are

class ω_1:	3.5	3.7	3.9	4.1	3.4	3.5	4.1	3.8	3.6	3.7
class ω_2:	3.2	3.6	3.1	3.4	3.0	3.4	2.8	3.1	3.3	3.6

The question is to check whether this feature is informative enough. If not, it will be discarded during the selection phase. To this end, we will test whether the values of the feature in the two classes differ significantly. We choose the significance level $\rho = 0.05$.

From the foregoing we have

$$\omega_1: \quad \bar{x} = 3.73 \quad \hat{\sigma}_1^2 = 0.0601$$

$$\omega_2: \quad \bar{y} = 3.25 \quad \hat{\sigma}_2^2 = 0.0672$$

For $N = 10$ we have

$$s_z^2 = \frac{1}{2}(\hat{\sigma}_1^2 + \hat{\sigma}_2^2)$$

$$q = \frac{(\bar{x} - \bar{y} - 0)}{s_z\sqrt{\frac{2}{N}}}$$

$$q = 4.25$$

From Table 5.2 and for $20 - 2 = 18$ degrees of freedom and significance level 0.05 we obtain $D = [-2.10, 2.10]$. Since 4.25 lies outside the interval D, we decide in favor of H_1; that is, the mean values differ significantly at the 0.05 level. *Hence, the feature is selected.*

5.4 THE RECEIVER OPERATING CHARACTERISTICS CROC CURVE

The hypothesis tests we have presented offer statistical evidence about the difference of the mean values of a single feature in the various classes. However, although this is useful information for *discarding* features, if the corresponding mean values are closely located, this information may not be sufficient to guarantee good discrimination properties of a feature passing the test. The mean values may differ significantly yet the spread around the means may be large enough to blur the class distinction. We will now focus on techniques providing information about the *overlap* between the classes.

Figure 5.2a illustrates an example of two overlapping probability density functions describing the distribution of a feature in two classes, together with a threshold (one pdf has been inverted for illustration purposes). We decide class ω_1 for values on the left of the threshold and class ω_2 for the values on the right. This decision is associated with an error probability, a, of reaching a wrong decision concerning class ω_1 (the probability of a correct decision is $1 - a$). This is equal to the shaded area under the corresponding curve. Similarly, let $\beta(1 - \beta)$ be the probability of a wrong (correct) decision concerning class ω_2. By moving the threshold over

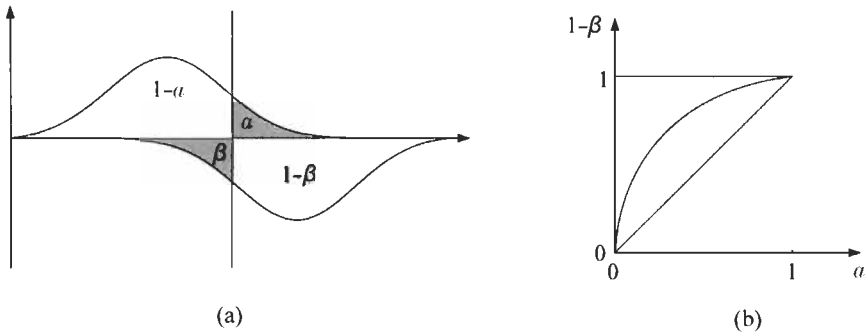


FIGURE 5.2: Example of (a) overlapping pdf's of the same feature in two classes and (b) the resulting ROC curve.

"all" possible positions, different values of a and β result. It takes little thought to realize that if the two distributions have complete overlap, then for *any* position of the threshold we get $a = 1 - \beta$. Such a case corresponds to the straight line in Figure 5.2b, where the two axes are a and $1 - \beta$. As the two distributions move apart, the corresponding curve departs from the straight line, as Figure 5.2b demonstrates. Once more, a little thought reveals that the less the overlap of the classes, the larger the area between the curve and the straight line. At the other extreme of two completely separated class distributions, moving the threshold to sweep the whole range of values for a in $[0, 1]$, $1 - \beta$ remains equal to unity. Thus, the aforementioned area varies between zero, for complete overlap, and $1/2$ (the area of the upper triangle), for complete separation, and *it is a measure of the class discrimination capability of the specific feature*. In practice, the ROC curve can easily be constructed by sweeping the threshold and computing percentages of wrong and correct classifications over the available training feature vectors. Other related criteria that test the overlap of the classes have also been suggested (see Problem 5.5).

5.5 CLASS SEPARABILITY MEASURES

The emphasis in the previous section was on techniques referring to the discrimination properties of *individual* features. However, such methods neglect to take into account the correlation that unavoidably exists among the various features and influences the classification capabilities of the feature vectors that are formed. Measuring the discrimination effectiveness of feature *vectors* will now become our major concern. This information will then be used in two ways. The first is to allow us to combine features appropriately and end up with the "best" feature vector for a given dimension l. The second is to transform the original data on

the basis of an optimality criterion in order to come up with features offering high classification power. In the sequel we will first state *class separability measures*, which will be used subsequently in feature selection procedures.

5.5.1 Divergence

Let us recall our familiar Bayes rule. Given two classes ω_1 and ω_2 and a feature vector $\boldsymbol{x}$, we select ω_1 if

$$P(\omega_1|\boldsymbol{x}) > P(\omega_2|\boldsymbol{x})$$

As pointed out in Chapter 2, the classification error probability depends on the difference between $P(\omega_1|\boldsymbol{x})$ and $P(\omega_2|\boldsymbol{x})$, e.g., equation (2.12). Hence, the ratio $\frac{P(\omega_1|\boldsymbol{x})}{P(\omega_2|\boldsymbol{x})}$ can convey useful information concerning the discriminatory capabilities associated with an adopted feature vector $\boldsymbol{x}$, with respect to the two classes ω_1, ω_2. Alternatively (for given values of $P(\omega_1)$, $P(\omega_2)$), the same information resides in the ratio $\ln \frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} \equiv D_{12}(\boldsymbol{x})$ and this can be used as a measure of the underlying discriminating information of class ω_1 with respect to ω_2. Clearly, for completely overlapped classes we get $D_{12}(\boldsymbol{x}) = 0$. Since $\boldsymbol{x}$ takes different values, it is natural to consider the mean value over class ω_1, that is,

$$D_{12} = \int_{-\infty}^{+\infty} p(\boldsymbol{x}|\omega_1) \ln \frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} d\boldsymbol{x} \tag{5.11}$$

Similar arguments hold for class ω_2 and we define

$$D_{21} = \int_{-\infty}^{+\infty} p(\boldsymbol{x}|\omega_2) \ln \frac{p(\boldsymbol{x}|\omega_2)}{p(\boldsymbol{x}|\omega_1)} d\boldsymbol{x} \tag{5.12}$$

The sum

$$d_{12} = D_{12} + D_{21}$$

is known as the *divergence* and can be used as a separability measure for the classes ω_1, ω_2, with respect to the adopted feature vector $\boldsymbol{x}$. For a multiclass problem, the divergence is computed for every class pair ω_i, ω_j

$$\begin{aligned} d_{ij} &= D_{ij} + D_{ji} \\ &= \int_{-\infty}^{+\infty} (p(\boldsymbol{x}|\omega_i) - p(\boldsymbol{x}|\omega_j)) \ln \frac{p(\boldsymbol{x}|\omega_i)}{p(\boldsymbol{x}|\omega_j)} d\boldsymbol{x} \end{aligned} \tag{5.13}$$

and the average class separability can be computed using the average divergence

$$d = \sum_{i=1}^{M} \sum_{j=1}^{M} P(\omega_i) P(\omega_j) d_{ij}$$

Divergence is basically a form of the Kullback–Leibler distance measure between density functions [Kulb 51] (Appendix A). The divergence has the following easily shown properties:

$$\begin{aligned} d_{ij} &\geq 0 \\ d_{ij} &= 0 \qquad \text{if } i = j \\ d_{ij} &= d_{ji} \end{aligned}$$

If the components of the feature vector are statistically independent, then it can be shown (Problem 5.8) that

$$d_{ij}(x_1, x_2, \ldots, x_l) = \sum_{r=1}^{l} d_{ij}(x_r)$$

Assuming now that the density functions are Gaussians $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$ and $\mathcal{N}(\boldsymbol{\mu}_j, \Sigma_j)$, respectively, the computation of the divergence is simplified and it is not difficult to show that

$$d_{ij} = \frac{1}{2}\text{trace}\left\{\Sigma_i^{-1}\Sigma_j + \Sigma_j^{-1}\Sigma_i - 2I\right\} + \frac{1}{2}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T\left(\Sigma_i^{-1} + \Sigma_j^{-1}\right)(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) \tag{5.14}$$

For the one-dimensional case this becomes

$$d_{ij} = \frac{1}{2}\left(\frac{\sigma_j^2}{\sigma_i^2} + \frac{\sigma_i^2}{\sigma_j^2} - 2\right) + \frac{1}{2}(\mu_i - \mu_j)^2\left(\frac{1}{\sigma_i^2} + \frac{1}{\sigma_j^2}\right)$$

As already pointed out, a class separability measure cannot depend only on the difference of the mean values; it must also be variance dependent. Indeed, divergence does depend explicitly on both the difference of the means and the respective variances. Furthermore, d_{ij} can be large even for equal mean values, *provided the variances differ significantly*. Thus, class separation is still possible even if the class means coincide. We will come to this later on.

Let us now investigate (5.14). If the covariance matrices of the two Gaussian distributions are equal, $\Sigma_i = \Sigma_j = \Sigma$, then the divergence is further simplified to

$$d_{ij} = (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T\Sigma^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$$

which is nothing other than the Mahalanobis distance between the corresponding mean vectors. This has another interesting implication. Recalling Problem 2.9 of Chapter 2, it turns out that in this case we have a direct relation between the divergence d_{ij} and the Bayes error, that is, the minimum error we can achieve by adopting the specific feature vector. This is a most desirable property for any class separability measure. Unfortunately, such a direct relation of the divergence

with the Bayes error is not possible for more general distributions. Furthermore, in [Swai 73, Rich 95] it is pointed out that the specific dependence of the divergence on the difference of the mean vectors may lead to misleading results, in the sense that small variations in the difference of the mean values can produce large changes in the divergence, which, however, are not reflected in the classification error. To overcome this, a variation of the divergence is suggested, called the *transformed divergence:*

$$\hat{d}_{ij} = 2\left(1 - \exp(-d_{ij}/8)\right)$$

In the sequel, we will try to define class separability measures with a closer relationship to the Bayes error.

5.5.2 Chernoff Bound and Bhattacharyya Distance

The minimum attainable classification error of the Bayes classifier for two classes ω_1, ω_2 can be written as:

$$P_e = \int_{-\infty}^{\infty} \min\left[P(\omega_i)p(\boldsymbol{x}|\omega_i),\, P(\omega_j)p(\boldsymbol{x}|\omega_j)\right] d\boldsymbol{x} \tag{5.15}$$

Analytic computation of this integral in the general case is not possible. However, an upper bound can be derived. The derivation is based on the inequality

$$\min[a, b] \leq a^s b^{1-s} \quad \text{for} \quad a, b \geq 0, \quad \text{and} \quad 0 \leq s \leq 1 \tag{5.16}$$

Combining (5.15) and (5.16), we get

$$P_e \leq P(\omega_i)^s P(\omega_j)^{1-s} \int_{-\infty}^{\infty} p(\boldsymbol{x}|\omega_i)^s\, p(\boldsymbol{x}|\omega_j)^{1-s}\, d\boldsymbol{x} \equiv \epsilon_{CB} \tag{5.17}$$

ϵ_{CB} is known as the *Chernoff bound*. The minimum bound can be computed by minimizing ϵ_{CB} with respect to s. A special form of the bound results for $s = 1/2$:

$$P_e \leq \epsilon_{CB} = \sqrt{P(\omega_i)P(\omega_j)} \int_{-\infty}^{\infty} \sqrt{p(\boldsymbol{x}|\omega_i)p(\boldsymbol{x}|\omega_j)}\, d\boldsymbol{x} \tag{5.18}$$

For Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i), \mathcal{N}(\boldsymbol{\mu}_j, \Sigma_j)$ and after a bit of algebra, we obtain

$$\epsilon_{CB} = \sqrt{P(\omega_i)P(\omega_j)}\exp(-B)$$

where

$$B = \frac{1}{8}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T \left(\frac{\Sigma_i + \Sigma_j}{2}\right)^{-1} (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \frac{1}{2}\ln \frac{\left|\frac{\Sigma_i + \Sigma_j}{2}\right|}{\sqrt{|\Sigma_i||\Sigma_j|}} \tag{5.19}$$

and $|\cdot|$ denotes the determinant of the respective matrix. The term B is known as the *Bhattacharyya distance* and it is used as a class separability measure. It can be shown (Problem 5.9) that it corresponds to the optimum Chernoff bound when $\Sigma_i = \Sigma_j$. It is readily seen that in this case the Bhattacharyya distance becomes proportional to the Mahalanobis distance between the means.

A comparative study of various distance measures for feature selection in the context of multispectral data classification in remote sensing can be found in [Maus 90]. A more detailed treatment of the topic is given in [Fuku 90].

Example 5.4. Assume that $P(\omega_1) = P(\omega_2)$ and that the corresponding distributions are Gaussians $\mathcal{N}(\mu, \sigma_1^2 I)$ and $\mathcal{N}(\mu, \sigma_2^2 I)$. The Bhattacharyya distance becomes

$$B = \frac{1}{2} \ln \frac{\left(\frac{\sigma_1^2+\sigma_2^2}{2}\right)^l}{\sqrt{\sigma_1^{2l}\sigma_2^{2l}}} = \frac{1}{2} \ln \left(\frac{\sigma_1^2 + \sigma_2^2}{2\sigma_1\sigma_2} \right)^l \tag{5.20}$$

For the one-dimensional case $l = 1$ and for $\sigma_1 = 10\sigma_2$, $B = 0.8097$ and

$$P_e \leq 0.2225$$

If $\sigma_1 = 100\sigma_2$, $B = 1.9561$ and

$$P_e \leq 0.0707$$

Thus, the greater the difference of the variances, the smaller the error bound. The decrease is bigger for higher dimensions due to the dependence on l. Figure 5.3

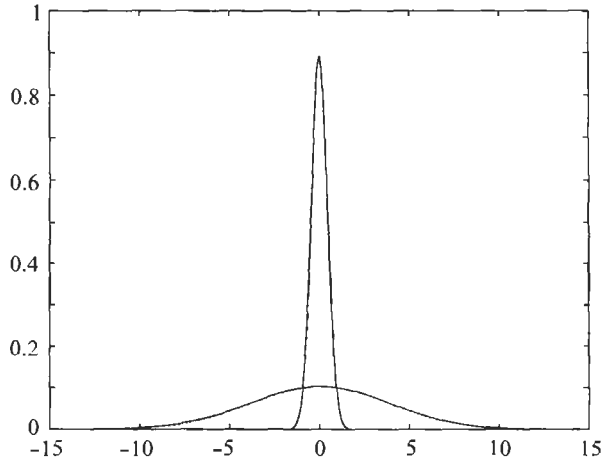


FIGURE 5.3: Gaussian pdf's with the same mean and different variances.

shows the pdf's for the same mean and $\sigma_1 = 1, \sigma_2 = 0.01$. The figure is self-explanatory as to how the Bayesian classifier discriminates between two classes of the same mean and significantly different variances. Furthermore, as $\sigma_2/\sigma_1 \longrightarrow 0$, the probability of error tends to zero (why?).

5.5.3 Scatter Matrices

A major disadvantage of the class separability criteria considered so far is that they are not easily computed, unless the Gaussian assumption is employed. We will now turn our attention on a set of simpler criteria, built upon information related to the way feature vector samples are scattered in the l-dimensional space. To this end, the following matrices are defined:

Within-class scatter matrix

$$S_w = \sum_{i=1}^{M} P_i S_i$$

where S_i is the covariance matrix for class ω_i

$$S_i = E[(\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^T]$$

and P_i the *a priori* probability of class ω_i. That is, $P_i \simeq n_i/N$, where n_i is the number of samples in class ω_i, out of a total of N samples. Obviously, trace$\{S_w\}$ is a measure of the average, over all classes, variance of the features.

Between-class scatter matrix

$$S_b = \sum_{i=1}^{M} P_i(\boldsymbol{\mu}_i - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_i - \boldsymbol{\mu}_0)^T$$

where $\boldsymbol{\mu}_0$ is the global mean vector

$$\boldsymbol{\mu}_0 = \sum_{i}^{M} P_i \boldsymbol{\mu}_i$$

trace$\{S_b\}$ is a measure of the average (over all classes) distance of the mean of each individual class from the respective global value.

Mixture scatter matrix

$$S_m = E[(\boldsymbol{x} - \boldsymbol{\mu}_0)(\boldsymbol{x} - \boldsymbol{\mu}_0)^T]$$

That is, S_m is the covariance matrix of the feature vector with respect to the global mean. It is not difficult to show (Problem 5.10) that

$$S_m = S_w + S_b$$

Its trace is the sum of variances of the features around their respective global mean. From these definitions it is straightforward to see that the criterion

$$J_1 = \frac{\text{trace}\{S_m\}}{\text{trace}\{S_w\}}$$

takes large values when samples in the l-dimensional space are well clustered around their mean, within each class, and the clusters of the different classes are well separated. Sometimes S_b is used in place of S_m. An alternative criterion results if determinants are used in the place of traces. This is justified for scatter matrices that are symmetric positive definite and thus their eigenvalues are positive (Appendix B). The trace is equal to the sum of the eigenvalues, while the determinant is equal to their product. Hence, large values of J_1 also correspond to large values of the criterion

$$J_2 = \frac{|S_m|}{|S_w|} = |S_w^{-1} S_m|$$

A variant of J_2 commonly encountered in practice is

$$J_3 = \text{trace}\{S_w^{-1} S_m\}$$

As we will see later on, criteria J_2 and J_3 have the advantage of being invariant under linear transformations, and we will adopt them to derive features in an optimal way. In [Fuku 90] a number of different criteria are also defined by using various combinations of S_w, S_b, S_m in a "trace" or "determinant" formulation. However, whenever a determinant is used, one should be careful with S_b, since $|S_b| = 0$ for $M < l$. This is because S_b is the sum of $Ml \times l$ matrices, of rank one each.

These criteria take a special form in the one-dimensional, two-class problem. In this case, it is easy to see that for equiprobable classes $|S_w|$ is proportional to

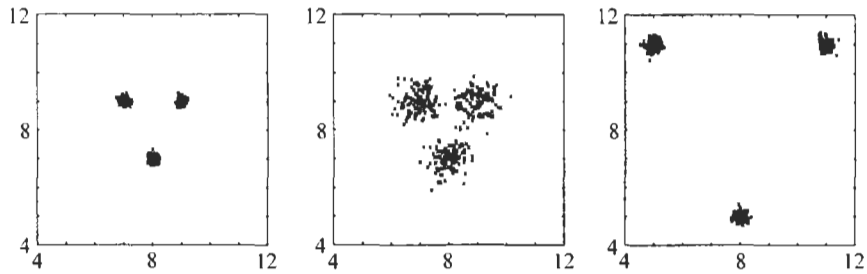


FIGURE 5.4: Classes with (a) small within-class variance and small between-class distances, (b) large within-class variance and small between-class distances and (c) small within-class variance and large between-class distances.

$\sigma_1^2 + \sigma_2^2$ and $|S_b|$ proportional to $(\mu_1 - \mu_2)^2$. Combining S_b and S_w, the so-called *Fisher's discriminant ratio* results

$$FDR = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

FDR is sometimes used to quantify the separability capabilities of individual features. *FDR* reminds us of the test statistic q appearing in the hypothesis statistical tests dealt with before. However, here the use of *FDR* is suggested in a more "primitive" fashion, independent of the underlying statistical distributions. For the multiclass case, averaging forms of *FDR* can be used. One possibility is

$$FDR_1 = \sum_{i}^{M} \sum_{j \neq i}^{M} \frac{(\mu_i - \mu_j)^2}{\sigma_i^2 + \sigma_j^2}$$

where the subscripts i, j refer to the mean and variance corresponding to the feature under investigation for the classes ω_i, ω_j, respectively.

Example 5.5. Figure 5.4 shows three cases of classes at different locations and within-class variances. The resulting values for the J_3 criterion involving the S_w and S_m matrices are 164.7, 12.5, and 620.9 for the cases in Figures 5.4a, b, and c, respectively. That is, the best is for distant well-clustered classes and the worst for the case of closely located classes with large within-class variance.

5.6 FEATURE SUBSET SELECTION

Having defined a number of criteria, measuring the classification effectiveness of individual features and/or feature vectors, we come to the heart of our problem,

that is, to select a subset of l features out of m originally available. There are two major directions to follow.

5.6.1 Scalar Feature Selection

Features are treated individually. Any of the class separability measuring criteria can be adopted, for example, *ROC*, *FDR*, one-dimensional divergence, and so on. The value of the criterion $C(k)$ is computed for each of the features, $k = 1, 2, \ldots, m$. Features are then ranked in order of descending values of $C(k)$. The l features corresponding to the l best values of $C(k)$ are then selected to form the feature vector.

All the criteria we have dealt with in the previous sections measure the classification capability with respect to a two-class problem. As we have already pointed out in a couple of places, in a multiclass situation a form of average or "total" value, over all classes, is used to compute $C(k)$. However, this is not the only possibility. In [Su 94] the one-dimensional divergence d_{ij} was used and computed for every pair of classes. Then, for each of the features, the corresponding $C(k)$ was set equal to

$$C(k) = \min_{i,j} d_{ij}$$

that is, the minimum divergence value over all class pairs, instead of an average value. Thus, selecting the features with the largest $C(k)$ values is equivalent to choosing features with the best "worst case" class separability capability, giving a "*maxmin*" flavor to the feature selection task. Such an approach may lead to more robust performance in certain cases.

The major advantage of dealing with features individually is computational simplicity. However, such approaches do not take into account existing correlations between features. Before we proceed to techniques dealing with vectors, we will comment on some "ad hoc" techniques that incorporate correlation information combined with criteria tailored for scalar features.

Let x_{nk}, $n = 1, 2, \ldots, N$ and $k = 1, 2, \ldots, m$, be the kth feature of the nth pattern. The cross-correlation coefficient between any two of them is given by

$$\rho_{ij} = \frac{\sum_{n=1}^{N} x_{ni} x_{nj}}{\sqrt{\sum_{n=1}^{N} x_{ni}^2 \sum_{n=1}^{N} x_{nj}^2}} \tag{5.21}$$

It can be shown that $|\rho_{ij}| \leq 1$ (Problem 5.11). The selection procedure evolves along the following steps:

- Select a class separability criterion C and compute its values for all the available features x_k, $k = 1, 2, \ldots, m$. Rank them in descending order and choose the one with the best C value. Let us say that this is x_{i_1}.

- To select the second feature, compute the cross-correlation coefficient defined in Eq. (5.21) between the chosen x_{i_1} and each of the remaining $m-1$ features, that is, $\rho_{i_1 j}$, $j \neq i_1$.
- Choose the feature x_{i_2} for which

$$i_2 = arg \max_j \left\{ \alpha_1 C(j) - \alpha_2 |\rho_{i_1 j}| \right\}, \quad \text{for all } j \neq i_1$$

where α_1, α_2 are weighting factors that determine the relative importance we give to the two terms. In words, for the selection of the next feature, we take into account not only the class separability measure C but also the correlation with the already chosen feature. This is then generalized for the kth step

- Select x_{i_k}, $k = 3, \ldots, l$, so that

$$i_k = arg \max_j \left\{ \alpha_1 C(j) - \frac{\alpha_2}{k-1} \sum_{r=1}^{k-1} |\rho_{i_r j}| \right\} \quad \text{for } j \neq i_r, \quad r = 1, 2, \ldots, k-1$$

That is, the average correlation with all previously selected features is taken into account.

There are variations of this procedure. For example, in [Fine 83] more than one criterion is adopted and averaged out. Hence, the best index is found by optimizing

$$\left\{ \alpha_1 C_1(j) + \alpha_2 C_2(j) - \frac{\alpha_3}{k-1} \sum_{r=1}^{k-1} |\rho_{i_r j}| \right\}$$

5.6.2 Feature Vector Selection

Treating features individually, that is, as scalars, has the advantage of computational simplicity but may not be effective for complex problems and for features with high mutual correlation. We will now focus on techniques measuring classification capabilities of feature vectors. It does not require much thought to see that computational burden is the major limiting factor of such an approach. Indeed, if we want to act according to what "optimality" suggests, we should form *all* possible vector combinations of l features out of the m originally available. For each combination we should use one of the separability criteria introduced previously

(e.g., Bhattacharrya distance, J_2) and select the best feature vector combination. Recalling our combinatorics basics, we obtain the total number of vectors as

$$\binom{m}{l} = \frac{m!}{l!(m-l)!} \tag{5.22}$$

This is a large number even for small values of l, m. Indeed, for $m = 20, l = 5$, the number equals 15504. Furthermore, in many practical cases the number l is not even known a priori. Thus, one has to try feature combinations for different values of l and select the "best" value for it (beyond which no gain in performance is obtained) and the corresponding "best" l-dimensional feature vector. As we will see in Chapter 10, sometimes it is desirable to base our feature selection decision not on the values of an adopted class separability criterion but on the performance of the classifier itself. That is, for each feature vector combination the classification error probability of the classifier has to be estimated and the combination resulting in the minimum error probability selected. This approach may increase the complexity requirements even more, depending, of course, on the classifier type. In order to reduce complexity, a number of efficient searching techniques have been suggested. Some of them are suboptimal and some optimal (under certain assumptions or constraints).

Suboptimal Searching Techniques

Sequential Backward Selection. We will demonstrate the method via an example. Let $m = 4$, and the originally available features are x_1, x_2, x_3, x_4. We wish to select two of them. The selection procedure consists of the following steps:

- Adopt a class separability criterion, C, and compute its value for the feature vector $[x_1, x_2, x_3, x_4]^T$.
- Eliminate one feature and for each of the possible resulting combinations, that is, $[x_1, x_2, x_3]^T$, $[x_1, x_2, x_4]^T$, $[x_1, x_3, x_4]^T$, $[x_2, x_3, x_4]^T$, compute the corresponding criterion value. Select the combination with the best value, say $[x_1, x_2, x_3]^T$.
- From the selected three-dimensional feature vector eliminate one feature and for each of the resulting combinations, $[x_1, x_2]^T$, $[x_1, x_3]^T$, $[x_2, x_3]^T$, compute the criterion value and select the one with the best value.

Thus, starting from m, at each step we drop out one feature from the "best" combination until we obtain a vector of l features. Obviously, this is a *suboptimal* searching procedure, since nobody can guarantee that the optimal two-dimensional vector has to originate from the optimal three-dimensional one. The number of combinations searched via this method is $1 + 1/2((m+1)m - l(l+1))$ (Problem 5.13), which is substantially less than that of the full search procedure.

Sequential Forward Selection. Here, the reverse to the preceding procedure is followed:

- Compute the criterion value for each of the features. Select the feature with the best value, say x_1.
- Form all possible two-dimensional vectors that contain the winner from the previous step, that is, $[x_1, x_2]^T$, $[x_1, x_3]^T$, $[x_1, x_4]^T$. Compute the criterion value for each of them and select the best one, say $[x_1, x_3]^T$.

If $l = 3$, then the procedure must continue. That is, we form all three-dimensional vectors springing from the two-dimensional winner, that is, $[x_1, x_3, x_2]^T$, $[x_1, x_3, x_4]^T$, and select the best one. For the general l, m case, it is simple algebra to show that the number of combinations searched with this procedure is $lm - l(l-1)/2$. Thus, from a computational point of view, the backward search technique is more efficient than the forward one for l closer to m than to 1.

Floating Search Methods. The preceding two methods suffer from the so-called *nesting effect*. That is, once a feature is discarded in the backward method, there is no possibility for it to be reconsidered again. The opposite is true for the forward procedure; once a feature is chosen, there is no way for it to be discarded later on. In [Pudi 94] a technique is suggested that offers the flexibility to reconsider features previously discarded and vice versa, to discard features previously selected. The technique is called the *floating search method*. There are two schemes that implement this technique. One springs from the forward selection and the other from the backward selection rationale. We will focus on the former. We consider a set of m features, and the idea is to search for the best subset of k of them for $k = 1, 2, \ldots, l \leq m$ so that a cost criterion C is optimized. Let $X_k = \{x_1, x_2, \ldots, x_k\}$ be the set of the best combination of k of the features and Y_{m-k} the set of the remaining $m - k$ features. We also keep all the lower dimension best subsets $X_2, X_3, \ldots, X_{k-1}$ of $2, 3, \ldots, k-1$ features, respectively. The rationale at the heart of the method is summarized as follows: At the next step the $k + 1$ best subset X_{k+1} is formed by "borrowing" an element from Y_{m-k}. Then, return to the previously selected lower dimension subsets to check whether the inclusion of this new element improves the criterion C. If it does, the new element replaces one of the previously selected features. The steps of the algorithm, when maximization of C is required are:

- *Step 1: Inclusion*
  $x_{k+1} = arg\ \max_{y \in Y_{m-k}} C(\{X_k, y\})$; that is, choose that element from Y_{m-k} which, combined with X_k, results to the best value of C.
  $X_{k+1} = \{X_k, x_{k+1}\}$

- *Step II: Test*

  1. $x_r = arg \max_{y \in X_{k+1}} C(X_{k+1} - \{y\})$; that is, find the feature that has the least effect on the cost when it is removed from X_{k+1}.
  2. If $r = k + 1$, change $k = k + 1$ and go to step I.
  3. If $r \neq k + 1$ AND $C(X_{k+1} - \{x_r\}) < C(X_k)$ go to step I; that is, if removal of x_r does not improve upon the cost of the previously selected best group of k, no further backward search is performed.
  4. If $k = 2$ put $X_k = X_{k+1} - \{x_r\}$ and $C(X_k) = C(X_{k+1} - \{x_r\})$; go to step I.

- *Step III: Exclusion*

  1. $X'_k = X_{k+1} - \{x_r\}$; that is, remove x_r.
  2. $x_s = arg \max_{y \in X'_k} C(X'_k - \{y\})$; that is, find the least significant feature in the new set.
  3. If $C(X'_k - \{x_s\}) < C(X_{k-1})$ then $X_k = X'_k$ and go to step I; no further backward search is performed.
  4. Put $X'_{k-1} = X'_k - \{x_s\}$ and $k = k - 1$.
  5. If $k = 2$ put $X_k = X'_k$ and $C(X_k) = C(X'_k)$ and go to step I.
  6. Go to step III.

The algorithm is initialized by running the sequential forward algorithm to form X_2. The algorithm terminates when l features have been selected. Although the algorithm does not guarantee finding all the best feature subsets, it results in substantially improved performance compared with its sequential counterpart, at the expense of increased complexity. The backward floating search scheme operates in the reverse direction but with the same philosophy.

Optimal Searching Techniques

These techniques are applicable when the *separability criterion is monotonic*, that is,

$$C(x_1, \ldots, x_i) \leq C(x_1, \ldots, x_i, x_{i+1})$$

This property allows identifying the optimal combination but at a considerably reduced computational cost with respect to (5.22). Algorithms based on the *dynamic programming* concept (Chapter 8) offer one possibility to approaching the problem. A computationally more efficient way is to formulate the problem as a combinatorial optimization task and employ the so-called *branch and bound* methods to obtain the optimal solution [Lawe 66, Yu 93]. These methods

compute the optimal value without involving exhaustive enumeration of all possible combinations. A more detailed description of the branch and bound methods is given in Chapter 15 and can also be found in [Fuku 90]. However, the complexity of these techniques is still higher than that of the previously mentioned suboptimal techniques.

Remarks

- The separability measures and feature selection techniques presented above, although they indicate the major directions followed in practice, do not cover the whole range of methods that have been suggested. For example, in [Bati 94, Kwak 02] the mutual information between the input features and the classifier's outputs is used as a criterion. The features that are selected maximize the input-output mutual information. In [Seti 97] a feature selection technique is proposed based on a decision tree by excluding features one by one and retraining the classifier. In [Zhan 02] the tabu combinatorial optimization technique is employed for feature selection.

A comparative study of various feature selection searching schemes can be found in [Kitt 78, Devi 82, Pudi 94, Jain 97, Brun 00].

5.7 OPTIMAL FEATURE GENERATION

So far, the class separability measuring criteria have been used in a rather "passive" way, that is, to measure the classification effectiveness of features generated in *some* way. In this section we will employ these measuring criteria in an "active" way, that is, as an integral part of the feature generation process itself. From this point of view, this section can be considered as a bridge between this chapter and the following one. Our major task can be summarized as follows: If $\boldsymbol{x}$ is an m-dimensional vector of measurement samples, transform it into another l-dimensional vector $\boldsymbol{y}$ so that an adopted class separability criterion is optimized. We will confine ourselves to linear transformations,

$$\boldsymbol{y} = A^T \boldsymbol{x}$$

where A^T is an $l \times m$ matrix. Any of the criteria exposed so far can be used. Obviously, the degree of complexity of the optimization procedure depends heavily on the chosen criterion. We will demonstrate the method via the J_3 scattering matrix criterion, involving S_w and S_b matrices. Its optimization is straightforward and at the same time it has some interesting implications. Let S_{xw}, S_{xb} be the within-class and between-class scatter matrices of $\boldsymbol{x}$. From the respective definitions, the corresponding matrices of $\boldsymbol{y}$ become

$$S_{yw} = A^T S_{xw} A, \qquad S_{yb} = A^T S_{xb} A$$

Thus, the J_3 criterion in the y subspace is given by

$$J_3(A) = \text{trace}\{(A^T S_{xw} A)^{-1}(A^T S_{xb} A)\}$$

Our task is to compute the elements of A so that this is maximized. Then A must necessarily satisfy

$$\frac{\partial J_3(A)}{\partial A} = 0$$

It can be shown that (Problem 5.14)

$$\frac{\partial J_3(A)}{\partial A} = -2S_{xw}A(A^T S_{xw}A)^{-1}(A^T S_{xb}A)(A^T S_{xw}A)^{-1} + 2S_{xb}A(A^T S_{xw}A)^{-1}$$
$$= 0$$

or

$$(S_{xw}^{-1} S_{xb})A = A(S_{yw}^{-1} S_{yb}) \tag{5.23}$$

An experienced eye will easily identify the affinity of this with an eigenvalue problem. It suffices to simplify its formulation slightly. Recall from Appendix B that the matrices S_{yw}, S_{yb} can be diagonalized simultaneously by a linear transformation

$$B^T S_{yw} B = I, \qquad B^T S_{yb} B = D \tag{5.24}$$

which are the within- and between-class scatter matrices of the transformed vector

$$\hat{\boldsymbol{y}} = B^T \boldsymbol{y} = B^T A^T \boldsymbol{x}$$

B is an $l \times l$ matrix and D an $l \times l$ diagonal matrix. Note that in going from $\boldsymbol{y}$ to $\hat{\boldsymbol{y}}$ there is no loss in the value of the cost J_3. This is because J_3 is invariant under linear transformations, within the l-dimensional subspace. Indeed,

$$J_3(\hat{\boldsymbol{y}}) = \text{trace}\{S_{\hat{y}w}^{-1} S_{\hat{y}b}\} = \text{trace}\{(B^T S_{yw} B)^{-1}(B^T S_{yb} B)\}$$
$$= \text{trace}\{B^{-1} S_{yw}^{-1} S_{yb} B\}$$
$$= \text{trace}\{S_{yw}^{-1} S_{yb} B B^{-1}\} = J_3(\boldsymbol{y})$$

Combining (5.23) and (5.24), we finally obtain

$$(S_{xw}^{-1} S_{xb})C = CD \tag{5.25}$$

where $C = AB$ is an $m \times l$ dimensional matrix. Equation (5.25) is a typical eigenvalue–eigenvector problem, with the diagonal matrix D having the eigenvalues of $S_{xw}^{-1} S_{xb}$ on its diagonal and C having the corresponding eigenvectors

as its columns. However, $S_{xw}^{-1}S_{xb}$ is an $m \times m$ matrix, and the question is which l out of a total of m eigenvalues we must choose for the solution of (5.25). From its definition, matrix S_{xb} is of rank $M-1$, where M is the number of classes (Problem 5.15). Thus, $S_{xw}^{-1}S_{xb}$ is also of rank $M-1$ and there are $M-1$ nonzero eigenvalues. Let us focus on the two possible alternatives separately.

- $l = M - 1$: We first form matrix C so that its columns are the unit norm $M - 1$ eigenvectors of $S_{xw}^{-1}S_{xb}$. Then we form the transformed vector

$$\hat{\boldsymbol{y}} = C^T \boldsymbol{x} \tag{5.26}$$

This guarantees the maximum J_3 value. *As a matter of fact, in reducing the number of data from m to* $M - 1$, *there is no loss in class separability power, as this is measured by* J_3. Indeed,

$$J_{3,x} = \text{trace}\{S_{xw}^{-1}S_{xb}\} = \lambda_1 + \cdots + \lambda_{M-1} + 0 \tag{5.27}$$

and

$$J_{3,\hat{y}} = \text{trace}\{(C^T S_{xw} C)^{-1}(C^T S_{xb} C)\} \tag{5.28}$$

Rearranging (5.25), we get

$$C^T S_{xb} C = C^T S_{xw} C D \tag{5.29}$$

Combining (5.28) and (5.29), we obtain

$$J_{3,\hat{y}} = \text{trace}\{D\} = \lambda_1 + \cdots + \lambda_{M-1} = J_{3,x} \tag{5.30}$$

It is most interesting to view this from a slightly different perspective. Let us recall the Bayesian classifier for an M class problem. Of the M conditional class probabilities, $P(\omega_i|\boldsymbol{x})$, $i = 1, 2, \ldots, M$, only $M - 1$ are independent, since they all add up to one. In general, $M - 1$ is the *minimum* number of discriminant functions needed for an M-class classification task (Problem 5.16). *Hence, the linear operation* $C^T\boldsymbol{x}$, *which computes the* $M - 1$ *components of* $\hat{\boldsymbol{y}}$, *can be seen as the optimal linear classifier, where optimality is with respect to* J_3. Therefore, this procedure can be viewed as a combination of the feature selection and classifier design stages, provided the classifier is a linear one. In Chapter 3 the optimal linear classifier was computed so as to minimize the mean (least) squares error. In this section it was designed to maximize J_3. From this point of view, this section can also be seen as a bridge with Chapter 3. This can be further strengthened by investigating the specific form that this classifier takes for the two-class problem. In this case, there is only one

nonzero eigenvalue and it is not difficult to show (Problem 5.17) that

$$\hat{y} = (\mu_1 - \mu_2)^T S_w^{-1} x$$

The resulting linear classifier is also known as *Fisher's linear discriminant*. For Gaussian random vectors, with equal covariance matrices in both classes, this is nothing other than the optimal Bayesian classifier with the exception of a threshold value (Problem 2.11). Recall from Problem 3.14 that this is also directly related to the linear MSE classifier.

- $l < M-1$: In this case C is formed from the eigenvectors corresponding to the l largest eigenvalues of $S_w^{-1} S_b$. The fact that J_3 is given as the sum of the corresponding eigenvalues guarantees its maximization. Of course, in this case there is loss of the available information because now $J_{3,\hat{y}} < J_{3,x}$.

Remarks

- If J_3 is used with another combination of matrices, such as S_w and S_m, then, in general, the rank of the corresponding matrix product involved in the trace is m and there are m nonzero eigenvalues. In such cases the transformation matrix C is formed so that its columns are the eigenvectors corresponding to the l *largest eigenvalues*. According to (5.30), this guarantees the maximum value of J_3.
- A geometric interpretation of (5.26) reveals that $\hat{y}$ is the projection of the original vector x onto the subspace spanned by the eigenvectors v_i of $S_w^{-1} S_b$. It must be pointed out that these *are not* mutually orthogonal. Indeed, although matrices S_w, S_b (S_m) are symmetric, products of the form $S_w^{-1} S_b$ are not; thus, the eigenvectors are not mutually orthogonal (Problem 5.18). Furthermore, as we saw during the proof, once we decide on which subspace to project (by selecting the appropriate combination of eigenvectors) *the value of J_3 remains invariant under any linear transformation within this subspace*. That is, it is independent of the coordinate system and its value depends only on the particular subspace. In general, projection of the original feature vectors onto a lower dimensional subspace is associated with some information loss. An extreme example is shown in Figure 5.5, where the two classes coincide after projection on the v_1 axis. The choice of the subspace corresponding to the optimal J_3 value guarantees no loss of information for $l = M - 1$ (as this is measured by the J_3 criterion). Thus, this is a good choice, provided that J_3 is a good criterion for the problem of interest. Of course, this is not always the case; it depends on the specific classification task. A more extensive treatment of the topic, also involving other optimizing criteria, can be found in [Fuku 90].
- No doubt, scattering matrix criteria are not the only ones to compute the optimal transformation matrix. For example, [Wata 97] suggested using a

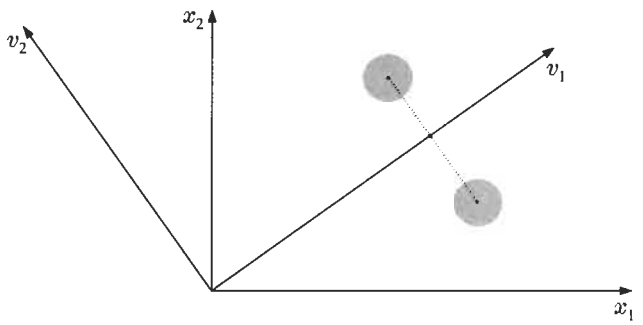


FIGURE 5.5: Geometry illustrating the loss of information associated with projections in lower dimensional subspaces.

different transformation matrix for each class and optimizing with respect to the classification error. This is within the spirit of the recent trend, to optimize directly with respect to the quantity of interest, which is the classification error probability. For the optimization, smooth versions of the error rate are used to guarantee differentiability. Other ways to compute the transformation matrix will be discussed in the next chapter.

- Besides linear ones, nonlinear transformations can also be employed for optimal feature selection. For example, in [Samm 69] a nonlinear technique is proposed that attempts to preserve maximally all the distances between vectors. Let $\boldsymbol{x}_i, \boldsymbol{y}_i, i = 1, 2, \ldots, N$, be the feature vectors in the original m-dimensional and the transformed l-dimensional space, respectively. The transformation into the lower dimensional space is performed so as to maximize

$$J = \frac{1}{\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d^o(i, j)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{(d^o(i, j) - d(i, j))^2}{d^o(i, j)} \tag{5.31}$$

where $d^o(i, j), d(i, j)$ are the (Euclidean) distances between vectors $\boldsymbol{x}_i$, and $\boldsymbol{x}_j$ in the original space and $\boldsymbol{y}_i, \boldsymbol{y}_j$ in the transformed space, respectively.

5.8 NEURAL NETWORKS AND FEATURE GENERATION/SELECTION

Recently, efforts have been made to use neural networks for feature generation and selection. A possible solution is via the so-called *auto-associative networks*.

A network is employed having m input and m output nodes and a single hidden layer with l nodes with linear activations. During training, the desired outputs are the same as the inputs. That is,

$$\mathcal{E}(i) = \sum_{k=1}^{m} (\hat{y}_k(i) - x_k(i))^2$$

where the notation of the previous chapter has been adopted. Such a network has a unique minimum and the outputs of the hidden layer constitute the projection of the input m-dimensional space onto an l-dimensional subspace. In [Bour 88] it is shown that this is basically a projection onto the subspace spanned by the l principal eigenvectors of the input correlation matrix, a topic on which we will focus in the next chapter. An extension of this idea is to use three hidden layers [Kram 91]. Such a network performs a nonlinear principal component analysis. The major drawback of such an architecture is that nonlinear optimization techniques have to be used for the training. Besides the computational load, the risk of being trapped in local minima is always present.

An alternative is to use neural networks, or any other (non)linear structure, to exploit properties of the LS cost function. In Chapter 3, we saw that the outputs of a network approximate posterior probabilities, provided that the weights have been trained so that the outputs match, in the LS sense, the class labels. In [Lowe 91] it is pointed out that, besides this property, another very interesting one is also valid. A multilayer perceptron was considered with linear output nodes. The network was trained to minimize the squared error between the actual and desired responses (i.e., class labels 1 and 0). It was shown that minimizing the squared error is equivalent to maximizing the criterion

$$J = \text{trace}\{S_m^{-1} S_b\} \tag{5.32}$$

where S_m is the mixture scatter matrix of the vectors formed by the outputs of the last hidden layer nodes and S_b the corresponding between-class scatter matrix in a weighted form (Problem 5.19). If the inverse of S_m does not exist, it is replaced by its pseudoinverse. In other words, such a network can be used as a *J-optimal nonlinear* transformer of the input m-dimensional vectors into l-dimensional vectors, where l is the number of nodes in the last hidden layer.

Another approach is to employ neural networks to perform the computations associated with the optimization of various class separability criteria discussed in this chapter. Although these techniques do not necessarily provide new approaches, the incorporation of neural networks offers the capability of adaptation in case the statistics of the input data is slowly varying. In [Chat 97, Mao 95] a number of such techniques are developed. The idea behind most of these techniques is to use a network that iteratively computes eigenvectors of correlation matrices, a step which, as we have seen, is at the heart of a number of optimality criteria.

An alternative technique has been suggested in [Lee 93, Lee 97]. They have shown that the discriminantly informative feature vectors have a component that is normal to the decision surface at least at one point on the surface. Furthermore, the less informative vectors are orthogonal to a vector normal to the decision surface at every point of the surface. This is natural, because vectors that do not have a component normal to the decision surface cannot cross it (and hence change classes) whatever their value is. Based on this observation, they estimate normal vectors to the decision boundary, using gradient approximation techniques, which are then used to formulate an appropriate eigenvalue–eigenvector problem leading to the computation of the transformation matrix.

Finally, pruning a neural network is a form of feature selection integrated into the classifier design stage. Indeed, the weights of the input nodes corresponding to less important features are expected to be small. As discussed in Chapter 4, the incorporation of appropriate regularization terms in the cost function encourages such weights to converge to zero and ultimately to be eliminated. This approach was followed, for example, in [Seti 97].

5.9 A HINT ON THE VAPNIK–CHERNOVENKIS LEARNING THEORY

So far in this book, two major issues have occupied us: the design of the classifier and its generalization capabilities. The former involved two stages: the choice of the classifier type and the choice of the optimality criterion. The generalization capabilities led us to seek ways to reduce the feature space dimensionality. In this section we will point out some important theoretical results that relate the size N of the training data set and the generalization performance of the designed classifier.

To this end, let us summarize a few necessary basic steps and definitions.

- Let $\mathcal{F}$ be the set of all the functions f that can be realized by the adopted classifier scheme. For example, if the classifier is a multilayer perceptron with a given number of neurons, then $\mathcal{F}$ is the set of all the functions that can be realized by the specific network structure. Functions f are mappings from $\mathcal{R}^l \rightarrow \{0, 1\}$. Thus, the response is either 1 or 0; that is, the two-class problem is considered and the mapping is either $f(\boldsymbol{x}) = 1$ or $f(\boldsymbol{x}) = 0$.
- Let $P_e^N(f)$ be the *empirical* classification error probability, based on the available input–desired output training pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, N$, which are considered to be independent and identically distributed (i.i.d.). Thus, $P_e^N(f)$ is the fraction of training samples for which an error occurs, that is, $f(\boldsymbol{x}_i) \neq y_i$. Obviously, this depends on the specific function f and the size N. The optimal function that results from minimizing this empirical cost is denoted by f^* and belongs to the set $\mathcal{F}$.

- $P_e(f)$ is the true classification error probability when a function f is realized. The corresponding empirical $P_e^N(f)$ can be very small, even zero, since a classifier can be designed to classify all training feature vectors correctly. However, $P_e(f)$ is the important performance measure, because it measures error probability based on the statistical nature of the data and not on the specific training set only. *For a classifier with good generalization capabilities, we expect the empirical and the true error probabilities to be close.*
- P_e denotes the minimum error probability over all the functions of the set, that is, $P_e = \min_{f \in \mathcal{F}} P_e(f)$.[1] Again, in practice we would like the optimal empirical error $P_e^N(f^*)$ to be close to P_e.

The Vapnik–Chernovenkis theorem is as follows.

Theorem. *Let $\mathcal{F}$ be the class of functions of the form $\mathcal{R}^l \rightarrow \{0, 1\}$. Then the empirical and true error probabilities corresponding to a function f in the class, satisfy*

$$prob\Big\{ \max_{f \in \mathcal{F}} |P_e^N(f) - P_e(f)| > \epsilon \Big\} \leq 8\mathcal{S}(\mathcal{F}, N) \exp(-N\epsilon^2/32) \qquad (5.33)$$

The term $\mathcal{S}(\mathcal{F}, N)$ is called the *shatter* coefficient of the class $\mathcal{F}$. This is defined as the *maximum* number of dichotomies of N points that can be formed by the functions in $\mathcal{F}$. From our combinatorics basics we know that the maximum number of dichotomies on a set of N points (separating them into two distinct subsets) is 2^N. However, not all these combinations can be implemented by a function $f : \mathcal{R}^l \rightarrow \{0, 1\}$. For example, we know that the set of functions realized by a perceptron (hyperplane) can form only 14 distinct dichotomies on 4 points out of the $16 = 2^4$ possibilities. The two XOR combinations cannot be realized. However, the class of functions realized by the perceptron can form all possible $8 = 2^3$ dichotomies for $N = 3$ points. This leads us to the following definition

Definition. *The largest integer $k \geq 1$ for which $\mathcal{S}(\mathcal{F}, k) = 2^k$ is called the Vapnik–Chernovenkis or VC dimension of the class $\mathcal{F}$ and is denoted by V_c. If $\mathcal{S}(\mathcal{F}, N) = 2^N$ for every N, then the VC dimension is infinite.*

Thus, in the two dimensional space the VC dimension of a single perceptron is 3. In the general l-dimensional space case the VC dimension of a perceptron is $l + 1$, as is easily verified from Section 4.13. It will not come as a surprise to say that the VC dimension and the shatter coefficient are related, because they have common origins. Indeed, this is true. It turns out that if the VC dimension is finite, then the

[1] Strictly speaking, in this section inf must be used instead of min and sup instead of max.

following bound is valid

$$S(\mathcal{F}, N) \leq N^{V_c} + 1 \tag{5.34}$$

That is, the shatter coefficient is either 2^N or is bounded as given in (5.34). This bound has a very important implication for the usefulness of (5.33). Indeed, for finite VC dimensions (5.34) guarantees that for large enough N the shatter coefficient is bounded by *polynomial growth*. Then the bound in (5.33) is dominated by its exponential decrease and it tends to zero as $N \to \infty$. In words, *for large N the probability of having large differences between the empirical and the true probability errors is very small! Thus, the network guarantees good generalization performance for large N*. Furthermore, the theory guarantees another strong result [Devr 96]

$$\text{prob}\left\{ P_e(f^*) - \min_{f \in \mathcal{F}} P_e(f) > \epsilon \right\} \leq 8S(\mathcal{F}, N) \exp(-N\epsilon^2/128) \tag{5.35}$$

That is, for large N we expect with high probability the performance of the empirical error optimal classifier to be close to the performance of the optimal one, over the specific class of functions.

Let us look at these theoretical results from a more intuitive perspective. Consider two different networks with VC dimensions $V_{c1} \ll V_{c2}$. Then if we fix N and ϵ, we expect the first network to have much better generalization performance, because the bound in (5.33) will be much tighter. Thus, the probability that the respective empirical and true errors will differ more than the predetermined quantity will be much smaller. We can think of the VC dimension as an *intrinsic capacity of a network, and only if the number of training vectors exceeds this number sufficiently can we expect good generalization performance.*

Another related bound of particular interest to us that holds with a probability at least $1 - \rho$ is the following:

$$P_e(f) \leq P_e^N(f) + \phi\left(\frac{V_c}{N}\right) \tag{5.36}$$

where V_c is the VC dimension of the corresponding class and

$$\phi\left(\frac{V_c}{N}\right) \equiv \sqrt{\frac{V_c\left(\ln\left(\frac{2N}{V_c} + 1\right)\right) - \ln(\rho/4)}{N}} \tag{5.37}$$

The interested reader may obtain more bounds and results concerning the Vapnik–Chernovenkis theory from [Devr 96, Vapn 95]. It will take some effort, but it is worth it! In some of the published literature the constants in the bounds are different. This depends on the way the bounds are derived. However, this is not of major practical importance, since the essence of the theory remains the same.

Due to the importance of the VC dimension, efforts have been made to compute it for certain classes of networks. In [Bau 89] it has been shown that the VC dimension of a multilayer perceptron with hard limiting activation functions in the nodes is bounded by

$$2\left\lfloor \frac{K_n^h}{2} \right\rfloor l \leq V_c \leq 2K_w \log_2(eK_n) \tag{5.38}$$

where K_n^h is the total number of hidden layer nodes, K_n the total number of nodes, K_w the total number of weights, l the input space dimension, e the base of the natural logarithm, and $\lfloor \cdot \rfloor$ the floor operator that gives the largest integer less than its argument. The lower bound holds only for networks with a single hidden layer and full connectivity between the layers. A similar upper bound is true for RBF networks too. Looking at this more carefully, one can say that for such networks the VC dimension is roughly given by the number of weights of the network, that is, the number of its free parameters to be determined! In practice, good generalization performance is expected if the number of training samples is a few times the VC dimension. A good rule of thumb is to choose N to be of the order of 10 times the VC dimension or more [Hush 93].

Besides the Vapnik–Chernovenkis theory, the published literature is rich in results concerning aspects of designing various classifiers using a finite data set N. Although they lack the elegance of the generality of the Vapnik–Chernovenkis theory, they provide further insight into this important task. For example, in [Raud 91] asymptotic analytic results are derived for a number of classifiers (linear, quadratic, etc.) under the Gaussian assumption of the involved densities. The classification error probability of a classifier designed using a finite set of N training samples is larger, by an amount Δ_N, than the error of the same classifier designed using an infinite ($N \rightarrow \infty$) set of data. It is shown that the mean of Δ_N (over different design sets) decreases as N tends to infinity. The rate of decrease depends on the specific type of classifier, on the dimensionality of the problem, and also on the value of the asymptotic ($N \rightarrow \infty$) error. It turns out that in order to keep the mismatch Δ_N within certain limits, *the number N of design samples must be a number of times larger than the dimension l*. Depending on the type of classifier, this proportionality constant can range from low values (e.g., 1.5) up to a few hundred! Furthermore, in [Fuku 90] it is shown that keeping N constant and increasing l, beyond a point, results in an increase of the classification error. This is also known as the *Hughes phenomenon*.

All these theoretical results provide useful guidelines in selecting appropriate values for N and l for certain types of classifiers. Moreover, they *make crystal clear the urge to keep the number of features as small as possible with respect to N and the importance of the feature selection stage in the design of a classification system*. In the fringes of this theoretical "happening," a substantial research effort

has also been devoted to experimental comparisons, involving different classifiers, with respect to their generalization capabilities; see, for example, [Mama 96] and the references therein. In practice, however, experience and empirical results retain an important part in the final decision. Engineering still has a flavor of art!

Structural Risk Minimization

In our discussion so far, we have focused on the effects of the finite size of the training data set, N, for a given class of functions, that is, a given classifier structure. Let us now touch on another important issue. If we allow N to grow indefinitely, does this luxury provide us with the means not only to have good generalization properties but also to improve our classification error so as to approach the optimal Bayesian performance? Recall that as N grows, we can expect to obtain the optimal performance with respect to all *allowable* sets of classifiers that can be implemented by the chosen network structure. However, the error rate of the corresponding optimal classifier may still be very far from that of the Bayesian classifier. Let us denote by P_B the Bayesian error probability. Then we can write

$$P_e(f^*) - P_B = \big(P_e(f^*) - P_e\big) + (P_e - P_B) \tag{5.39}$$

The right-hand side consists of two conflicting terms. If the class $\mathcal{F}$ is too small, then the first term is expected to be small but the second term is likely to be large. If, on the other hand, the class of functions $\mathcal{F}$ is large, then the second term is expected to be small but the first term is probably large. This is natural, because the larger the set of functions, the higher the probability of including in the set a good approximation of the Bayesian classifier. Moreover, the smaller the class, the less the variation between its members. This reminds us of the bias–variance dilemma we discussed in Chapter 3. A little thought suffices to reveal that the two problems are basically the same, seen from a different viewpoint. Then the natural question arises once more, can we make both terms small and how? The answer is that this is possible only asymptotically, provided that *at the same time* the size of the class $\mathcal{F}$ grows appropriately. An elegant strategy to achieve this has been suggested by Vapnik and Chernovenkis [Vapn 82].

Let $\mathcal{F}^{(1)}, \mathcal{F}^{(2)}, \ldots$ be a sequence of nested classes of functions, that is,

$$\mathcal{F}^{(1)} \subset \mathcal{F}^{(2)} \subset \mathcal{F}^{(3)} \subset \cdots \tag{5.40}$$

with an increasing, yet finite, VC dimension,

$$V_{c,\mathcal{F}^{(1)}} \leq V_{c,\mathcal{F}^{(2)}} \leq V_{c,\mathcal{F}^{(3)}} \leq \cdots \tag{5.41}$$

Also let

$$\lim_{i \to \infty} \inf_{f \in \mathcal{F}^{(i)}} P_e(f) = P_B \tag{5.42}$$

For each N and class of functions $\mathcal{F}^{(i)}$, $i = 1, 2, \ldots$, compute the optimum, $f^*_{N,i}$, with respect to the *empirical* error using the N training pairs of input–output samples. Vapnik and Chernovenkis suggest choosing for each N the function f^*_N according to the *structural risk minimization principle* (SRM). This consists of the following two steps. First we select the classifier $f^*_{N,i}$ from every class $\mathcal{F}^{(i)}$ that minimizes the corresponding empirical error over the class of functions. Then, from all these classifiers, we choose the one that minimizes the upper bound in (5.36), over all i. More precisely, form the so-called *guaranteed error bound*,

$$\tilde{P}_e(f^*_{N,i}) \equiv P_e^N(f^*_{N,i}) + \phi\left(\frac{V_{c,\mathcal{F}^{(i)}}}{N}\right) \tag{5.43}$$

and choose

$$f^*_N = arg \min_i \tilde{P}_e(f^*_{N,i}) \tag{5.44}$$

Then, as $N \to \infty$, $P_e(f^*_N)$ tends to P_B with probability one. Note that the second term in the minimized bound, $\phi\left(\frac{V_{c,\mathcal{F}^{(i)}}}{N}\right)$, is a *complexity penalty term* that increases as the network complexity increases (i.e., with the size of the class of functions and $V_{c,\mathcal{F}^{(i)}}$). If the classifier model is too simple, the penalty term is small but the empirical error term will be large in (5.43). On the other hand, if the model is complex, the empirical error is small but the penalty term large. The structural risk minimization criterion aims at achieving the best trade-off between these two terms.

From this point of view, the structural risk minimization principle belongs to a more general class of approaches that try to estimate the order of a system, by considering simultaneously the model complexity and a performance index. Depending on the function used to measure the model complexity and the corresponding performance index, different criteria result. For example, in [Akai 74] the place of the empirical error is taken by the value of the loglikelihood function, corresponding to the maximum likelihood estimates of the unknown parameters and the complexity term is proportional to the number of the free parameters to be estimated. A number of alternatives have also been suggested, see, for example, [Riss 83, Mood 92, Leth 96, Wang 98].

Remarks

- The SRM procedure provides a theoretical guideline for constructing a classifier that converges asymptotically to the optimal Bayesian one. However, the bound in (5.36) which is exploited in order to reach this limit, must not be misinterpreted. For any bound to be useful in practice, one needs an extra piece of information. *Is this bound loose or tight?* In general, until now, there has been no result that provides this information. We can construct

classifiers whose VC dimension is large, yet their performance can be good. A typical example is the nearest neighbor (NN) classifier. Its VC dimension is infinite. Indeed, since we know the class label of all the N training points, the NN classifier classifies correctly all training points and the corresponding shatter coefficient is 2^N. Yet, it is generally accepted that the generalization performance of this classifier can be quite good in practice. In contrast, one can build a classifier with finite VC dimension, yet whose performance is always bad ([Burg 98]). Concluding this remark, we have to keep in mind that *if two classifiers have the same empirical error, it does not, necessarily, mean that the one with the smaller VC dimension leads to better performance.*

Support Vector Machines: A Last Touch

We have already discussed that the VC dimension of a linear classifier in the l-dimensional space is $l + 1$. However, the VC dimension of hyperplanes that are *constrained* to leave the maximum margin between the classes *may have a smaller VC dimension.*

Let us assume that r is the radius of the *smallest* (hyper)sphere that encloses all the data (Problem 5.20), i.e.,

$$\|\boldsymbol{x}_i\| \leq r, \ i = 1, 2, \ldots, N$$

Then if a hyperplane satisfies the conditions in Equation (3.61) and

$$\|\boldsymbol{w}\|^2 \leq c$$

where c is a constant, then its VC dimension, V_c, is bounded by ([Vapn 98])

$$V_c \leq \min(r^2 c, l) + 1 \tag{5.45}$$

That is, the capacity of the classifier can be *controlled independently of the dimensionality* of the feature space. This is very interesting indeed. It basically states that the capacity of a classifier may not, necessarily, be related to the number of unknown parameters! This is a more general result. To emphasize it further, note that it is possible to construct a classifier with only one free parameter, yet with infinite VC dimension; see, for example, [Burg 98]. Let us now consider a sequence of bounds

$$c_1 < c_2 < c_3 < \ldots$$

This defines the following sequence of classifiers:

$$\mathcal{F}^i : \left\{ \boldsymbol{w}^T \boldsymbol{x} + w_0 : \|\boldsymbol{w}\|^2 \leq c_i \right\} \tag{5.46}$$

where

$$\mathcal{F}^i \subset \mathcal{F}^{i+1}$$

If the classes are separable, then the empirical error is zero. Thus minimizing the norm $\|\boldsymbol{w}\|$ is equivalent with minimizing the VC dimension (to be fair, the upper bound of the VC dimension). Thus, we can conclude that, the design of an SVM classifier senses the spirit of the SRM principle. Hence, keeping the VC dimension minimum suggests that we can expect support vector machines to exhibit good generalization performance. More on these issues can be found in [Vapn 98, Burg 98].

Problems

5.1 If x_i, y_i, $i = 1, 2, \ldots N$ are independent samples of two Gaussian distributions of the same variance σ^2, show that the random variable $\frac{(2N-2)s_z^2}{\sigma^2}$, where

$$s_z^2 = \frac{1}{2N-2}\left(\sum_{i=1}^{N}(x_i - \bar{x})^2 + \sum_{i=1}^{N}(y_i - \bar{y})^2\right)$$

where $\bar{x}$, $\bar{y}$ are the respective sample mean values, is chi-square distributed with $2N-2$ degrees of freedom.

5.2 Let N_1, N_2 be the available values of a feature in two classes, respectively. The feature is assumed to follow a Gaussian distribution with the same variance in each class. Define the test statistic

$$q = \frac{(\bar{x} - \bar{y}) - (\mu_1 - \mu_2)}{s_z\sqrt{\frac{1}{N_1} + \frac{1}{N_2}}} \tag{5.47}$$

where

$$s_z^2 = \frac{1}{N_1 + N_2 - 2}\left(\sum_{i=1}^{N_1}(x_i - \bar{x})^2 + \sum_{i=1}^{N_2}(y_i - \bar{y})^2\right)$$

and μ_1, μ_2 are the respective true mean values. Show that q follows the t-distribution with $N_1 + N_2 - 2$ degrees of freedom.

5.3 Show that the matrix

$$A = \begin{bmatrix} \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \cdots & \frac{1}{\sqrt{n}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \cdots & 0 \\ \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{-1}{\sqrt{n(n-1)}} & \frac{-1}{\sqrt{n(n-1)}} & \frac{-1}{\sqrt{n(n-1)}} & \cdots & \frac{n-1}{\sqrt{n(n-1)}} \end{bmatrix}$$

is orthogonal, that is, $AA^T = I$.

5.4 Show that if $x_i, i = 1, 2, \ldots, l$, are jointly Gaussian, then the l variables $y_i, i = 1, 2, \ldots, l$, resulting from a linear transformation of them are also jointly Gaussian. Furthermore, if x_i are mutually independent and the transformation is orthogonal, then y_i are also mutually independent and Gaussian.

5.5 Let $\omega_i, i = 1, 2, \ldots, M$, be the classes for a classification task. Divide the interval of the possible values of a feature into subintervals $\Delta_j, j = 1, 2, \ldots, K$. If $P(\Delta_j)$ is the probability of having values in the respective subinterval and $P(\omega_i|\Delta_j)$ the probability of occurrence of ω_i in this interval, show that the so-called *ambiguity function*

$$A = -\sum_i \sum_j P(\Delta_j) P(\omega_i|\Delta_j) \log_M (P(\omega_i|\Delta_j))$$

is equal to 1 for completely overlapped distributions and is equal to 0 for perfectly separated ones. For all other cases it takes intermediate values. Thus, it can be used as a distribution overlap criterion [Fine 83].

5.6 Show that if $d_{ij}(x_1, x_2, \ldots, x_m)$ is the class divergence based on m features, adding a new one x_{m+1} cannot decrease the divergence, that is,

$$d_{ij}(x_1, x_2, \ldots, x_m) \leq d_{ij}(x_1, x_2, \ldots, x_m, x_{m+1})$$

5.7 Show that if the density functions are Gaussians in both classes with the same covariance matrix Σ, then on adding a new feature x_{m+1} to the feature vector the new divergence is recursively computed by

$$d_{ij}(x_1, \ldots, x_{m+1}) = d_{ij}(x_1, \ldots, x_m) + \frac{[(\mu_i - \mu_j) - (\boldsymbol{\mu_i} - \boldsymbol{\mu_j})^T \Sigma^{-1} \boldsymbol{r}]^2}{\sigma^2 - \boldsymbol{r}^T \Sigma^{-1} \boldsymbol{r}}$$

where μ_i, μ_j are the mean values of x_{m+1} for the two classes, σ^2 is its variance, $\boldsymbol{r}$ is its cross-covariance vector with the other elements of $\boldsymbol{x}$, and $\boldsymbol{\mu_i}, \boldsymbol{\mu_j}$ are the mean vectors of $\boldsymbol{x}$ prior to x_{m+1}. If x_{m+1} is now uncorrelated with the previously selected features $x_1, \ldots, x_m$, then this becomes

$$d_{ij}(x_1, \ldots, x_{m+1}) = d_{ij}(x_1, \ldots, x_m) + \frac{(\mu_i - \mu_j)^2}{\sigma^2}$$

5.8 Show that if the features are statistically independent, then the divergence is given by

$$d_{ij}(x_1, x_2, \ldots, x_l) = \sum_{i=1}^{l} d_{ij}(x_i)$$

5.9 Show that in the case of Gaussian distributions the Chernoff bound becomes

$$\epsilon_{CB} = \exp(-b(s))$$

where

$$b(s) = \frac{s(1-s)}{2}(\mu_i - \mu_j)^T[s\Sigma_j + (1-s)\Sigma_i]^{-1}(\mu_i - \mu_j) + \frac{1}{2}\ln\frac{|\, s\Sigma_j + (1-s)\Sigma_i \,|}{|\, \Sigma_j \,|^s |\, \Sigma_i \,|^{1-s}}$$

Then take the derivative with respect to s and show that for equal covariance matrices the optimum is achieved for $s = 1/2$. Thus, in this case $b(s)$ equals the Bhattacharyya distance.

5.10 Show that the mixture scatter matrix is the sum of the within-class and between-class scatter matrices.

5.11 Show that the cross-correlation coefficient in (5.21) lies in the interval $[-1, 1]$.
Hint: Use Schwartz's inequality $|x^T y| \leq \|x\|\|y\|$.

5.12 Show that for a two-class problem and Gaussian distributed feature vectors, with the same covariance matrix in the two classes, which are assumed equiprobable, the divergence is equal to

$$\text{trace}\{S_w^{-1} S_b\}$$

5.13 Show that the number of combinations to be searched using the backward search technique is given by

$$1 + 1/2((m+1)m - l(l+1))$$

5.14 Show that

$$\frac{\partial}{\partial A}\text{trace}\{(A^T S_1 A)^{-1}(A^T S_2 A)\} = -2S_1 A(A^T S_1 A)^{-1}(A^T S_2 A)(A^T S_1 A)^{-1} + 2S_2 A(A^T S_1 A)^{-1}$$

5.15 Show that for an M-class problem the matrix S_b is of rank $M - 1$.
Hint: Recall that $\mu_0 = \sum_i P_i \mu_i$.

5.16 Show that if $f_i(x), i = 1, \ldots, M$, are the discriminant functions of an M-class problem, we can construct from them $M - 1$ new functions that are, in principle, sufficient for the classification.
Hint: Consider the differences $f_i(x) - f_j(x)$.

5.17 Show that for a two-class problem the nonzero eigenvalue of matrix $S_w^{-1} S_b$ is equal to

$$\lambda_1 = P_1 P_2(\mu_1 - \mu_2)^T S_{xw}^{-1}(\mu_1 - \mu_2)$$

and the corresponding eigenvector

$$v_1 = S_{xw}^{-1}(\mu_1 - \mu_2)$$

where P_1, P_2 are the respective class probabilities.

5.18 Show that if matrices S_1, S_2 are two covariance matrices, then the eigenvectors of $S_1^{-1} S_2$ are orthogonal with respect to S_1, that is,

$$\boldsymbol{v}_i^T S_1 \boldsymbol{v}_j = \delta_{ij}$$

Hint: Use the fact that S_1, S_2 can be simultaneously diagonalized (Appendix B).

5.19 Show that in a multilayer perceptron with a linear output node, minimizing the squared error is equivalent with maximizing (5.32).

Hint: Assume the weights of the nonlinear nodes fixed and compute first the LS optimal weights driving the linear output nodes. Then substitute these values into the sum of error squares cost function.

5.20 Compute the minimal enclosure (hyper)sphere, i.e., radius as well as its center, of a set of points $\boldsymbol{x}_i$, $i = 1, 2, \ldots, N$.

References

[Akai 74] Akaike H. "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, Vol. 19(6), pp. 716–723, 1974.

[Bati 94] Batiti R. "Using mutual information for selecting features in supervised neural network learning," *IEEE Transactions on Neural Networks*, Vol. 5(8), pp. 537–550, 1994.

[Bau 89] Baum E.B., Haussler D. "What size net gives valid generalization," *Neural Computation*, Vol. 1(1), pp. 151–160, 1989.

[Bish 95] Bishop C. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[Bour 88] Bourland H., Kamp Y. "Auto-association by multilayer perceptrons and singular value decomposition," *Biological Cybernetics*, Vol. 59, pp. 291–294, 1988.

[Brun 00] Brunzell H., Eriksson J. "Feature reduction for classification of multidimensional data," *Pattern Recognition*, Vol. 33, pp. 1741–1748, 2000.

[Burg 98] Burges C.J.C. "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, Vol. 2(2), pp. 1–47, 1998.

[Chat 97] Chatterjee C., Roychowdhury V. "On self-organizing algorithms and networks for class-separability features," *IEEE Transactions on Neural Networks*, Vol. 8(3), pp. 663–678, 1997.

[Devi 82] Devijver P.A., Kittler J. *Pattern Recognition; A Statistical Approach*, Englewood Cliffs, NJ: Prentice Hall, 1982.

[Devr 96] Devroye L., Gyorfi L., Lugosi G. *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag, 1996.

[Fine 83] Finette S., Bleier A., Swindel W. "Breast tissue classification using diagnostic ultrasound and pattern recognition techniques: I. Methods of pattern recognition," *Ultrasonic Imaging*, Vol. 5, pp. 55–70, 1983.

[Fras 58] Fraser D.A.S. *Statistics: An Introduction*, John Wiley, 1958.

[Fuku 90] Fukunaga K. *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press, 1990.

[Ghah 94] Ghaharamani Z., Jordan M.I. "Supervised learning from incomplete data via the EM approach," in *Advances in Neural Information Processing Systems* (Cowan J.D.,

Tesauro G.T., Alspector J., eds), Vol. 6, pp. 120–127, Morgan Kaufmann, San Mateo, CA, 1994.

[Hube 81] Huber P.J. *Robust Statistics*, John Wiley, 1981.

[Hush 93] Hush D.R., Horne B.G. "Progress in supervised neural networks," *Signal Processing Magazine*, Vol. 10(1), pp. 8–39, 1993.

[Jain 97] Jain A., Zongker D. "Feature selection: Evaluation, application, and small sample performance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(2), pp. 153–158, 1997.

[Kitt 78] Kittler J. "Feature set search algorithms," in *Pattern Recognition and Signal Processing* (Chen C.H., ed.), pp. 41–60, Sijthoff and Noordhoff, Alphen aan den Rijn, The Netherlands, 1978.

[Kram 91] Kramer M.A. "Nonlinear principal component analysis using auto-associative neural networks," *AIC Journal*, Vol. 37(2), pp. 233–243, 1991.

[Kulb 51] Kullback S., Liebler R.A. "On information and sufficiency," *Annals of Mathematical Statistics*, Vol. 22, pp. 79–86, 1951.

[Kwak 02] Kwak N., Choi C.-H. "Input feature selection for classification problems," *IEEE Transactions on Neural Networks*, Vol. 13(1), pp. 143–159, 2002.

[Lawe 66] Lawer E.L., Wood D.E. "Branch and bound methods: A survey," *Operational Research*, Vol. 149(4), 1966.

[Leth 96] Lethtokanga S.M., Saarinen J., Huuhtanen P., Kaski K. "Predictive minimum description length criterion for time series modeling with neural networks," *Neural Computation*, Vol. 8, pp. 583–593, 1996.

[Lee 93] Lee C., Landgrebe D.A. "Decision boundary feature extraction for nonparametric classifiers," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 23, pp. 433–444, 1993.

[Lee 97] Lee C., Landgrebe D. "Decision boundary feature extraction for neural networks," *IEEE Transactions on Neural Networks*, Vol. 8(1), pp. 75–83, 1997.

[Lowe 90] Lowe D., Webb A.R. "Exploiting prior knowledge in network optimization: An illustration from medical prognosis," *Network: Computation in Neural Systems*, Vol. 1(3), pp. 299–323, 1990.

[Lowe 91] Lowe D., Webb A.R. "Optimized feature extraction and the Bayes decision in feed-forward classifier networks," *IEEE Transactions in Pattern Analysis and Machine Intelligence*, Vol. 13(4), pp. 355–364, 1991.

[Mama 96] Mamamoto Y., Uchimura S., Tomita S. "On the behaviour of artificial neural network classifiers in high dimensional spaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18(5), pp. 571–574, 1996.

[Mao 95] Mao J., Jain A.K. "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Transactions on Neural Networks*, Vol. 6(2), pp. 296–317, 1997.

[Maus 90] Mausel P.W., Kramber W.J., Lee J.K. "Optimum band selection for supervised classification of multispectra data," *Photogrammetric Engineering and Remote Sensing* Vol. 56, pp. 55–60, 1990.

[Mood 92] Moody J.E. "The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems" in *Advances in Neural Computation* (Moody J.E., Hanson S.J., Lippman R.R., eds.), pp. 847–854, San Mateo, C.A., Morgan Kaufman, 1992.

[Papo 91] Papoulis A. *Probability Random Variables and Stochastic Processes*, 3rd ed., McGraw-Hill, 1991.

[Pudi 94] Pudil P., Novovicova J., Kittler J. "Floating search methods in feature selection," *Pattern Recognition Letters*, Vol. 15, pp. 1119–1125, 1994.

[Raud 91] Raudys S.J., Jain A.K. "Small size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13(3), pp. 252–264, 1991.

[Rich 95] Richards J. *Remote Sensing Digital Image Analysis*, 2nd ed., Springer-Verlag, 1995.

[Riss 83] Rissanen J. "A universal prior for integers and estimation by minimum description length," *The Annals of Statistics*, Vol. 11(2), pp. 416–431, 1983.

[Samm 69] Sammon J.W. "A nonlinear mapping for data structure analysis," *IEEE Transactions on Computers*, Vol. 18, pp. 401–409, 1969.

[Seti 97] Setiono R., Liu H. "Neural network feature selector," *IEEE Transactions on Neural Networks*, Vol. 8(3), pp. 654–662, 1997.

[Su 94] Su K.Y, Lee C.H. "Speech recognition using weighted HMM and subspace projection approaches," *IEEE Transactions on Speech and Audio Processing*, Vol. 2(1), pp. 69–79, 1994.

[Swai 73] Swain P.H., King R.C. "Two effective feature selection criteria for multispectral remote sensing," *Proceedings of the 1st International Conference on Pattern Recognition*, pp. 536–540, 1973.

[Tou 74] Tou J., Gonzalez R.C. *Pattern Recognition Principles*, Addison-Wesley, 1974.

[Yu 93] Yu B., Yuan B. "A more efficient branch and bound algorithm for feature selection," *Pattern Recognition*, Vol. 26(6), pp. 883–889, 1993.

[Walp 78] Walpole R.E., Myers R.H. *Probability and Statistics for Engineers and Scientists*, Macmillan, 1978.

[Wang 98] Wang Y., Adali T., Kung S.Y., Szabo Z. "Quantization and segmentation of brain tissues from MR images: A probabilistic neural network approach," *IEEE Transactions on Image Processing*, Vol. 7(8), 1998.

[Wata 97] Watanabe H., Yamaguchi T., Katagiri S. "Discriminative metric for robust pattern recognition," *IEEE Transactions on Signal Processing*, Vol. 45(11), pp. 2655–2663, 1997.

[Vapn 82] Vapnik V.N. *Estimation of Dependencies Based on Empirical Data*, Springer-Verlag, 1982.

[Vapn 95] Vapnik V.N. *The Nature of Statistical Learning Theory*, Springer-Verlag, 1995.

[Vapn 98] Vapnik, V.N. *Statistical Learning Theory*, John Wiley & Sons, 1998.

[Zhan 02] Zhang H., Sun G. "Feature selection using tabu search method," *Pattern Recognition*, Vol. 35, pp. 701–711, 2002.

CHAPTER 6

FEATURE GENERATION I: LINEAR TRANSFORMS

6.1 INTRODUCTION

The goal of this chapter is the generation of features via linear transforms of the input (measurement) samples. A number of transforms will be presented and reviewed. The basic concept is to transform a given set of measurements to a new set of features. If the transform is suitably chosen, transform domain features can exhibit high "*information packing*" properties compared with the original input samples. This means that most of the classification-related information is "squeezed" in a relatively small number of features, leading to a reduction of the necessary feature space dimension.

The basic reasoning behind transform-based features is that an appropriately chosen transform can exploit and remove information redundancies, which usually exist in the set of samples obtained by the measuring devices. Let us take for example an image resulting from a measuring device, for example, X-rays or a camera. The pixels (i.e., the input samples) at the various positions in the image have a large degree of correlation, due to the internal morphological consistencies of real-world images that distinguish them from noise. Thus, if one uses the pixels as features, there will be a large degree of redundant information. Alternatively, if one obtains the Fourier transform, for example, of a typical real-world image, it turns out that most of the energy lies in the low-frequency components, due to the high correlation between the pixels' gray levels. Hence, using the Fourier coefficients as features seems a reasonable choice, because the low-energy, high-frequency coefficients can be neglected, with little loss of information. In this chapter we will see that the Fourier transform is just one of the tools from a palette of possible transforms.

6.2 BASIS VECTORS AND IMAGES

Let $x(0), x(1), \ldots, x(N-1)$ be a set of input samples and $\boldsymbol{x}$ be the $N \times 1$ corresponding vector,

$$\boldsymbol{x}^T = [x(0), \ldots, x(N-1)]$$

Given a unitary $N \times N$ matrix A[1] we define the transformed vector $\boldsymbol{y}$ of $\boldsymbol{x}$ as

$$\boldsymbol{y} = A^H \boldsymbol{x} \equiv \begin{bmatrix} \boldsymbol{a}_0^H \\ \vdots \\ \boldsymbol{a}_{N-1}^H \end{bmatrix} \boldsymbol{x} \tag{6.1}$$

where "H" denotes the Hermitian operation, that is, complex conjugation and transposition. From (6.1) and the definition of unitary matrices we have

$$\boldsymbol{x} = A\boldsymbol{y} = \sum_{i=0}^{N-1} y(i)\boldsymbol{a}_i \tag{6.2}$$

The columns of A, $\boldsymbol{a}_i, i = 0, 1, \ldots, N-1$, are called the *basis vectors* of the transform. The elements $y(i)$ of $\boldsymbol{y}$ are nothing but the projections of $\boldsymbol{x}$ onto these basis vectors. Indeed, taking the inner product of $\boldsymbol{x}$ with $\boldsymbol{a}_j$ we have

$$\langle \boldsymbol{a}_j, \boldsymbol{x} \rangle \equiv \boldsymbol{a}_j^H \boldsymbol{x} = \sum_{i=0}^{N-1} y(i)\langle \boldsymbol{a}_j, \boldsymbol{a}_i \rangle = \sum_{i=0}^{N-1} y(i)\delta_{ij} = y(j) \tag{6.3}$$

This is due to the unitary property of A, that is, $A^H A = I$ or $\langle \boldsymbol{a}_i, \boldsymbol{a}_j \rangle = \boldsymbol{a}_i^H \boldsymbol{a}_j = \delta_{ij}$.

In many problems, such as in image analysis, the input set of samples is a two-dimensional sequence $X(i, j), i, j = 0, 1, \ldots, N-1$, defining an $N \times N$ matrix X instead of a vector. In such cases, one can define an equivalent N^2 vector $\boldsymbol{x}$, for example, by ordering the rows of the matrix one after the other (*lexicographic ordering*)

$$\boldsymbol{x}^T = [X(0,0), \ldots, X(0, N-1), \ldots, X(N-1, 0), \ldots, X(N-1, N-1)]$$

and then transform this equivalent vector. However, this is not the most efficient way to work. The number of operations required to multiply an $N^2 \times N^2$ square matrix (A) with an $N^2 \times 1$ vector $\boldsymbol{x}$ is of the order of $O(N^4)$, which is prohibitive for many applications. An alternative possibility is to transform matrix X via a set of *basis matrices* or *basis images*. Let U and V be unitary $N \times N$ matrices.

[1] A complex matrix is called unitary if $A^{-1} = A^H$. Real matrices are equivalently called *orthogonal* if $A^{-1} = A^T$.

Define the transformed matrix Y of X as

$$Y = U^H X V \tag{6.4}$$

or

$$X = U Y V^H \tag{6.5}$$

The number of operations is now reduced to $O(N^3)$. Equation (6.5) can alternatively be written (Problem 6.1) as

$$X = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} Y(i, j) \boldsymbol{u}_i \boldsymbol{v}_j^H \tag{6.6}$$

where $\boldsymbol{u}_i$ are the column vectors of U and $\boldsymbol{v}_j$ the column vectors of V. Each of the outer products $\boldsymbol{u}_i \boldsymbol{v}_j^H$ is an $N \times N$ matrix

$$\boldsymbol{u}_i \boldsymbol{v}_j^H = \begin{bmatrix} u_{i0} v_{j0}^* & \cdots & u_{i0} v_{jN-1}^* \\ \vdots & \vdots & \vdots \\ u_{iN-1} v_{j0}^* & \cdots & u_{iN-1} v_{jN-1}^* \end{bmatrix} \equiv \mathcal{A}_{ij}$$

and (6.6) is an expansion of matrix X in terms of these N^2 basis images (matrices). The $*$ denotes complex conjugation. Furthermore, if Y turns out to be diagonal, then (6.6) becomes

$$X = \sum_{i=0}^{N-1} Y(i, i) \boldsymbol{u}_i \boldsymbol{v}_i^H$$

and the number of basis images is reduced to N. An interpretation similar to (6.3) is also possible. To this end, let us define the inner product between two matrices as

$$\langle A, B \rangle \equiv \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} A^*(m, n) B(m, n) \tag{6.7}$$

Then it is not difficult to show that (Problem 6.1)

$$Y(i, j) = \langle \mathcal{A}_{ij}, X \rangle \tag{6.8}$$

In words, the (i, j) element of the transformed matrix results from multiplying each element of X by the conjugate of the corresponding element of $\mathcal{A}_{ij}$ and summing up all products.

Transformations of the type (6.4) are also known as *separable* (Problem 6.2). The reason is that one can look at them as a succession of one-dimensional transforms, first applied on column vectors and then on row vectors. For example, the

intermediate result in (6.4), $Z = U^H X$, is equivalent to N transforms applied to the column vectors of X, and $(U^H X)V = (V^H Z^H)^H$ is equivalent to a second sequence of N transforms acting upon the rows of Z. All the two-dimensional transforms that we will deal with in this chapter are separable ones.

Example 6.1. Given the image X and the orthogonal transform matrix U

$$X = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}, \quad U = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

the transformed image $Y = U^T X U$ is

$$Y = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & -1 \\ -1 & 0 \end{bmatrix}$$

The corresponding basis images are

$$\mathcal{A}_{00} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} [1,1] = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\mathcal{A}_{11} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \end{bmatrix} [1,-1] = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

and similarly

$$\mathcal{A}_{01} = \mathcal{A}_{10}^T = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

Now verify that the elements of Y are obtained via the matrix inner products $\langle X, \mathcal{A}_{ij} \rangle$.

6.3 THE KARHUNEN–LOÈVE TRANSFORM

Let $\boldsymbol{x}$ be the vector of input samples. In the case of an image array, $\boldsymbol{x}$ may be formed by lexicographic ordering of the array elements. We have already mentioned in a number of places in this book that a desirable property of the generated features is to be mutually uncorrelated in order to avoid information redundancies. The goal of this section is to generate features that are optimally uncorrelated, that is, $E[y(i)y(j)] = 0, i \neq j$. Let[2]

$$\boldsymbol{y} = A^T \boldsymbol{x} \tag{6.9}$$

From the definition of the correlation matrix we have

$$R_y \equiv E[\boldsymbol{y}\boldsymbol{y}^T] = E[A^T \boldsymbol{x}\boldsymbol{x}^T A] = A^T R_x A \tag{6.10}$$

However, R_x is a symmetric matrix, and hence its eigenvectors are mutually orthogonal (Appendix B). Thus, if matrix A is chosen so that its columns are the

[2]We deal with real data. The complex case is a straightforward extension.

orthonormal eigenvectors $\boldsymbol{a}_i, i = 0, 1, \ldots, N-1$, of R_x, then R_y is diagonal (Appendix B)

$$R_y = A^T R_x A = \Lambda \tag{6.11}$$

where Λ is the diagonal matrix having as elements on its diagonal the respective eigenvalues $\lambda_i, i = 0, 1, \ldots, N-1$, of R_x. Furthermore, assuming R_x to be positive definite (Appendix B) the eigenvalues are positive. The resulting transform is known as the *Karhunen–Loève (KL)* transform, and it achieves our original goal of generating mutually uncorrelated features. The KL transform is of fundamental significance in pattern recognition and in a number of signal and image processing applications. Let us look at some of its important properties.

Mean square error approximation. From Eqs. (6.2) and (6.3) we have

$$\boldsymbol{x} = \sum_{i=0}^{N-1} y(i)\boldsymbol{a}_i \quad \text{and} \quad y(i) = \boldsymbol{a}_i^T \boldsymbol{x} \tag{6.12}$$

Let us now define a new vector in the m-dimensional subspace

$$\hat{\boldsymbol{x}} = \sum_{i=0}^{m-1} y(i)\boldsymbol{a}_i \tag{6.13}$$

where only m of the basis vectors are involved. Obviously, this is nothing but the projection of $\boldsymbol{x}$ onto the subspace spanned by the m (orthonormal) eigenvectors involved in the summation. If we try to approximate $\boldsymbol{x}$ by its projection $\hat{\boldsymbol{x}}$, the resulting mean square error is given by

$$E\left[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2\right] = E\left[\left\|\sum_{i=m}^{N-1} y(i)\boldsymbol{a}_i\right\|^2\right] \tag{6.14}$$

Our goal now is to choose the eigenvectors that result in the minimum MSE. From (6.14) and taking into account the orthonormality property of the eigenvectors, we have

$$E\left[\left\|\sum_{i=m}^{N-1} y(i)\boldsymbol{a}_i\right\|^2\right] = E\left[\sum_i \sum_j (y(i)\boldsymbol{a}_i^T)(y(j)\boldsymbol{a}_j)\right] \tag{6.15}$$

$$= \sum_{i=m}^{N-1} E[y^2(i)] = \sum_{i=m}^{N-1} \boldsymbol{a}_i^T E[\boldsymbol{x}\boldsymbol{x}^T]\boldsymbol{a}_i \tag{6.16}$$

Combining this with (6.14) and the eigenvector definition, we finally get

$$E\left[\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|^2\right]=\sum_{i=m}^{N-1}\boldsymbol{a}_i^T\lambda_i\boldsymbol{a}_i=\sum_{i=m}^{N-1}\lambda_i \tag{6.17}$$

Thus, if we choose in (6.13) the eigenvectors corresponding to the *m largest* eigenvalues of the *correlation matrix*, then the error in (6.17) is *minimized, being the sum of the $N-m$ smallest eigenvalues. Furthermore, it can be shown (Problem 6.3) that this is also the minimum MSE, compared with any other approximation of $\boldsymbol{x}$ by an m-dimensional vector*. This is the reason that the KL transform is also known as Principal Component Analysis (PCA).

A different form of the KL transform results if we compute A in terms of the eigenvectors of the covariance matrix. This transform diagonalizes the covariance matrix Σ_y,

$$\Sigma_y=A^T\Sigma_x A=\Lambda \tag{6.18}$$

In general, the two are different and coincide for zero mean random vectors. In practice this is usually the case, because if it is not true one can replace each vector by $\boldsymbol{x}-E[\boldsymbol{x}]$. Despite that, it is still interesting to point out a difference between the two variants of the KL transform. It can be shown (Problem 6.4) that in this case, the resulting orthonormal basis (Σ_x eigenvectors $\hat{a}_i$) guarantees that the mean square error between $\boldsymbol{x}$ and its approximation given by

$$\hat{\boldsymbol{x}}=\sum_{i=0}^{m-1}y(i)\hat{\boldsymbol{a}}_i+\sum_{i=m}^{N-1}E[y(i)]\hat{\boldsymbol{a}}_i,\quad y(i)\equiv\hat{\boldsymbol{a}}_i^T\boldsymbol{x} \tag{6.19}$$

is minimum. In words, the last $N-m$ components are not random but are frozen to their respective mean values.

The optimality of the KL transform, with respect to the MSE approximation, leads to excellent information packing properties and offers us a tool to select the m dominant features out of N measurement samples. However, although this may be a good criterion, in many cases it does not necessarily lead to maximum class separability in the lower dimensional subspace. This is reasonable, since the dimensionality reduction is not optimized with respect to class separability, as was, for example, the case with the scattering matrix criteria of the previous chapter. This is demonstrated via the example of Figure 6.1. The feature vectors in the two classes follow the Gaussian distribution with the same covariance matrix. The ellipses show the curves of constant pdf values. We have computed the eigenvectors of the overall correlation matrix, and the resulting eigenvectors are shown in the figure. Eigenvector $\boldsymbol{a}_1$ is the one that corresponds to the largest eigenvalue. It does not take time for someone to realize that projection on $\boldsymbol{a}_1$ makes the two classes almost coincide. However, projecting on $\boldsymbol{a}_2$ keeps the two class separable.

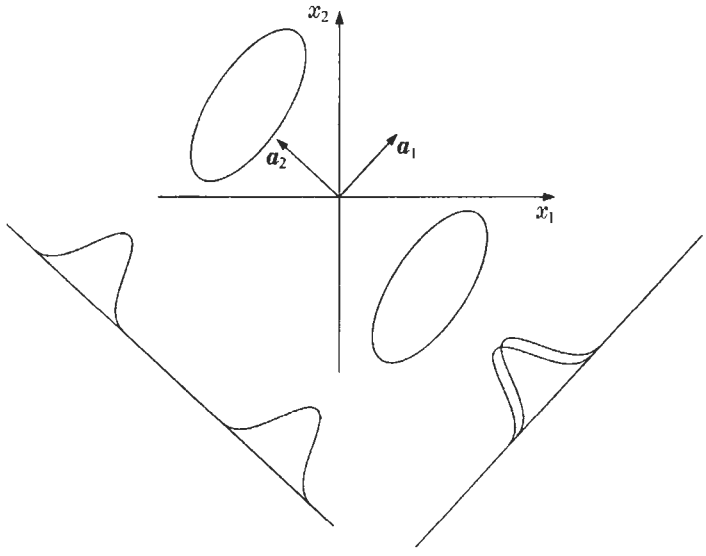


FIGURE 6.1: The KL transform is not always best for pattern recognition. In this example, projection on the eigenvector with the larger eigenvalue makes the two classes coincide. On the other hand, projection on the other eigenvector keeps the classes separated.

Total variance. Let $E[\boldsymbol{x}]$ be zero. If this is not the case, the mean can always be subtracted. Let $\boldsymbol{y}$ be the KL transformed vector of $\boldsymbol{x}$. From the respective definitions we have that $\sigma^2_{y(i)} \equiv E[y^2(i)] = \lambda_i$. That is, *the eigenvalues of the input correlation matrix are equal to the variances of the transformed features.* Thus, selecting those features, $y(i) \equiv \boldsymbol{a}_i^T \boldsymbol{x}$, corresponding to the m largest eigenvalues makes their sum variance $\sum_i \lambda_i$ maximum. In other words, the selected m features retain most of the total variance associated with the original random variables $x(i)$. Indeed, the latter is equal to the trace of R_x, which we know from linear algebra to be equal to the sum of the eigenvalues $\sum_{i=0}^{N-1} \lambda_i$ [Stra 80]. It can be shown that this is a more general property. That is, from all possible sets of m features, obtained via any orthogonal linear transformation on $\boldsymbol{x}$, the ones resulting from the KL transform have the largest sum variance (Problem 6.3).

Entropy. We know from Chapter 2 that the entropy of a process is defined as

$$H_y = -E[\ln p_y(\boldsymbol{y})]$$

and it is a measure of the randomness of the process. For a zero mean Gaussian multivariable m-dimensional process the entropy becomes

$$H_y = \frac{1}{2}E[\boldsymbol{y}^T R_y^{-1}\boldsymbol{y}] + \frac{1}{2}\ln|R_y| + \frac{m}{2}\ln(2\pi) \tag{6.20}$$

However,

$$E[\boldsymbol{y}^T R_y^{-1}\boldsymbol{y}] = E[\text{trace}\{\boldsymbol{y}^T R_y^{-1}\boldsymbol{y}\}] = E[\text{trace}\{R_y^{-1}\boldsymbol{y}\boldsymbol{y}^T\}] = \text{trace}(I) = m$$

and using the known property from linear algebra the determinant is

$$\ln|R_y| = \ln(\lambda_0\lambda_1\ldots\lambda_{m-1})$$

In words, selection of the m features that correspond to the m largest eigenvalues maximizes the entropy of the process. This is expected, because variance and randomness are directly related.

Remarks

- The concept of principal eigenvectors subspace has also been exploited as a classifier. First, the sample mean of the whole training set is subtracted from the feature vectors. For each class, ω_i, the correlation matrix R_i is estimated and the principal m eigenvectors (corresponding to the m largest eigenvalues) are computed. A matrix A_i is then formed using the respective eigenvectors as columns. An unknown feature vector $\boldsymbol{x}$ is then classified in the class ω_j for which

$$\|A_j^T\boldsymbol{x}\| > \|A_i^T\boldsymbol{x}\|, \quad \forall i \neq j \tag{6.21}$$

that is, the class corresponding to the maximum norm subspace projection of $\boldsymbol{x}$ [Wata 73]. From Pythagoras' theorem this is equivalent to classifying a vector in its *nearest class subspace*. The decision surfaces are hyperplanes if all the subspaces have the same dimension or quadric surfaces in the more general case. *Subspace classification integrates the stages of feature generation/selection and classifier design.*

If this approach results in a relatively high classification error, the performance may be improved by suitable modifications known as *learning subspace methods*. For example, one can iteratively rotate the subspaces to adjust the lengths of the projections of the training vectors. The basic idea is to increase the length of a projection in the subspace of the correct class and decrease it for the rest. Such techniques have been applied successfully in a number of applications, such as speech recognition, texture classification, and character recognition. The interested reader may consult, for example, [Oja 83, Koho 89, Prak 97].

- For the computation of the correlation matrix eigenvectors, a number of iterative schemes have been developed. The computation is performed working directly with the vectors, without having to estimate the corresponding correlation matrix, using neural network concepts [Oja 83, Diam 96].

Example 6.2. The correlation matrix of a vector $\boldsymbol{x}$ is given by

$$R_x = \begin{bmatrix} 0.3 & 0.1 & 0.1 \\ 0.1 & 0.3 & -0.1 \\ 0.1 & -0.1 & 0.3 \end{bmatrix}$$

Compute the KL transform of the input vector.

The eigenvalues of R_x are $\lambda_0 = 0.1, \lambda_1 = \lambda_2 = 0.4$. Since the matrix R_x is symmetric, we can always construct orthonormal eigenvectors. For this case we have

$$\boldsymbol{a}_0 = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, \quad \boldsymbol{a}_1 = \frac{1}{\sqrt{6}} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{a}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}$$

The KL transform is then given by

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \end{bmatrix} = \begin{bmatrix} 2/\sqrt{6} & 1/\sqrt{6} & 1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{3} & -1/\sqrt{3} & -1/\sqrt{3} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \end{bmatrix}$$

where $y(0)$, $y(1)$ correspond to the two largest eigenvalues.

6.4 THE SINGULAR VALUE DECOMPOSITION

Given a matrix X of rank r, we will show that there exist $N \times N$ unitary matrices U and V so that

$$X = U \begin{bmatrix} \Lambda^{\frac{1}{2}} & \mathrm{O} \\ \mathrm{O} & 0 \end{bmatrix} V^H \quad \text{or} \quad Y \equiv \begin{bmatrix} \Lambda^{\frac{1}{2}} & \mathrm{O} \\ \mathrm{O} & 0 \end{bmatrix} = U^H X V \tag{6.22}$$

where $\Lambda^{\frac{1}{2}}$ is the $r \times r$ diagonal matrix with elements $\sqrt{\lambda_i}$, and λ_i are the r nonzero eigenvalues of the associated matrix $X^H X$. O denotes a zero element matrix. *In other words, there exist unitary matrices U and V so that the transformed matrix Y is diagonal.* From (6.22) it is readily shown that

$$X = \sum_{i=0}^{r-1} \sqrt{\lambda_i} \boldsymbol{u}_i \boldsymbol{v}_i^H \tag{6.23}$$

where $\boldsymbol{u}_i$, $\boldsymbol{v}_i$ are the first r columns of U and V, respectively. More precisely, $\boldsymbol{u}_i$, $\boldsymbol{v}_i$ are the eigenvectors of XX^H and X^HX, respectively. The eigenvalues λ_i are known as *singular values* of X and the expansion in (6.23) as the *singular value decomposition* (SVD) of X or the *spectral representation* of X.

Proof. Given a matrix X of rank r, it is known from linear algebra [Stra 80] that the two square matrices X^HX and XX^H are also of rank r. Furthermore, the two matrices X^HX and XX^H have the same eigenvalues but different (yet related) eigenvectors (Problem 6.5),

$$XX^H\boldsymbol{u}_m = \lambda_m\boldsymbol{u}_m \tag{6.24}$$

$$X^HX\boldsymbol{v}_m = \lambda_m\boldsymbol{v}_m \tag{6.25}$$

Since both matrices are Hermitian and nonnegative (i.e., $(XX^H)^H = XX^H$), they have nonnegative real eigenvalues and orthogonal eigenvectors (Appendix B). In (6.24) and (6.25) we have also chosen them to be orthonormal, that is, $\boldsymbol{u}_m^H\boldsymbol{u}_m = 1$ and $\boldsymbol{v}_m^H\boldsymbol{v}_m = 1$. It is straightforward to see from (6.24) and (6.25) that

$$\boldsymbol{u}_m = \frac{1}{\sqrt{\lambda_m}}X\boldsymbol{v}_m, \quad \text{for } \lambda_m \neq 0 \tag{6.26}$$

Indeed, premultiplying (6.25) by X results in

$$(XX^H)X\boldsymbol{v}_m = \lambda_m X\boldsymbol{v}_m$$

That is, $\boldsymbol{u}_m = \alpha X\boldsymbol{v}_m$, where (without loss of generality) the scaling factor α can be taken as positive and it is found from

$$\|\boldsymbol{u}_m\|^2 = 1 = \alpha^2\boldsymbol{v}_m^HX^HX\boldsymbol{v}_m = \alpha^2\lambda_m\|\boldsymbol{v}_m\|^2 \Rightarrow \alpha = \frac{1}{\sqrt{\lambda_m}}$$

Let us now assume that $\boldsymbol{u}_i$, $\boldsymbol{v}_i$, $i = 0, 1, \ldots, r-1$, are the eigenvectors corresponding to the nonzero eigenvalues and $\boldsymbol{u}_i$, $\boldsymbol{v}_i$, $i = r, \ldots, N-1$, to the zero eigenvalues. Then, for the latter ones we have

$$X^HX\boldsymbol{v}_i = 0 \Rightarrow \boldsymbol{v}_i^HX^HX\boldsymbol{v}_i = 0 \Rightarrow \|X\boldsymbol{v}_i\|^2 = 0$$

Hence

$$X\boldsymbol{v}_i = \boldsymbol{0}, \quad i = r, \ldots, N-1 \tag{6.27}$$

In a similar way one can show that

$$X^H\boldsymbol{u}_i = \boldsymbol{0}, \quad i = r, \ldots, N-1 \tag{6.28}$$

Combining (6.26) and (6.27), we show that the right-hand side of (6.23) is

$$\sum_{i=0}^{r-1} \sqrt{\lambda_i}\boldsymbol{u}_i\boldsymbol{v}_i^H = X\sum_{i=0}^{r-1} \sqrt{\lambda_i}\frac{1}{\sqrt{\lambda_i}}\boldsymbol{v}_i\boldsymbol{v}_i^H = X\sum_{i=0}^{N-1} \boldsymbol{v}_i\boldsymbol{v}_i^H \tag{6.29}$$

Let us now define a matrix V that has as columns the orthonormal eigenvectors $\boldsymbol{v}_i$,

$$V = [\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N-1}]$$

Orthonormality of the columns results in $V^H V = I$, that is, V is unitary and $VV^H = I$. Hence

$$I = VV^H = [\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N-1}]\begin{bmatrix} \boldsymbol{v}_0^H \\ \vdots \\ \boldsymbol{v}_{N-1}^H \end{bmatrix} = \sum_{i=0}^{N-1} \boldsymbol{v}_i\boldsymbol{v}_i^H \tag{6.30}$$

From (6.29) and (6.30) we have that

$$X = \sum_{i=0}^{r-1} \sqrt{\lambda_i}\boldsymbol{u}_i\boldsymbol{v}_i^H \tag{6.31}$$

and X can be written as

$$X = U\begin{bmatrix} \Lambda^{\frac{1}{2}} & \mathrm{O} \\ \mathrm{O} & 0 \end{bmatrix} V^H \tag{6.32}$$

where U is the unitary matrix with columns the orthonormal eigenvectors $\boldsymbol{u}_i$. ■

Remarks

- If X is approximated by

$$\hat{X} = \sum_{i=0}^{k-1} \sqrt{\lambda_i}\boldsymbol{u}_i\boldsymbol{v}_i^H, \quad k \leq r-1$$

then $\hat{X}$, being the sum of k $N \times N$ rank-one independent matrices, is of rank k. If the k largest eigenvalues are involved, it can be shown that the squared error

$$\epsilon^2 = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} |X(m,n) - \hat{X}(m,n)|^2$$

is the minimum one with respect to all rank-k $N \times N$ matrices. The square root of this is also known as the Frobenius norm $\|X - \hat{X}\|_F$ of the difference

matrix $X - \hat{X}$. The error in the approximation turns out to be (Problem 6.6)

$$\epsilon^2 = \sum_{i=k}^{r-1} \lambda_i \tag{6.33}$$

Hence, if we order the eigenvalues in descending order, $\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{r-1}$, then for a given number of k terms in the expansion, the SVD leads to the minimum square error. *Thus, $\hat{X}$ is the best rank-k approximation of X in the Frobenius norm sense.* This reminds us of the Karhunen–Loève expansion. However, in the latter case the optimality was with respect to the mean square error. This is a major difference in philosophy between SVD and KL. *The former is related to a single set of samples and the latter to an ensemble of them.*

- Due to its optimal approximation properties, the SVD transform also has excellent "information packing" properties, and the image array can be represented efficiently by a few of its singular values. Thus, SVD is a natural candidate as a tool for feature generation or selection in classification.

Example 6.3. Consider the matrix

$$X = \begin{bmatrix} 6 & 6 \\ 0 & 1 \\ 4 & 0 \\ 0 & 6 \end{bmatrix}$$

The goal is to compute its singular value decomposition.

- Step 1: Find the eigenvalues and eigenvectors of

$$X^T X = \begin{bmatrix} 52 & 36 \\ 36 & 73 \end{bmatrix}$$

These are $\lambda_0 = 100, \lambda_1 = 25$ and the corresponding eigenvectors $\boldsymbol{v}_0 = [0.6, 0.8]^T$, $\boldsymbol{v}_1 = [0.8, -0.6]^T$.

- Step 2: Compute the eigenvectors of XX^T. This is a 4×4 matrix of rank 2. The eigenvectors corresponding to the nonzero eigenvalues λ_0, λ_1 are computed via (6.26), that is, $\boldsymbol{u}_0 = 0.1X\boldsymbol{v}_0, \boldsymbol{u}_1 = 0.2X\boldsymbol{v}_1$ or $[0.84, 0.08, 0.24, 0.48]^T$ and $[0.24, -0.12, 0.64, -0.72]^T$ respectively.
- Step 3: Compute the SVD of X

$$X = 10[0.84, 0.08, 0.24, 0.48]^T[0.6, 0.8] + 5[0.24, -0.12, 0.64, -0.72]^T[0.8, -0.6]$$

If we keep the first of the two terms, then the resulting approximation is the best, in the Frobenius sense, rank-1 approximation of X.

6.5 INDEPENDENT COMPONENT ANALYSIS

As we have already seen, the Principal Component Analysis (PCA) performed by the Karhunen–Loève transform produces features $y(i), i = 0, 1, \ldots, N-1$, that are mutually uncorrelated. The obtained by the KL transform solution is optimal when dimensionality reduction is the goal and one wishes to minimize the approximation mean square error. However, for certain applications, such as the one illustrated in Figure 6.1, the obtained solution falls short of the expectations. In contrast, the more recently developed *Independent Component Analysis* (ICA) theory, e.g., [Hyva 01, Como 94, Jutt 91, Hayk 00, Lee 98], tries to achieve much more than simple decorrelation of the data. The ICA task is casted as follows: Given the set of input samples $\boldsymbol{x}$, determine an $N \times N$ invertible matrix W such that the entries $y(i), i = 0, 1, \ldots, N-1$, of the transformed vector

$$\boldsymbol{y} = W\boldsymbol{x} \tag{6.34}$$

are mutually independent. The goal of statistical independence is a stronger condition than the uncorrelatedness required by the PCA. The two conditions are equivalent *only* for Gaussian random variables.

Searching for independent rather than uncorrelated features gives us the means of exploiting a lot more of information, hidden in the higher order statistics of the data. As the example of Figure 6.1 suggests, constraining the search by digging information in the second-order statistics only results in the least interesting, for our problem, projection direction, i.e., that of $\boldsymbol{a}_1$. However, $\boldsymbol{a}_2$ is, no doubt, the most interesting direction from the class separation point of view. In contrast, employing ICA can unveil from the higher order statistics of the data the piece of information that points $\boldsymbol{a}_2$ as the most interesting one. Furthermore, searching for statistically independent features is in line with the way nature builds up the "cognitive" maps of the outside world in the brain, by processing the (input) sensory data. Barlow [Barl 89], in the so-called Barlow's hypothesis, suggests that the outcome of the early processing performed in our visual cortical feature detectors might be the result of a *redundancy reduction* process. In other words, the neural outputs are mutually as statistically independent as possible, conditioned, of course, on the received sensory messages. The interested reader can find more on issues related to redundancy reduction and also to a number of methodologies inspired by it in [Atti 92, Fiel 94, Deco 95, Bell 00].

Before we proceed to develop techniques for performing ICA, we need to be sure that such a problem is well defined and has a solution and under what conditions. To this end, let us assume that our input random data vector $\boldsymbol{x}$ is indeed generated by a linear combination of statistically independent and *stationary in the strict sense* components (sources), i.e.,

$$\boldsymbol{x} = A\boldsymbol{y} \tag{6.35}$$

The task now is under what conditions a matrix, W, can be computed so as to recover the components of $\boldsymbol{y}$ from eq. (6.34), by exploiting information hidden in $\boldsymbol{x}$. Usually $\mathcal{A}$ is known as the mixing and W as the de-mixing matrix, respectively. The following condition is proved in [Como 94].

Identifiability condition of the ICA model. All independent components $y(i)$, $i = 1, 2, \ldots, N$, with the possible exception of one, must be non-Gaussian. A second condition is that matrix $\mathcal{A}$ must be invertible. In the more general case where $\mathcal{A}$ is a nonsquare $l \times N$ matrix, then l must be greater than N and $\mathcal{A}$ must be of full column rank.

In other words, in contrast to PCA that can always be performed, ICA is meaningful only if the involved random variables are non-Gaussian. Indeed, as has already been stated, for Gaussian random variables independence is equivalent to uncorrelatedness and PCA suffices. From a mathematical point of view, the ICA problem is ill-posed for Gaussian processes. Indeed, if we assume that the independent components $y(i), i = 0, 1, \ldots, N-1$, are all Gaussian, then a linear transformation of them by *any* unitary matrix will also be a solution (see Problem 5.4). PCA achieves a unique solution by imposing a *specific orthogonal structure* onto the transformation matrix.

Under the above stated conditions, it can be shown that each one of the resulting independent components is *uniquely* estimated up to a multiplicative constant, which is a rather insignificant indeterminacy associated with the method. This is the reason that many times the components are considered of unit variance. Finally, it is interesting to state that the independent components result in no specific ordering, in contrast to the PCA, where a specific ordering is associated with the values of the corresponding eigenvalues. However, in practice, some form of ordering can be adopted. For example, the components can be ordered according to the degree of "non-Gaussianity," measured by an appropriate index, e.g., the fourth-order cumulant (see Appendix A). Although such an index may seem a bit strange to a newcomer, its physical interpretation will become clearer as we go on. After all, from a common sense point of view, a Gaussian pdf must be the least interesting one. Recall from Chapter 2 that maximizing the entropy, constraining the solution to be within the family of random variables with given mean and variance, the result is a Gaussian pdf. That is, the Gaussian is the most "random" of all the pdfs describing this family of random variables and from this point of view the least informative one with respect to the underlying structure of the data. In contrast, distributions that have the "least resemblance" to the Gaussian are more interesting since they display some structure associated with the data. This observation is at the heart of a closely related to ICA family of techniques known as *projection pursuit.* The essence behind such techniques is to search for directions (subspaces) in the feature space so that the corresponding data vector projections are described by "interesting" non-Gaussian distributions.

For a more rigorous discussion on such issues the reader may refer to, e.g., [Hube 85, Jone 87].

6.5.1 ICA Based on Second- and Fourth-Order Cumulants

This approach in performing ICA is a direct generalization of the PCA technique. The Karhunen–Loève transform focuses on the second-order statistics and demands the cross-correlations $E[y(i)y(j)]$ to be zero. Since in ICA we demand the components of $\boldsymbol{y}$ to be statistically independent, this is equivalent to demanding all the higher order cross-cumulants to be zero (see Appendix A). In [Como 94] it is suggested that restricting ourselves up to the fourth-order cumulants is sufficient for many applications. Recall from Appendix A that the first three cumulants are equal to the first three moments, i.e.,

$$\kappa_1(y(i)) = E[y(i)] = 0$$

$$\kappa_2(y(i)y(j)) = E[y(i)y(j)]$$

$$\kappa_3(y(i)y(j)y(k)) = E[y(i)y(j)y(k)]$$

and the fourth-order cumulants are given by

$$\begin{aligned}\kappa_4(y(i)y(j)y(k)y(r)) = {} & E[y(i)y(j)y(k)y(r)] - E[y(i)y(j)]E[y(k)y(r)] \\ & - E[y(i)y(k)]E[y(j)y(r)] \\ & - E[y(i)y(r)]E[y(j)y(k)]\end{aligned}$$

where zero mean processes have been assumed. Another assumption which is usually encountered in practice, and will be adopted here, is that the associated pdfs are symmetric. This makes all odd order cumulants zero. Thus the problem has now been reduced to finding a matrix, W, so that the second-order (cross-correlations) and fourth-order cross-cumulants of the transformed variables are zero. In [Como 94] this is achieved by the following steps:

Step 1. Perform a PCA on the input data, i.e.,

$$\hat{\boldsymbol{y}} = A^T \boldsymbol{x} \tag{6.36}$$

A is our familiar unitary transformation matrix of the Karhunen–Loève transform. The components of the transformed random vector $\hat{\boldsymbol{y}}$ are, thus, uncorrelated.

Step 2. Compute another unitary matrix, $\hat{A}$, so that the fourth-order cross-cumulants of the components of the transformed random vector

$$y = \hat{A}^T \hat{y} \tag{6.37}$$

are zero. This is equivalent to searching for a matrix $\hat{A}$ that makes the sum of the squares of the fourth-order auto-cumulants maximum, i.e.,

$$\max_{\hat{A}\hat{A}^T = I} \Psi(\hat{A}) \equiv \sum_{i=0}^{N-1} \kappa_4(y(i))^2 \tag{6.38}$$

Step 2 is justified as follows. It can be shown [Como 94] that the sum of the squares of the fourth-order cumulants is invariant under a linear transformation by a unitary matrix. Therefore, since the sum of squares of the fourth-order cumulants is fixed for $\hat{y}$, maximizing the sum of squares of the auto-cumulants of y will force the corresponding cross-cumulants to zero. Observe that this is basically a diagonalization problem of the fourth-order cumulant multidimensional array. In practice, this is achieved by generalizing the method of Givens rotations, used for matrix diagonalization [Como 94]. Note that the right hand side in Eq. (6.38) is a function of (a) the elements of the unknown matrix $\hat{A}$, (b) the elements of the known (for this step) matrix A, and (c) the cumulants of the random components of the input data vector $\boldsymbol{x}$, which have to be estimated prior to the application of the method. In practice, it may turn out that nulling of the cross-cumulants is only approximately achieved. This is because (a) the input data may not obey the linear model of Eq. (6.35), (b) the input data are corrupted by noise, which has not been taken into account, and (c) the cumulants of the input are only approximately known, since they are estimated by the available input data set.

Once the two steps have been completed, the final feature vector with (approximately) independent components is given by the combined transform

$$y = (A\hat{A})^T \boldsymbol{x} \equiv W\boldsymbol{x} \tag{6.39}$$

Notice that since $\hat{A}$ is unitary the uncorrelatedness achieved in the first step is inherited by the elements of y, which now has its second- and fourth-order cross-cumulants (at least approximately) zero.

6.5.2 ICA Based on Mutual Information

The approach based on nulling the second- and fourth-order cross-cumulants, although one of the most widely used in practice, somehow lacks in generality and also imposes, externally, a structure in the transformation matrix. An alternative, theoretically more pleasing approach is estimating W by minimizing the *mutual*

information between the transformed random variables. The mutual information, $I(\boldsymbol{y})$, between the components of $\boldsymbol{y}$ is defined as

$$I(\boldsymbol{y}) = -H(\boldsymbol{y}) + \sum_{i=0}^{N-1} H(y(i)) \tag{6.40}$$

where $H(y(i))$ is the associated entropy of $y(i)$, defined as ([Papo 91])

$$H(y(i)) = -\int p_i(y(i)) \ln p_i(y(i))\, dy(i) \tag{6.41}$$

where $p_i(y(i))$ is the marginal pdf of $y(i)$. In Appendix A, it is shown that $I(\boldsymbol{y})$ is equal to the Kullback–Leibler probability distance between the joint pdf $p(\boldsymbol{y})$ and the product of the respective marginal probability densities $\prod_{i=0}^{N-1} p_i(y(i))$. This distance (and hence the associated mutual information $I(\boldsymbol{y})$) is zero if the components $y(i)$ are statistically independent. This is because only in this case is the joint pdf equal to the product of the corresponding marginal pdfs and the Kullback–Leibler distance becomes zero. Hence, what is more natural than trying to compute W so as to force $I(\boldsymbol{y})$ to be minimum, since this will make the components of $\boldsymbol{y}$ *as independent as possible*? Combining Eqs. (6.34), (6.40), and (6.41) and taking into account the formula that relates the two pdfs associated with $\boldsymbol{x}$ and $\boldsymbol{y}$ ($\boldsymbol{y}$ is a function of $\boldsymbol{x}$), e.g., [Papo 91], we end up with

$$I(\boldsymbol{y}) = -H(\boldsymbol{x}) - \ln|det(W)| - \sum_{i=0}^{N-1} \int p_i(y(i)) \ln p_i(y(i))\, dy(i) \tag{6.42}$$

where $det(W)$ denotes the determinant of W. The elements of the unknown matrix W are hidden in the marginal pdfs of the transformed variables, $y(i)$. However, it is not easy to express this dependence explicitly. An approach currently used is to expand each of the marginal probabilities around the Gaussian pdf, $g(y)$, following Edgeworth's expansion (Appendix A), and truncate the series to a reasonable approximation. For example, keeping the first two terms in the Edgeworth expansion we have

$$p(y) = g(y)\left(1 + \frac{1}{3!}\kappa_3(y)H_3(y) + \frac{1}{4!}\kappa_4(y)H_4(y)\right) \tag{6.43}$$

where $H_k(y)$ is the Hermite polynomial of order k (Appendix A). To obtain an approximate expression for $I(\boldsymbol{y})$ in terms of cumulants of $y(i)$ and W, we can (a) insert in Eq. (6.42) the pdf approximation in Eq. (6.43), (b) adopt the approximation $\ln(1+y) \simeq y - y^2$, and (c) perform the integrations. This is no doubt a rather

painful task! For the case of Eq. (6.43) and constraining W to be unitary, the following is obtained ([Hyva 01]):

$$I(\boldsymbol{y}) \simeq C - \sum_{i=0}^{N-1} \left(\frac{1}{12}\kappa_3^2(y(i)) + \frac{1}{48}\kappa_4^2(y(i)) + \frac{7}{48}\kappa_4^4(y(i)) - \frac{1}{8}\kappa_3^2(y(i))\kappa_4(y(i)) \right) \tag{6.44}$$

where C in a variable independent of W. Under the assumption that the pdfs are symmetric (thus, third order cumulants are zero,) it can be shown that minimizing the approximate expression of the mutual information in Eq. (6.44) is equivalent to maximizing the sum of the squares of the fourth-order cumulants. Of course, the unitary W constraint is not necessary, and in this case other approximate expressions for $I(\boldsymbol{y})$ result, e.g., [Hayk 99].

Minimization of $I(\boldsymbol{y})$ in Eq. (6.44) can be carried out by a gradient descent technique (Appendix C), where the involved expectations (associated with the cumulants) are replaced by the respective instantaneous values. Although a detailed treatment of the optimization procedure is beyond the scope of this book, it is worth pointing out some of its aspects.

Let us go back to Eq. (6.42), before we apply the approximations. Since $H(\boldsymbol{x})$ does not depend on W, minimizing $I(\boldsymbol{y})$ is equivalent with the maximization of

$$J(W) = \ln|det(W)| + E\left[\sum_{i=0}^{N-1} \ln p_i(y(i))\right] \tag{6.45}$$

Taking the gradient of the cost function with respect to W results in

$$\frac{\partial J(W)}{\partial W} = W^{-T} - E[\boldsymbol{\phi}(\boldsymbol{y})\boldsymbol{x}^T] \tag{6.46}$$

where

$$\boldsymbol{\phi}(\boldsymbol{y}) \equiv \left[-\frac{p_0'(y(0))}{p_0(y(0))}, \ldots, -\frac{p_{N-1}'(y(N-1))}{p_{N-1}(y(N-1))} \right]^T \tag{6.47}$$

and

$$p_i'(y(i)) \equiv \frac{dp_i(y(i))}{dy(i)} \tag{6.48}$$

Obviously the derivatives of the marginal probability densities depend on the type of approximation adopted in each case. The general gradient ascent scheme

at the tth iteration step can now be written as

$$W(t) = W(t-1) + \mu(t)\left(W^{-T}(t-1) - E[\boldsymbol{\phi}(\boldsymbol{y})\boldsymbol{x}^T]\right)$$

$$W(t) = W(t-1) + \mu(t)\left(I - E[\boldsymbol{\phi}(\boldsymbol{y})\boldsymbol{y}^T]\right)W^{-T}(t-1) \tag{6.49}$$

In practice, the expectation operator is neglected, in the spirit of the stochastic approximation rationale (Section 3.4.2).

Remarks

- From the gradient in Eq. (6.46) it is easy to see that at a stationary point the following is true:

$$\frac{\partial J(W)}{\partial W}W^T = E[I - \boldsymbol{\phi}(\boldsymbol{y})\boldsymbol{y}^T] = 0 \tag{6.50}$$

  In other words, what we achieve with ICA is a nonlinear generalization of PCA. Recall that for the latter, the uncorrelatedness condition can be written as

$$E[I - \boldsymbol{y}\boldsymbol{y}^T] = 0 \tag{6.51}$$

  The presence of the *nonlinear* function $\boldsymbol{\phi}(\cdot)$ takes us beyond simple uncorrelatedness, and brings the cumulants into the scene. As a matter of fact Eq. (6.50) was the one that inspired the early pioneering work on ICA, as a direct nonlinear generalization of PCA [Jutt 91].

- The update equation in Eq. (6.49) involves the inversion of the transpose of the current estimate of W. Besides the computational complexity issues, there is no guarantee of the invertibility in the process of adaptation. The use of the so called *natural gradient* [Doug 00], instead of the gradient in Eq. (6.46), results in

$$W(t) = W(t-1) + \mu(t)\left(I - E[\boldsymbol{\phi}(\boldsymbol{y})\boldsymbol{y}^T]\right)W(t-1) \tag{6.52}$$

  which does not involve matrix inversion and at the same time improves convergence. A more detailed treatment of this issue is beyond the scope of the present book. Just to give an incentive to the mathematically inclined reader for indulging more deeply this field, it suffices to say that our familiar gradient, i.e, Eq. (6.46), points to the steepest ascent direction if the space is Euclidean. However, in our case the parameter space consists of all the nonsingular $N \times N$ matrices, which is a multiplicative group. The space is Riemannian and it turns out that the natural gradient, pointing to the steepest ascent direction, results if we multiply the gradient in Eq. (6.46) by $W^T W$, which is the corresponding Riemannian metric tensor [Doug 00].

6.5.3 An ICA Simulation Example

The example is a realization of the case shown in Figure 6.1. A number of 1024 samples of a two-dimensional normal distribution was generated.

The mean and covariance matrix of the normal pdf were

$$\boldsymbol{\mu} = [-2.6042, 2.5]^T, \quad \Sigma = \begin{bmatrix} 10.5246 & 9.6313 \\ 9.6313 & 11.3203 \end{bmatrix}$$

Similarly, 1024 samples from a second normal pdf were generated with the same covariance and mean $-\boldsymbol{\mu}$. For the ICA the method based on the second- and fourth-order cumulants, presented in this section, was used. The resulting transformation matrix W is

$$W = \begin{bmatrix} -0.7088 & 0.7054 \\ 0.7054 & 0.7088 \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{a}_2^T \\ \boldsymbol{a}_1^T \end{bmatrix}$$

The vectors $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ point in the principal and minor axis directions, respectively, obtained from the PCA analysis. However, the most interesting direction for projection, according to the ICA analysis, is that of $\boldsymbol{a}_2$ and not of $\boldsymbol{a}_1$. Indeed, the kurtosis of the obtained transformed variables $[y_1, y_2]^T = W\boldsymbol{x}$ is

$$\kappa_4(y_1) = -1.7$$
$$\kappa_4(y_2) = 0.1$$

Thus, projection in the principal axis direction results in a variable with a pdf close to a Gaussian. The projection to the minor axis direction results in a variable with a pdf that deviates from the Gaussian (Figures 6.1, 6.2) and is more appropriate from the classification point of view.

6.6 THE DISCRETE FOURIER TRANSFORM (DFT)

We have already seen that the basis vectors/images for the KL and SVD expansions are not fixed but are "problem dependent" and they are the result of an optimization process. This is the reason for their optimality with respect to the decorrelation and information packing properties, but at the same time this accounts for their major disadvantage, that is, their high computational complexity. For the rest of the chapter we will be concerned with transforms that use *fixed* basis vectors/images. Their suboptimality with respect to decorrelation and information packing properties is most often compensated by their low computational requirements.

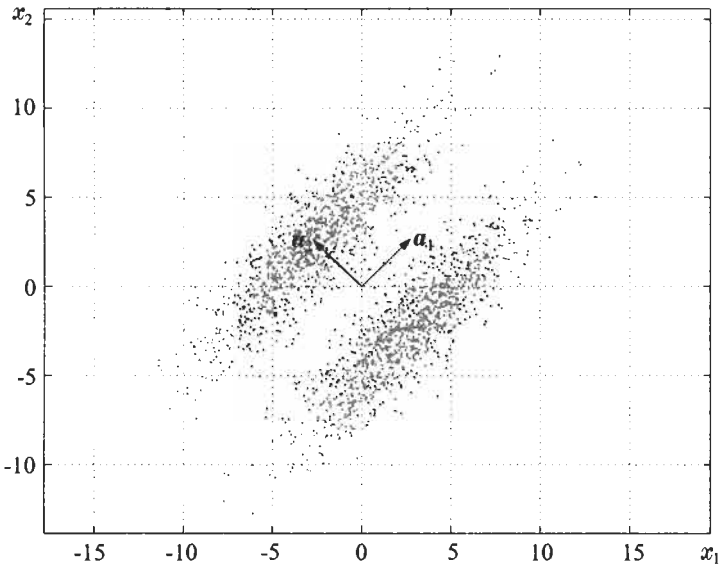


FIGURE 6.2: The setup for the ICA simulation example. The two vectors point to the projection directions resulting from the analysis.

6.6.1 One-Dimensional DFT

Given N input samples $x(0), x(1), \ldots, x(N-1)$, their DFT is defined as

$$y(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) \exp\left(-j\frac{2\pi}{N}kn\right), \quad k = 0, 1, \ldots, N-1 \tag{6.53}$$

and the inverse DFT as

$$x(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} y(k) \exp\left(j\frac{2\pi}{N}kn\right), \quad n = 0, 1, \ldots, N-1 \tag{6.54}$$

where $j \equiv \sqrt{-1}$. Collecting all $x(n)$ and $y(k)$ together into two $N \times 1$ vectors and defining

$$W_N \equiv \exp\left(-j\frac{2\pi}{N}\right) \tag{6.55}$$

(6.53), (6.54) are written in a matrix form as

$$\boldsymbol{y} = W^H \boldsymbol{x}, \; \boldsymbol{x} = W\boldsymbol{y} \tag{6.56}$$

where

$$W^H = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)(N-1)} \end{bmatrix} \tag{6.57}$$

It is not difficult to see that W is a unitary and symmetric matrix

$$W^{-1} = W^H = W^*$$

The basis vectors are the columns of W. For example, for $N = 4$

$$W = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}$$

and the basis vectors are

$$\boldsymbol{w}_0 = \frac{1}{2}[1, 1, 1, 1]^T$$

$$\boldsymbol{w}_1 = \frac{1}{2}[1, j, -1, -j]^T$$

$$\boldsymbol{w}_2 = \frac{1}{2}[1, -1, 1, -1]^T$$

$$\boldsymbol{w}_3 = \frac{1}{2}[1, -j, -1, j]^T$$

and

$$\boldsymbol{x} = \sum_{i=0}^{3} y(i)\boldsymbol{w}_i$$

The direct computation of (6.56) requires $O(N^2)$ computations. However, taking advantage of the specific structure of the matrix W, a substantial saving in computations is possible via the celebrated FFT algorithm, which computes each equation of (6.56) in $O(N \log_2 N)$ operations [Proa 92].

So far, the DFT has been introduced as a special type of a linear unitary transform of one vector to another. Another point of view, which will be useful later on in this chapter, is to see the DFT as a means of expanding a sequence $x(n)$ into a set

of N *basis sequences* $h_k(n)$

$$x(n) = \sum_{k=0}^{N-1} y(k) h_k(n)$$

where

$$h_k(n) = \begin{cases} \frac{1}{\sqrt{N}} \exp(j\frac{2\pi}{N}kn), & n = 0, 1, \ldots, N-1 \\ 0, & \text{otherwise} \end{cases}$$

and $y(k)$ are the coefficients of the expansion. The DFT basis sequences belong to a more general class of sequences known as orthonormal, that is,

$$\langle h_l(n), h_k(n) \rangle \equiv \sum_n h_k(n) h_l^*(n) = \delta_{kl} \tag{6.58}$$

where $\langle ., . \rangle$ is known as the inner product of the sequences $h_k(n)$, $h_l(n)$. For the DFT expansion we have

$$\begin{aligned} \langle h_k(n), h_l(n) \rangle &= \frac{1}{N} \sum_{n=0}^{N-1} \exp\left(j\frac{2\pi}{N}kn\right) \exp\left(-j\frac{2\pi}{N}ln\right) \\ &= \frac{1}{N} \sum_{n=0}^{N-1} \exp\left(j\frac{2\pi}{N}(k-l)n\right), \quad k, l = 0, 1, \ldots, N-1 \end{aligned}$$

However, it can easily be shown (Problem 6.8) that

$$\frac{1}{N} \sum_{n=0}^{N-1} \exp\left(j\frac{2\pi}{N}(k-l)n\right) = \begin{cases} 1, & l = k + rN, \quad r = 0, \pm 1, \pm 2, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{6.59}$$

Hence

$$\langle h_k(n), h_l(n) \rangle = \delta_{kl}$$

6.6.2 Two-Dimensional DFT

Given an $N \times N$ matrix/image, its two-dimensional DFT is defined as

$$Y(k, l) = \frac{1}{N} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} X(m, n) W_N^{km} W_N^{ln} \tag{6.60}$$

and the inverse DFT as

$$X(m,n) = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} Y(k,l)W_N^{-km}W_N^{-ln} \tag{6.61}$$

It is readily seen that this can be written in a compact form as

$$Y = W^H X W^H, \quad X = WYW \tag{6.62}$$

Thus, the 2-D DFT is a separable transform with basis images $\boldsymbol{w}_i\boldsymbol{w}_j^T$, $i, j = 0, 1, \ldots, N-1$. It is apparent from (6.62) that the number of operations required for the respective computations is $O(N^2 \log_2 N)$, that is, the number of additions and multiplications needed for $2N$ one-dimensional DFTs, via the FFT algorithm.

6.7 THE DISCRETE COSINE AND SINE TRANSFORMS

Given N input samples $x(0), x(1), \ldots, x(N-1)$ their *discrete cosine transform* (DCT) is defined as

$$y(k) = \alpha(k)\sum_{n=0}^{N-1} x(n)\cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad k = 0, 1, \ldots, N-1 \tag{6.63}$$

and the inverse DCT is given by

$$x(n) = \sum_{k=0}^{N-1} \alpha(k)y(k)\cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad n = 0, 1, \ldots, N-1 \tag{6.64}$$

where

$$\alpha(k) = \begin{cases} \sqrt{\frac{1}{N}}, & k = 0 \\ \sqrt{\frac{2}{N}}, & k \neq 0 \end{cases}$$

In vector form the transform is written as

$$\boldsymbol{y} = C^T\boldsymbol{x}$$

where the elements of the matrix C are given by

$$C(n,k) = \frac{1}{\sqrt{N}}, \quad k = 0,\ 0 \leq n \leq N-1$$

$$C(n,k) = \sqrt{\frac{2}{N}}\cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad 1 \leq k \leq N-1,\ 0 \leq n \leq N-1$$

Matrix C has real elements, and it is easy to see that it is orthogonal,

$$C^{-1} = C^T$$

The 2-D DCT is the separable transform defined as

$$Y = C^T X C, \quad X = C Y C^T \tag{6.65}$$

The *discrete sine transform* (DST) is defined via the transform matrix

$$S(k, n) = \sqrt{\frac{2}{N+1}} \sin\left(\frac{\pi(k+1)(n+1)}{N+1}\right), \quad k, n = 0, 1, \ldots, N-1$$

and it is also an orthogonal transform. The DCT and DST belong to the family of transforms that can be computed via a fast method in $O(N \log_2 N)$ operations [Jain 89, Lim 90].

Remark

- The DCT and DST have very good information packing properties for most of the images, in the sense that they concentrate most of the energy in a few coefficients. An explanation for this property is that both offer a close approximation to the (KL) transform for a class of random signals, known as first-order Markov processes, which can approximately model a number of real-world images [Jain 89]. Figure 6.3 shows an image and the resulting DFT (magnitude), DST, and DCT transforms. It is apparent from the figure that the high-intensity coefficients of the transforms (dark) are concentrated in a small area, whose size depends on the energy compaction properties of the respective transform. This area is smaller for DCT and DST than for DFT.

6.8 THE HADAMARD TRANSFORM

The Hadamard transform and the Haar transform, to be considered in the next section, share a serious computational advantage over the previously considered DFT, DCT, and DST transforms. Their unitary matrices consist of ± 1 and the transforms are computed via additions and subtractions only, with no multiplications being involved. Hence, for processors for which multiplication is a time-consuming operation a substantial saving is obtained.

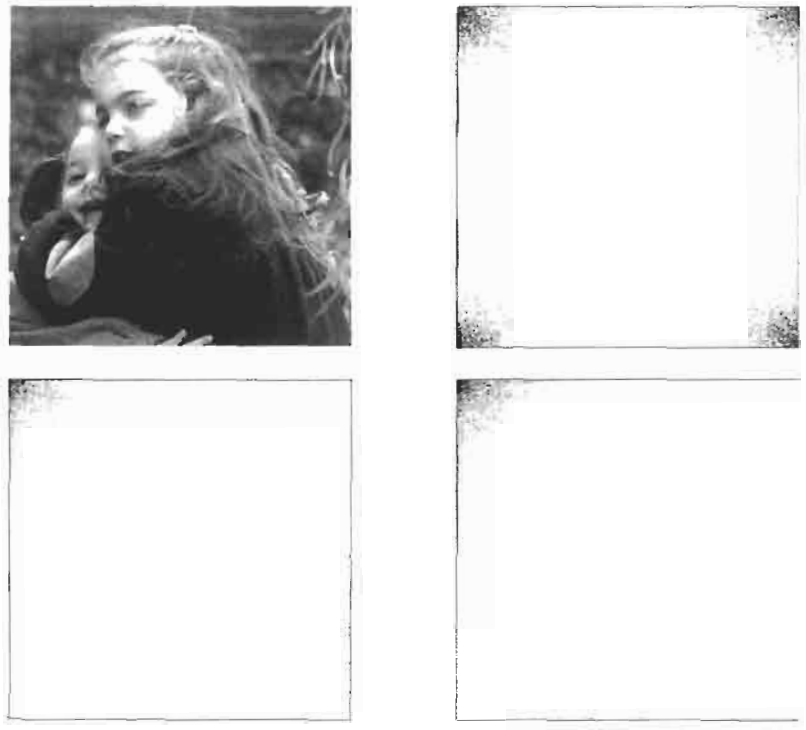


FIGURE 6.3: Example of an image and its magnitude DFT, DST, and DCT transforms, shown from top left to bottom left in the clockwise sense.

The Hadamard unitary matrix of order n is the $N \times N$ matrix, $N = 2^n$, generated by the following iteration rule:

$$H_n = H_1 \otimes H_{n-1} \tag{6.66}$$

where

$$H_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{6.67}$$

and $\otimes$ denotes the Kronecker product of two matrices

$$A \otimes B = \begin{bmatrix} A(1,1)B & A(1,2)B & \dots & A(1,N)B \\ \vdots & \vdots & \vdots & \vdots \\ A(N,1)B & A(N,2)B & \dots & A(N,N)B \end{bmatrix}$$

where $A(i, j)$ is the (i, j) element of A, $i, j = 1, 2, \ldots, N$. Thus, according to (6.66), (6.67), it is

$$H_2 = H_1 \otimes H_1 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

and for $n = 3$

$$H_3 = H_1 \otimes H_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix}$$

It is not difficult to show the orthogonality of H_n, $n = 1, 2, \ldots$, that is,

$$H_n^{-1} = H_n^T = H_n$$

For a vector $\boldsymbol{x}$ of N samples and $N = 2^n$ the transform pair is

$$\boldsymbol{y} = H_n \boldsymbol{x}, \quad \boldsymbol{x} = H_n \boldsymbol{y} \tag{6.68}$$

The 2-D Hadamard transform is given by

$$Y = H_n X H_n, \quad X = H_n Y H_n \tag{6.69}$$

The Hadamard transform has good to very good energy packing properties. Fast algorithms for its computation in $O(N \log_2 N)$ subtractions and/or additions are also available [Jain 89].

Remark

- Experimental results using the DCT, DST, and Hadamard transforms for texture discrimination have shown that the performance obtained was close to that of the optimal KL transform [Unse 86, Unse 89]. At the same time, this near-optimal performance is obtained at substantially reduced complexity, due to the availability of fast computational schemes as reported before.

6.9 THE HAAR TRANSFORM

The starting point for the definition of the Haar transform is the Haar functions $h_k(z)$, which are defined in the closed interval $[0, 1]$. The order k of the function is uniquely decomposed into two integers p, q

$$k = 2^p + q - 1, \quad k = 0, 1, \ldots, L - 1, \text{ and } L = 2^n$$

Table 6.1: Parameters for the Haar functions

k	0	1	2	3	4	5	6	7
p	0	0	1	1	2	2	2	2
q	0	1	1	2	1	2	3	4

where

$$0 \leq p \leq n-1,\ 0 \leq q \leq 2^p \quad \text{for } p \neq 0 \text{ and } q = 0 \text{ or } 1 \text{ for } p = 0$$

Table 6.1 summarizes the respective values for $L = 8$. The Haar functions are

$$h_0(z) \equiv h_{00}(z) = \frac{1}{\sqrt{L}}, \quad z \in [0, 1] \tag{6.70}$$

$$h_k(z) \equiv h_{pq}(z) = \frac{1}{\sqrt{L}} \begin{cases} 2^{\frac{p}{2}} & \frac{q-1}{2^p} \leq z < \frac{q-\frac{1}{2}}{2^p} \\ -2^{\frac{p}{2}} & \frac{q-\frac{1}{2}}{2^p} \leq z < \frac{q}{2^p} \\ 0 & \text{otherwise in } [0, 1] \end{cases} \tag{6.71}$$

The Haar transform matrix of order L consists of rows resulting from the preceding functions computed at the points $z = \frac{m}{L}, m = 0, 1, 2, \ldots, L-1$. For example, the 8×8 transform matrix is

$$H = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix} \tag{6.72}$$

It is not difficult to see that

$$H^{-1} = H^T$$

that is, H is orthogonal. The energy packing properties of the Haar transform are not very good. However, its importance for us lies beyond that. We will use it as the vehicle to take us from the world of unitary transforms to that of

multiresolution analysis. To this end, let us look carefully at the Haar transform matrix. We readily observe its sparse nature with a number of zeros, whose location reveals an underlying cyclic shift mechanism. To satisfy our curiosity as to why this happens, let us look at the Haar transform from a different perspective.

6.10 THE HAAR EXPANSION REVISITED

Let us split our original set of N input samples (N even) $x(0), x(1), \ldots, x(N-1)$ into successive blocks of two, that is, $(x(2k), x(2k+1)), k = 0, 1, \ldots, \frac{N}{2} - 1$, and apply the Haar transform of order $L = 2$. For each pair of input samples, a pair of transformed samples is obtained,

$$\begin{bmatrix} y_1(k) \\ y_0(k) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x(2k) \\ x(2k+1) \end{bmatrix}, \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{6.73}$$

That is,

$$y_1(k) = \frac{1}{\sqrt{2}}(x(2k) + x(2k+1)) \tag{6.74}$$

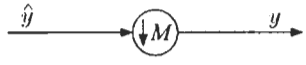
$$y_0(k) = \frac{1}{\sqrt{2}}(x(2k) - x(2k+1)), \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{6.75}$$

This can be interpreted as the action—on the sequence of N input samples—of two (noncausal) filters with impulse responses $(h_1(0) = \frac{1}{\sqrt{2}}, h_1(-1) = \frac{1}{\sqrt{2}})$ and $(h_0(0) = \frac{1}{\sqrt{2}}, h_0(-1) = -\frac{1}{\sqrt{2}})$, respectively. The corresponding transfer functions (Appendix D) are

$$H_1(z) = \frac{1}{\sqrt{2}}(1 + z) \tag{6.76}$$

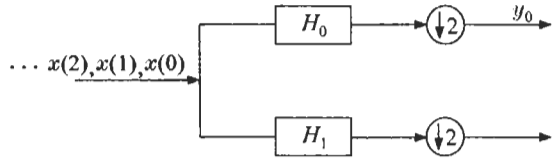
$$H_0(z) = \frac{1}{\sqrt{2}}(1 - z) \tag{6.77}$$

In other words, the order $L = 2$ Haar transform computes the output samples of the two filters when they are fed with the input sequence $x(n), n = 0, 1, 2, \ldots, N-1$. *Furthermore, the output sequence samples are computed for every other sample of the input sequence, at even time instants 0,2,4,...,* as (6.74) and (6.75) suggest. This operation is portrayed in Figure 6.4b. The operation at the output of the two filters is known as *subsampling by M*, in this case $M = 2$, and it is defined in Figure 6.4a. In other words, from the samples generated at the filter output we keep one every $M(= 2)$. In the time domain and for an input sequence consisting



$$y(k) = \hat{y}(kM)$$

(a)



(b)

FIGURE 6.4: (a) Subsampling operation and (b) filtering interpretation of the Haar transform.

of eight samples, the output, $y_0(k)$, of the H_0 branch of Figure 6.4b will consist of four samples given by

$$\begin{bmatrix} y_0(0) \\ y_0(1) \\ y_0(2) \\ y_0(3) \end{bmatrix} = \left[\begin{array}{cc|cc|cc|cc} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{array}\right] \begin{bmatrix} x(0) \\ x(1) \\ -- \\ x(2) \\ x(3) \\ -- \\ x(4) \\ x(5) \\ -- \\ x(6) \\ x(7) \end{bmatrix} \tag{6.78}$$

Well, this is nothing other than the action of the last four rows of the 8×8 Haar transform in (6.72)! What about the rest? Let us carry on the splitting of Figure 6.4b one step further, as shown in Figure 6.5. Using the easily shown Noble identity illustrated in Figure 6.6a (Problem 6.17), the structure of Figure 6.5 turns out to be equivalent to that of Figure 6.6b. Taking into account the subsampling operation of the lower branch after the filters H_0 and H_1, the Noble identity leads to

$$\hat{F}_1(z) = \frac{1}{2}(1+z)(1-z^2) = \frac{1}{2}(1+z-z^2-z^3) \tag{6.79}$$

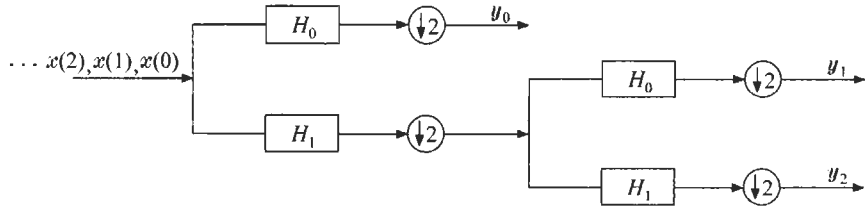


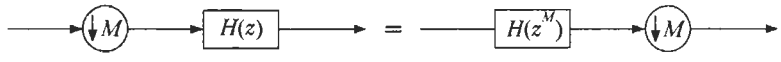
FIGURE 6.5: Two-stage filtering followed by subsampling operation.

$$\hat{F}_2(z) = \frac{1}{2}(1+z)(1+z^2) = \frac{1}{2}(1+z+z^2+z^3) \tag{6.80}$$

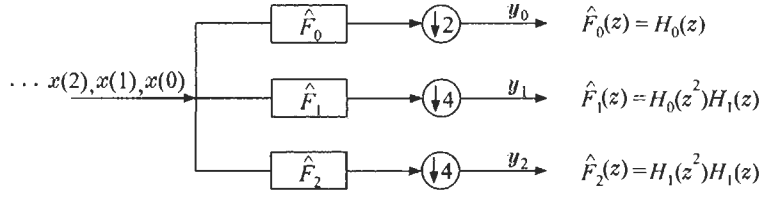
From the transfer function $\hat{F}_1(z)$ and taking into account the subsampling by 4 (2×2) operation, the first two samples of the $y_1(k)$ sequence are given by

$$\begin{bmatrix} y_1(0) \\ y_1(1) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(7) \end{bmatrix} \tag{6.81}$$

This is nothing but the action of the third and fourth rows of the 8×8 Haar transform on the input vector. If we now carry on the splitting one step further, as



(a)



(b)

FIGURE 6.6: (a) Noble identity I and (b) equivalent filter bank of Figure 6.5.

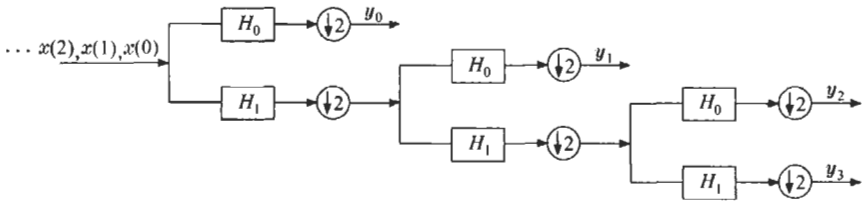


FIGURE 6.7: Tree-structured filter bank.

in Figure 6.7, it is straightforward to show by repeating the preceding arguments that

$$y_2(0) = \frac{1}{\sqrt{8}}\begin{bmatrix}1 & 1 & 1 & 1 & -1 & -1 & -1 & -1\end{bmatrix}\begin{bmatrix}x(0)\\x(1)\\\vdots\\x(7)\end{bmatrix} \tag{6.82}$$

and

$$y_3(0) = \frac{1}{\sqrt{8}}\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix}x(0)\\x(1)\\\vdots\\x(7)\end{bmatrix} \tag{6.83}$$

These equations are the actions of the second and first rows of the Haar transform on the input vector. The structure of Figure 6.7 is known as a (three-level) *tree-structured filter bank* generated by the filters $H_0(z)$ and $H_1(z)$. Figure 6.8 shows the frequency responses of these two filters. One ($H_0(z)$) is a high-pass and the other a low-pass filter. Herein lies the importance of the filter bank interpretation of the Haar transform. The input sequence $x(n)$ is first split into two versions *of lower resolution* with respect to the original one: a low-pass (average) *coarser* resolution version and a high-pass (difference) *detailed* resolution one. In the sequel the coarser resolution version is further split into two versions, and so on. This leads to a number of versions with an hierarchy of resolutions. This decomposition is known as *multiresolution decomposition.*

The idea of multiresolution decomposition has been around for some time [Burt 83, Akan 93] and has been exploited in various applications and for a number of reasons. Its popularity as a tool in pattern recognition is mainly due to the information compaction capabilities associated with such a decomposition, *provided filters H_0, H_1 are properly designed.* For many types of signals, such as, speech and images, most of the information is localized in certain resolution levels.

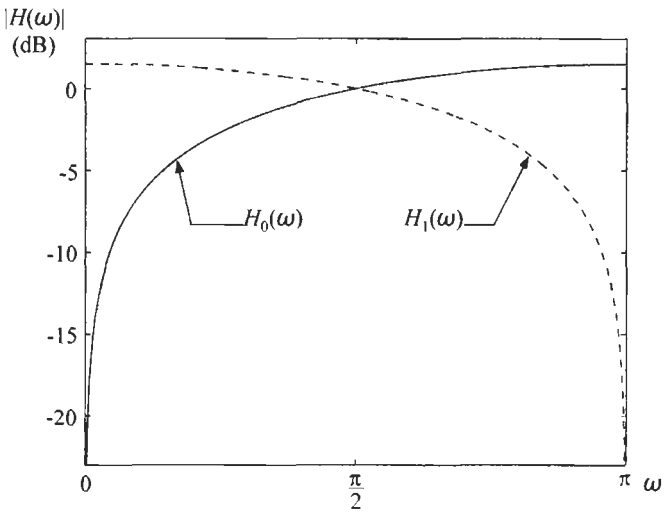


FIGURE 6.8: Magnitude of the frequency response for the two Haar filters. H_1 is lowpass and H_0 a highpass one.

Thus, most of the energy is concentrated in a (relatively) small number of samples, which carry most of the necessary information [Este 77, Mall 89]. Some fundamental issues related to the multiresolution decomposition and the design of filter banks will be highlighted next.

6.11 DISCRETE TIME WAVELET TRANSFORM (DTWT)

The goal of this section is twofold. We first free ourselves from the Haar functions, and we seek the possibility of using other filters in place of H_0, H_1. There is more than one reason for this generalization. An obvious reason is that the frequency responses of the Haar filters are far from ideal. If our aim is to split the original sequence into a hierarchy of "coarse" and "detailed" versions, we should require the filters that perform the splitting to be as close as possible to the ideal low/high-pass ones (Figure 6.9). Our next concern in this section is the inversion problem. That is, if we know the lower resolution versions, can we obtain the original sequence, $x(n)$, as was the case with the unitary transforms? We will show that under certain constraints in the design of H_0 and H_1 this is indeed possible.

The Two-Band Case

Let us start with the simple two-band case of Figure 6.4b, where we now assume that the filters are not the Haar ones. If $h_0(k)$, $h_1(k)$ are the respective impulse

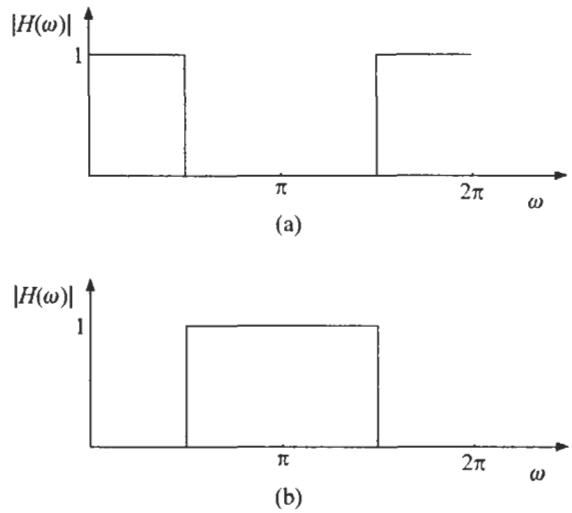


FIGURE 6.9: Ideal frequency responses for (a) low-pass and (b) high-pass filters.

responses, then we can write

$$y_0(k) = \sum_l x(l)h_0(n-l)|_{n=2k}$$

$$y_1(k) = \sum_l x(l)h_1(n-l)|_{n=2k}$$

where $y_1(k)$ is the output of the lower branch of Figure 6.4b. Collecting all $y_0(k), y_1(k), k = 0, 1, 2, \ldots$, together in a vector we have

$$\begin{bmatrix} \vdots \\ y_0(0) \\ y_1(0) \\ y_0(1) \\ y_1(1) \\ y_0(2) \\ y_1(2) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \ldots & h_0(2) & h_0(1) & h_0(0) & h_0(-1) & h_0(-2) & \ldots \\ \ldots & h_1(2) & h_1(1) & h_1(0) & h_1(-1) & h_1(-2) & \ldots \\ \ldots & \ldots & \ldots & h_0(2) & h_0(1) & h_0(0) & \ldots \\ \ldots & \ldots & \ldots & h_1(2) & h_1(1) & h_1(0) & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & h_0(2) & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & h_1(2) & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ x(0) \\ x(1) \\ x(2) \\ \vdots \end{bmatrix}$$

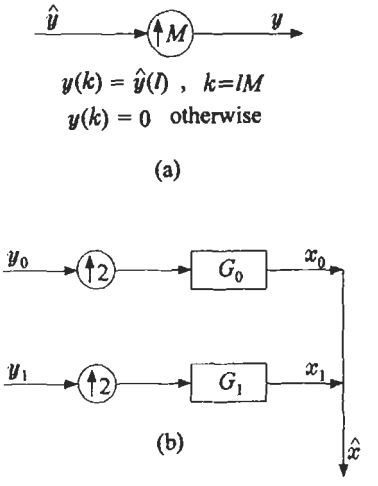


FIGURE 6.10: Tree-structured synthesis filter bank.

or

$$y = T_i x \tag{6.84}$$

Here, we have assumed that the filters can be noncausal but they are of *finite impulse response* (FIR) (the impulse response has a finite nonzero number of terms). The latter assumption is imposed here in order to avoid issues of convergence of infinite series. Observe the structure of T_i. It basically consists of two rows, one with the impulse response of H_0 and the other with that of H_1, which are then shifted each time by two to the right, to form the rest of the rows. This is the result of the subsampling, by two, operation. Figure 6.10b shows a structure that combines $y_0(k)$, $y_1(k)$, through the filters G_0, G_1, to form a sequence $\hat{x}$. The symbol at the input of the filters denotes the *upsampling* by M operation, which is defined in Figure 6.10a. In this case $M = 2$. In other words, this is equivalent to stuffing $M - 1$ zeros between every two samples. That is, the input sequences of the filters G_0, G_1 will be

$$\ldots 0\ y_0(0)\ 0\ y_0(1)\ 0\ y_0(2)\ 0 \ldots$$
$$\ldots 0\ y_1(0)\ 0\ y_1(1)\ 0\ y_1(2)\ 0 \ldots$$

respectively. Thus, every other sample of the impulse response hits a zero and

$$x_0(n) = \sum_k y_0(k) g_0(n - 2k)$$

$$x_1(n) = \sum_k y_1(k) g_1(n - 2k)$$

$$\hat{x}(n) = x_0(n) + x_1(n)$$

Filters G_i are known as the *synthesis filters* and the corresponding H_i, of Figure 6.4b, as the *analysis filters*. Collecting all $\hat{x}(n)$ together, it is not difficult to see that

$$\begin{bmatrix} \vdots \\ \hat{x}(0) \\ \hat{x}(1) \\ \hat{x}(2) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & g_0(0) & g_1(0) & g_0(-2) & g_1(-2) & \cdots \\ \cdots & g_0(1) & g_1(1) & g_0(-1) & g_1(-1) & \cdots \\ \cdots & g_0(2) & g_1(2) & g_0(0) & g_1(0) & \cdots \\ \cdots & g_0(3) & g_1(3) & g_0(1) & g_1(1) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ y_0(0) \\ y_1(0) \\ y_0(1) \\ y_1(1) \\ \vdots \end{bmatrix}$$

or

$$\hat{\boldsymbol{x}} = T_o \boldsymbol{y} \tag{6.85}$$

In order that $\hat{\boldsymbol{x}} = \boldsymbol{x}$ we require that [Vett 92]

$$T_o T_i = I = T_i T_o \tag{6.86}$$

Multiplying rows of T_i with columns of T_o, (6.86) becomes equivalent to

$$\sum_n h_i(2k - n) g_j(n - 2l) = \delta_{kl} \delta_{ij}, \quad i, j = 0, 1 \tag{6.87}$$

or according to the definition of the inner product in (6.58),

$$\langle h_i(2k - n), g_j(n - 2l) \rangle = \delta_{kl} \delta_{ij}$$

If (6.87) is satisfied, we say that the two-band filter bank is a *perfect reconstruction* one and $\hat{x}(n) = x(n)$. Thus,

$$x(n) = \sum_k y_0(k) g_0(n - 2k) + \sum_k y_1(k) g_1(n - 2k) \tag{6.88}$$

Equation (6.88) can also be seen from a different perspective. It is an expansion of $x(n)$ into a set of *basis sequences*

$$\{g_0(n-2k), g_1(n-2k)\}, \quad k \in Z$$

where Z is the set of the integer numbers. From such a point of view $y_0(k)$, $y_1(k)$ are the respective coefficients of the expansion. This is known as the *discrete time wavelet transform (DTWT)* and the coefficients $y_0(k)$, $y_1(k)$ as the *discrete time wavelet coefficients*. Thus, given a perfect reconstruction two-band filter bank (i.e., condition (6.87) is satisfied) the following transform pair is defined

$$y_i(k) = \sum_n x(n) h_i(2k-n) \quad (a)$$

$$x(n) = \sum_{i=0}^{1} \sum_k y_i(k) g_i(n-2k) \quad (b) \tag{6.89}$$

Remarks

- Two sets of basis functions are involved, namely

  $$h_i(2k-n) \equiv \phi_{ik}(n), \; g_j(n-2l) \equiv \psi_{jl}(n) \quad i, j = 0, 1 \text{ and } k, l \in Z$$

  Equation (6.87) is an orthogonality condition between $\phi_{ik}(n)$ and $\psi_{jl}(n)$, that is,

  $$\langle \phi_{ik}(n), \psi_{jl}(n) \rangle = \delta_{ij}\delta_{kl}$$

  and it is known as the *biorthogonality condition*. The discrete time wavelet transform pair in (6.89) is a *biorthogonal expansion*.
- The basis sequences $\phi_{ik}(n)$ and $\psi_{jl}(n)$ of the expansion are *shifts by an even number of samples* of four basic mother sequences $g_0(n)$, $g_1(n)$, $h_0(-n)$, $h_1(-n)$, which are the impulse responses of the synthesis and the time-reversed analysis filters. For the recovery of $x(n)$ from its discrete time wavelet coefficients, each coefficient $y_i(k)$ weighs and adds a copy of the mother sequences $g_i(n)$ shifted by $2k$.
- When the sequences $\phi_{ik}(n) = h_i(2k-n)$ are themselves orthogonal, that is,

  $$\sum_n h_i(2k-n) h_j(2l-n) = \delta_{kl}\delta_{ij}, \quad i, j = 0, 1 \text{ and } k, l \in Z$$

  then

  $$g_i(n) = h_i(-n)$$

Table 6.2: Daubechies' low-pass filters of length 4, 6, 8, and 10

$h_1(0)$	0.4829629	0.33267	0.2303778	0.1601024
$h_1(1)$	0.8365163	0.806891	0.7148466	0.6038293
$h_1(2)$	0.2241439	0.459877	0.6308808	0.7243085
$h_1(3)$	−0.1294095	−0.135011	−0.0279838	0.1384281
$h_1(4)$		−0.08544	−0.1870348	−0.2422949
$h_1(5)$		0.03522	0.0308414	−0.0322449
$h_1(6)$			0.0328830	0.0775715
$h_1(7)$			−0.0105974	−0.0062415
$h_1(8)$				−0.0125807
$h_1(9)$				0.0033357

That is, the synthesis filters are the time reverse of the analysis ones. Such a filter bank is known as orthogonal or *paraunitary*, and we have the same set of mother sequences (h_i only) involved in both equations of the discrete time wavelet transform (6.89).

- A number of orthogonal and biorthogonal perfect reconstruction filter pairs have been proposed in the literature, [Daub 90, Vett 95]. Table 6.2 gives the coefficients for the first four of Daubechies' maximally flat orthogonal filters. The low-pass version $h_1(n)$ is shown. The high-pass versions are obtained as $h_0(n) = (-1)^n h_1(-n + 2L - 1)$, where L is the length of the filters.
- Besides the case of wavelet basis sequences with predefined values, a large research effort has been devoted to constructing such sequences that are optimized to the specific problem of interest. This has also been used in pattern recognition applications. For example, in [Mall 97] it is proposed to design the filters of the bank to optimize a class discriminant criterion. A different approach is followed in [Szu 92], where an optimal linear combination of predefined bases is sought for classification of speech signals.
- When implementing filter banks in practice, noncausal filters have to be appropriately delayed to make them realizable (Appendix D). This makes it necessary to involve certain delay elements at different points, in order to safeguard the perfect reconstruction property of the analysis–synthesis bank (Problem 6.19).
- In practice, the number of input samples $x(n)$ is finite, that is, $n = 0, 1, \ldots, N - 1$. Thus, for the computation of (6.89) some initial conditions are required. Zero, periodic, or symmetric extensions of the data are popular alternatives. Such implementation issues as well as algorithms for the efficient computation of the DTWT coefficients are discussed in [Vett 95, Chapter 6].

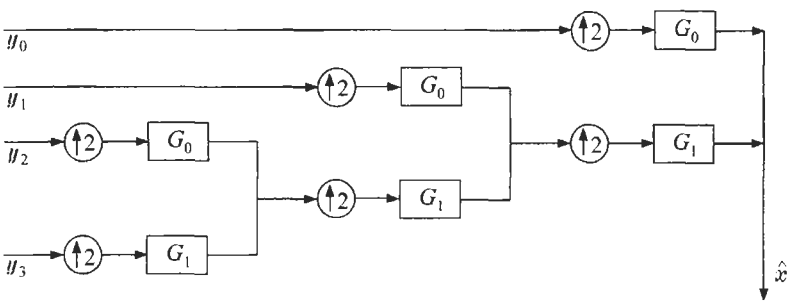


FIGURE 6.11: Tree-structured synthesis filter bank.

Many Bands Case

Figure 6.11 shows the synthesis part corresponding to the analysis bank of Figure 6.7, and it is a generalization of the two-band synthesis concept. Using the easily shown Noble identity given in Figure 6.12 (Problem 6.17), we end up with the equivalent structure of the synthesis part, shown in Figure 6.13. Let $f_i(n)$ be the impulse responses of the F_i filters. It is easy to see that the respective contribution of each $y_i(k)$ sequence to the output $\hat{x}(n)$ is

$$x_i(n) = \sum_k y_i(k) f_i(n - 2^{i+1}k) \quad i = 0, 1, \ldots, J-2$$

$$x_{J-1}(n) = \sum_k y_{J-1}(k) f_{J-1}(n - 2^{J-1}k)$$

$$\hat{x}(n) = \sum_{i=0}^{J-1} x_i(n)$$

where J is the number of bands, with $J = 4$ in the case of Figures 6.11 and 6.13. It can be shown [Vaid 93] that if the mother filters G_0, G_1, which generate the synthesis part, and the mother filters H_0, H_1 of the analysis part *satisfy the biorthogonality condition* (6.87), *then the J-level analysis–synthesis bank is also*

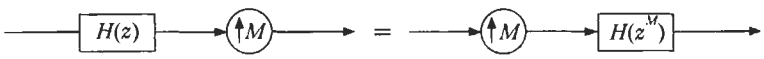


FIGURE 6.12: Noble identity II.

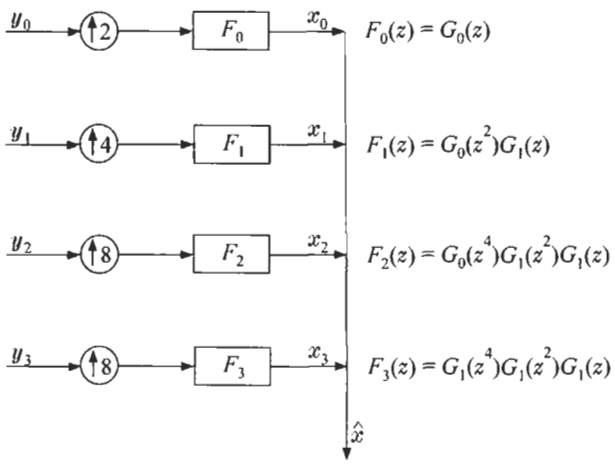


FIGURE 6.13: Equivalent of the tree-structured filter bank of Figure 6.11.

a perfect reconstruction filter bank, that is,

$$\hat{x}(n) = x(n) = \sum_{i=0}^{J-2} \sum_{k} y_i(k) f_i(n - 2^{i+1}k) + \sum_{k} y_{J-1}(k) f_{J-1}(n - 2^{J-1}k) \tag{6.90}$$

where

$$y_i(k) = \sum_{n} x(n) \hat{f}_i(2^{i+1}k - n), \quad i = 0, 1, \ldots, J-2 \tag{6.91}$$

$$y_{J-1}(k) = \sum_{n} x(n) \hat{f}_{J-1}(2^{J-1}k - n) \tag{6.92}$$

with $\hat{f}_i(k)$ being the impulse responses of the corresponding analysis band, in analogy with Figure 6.6b. To summarize our findings, let us define

$$\psi_{ik}(n) = f_i(n - 2^{i+1}k), \quad i = 0, 1, \ldots, J-2$$

$$\psi_{(J-1)k}(n) = f_{J-1}(n - 2^{J-1}k)$$

$$\phi_{ik}(n) = \hat{f}_i(2^{i+1}k - n) \quad i = 0, 1, \ldots, J-2$$

$$\phi_{(J-1)k}(n) = \hat{f}_{J-1}(2^{J-1}k - n)$$

Then from (6.90), (6.91), and (6.92) we obtain Table 6.3.

Table 6.3: The discrete time wavelet transform

$y_i(k) = \sum_n x(n)\phi_{ik}(n)$	DTWT
$x(n) = \sum_i \sum_k y_i(k)\psi_{ik}(n)$	Inverse DTWT
$\sum_n \phi_{ik}(n)\psi_{jl}(n) = \delta_{kl}\delta_{ij}$	Biorthogonal expansion
$\psi_{ik}(n) = \phi_{ik}(n)$ $\sum_n \phi_{ik}(n)\phi_{jl}(n) = \delta_{kl}\delta_{ij}$	Orthonormal expansion

Remarks

- A notable characteristic of the DTWT is that the basis sequences for each level i are power of 2 shifts of a corresponding mother sequence

$$\begin{aligned} \psi_{ik}(n) = \psi_{i0}(n - 2^r k), \quad & r = i + 1 \quad && \text{for } i \neq J - 1 \\ & \text{or } r = J - 1 && \text{for } i = J - 1 \\ \phi_{ik}(n) = \phi_{i0}(n - 2^r k), \quad & r = i + 1 && \text{for } i \neq J - 1 \\ & \text{or } r = J - 1 && \text{for } i = J - 1 \end{aligned}$$

  In the more elegant theory of continuous wavelet transform all analysis (synthesis) basis functions are produced from a *single* analysis (synthesis) mother function by dilations (time scaling) and shifts [Meye 93, Daub 90, Vett 95].
- The magic number 2, whose powers determine the shifts in the mother basis sequences, results from the successive splitting by two in the tree-structured filter banks, which we have adopted to introduce the DTWT. Filter banks of this type are known as *octave-band filter banks*. Their characteristic is that the bandwidth of each of the filters in the bank is the same in a logarithmic scale. Sometimes they are also called *constant-Q filter banks* to stress the fact that the ratio of the filters' bandwidth to the respective central frequency is constant. Generalizations of DTWT with another integer M in place of 2 can also be defined and used [Stef 93].

Example 6.4. The Haar Transform—The Epilogue

We have already seen that the Haar transform is equivalent to a tree-structured analysis filter bank. Let us now look at the synthesis problem. For the 8 × 8 Haar transform and

after a row reshuffling of the corresponding Haar matrix, we have

$$\begin{bmatrix} y_0(0) \\ y_0(1) \\ y_0(2) \\ y_0(3) \\ y_1(0) \\ y_1(1) \\ y_2(0) \\ y_3(0) \end{bmatrix} = \frac{1}{\sqrt{8}} \begin{bmatrix} 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(7) \end{bmatrix}$$

or

$$y = \hat{H}x$$

Thus the 8×8 Haar transform gives four coefficients at the finest resolution level 0, two at level 1, and one for each of the coarsest resolution levels 2 and 3. We will now design the corresponding synthesis bank to obtain $x(n)$ from these coefficients. The impulse responses of the Haar analysis filters are

$$h_1(n) = \begin{cases} \frac{1}{\sqrt{2}} & n = 0 \text{ or } n = -1 \\ 0 & \text{otherwise} \end{cases}$$

$$h_0(n) = \begin{cases} \frac{1}{\sqrt{2}} & n = 0 \\ -\frac{1}{\sqrt{2}} & n = -1 \\ 0 & \text{otherwise} \end{cases}$$

It is readily seen that

$$\sum_n h_i(2k - n)h_j(2l - n) = \delta_{ij}\delta_{kl}, \quad i, j = 0, 1$$

That is, the Haar filter bank is paraunitary. Thus, the synthesis filters can be defined as

$$g_i(n) = h_i(-n), \quad i = 0, 1$$

Hence

$$G_0(z) = \frac{1}{\sqrt{2}}(1 - z^{-1})$$

$$G_1(z) = \frac{1}{\sqrt{2}}(1 + z^{-1})$$

From the equivalent structure of the synthesis bank of Figure 6.13 we have

$$F_1(z) = G_1(z)G_0(z^2) = \frac{1}{2}(1 + z^{-1} - z^{-2} - z^{-3})$$

and the respective impulse response is

$$f_1(n) = \begin{cases} \frac{1}{2} & n = 0, 1 \\ -\frac{1}{2} & n = 2, 3 \\ 0 & \text{otherwise} \end{cases}$$

Following similar arguments, we have

$$f_2(n) = \begin{cases} \frac{1}{\sqrt{8}} & n = 0, 1, 2, 3 \\ -\frac{1}{\sqrt{8}} & n = 4, 5, 6, 7 \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(n) = \begin{cases} \frac{1}{\sqrt{8}} & n = 0, 1, \ldots, 7 \\ 0 & \text{otherwise} \end{cases}$$

If we now insert these values in (6.90) and collect the values of $x(n)$ together, we get

$$\begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(7) \end{bmatrix} = \frac{1}{\sqrt{8}} \begin{bmatrix} 2 & 0 & 0 & 0 & \sqrt{2} & 0 & 1 & 1 \\ -2 & 0 & 0 & 0 & \sqrt{2} & 0 & 1 & 1 \\ 0 & 2 & 0 & 0 & -\sqrt{2} & 0 & 1 & 1 \\ 0 & -2 & 0 & 0 & -\sqrt{2} & 0 & 1 & 1 \\ 0 & 0 & 2 & 0 & 0 & \sqrt{2} & -1 & 1 \\ 0 & 0 & -2 & 0 & 0 & \sqrt{2} & -1 & 1 \\ 0 & 0 & 0 & 2 & 0 & -\sqrt{2} & -1 & 1 \\ 0 & 0 & 0 & -2 & 0 & -\sqrt{2} & -1 & 1 \end{bmatrix} \begin{bmatrix} y_0(0) \\ y_0(1) \\ y_0(2) \\ y_0(3) \\ y_1(0) \\ y_1(1) \\ y_2(0) \\ y_3(0) \end{bmatrix}$$

or

$$\boldsymbol{x} = \hat{H}^T \boldsymbol{y}$$

That is, we reobtain the inverse (within a permutation) Haar transform. Hence, we can now state that *the Haar transform and its inverse form a DTWT pair, using the orthogonal Haar sequences as the basis for the wavelet expansion.*

6.12 THE MULTIRESOLUTION INTERPRETATION

The goal of this section is to highlight, without resorting to mathematical details, an important aspect of the wavelet transform that accounts for its success as a tool in pattern recognition as well as in numerous other applications. Let us assume for simplicity that the two filters in the analysis–synthesis bank of a paraunitary filter bank are ideal low/high pass. Figure 6.14 shows the magnitude responses

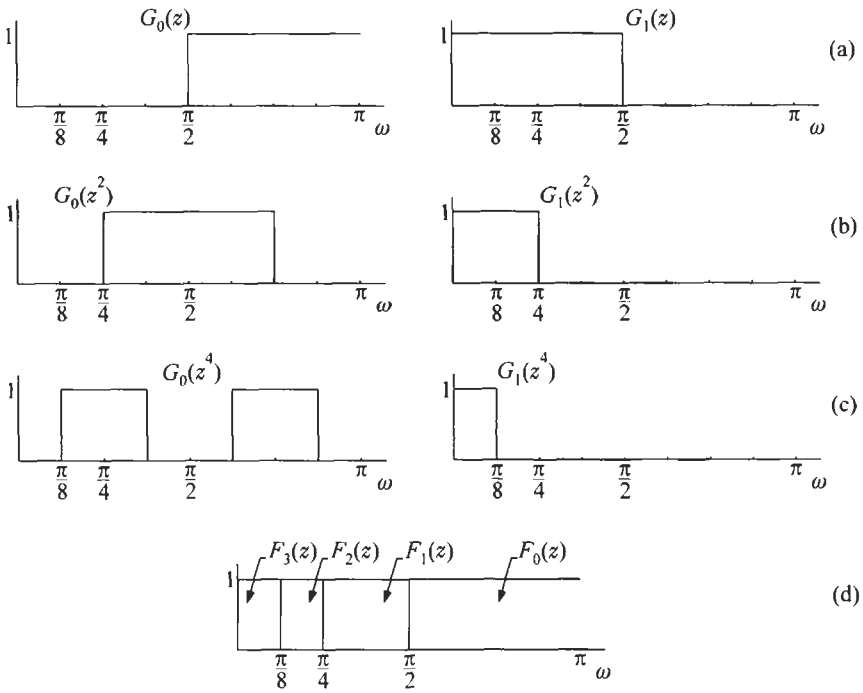


FIGURE 6.14: Filter bandwidths in an octave band filter bank.

of the respective filters in the equivalent of the tree-structured octave band filter bank of Figure 6.13. The width of the frequency response (bandwidth) is halved for each level of the tree (Figure 6.14d). That is, the "detail" resolution (high-pass) filters have a wide bandwidth and the "coarse" resolution (low-pass) filters are of narrow bandwidth. Filters F_3 and F_2, the two coarser resolution ones, are of the same bandwidth. These observations are true for any octave band filter bank of any number of levels J. That is, the width of $F_i(z)$ is half of the width of $F_{i-1}(z)$ and the widths of F_{J-2} and F_{J-1} are equal. This multiresolution viewpoint of the DTWT is a source of its power as a tool, and it is worth spending some time on it.

It is known (uncertainty principle) that filters with narrow bandwidth have long impulse responses and filters with wide bandwidth have short impulse responses. Let us take for example the Haar filter bank. The filter impulse response at level zero, $h_0(n) = \hat{f}_0(n)$, is $(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$ and at level three (i.e., corresponding to Eq. (6.83)), $\hat{f}_3(n)$, is $\frac{1}{\sqrt{8}}(1, 1, 1, 1, 1, 1, 1, 1)$. Since the output of each filter of the analysis bank, that is, the DTWT coefficients, is the convolution of the input sequence with the respective impulse response, the filter tends to spread out the

input activity. For example, a single impulse in the input of the $\hat{F}_3(z)$ filter produces a sequence of eight samples at the output. Thus, if our goal is to identify sudden (short in time) changes in an input signal, it is apparent that one should use filters of short impulse response to be able to obtain good time locality. Otherwise, the sudden activity will spread in time. Hence, for sudden changes in time (rich in high-frequency components) one needs a "detailed" analysis filter, that is, a short impulse response. For slowly time-varying activities, rich in low frequencies, one needs a "less detailed" analysis filter, so as to be able to "see" the longer scale variations. Thus, detail is not of interest here and long impulse responses are required. In other words, *the resolution should match the scale of the activity under investigation.*

The wavelet transform provides the means of analyzing the input signal into a number of different resolution levels in a hierarchical fashion. This is also known as *multiresolution analysis*. Thus, signal components corresponding to different physical activities can be best represented at different resolution levels: short high-frequency activities at the finer resolution and long low-frequency ones at the coarser resolution levels. It turns out that this coarse-to-fine analysis strategy is appropriate for a number of pattern recognition tasks.

On the synthesis part, the signal can be reconstructed from its multiresolution components. See, for example, Figure 6.13. The sequence $x(n)$ is synthesized first by its coarser component $x_3(n)$ and then higher frequency (detailed) components are added, resulting in a *successively finer approximation*. When the component of the finest detail, $x_0(n)$, is added, the original signal is obtained. This philosophy is at the heart of a number of signal compression schemes.

Remarks

- The analysis of a signal in a number of components via a filter bank is not new and goes back to the work of Gabor in the 1940s. It is directly related to the *short-time Fourier transform* defined as [Gabo 46, Vett 95]

$$X_s(\omega, m) = \sum_{n=-\infty}^{\infty} x(n)\theta(n-m)\exp(-j\omega n) \tag{6.93}$$

  where $\theta(n)$ is a window sequence, whose center is successively moved to the different points m. Thus, each time, the part of the sequence $x(n)$ around m (depending on the window's effective width) is selected and Fourier transformed. It can be shown that this is equivalent to filtering the signal $x(n)$ by a bank of filters, each centered at a different frequency but all of them having the *same bandwidth* (Problem 6.20). This is its drawback, because low- and high-frequency signal components are "looked" at through the same window in time, resulting in poor overall localization of the events. What is really needed is a long window to analyze slowly time-varying low-frequency

components and a narrow window to detect high-frequency short-time activities. As we saw, this is offered by a tree-structured octave-band filter bank, associated with the DTWT.

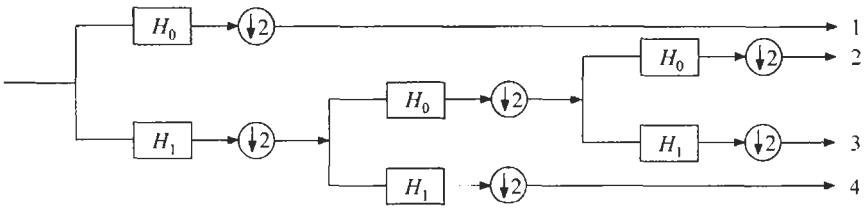
- All we have said about wavelet transforms and multiresolution analysis is just a glimpse of the whole story, a story that is really worth further effort see, for example, [Daub 90].

6.13 WAVELET PACKETS

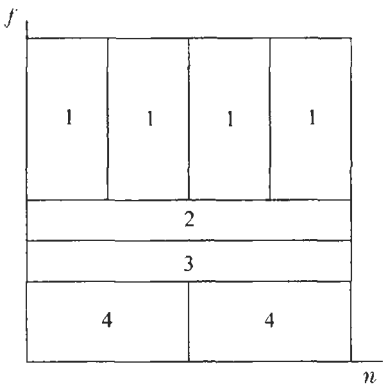
The DTWT has been introduced via an octave-band filter bank, and the wavelet coefficients result at the outputs of the bank, when its input is fed with the signal of interest. The octave-band filter bank is constructed by successively splitting by two the lowest frequency band (leaf) of the tree-structured bank (Figure 6.7). However, there are many cases in which most of the activity is not in the low-frequency band but in the middle or high-frequency parts of the spectrum. In such cases, it may be useful to be able to allocate finer frequency bandwidths in the bands where the activity occurs. As we will see later on in the chapter, this can boost the discriminatory power of our system from a classification point of view, which is always our main interest. Figure 6.15a shows an example of a tree-structured filter bank but with the finer frequency splitting occurring at a midfrequency band. Figure 6.15b shows the resulting bandwidths for each of the (ideal) filters in the bank (f-axis) and the respective window length of the impulse responses in the time domain (n-axis). In other words, filters 2 and 3 have half the bandwidth and twice the impulse response of 4. Furthermore, they have one fourth of the bandwidth and a four times longer impulse response than that of 1. For comparison, Figure 6.16 shows the frequency–time resolution plots for an octave-band filter bank (a) and for a bank with equal bandwidths (b), associated with the DTWT and the short-time Fourier transform, respectively. Having freed ourselves from the octave-band tree structure, filter banks can be constructed by various tree growth strategies, with that of Figure 6.15 being just one possibility. As was the case with the octave-band philosophy, these arbitrary tree structures also lead to a set of basis sequences for discrete signal expansions [Coif 92] called *wavelet packets*. Following arguments similar to those in Section 6.11 and using filters with the perfect reconstruction property, the basis sequences for the wavelet packets result from the respective impulse responses of the synthesis bank, after the appropriate, for each level, power of two time shifts.

6.14 A LOOK AT TWO-DIMENSIONAL GENERALIZATIONS

All the concepts discussed so far can be carried over in the two-dimensional case. No doubt, the task is even more challenging now. How can one define



(a)



(b)

FIGURE 6.15: Wavelet packet tree structure.

subsampling here? The straightforward way is via the "separable" philosophy. That is, we first transform (filter) the columns of the two-dimensional sequence and then the resulting rows. This leads to the subsampling shown in Figure 6.17. In other words, we leave out every other row and every other column (for subsampling by 2). Figure 6.18 shows the filter bank structure that complies with this philosophy. The image sequence $I(m, n)$ appears in the filters of stage 1 column after column and the respective outputs are subsampled by 2. The resulting subsampled images are in turn filtered at stage two, but now they are fed into the filters row after row.

Assuming H_0 to be the (ideal) high-pass and H_1 the low-pass filter, the four frequency bands that are formed by the previous procedure are illustrated in Figure 6.19a. The area H_1H_1 corresponds to low-pass columns and rows, H_1H_0 to low-pass columns and high-pass rows, and so forth. Figure 6.19b shows the resulting segmentation of the frequency domain when the low-pass area H_1H_1 is successively split by repeating the procedure.

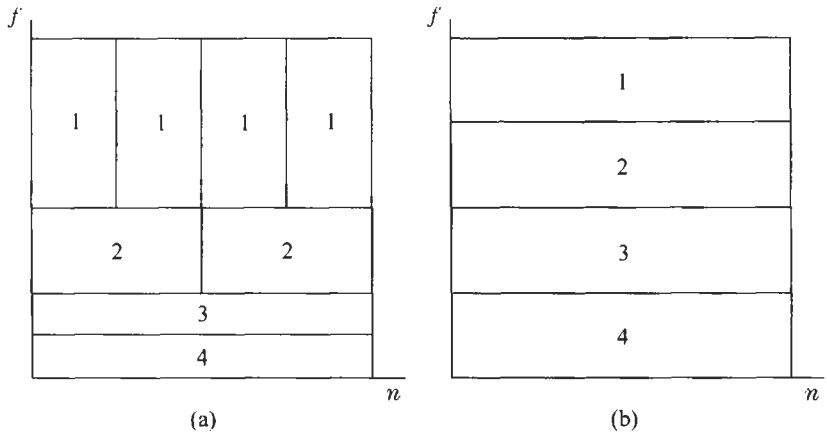


FIGURE 6.16: Frequency versus time resolution for (a) octave band and (b) equal bandwidth filter banks.

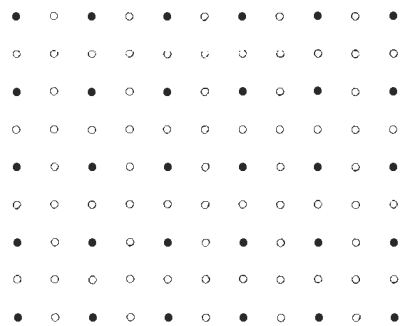


FIGURE 6.17: Separable subsampling by 2 for images.

Example 6.5. Figure 6.20 shows a 64 × 64 image of a triangle. The three "line" images are the 32 × 32 images resulting when passing the triangle image through the structure of Figure 6.18. In the columnwise filtering of the first stage the vertical line goes through the low-pass H_1 filter (no variation across it) and the horizontal and diagonal lines go through the high-pass H_0. This is because in a columnwise filtering these appear as impulses in each column, thus rich in high frequencies. In the row scanning of the second stage it will be the horizontal line that will go through the low-pass filter. Similar reasoning explains the position of the various parts of the triangle in the different bands.

Although this is obviously a very simplified example, it is quite instructive. It demonstrates how the original image can be obtained from its multiresolution components and

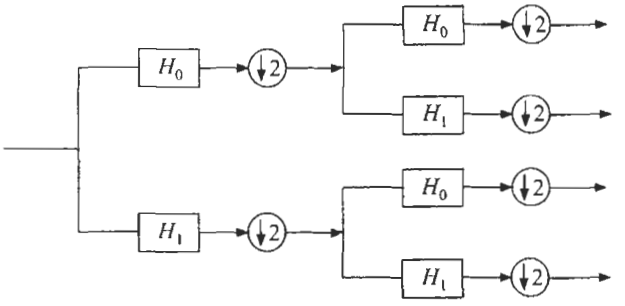


FIGURE 6.18: Basic element for a two-dimensional filter bank, leading to separable subsampling by 2.

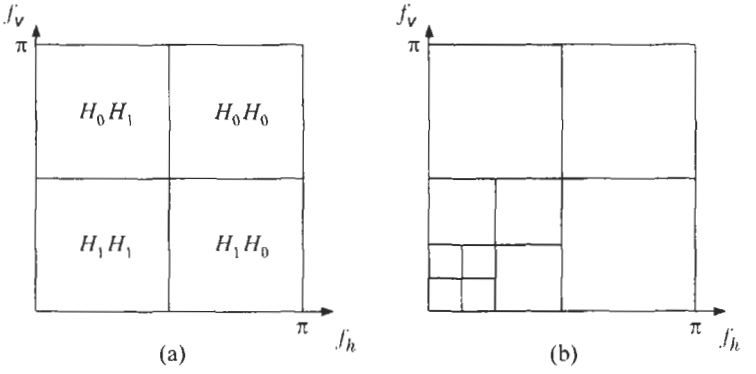


FIGURE 6.19: (a) Frequency domain division corresponding to the filter bank of Figure 6.18 and (b) the result of a successive division of the low-pass part of the spectrum.

also how different characteristics (directional in this case) of the whole may be isolated at different bands.

6.15 APPLICATIONS

All the transforms we have studied in this chapter are good candidates for feature generation, and they have been used extensively in various pattern recognition tasks. However, the wavelet transform offers an extra advantage, which in some cases can be beneficially exploited. *Its multiresolution properties conform to the way perception is achieved by humans, through their hearing and visual systems.*

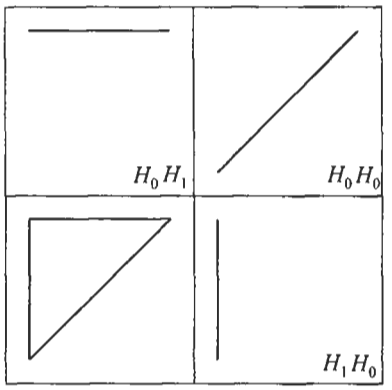


FIGURE 6.20: A triangle image and its filtered versions through the filter bank of Figure 6.18.

The human ear exhibits decreasing resolution at higher frequencies, in a way that is uniform on a logarithmic scale (octave bands) [Flan 72]. Experiments in psychophysics and physiology show that the human visual cortex perceives by decomposing the stimuli in a number of frequency bands [Camp 68, Levi 85], which are also dependent on the spatial orientation [Camp 66]. Experimental results in [Nach 75] indicate that these frequency bands have the approximate bandwidth of an octave. The similarities between the mechanism with which human perception systems treat the respective stimuli and the processing techniques that split the signal into various (spatial) frequency bands, in a way similar to the wavelet transform, justify the use of the latter in pattern recognition tasks [Mall 89].

The following examples come from two of the most important areas of interest in today's pattern recognition applications.

Recognition of Handwritten Characters

The development of OCR systems is of particular importance in various application areas. One of the most challenging among them is the recognition of handwritten characters. Figure 6.21 shows the character "3" as well as its boundary contour after the application of a contour tracing algorithm [Pita 92]. The task now becomes one of shape recognition. As discussed in more detail in Chapter 7, the boundary can be represented as a closed parametric curve in the complex plane

$$u(n) = x(n) + jy(n), \quad 0 \leq n \leq N-1 \tag{6.94}$$

with N being the number of samples (pixels) found tracing the contour and $x(n)$, $y(n)$ the corresponding coordinates. The first point $(x(0), y(0))$ of the

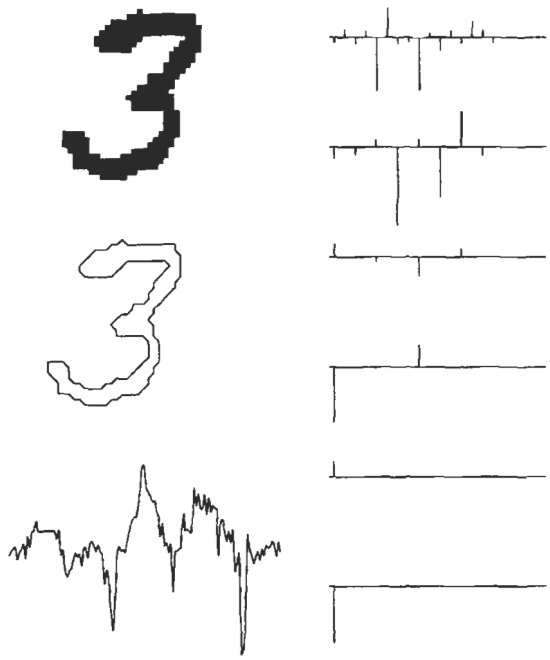


FIGURE 6.21: Wavelet coefficients corresponding to the curvature of the boundary of number "3."

sequence is considered as the origin. Fourier methods have been used extensively in such classification tasks, by obtaining the DFT of $u(n)$ and keeping a sufficient number, from the total of N, of Fourier components as features. An alternative way is to extract the features from the wavelet domain. In other words, $x(n)$ and $y(n)$ are independently filtered through a tree-structured filter bank of appropriate resolution depth and the resulting wavelet coefficients are used as features. The low-frequency components account for the basic shape of the character and are less sensitive to varying writing styles. The high-frequency ones account for the details and are more sensitive to the specific handwriting style. In [Wuns 95] a comparative study was carried out using the same number of DTWT coefficients and Fourier-based features, with a neural network classifier. The experiments showed that classification based on wavelet coefficients resulted in reduced error rates. Furthermore, it was pointed out that Fourier-based features exhibited larger within class variance and weaker between class separation than the wavelet-based ones. A major disadvantage associated with the wavelet coefficients is that *they are not shift invariant*. That is, if we rotate/translate a character, the resulting coefficients will not be the same. This is a consequence of the subsampling process [Mall 89] (Problem 6.21).

In other words, if

$$x'(n) = x(n - n_0)$$

and $y'(n)$, $y(n)$ are the sequences of wavelet coefficients of $x'(n)$ and $x(n)$, respectively, then, in general

$$y'(n) \neq y(n - n_0)$$

This has led to research in designing filter banks that aim to overcome this property [Marco 95, Hui 96]. The shift dependence obviously makes the wavelet coefficients also sensitive to the choice of the initial point from which the contour is traced, as already described. As we will see in more detail in Chapter 7, the Fourier coefficients are also dependent on shifts *but* in a deterministic way. Thus, various normalizing techniques exist that result in shift-invariant feature parameters [Crim 82, Arbt 90]. To overcome the problems associated with the choice of the initial point within the contour, in order to minimize its effects on the wavelet coefficients, a number of techniques have been suggested and used in practice. A simple method is to select a specific point resulting during the scanning process, for example, the first pixel when scanning the character from left to right. Other more "intelligent" ways have also been suggested [Wuns 95, Chua 96].

An alternative approach to that of Eq. (6.94) is to describe a contour in terms of the arc length between a given point and the origin, within the contour. As arc length t at a given point, we define the number of consecutive pixels between the given point and the point considered as the origin. The contour description can now be achieved via a one-parameter real-valued function, the arc tangent angle $\theta(t)$ or the corresponding curvature $\kappa(t)$, defined as

$$\theta(t) = \tan^{-1}\left[\frac{dy(t)}{dx(t)}\right]$$

$$\kappa(t) = \frac{d\theta(t)}{dt}$$

where $x(t)$, $y(t)$ are the coordinates of the respective point as a function of length t from the origin and $dt = \sqrt{dy^2 + dx^2}$. A further discussion of this is provided in Chapter 7. [Kapo 96] suggests wavelet transforming the curvature of the contour and using the corresponding wavelet coefficients as features. Figure 6.21 shows the coefficients resulting from the wavelet analysis of the curvature (bottom left) of the boundary contour of number 3. The wavelet basis used for the analysis was Daubechies' biorthogonal pairs (3,9) [Vett 95]. Six successive resolution levels are shown, the finest being on top and the coarsest at the bottom. Extensive experimentation with a number of different characters shows that the use of wavelet coefficients from more than six resolution levels adds no further discriminatory

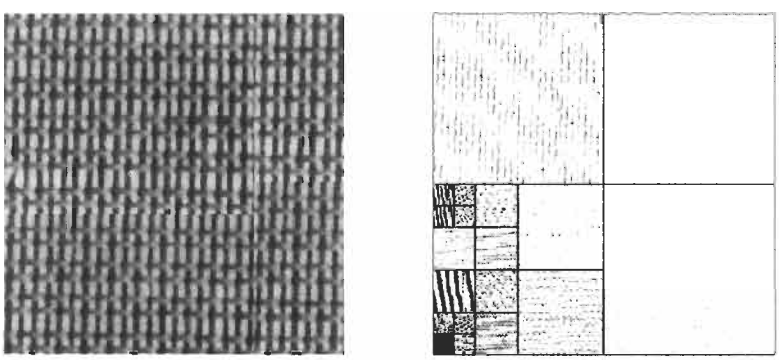
information to the system. Thus, each of the resulting feature vectors has 32 components. A different philosophy is followed in [Geze 00, Geze 02], in the context of an OCR system for the Greek Orthodox Byzantine music notation. The wavelet transform is applied to the vector combining the four projections (horizontal, vertical, left-diagonal, right-diagonal) of the characters. It turns out that such an approach leads to an efficient coding of the directional properties of the characters.

Texture Classification

Texturecharacterization in image analysis tasks is another area where the wavelet transform, as well as the other transforms discussed in this chapter, has been heavily utilized. The basic approach is similar to that in the OCR example. However, because texture is a property of the image region and not of its boundary, the 2-D variants of the transforms are used. The 2-D wavelet transform offers the advantages of spatial frequency and orientation selectivity, provided the appropriate bases are chosen. The information compaction properties result from the fact that most of the energy activity is concentrated in certain resolution levels, as was the case with the OCR example considered earlier. The wavelet coefficients of these levels are then selected as features to form the feature vectors. Sometimes a function of the features is employed, such as the energy, $\sum_i y_i^2$, or the entropy, $\sum_i y_i^2 \log y_i^2$, where y_i are the respective wavelet coefficients at each resolution level [Lain 93].

In many cases, the underlying image texture exhibits a great deal of activity in middle or high-frequency bands.

Figure 6.22a is an example of a textured image taken from [Brod 66]. In such cases, one must pay attention in the bands of high energy, instead of looking with



(a) (b)

FIGURE 6.22: An example of (a) a textured image and (b) its corresponding wavelet packet transform.

fine frequency bandwidths at low-frequency bands of low energy. This leads to the adoption of wavelet packets, discussed in Section 6.13. There, we saw that there is a number of choices for splitting the frequency bands, leading to different wavelet packets. In [Chan 93] a dynamic procedure is suggested, depending on the particular texture image. A threshold C is selected prior to the analysis. If the output energy in a band is less than C, no splitting of the band is carried out. If it is higher than C, the band is further split. Splitting terminates if subimages, after filtering and subsampling, become small, for example, 16×16. It is then suggested that the energy values at the J (a preselected number) most dominant (energywise) bands be used as features. In [Mojs 00] the effects on texture characterization of the properties of the analysis band are considered. It is demonstrated that the choice of the analysis filters may have a significant influence on the classification performance. In [Unse 95, Lain 96] another variation of the DTWT is adopted, called the *discrete wavelet frame*. The difference from DTWT is that the filter outputs in the bank are not subsampled. Although this leads to a redundant representation, it leads to a texture description tolerant to translations.

A procedure similar in concept to the DTWT one is to employ a bank of 2-D Gabor filters to perform the splitting of the image into a number of frequency bands. The impulse response (point spread function, Appendix D) of the complex 2-D *Gabor filter* is given as the product of a Gaussian low-pass filter with a complex exponential, that is [Bovi 91],

$$h(x, y) = g'(x, y) \exp(j(\omega_x x + \omega_y y)) \tag{6.95}$$

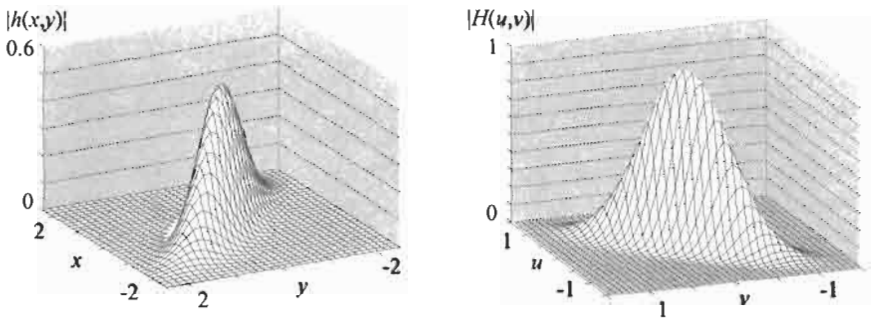
where

$$g'(x, y) = \frac{1}{\lambda\sigma^2} g\left(\frac{x'}{\lambda\sigma}, \quad \frac{y'}{\sigma}\right), g(x, y) = \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right) \tag{6.96}$$

and

$$\begin{aligned} x' &= x \cos\theta + y \sin\theta \\ y' &= -x \sin\theta + y \cos\theta \end{aligned} \tag{6.97}$$

That is, $g'(x, y)$ is a version of the Gaussian $g(x, y)$ that is spatially scaled and rotated by θ. The parameter σ is the spatial scaling, which controls the width of the filter impulse response, and λ defines the aspect ratio of the filter, which determines the directionality of the filter, that is no longer circularly symmetric. The orientation angle θ is usually chosen to be equal to the direction of the filter's



(a) (b)

FIGURE 6.23: Plot of (a) the magnitude of point spread function of a Gabor filter and (b) the magnitude of its Fourier transform.

center circular frequency

$$\omega = \sqrt{\omega_x^2 + \omega_y^2} \tag{6.98}$$

That is,

$$\theta = \tan^{-1} \frac{\omega_y}{\omega_x} \tag{6.99}$$

By varying the free parameters σ, λ, ω, and θ, filters of arbitrary orientation and bandwidth characteristics are obtained (Problem 6.22). Figure 6.23a shows the magnitude of the complex Gabor filter for $\sigma = 1.0$, $\lambda = 0.3$. Figure 6.23b shows the magnitude of the corresponding spatial frequency response. The choice of the Gabor filters is justified by the fact that these filters offer the optimal trade-off between spectral bandwidth and spatial localization. In Section 6.12 we saw that the shorter the filter's impulse response (spatial localization), the wider its frequency bandwidth and vice versa, according to the *uncertainty principle* [Papo 91], that is,

$$\begin{aligned} \Delta x \Delta \omega_x &\geq \frac{1}{2} \\ \Delta y \Delta \omega_y &\geq \frac{1}{2} \end{aligned} \tag{6.100}$$

In [Daug 85] it has been shown that the 2-D Gabor filters attain the minimum uncertainty bound. For digital images, a sampling of the above functions has to be performed, which introduces aliasing errors regardless of the sampling interval. This happens because Gabor filters are not bandlimited, but have a Gaussian-shaped frequency response (Problem 6.23). The topic is treated in [Bovi 90].

Gabor filter banks for analyzing the image in a number of bands, in the context of texture classification, have been successfully employed in a number of cases, [Jain 91, Turn 86, Bovi 90, Hale 95, Hale 99, Weld 96]. A set of Gabor filters centered at different frequencies and having different orientations are used to cover the frequency range of interest, using various frequency and orientation bandwidths. Images are then filtered through this set of filters and the features are generated from the resulting output samples. For example, the output energies of the Gabor filters may be chosen to be the respective features. Thus, using this strategy, the generated features encode classification information related to the spatial frequency as well as the orientation activity of the various textures. In order to grasp most of the textural information, techniques have been proposed to place the centers of the Gabor filters in the most "important" image frequencies [Pich 96]. A comparative study of various transform-based features is provided, in the context of texture classification, in [Chan 93].

Problems

6.1 Show the equivalence (a) between (6.5) and (6.6) and (b) between (6.7) and (6.8).

6.2 Consider the separable transform $Y = UXV^T$. Then show that if Y, X are turned into the row-ordered vectors $\boldsymbol{y}, \boldsymbol{x}$, respectively, then $\boldsymbol{y} = (U \otimes V)\boldsymbol{x}$, where $\otimes$ denotes the Kronecker product of two matrices.

6.3 Let $\boldsymbol{e}_i, i = 0, 1, \ldots, N-1$, be any orthonormal basis in the N-dimensional space. Show that the MSE between an N-dimensional vector and an m-dimensional projection of it is minimized if (a) the basis consists of the eigenvectors of R_x and (b) the m-dimensional subspace is the one spanned by the eigenvectors corresponding to the m largest eigenvalues. Furthermore, the projection onto the latter subspace is the one that maximizes the sum of the variances of its components.
Hint: Minimize the mean square error $E[\|\boldsymbol{\epsilon}\|^2]$ subject to the constraint $\boldsymbol{e}_i^T \boldsymbol{e}_i = 1$.

6.4 Consider an N-dimensional random vector $\boldsymbol{x}$, which is approximated by

$$\hat{\boldsymbol{x}} = \sum_{i=0}^{m-1} y_i \boldsymbol{e}_i + \sum_{i=m}^{N-1} c_i \boldsymbol{e}_i$$

where c_i are nonrandom constants and $\boldsymbol{e}_i, i = 0, 1, 2, \ldots, N-1$, constitute an orthonormal basis. Show that the minimum mean square error $E\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2$ is achieved if (a) $c_i = E[y_i], i = m, \ldots, N-1$; (b) the orthonormal basis consists of the eigenvectors of Σ_x; and (c) $\boldsymbol{e}_i, i = m, \ldots, N-1$, correspond to the $N-m$ smallest eigenvalues.

6.5 If X is a rank r matrix, show that the two square matrices XX^H and X^HX have the same nonzero eigenvalues.

6.6 Show Eq. (6.33).

6.7 Given the matrix

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 3 \end{bmatrix}$$

compute its SVD representation.

6.8 Show the orthogonality of the DFT matrix W and also identity (6.59).

6.9 Given the image array

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

compute its two-dimensional DFT transform.

6.10 For one of the images available at the web site of the book, write a program to compute its DFT transform. Use a routine to implement the fast Fourier transform and plot the magnitude of the resulting Fourier transform.

6.11 Show the orthogonality of the DCT transform.

6.12 Compute the DCT of the image array of Problem 6.9.

6.13 Develop a program to compute the DCT for one of the images available from the web site of the book.

6.14 Show the orthogonality of the Hadamard transform.

6.15 Compute the Hadamard transform for a 2×2 submatrix of the matrix of Problem 6.9.

6.16 Show the orthogonality of the Haar transform.

6.17 Show the two Noble identities of Figures 6.6a and 6.13.

6.18 Show the equivalence between the tree structure of Figure 6.5 and that of Figure 6.6b.

6.19 Consider the perfect reconstruction two-band Haar bank. Show that

(a) If we make the analysis filters causal by delaying each of them by one sample, then the reconstructed sequence $\hat{x}(n)$ is delayed by one sample, that is, $\hat{x}(n) = x(n-1)$. In the more general case, if both the analysis filters have to be delayed by L, then the output of the band is also delayed by L.

(b) If this is repeated with the more general N-band case, show that the output is delayed as $\hat{x}(n) = x(n - (2^{N-1} - 1)L)$ and a delay must be inserted in each band to safeguard perfect reconstruction. Figure 6.24 shows the three-band case. In the general case, the delay element in each band is $z^{-(2^{N-i-1}-1)L}$, $i = 0, 1, \ldots, N-1$.

6.20 Show that the short-time Fourier transform defined in (6.97) is equal to

$$X_s(\omega, m) = \exp(-j\omega m) \sum_{n=-\infty}^{\infty} x(n)\theta(n-m)\exp(j\omega(m-n))$$

Verify that this is equivalent to filtering the sequence $x(n)$ with different filters of the same bandwidth but centered at different frequencies.

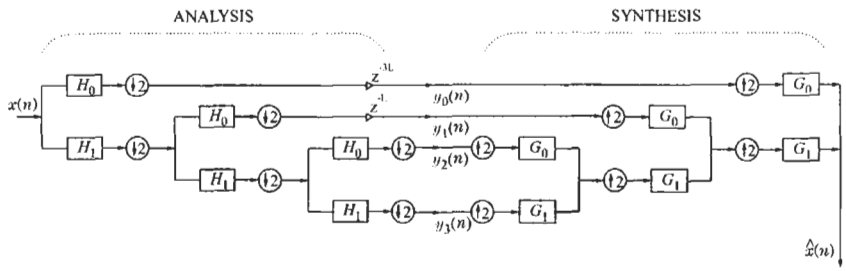


FIGURE 6.24: A three-band perfect reconstruction filter bank with causal analysis and synthesis filters.

6.21 Show that the process of filtering and then subsampling is equivalent to the action of a linear but time-varying system. As a matter of fact, it is a periodically time-varying linear system. The same is true for the process of upsampling followed by linear filtering.

6.22 Show that the frequency (octave) and orientation (radians) half-peak bandwidths for Gabor filters are given by B_f, B_θ, respectively, where

$$B_f = \log_2 \frac{\omega\lambda\sigma + \sqrt{2\ln 2}}{\omega\lambda\sigma - \sqrt{2\ln 2}}$$

$$B_\theta = 2\tan^{-1}\frac{\sqrt{2\ln 2}}{\omega\sigma}$$

Compute these for different values of λ, σ.

6.23 Show that the Fourier transform of the 2-D Gabor filter response $h(x, y)$ is given by

$$H(u, v) = \exp\left(-\frac{\sigma^2}{2}\{(u' - \omega'_x)^2\lambda^2 + (v' - \omega'_y)^2\}\right)$$

where

$$u' = u\cos\theta + v\sin\theta$$
$$v' = -u\sin\theta + v\cos\theta$$

and ω'_x, ω'_y the corresponding versions of ω_x, ω_y rotated by θ. Draw its magnitude versus frequency (u, v) for different values of λ and θ.

References

[Akan 93] Akansu A.N., Hadda R.A. *Multiresolution Signal Decomposition*, Academic Press, 1992.

[Arbt 90] Arbter K., Snyder W. E., Burkhardt H., Hirzinger G. "Application of affine-invariant Fourier descriptors to recognition of 3-D objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12(7), pp. 640–647, 1990.

[Atti 92] Attick J.J. "Entropy minimization: A design principle for sensory perception," *International Journal of Neural Systems*, Vol. 3, pp. 81–90, 1992.

[Barl 89] Barlow H.B. "Unsupervised learning," *Neural Computation*, Vol. 1, pp. 295–311, 1989.

[Bell 00] Bell A.J. "Information theory, independent component analysis, and applications," in *Unsupervised Adaptive Filtering, Part I: Blind Source Separation* (Haykin S., ed.), pp. 237–264, John Wiley & Sons, 2000.

[Bovi 90] Bovic A.C., Clark M., Geisler W.S. "Multichannel texture analysis using localized spatial filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12(1), pp. 55–73, 1990.

[Bovi 91] Bovic A.C. "Analysis of multichannel narrow-band filters for image texture segmentation," *IEEE Transactions on Signal Processing*, Vol. 39(9), pp. 2025–2044, 1991.

[Brod 66] Brodatz P. *Textures: A Photographic Album for Artists and Designers*, Dover, New York, 1966.

[Burt 83] Burt P.J., Adelson E.H. "The Laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, Vol. 31(4), pp. 532–540, 1983.

[Camp 66] Campell F., Kulikowski J. "Orientation selectivity of the human visual system," *Journal of Physiology*, Vol. 197, pp. 437–441, 1966.

[Camp 68] Campell F., Robson J. "Application of Fourier analysis to the visibility of gratings," *Journal of Physiology*, Vol. 197, pp. 551–566, 1968.

[Chan 93] Chang T., Kuo C.C.J. "Texture analysis and classification with tree structured wavelet transform," *IEEE Transactions on Image Processing*, Vol. 2(4), pp. 429–442, 1993.

[Chua 96] Chuang G.C.H., Kuo C.C.J. "Wavelet descriptor of planar curves: Theory and applications," *IEEE Transactions on Image Processing*, Vol. 5(1), pp. 56–71, 1996.

[Coif 92] Coifman R.R., Meyer Y., Wickerhauser M.V. "Wavelet analysis and signal processing," in *Wavelets and Their Applications* (Ruskai M.B. et al. eds.), pp. 153–178, Jones and Barlett, 1992.

[Como 94] Comon P. "Independent component analysis—A new concept?" *Signal Processing*, Vol. 36, pp. 287–314, 1994.

[Crim 82] Crimmins T.R. "A complete set of Fourier descriptors for two dimensional shapes," *IEEE Transactions on Systems, Man Cybernetics*, Vol. 12(6), pp. 848–855, 1982.

[Daub 90] Daubechies I. *Ten Lectures on Wavelets*, SIAM, Philadelphia, 1991.

[Daug 85] Daugman J.G. "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two dimensional visual cortical filters," *Journal of Optical Society of America*, Vol. 2, pp. 1160–1169, 1985.

[Deco 95] Deco G., Obradovic D. "Linear redundancy reduction learning," *Neural Networks*, Vol. 8(5), pp. 751–755, 1995.

[Diam 96] Diamantaras K.I., Kung S.Y. *Principal Component Neural Networks*, John Wiley, 1996.

[Doug 00] Douglas S.C., Amari S. "Natural gradient adaptation," in *Unsupervised Adaptive Filtering, Part I: Blind Source Separation* (Haykin S. ed.), pp. 13–61, John Wiley & Sons, 2000.

[Este 77] Esteban D., Galand C. "Application of quadrature mirror filters to split band voice coding schemes," *Proceedings of the IEEE Conference on Acoustics Speech and Signal Procesing*, pp. 191–195, May 1977.

[Fiel 94] Field D.J. "What is the goal of sensory coding?" *Neural Computation*, Vol. 6, pp. 559–601, 1994.

[Flan 72] Flanagan J.L. *Speech Analysis, Synthesis and Perception*, Springer Verlag, New York, 1972.

[Gabo 46] Gabor D. "Theory of communications," *Journal of the Institute of Elec. Eng.*, Vol. 93, pp. 429–457, 1946.

[Geze 00] Gezerlis V., Theodoridis, S. "An optical music recognition system for the notation of Orthodox Hellenic Byzantine music," *Proceedings of the International Conference on Pattern Recognition (ICPR)*, Barcelona, 2000.

[Geze 02] Gezerlis V., Theodoridis S. "Optical character recognition of the Orthodox Hellenic Byzantine music," *Pattern Recognition*, Vol. 35(4), pp. 895–914, 2002.

[Hale 95] Haley G., Manjunath B.S. "Rotation-invariant texture classification using modified Gabor filters," *IEEE International Conference on Image Processing*, pp. 262–265, 1995.

[Hale 99] Haley G., Manjunath B.S. "Rotation-invariant texture classification using complete space frequency model," *IEEE Transactions on Image Processing*, Vol. 8(2), pp. 255–269, 1999.

[Hayk 99] Haykin S. *Neural Networks—A Comprehensive Foundation*, 2nd edition, Prentice Hall, 1999.

[Hayk 00] Haykin S. (ed.) *Unsupervised Adaptive Filtering, Part I: Blind Source Separtion*, John Wiley & Sons, 2000.

[Hube 85] Huber P.J. "Projection pursuit," *The Annals of Statistics*, Vol. 13(2), pp. 435–475, 1985.

[Hui 96] Hui Y., Kok C.W., Nguyen T.Q. "Theory and design of shift invariant filter banks," *Proceeding of IEEE TFTS'96*, June 1996.

[Hyva 01] Hyvärien A., Karhunen J., Oja E. *Independent Component Analysis*, Wiley Interscience, 2001.

[Jain 89] Jain A.K. *Fundamentals of Digital Image Processing*, Prentice Hall, 1989.

[Jain 91] Jain A.K., Farrokhnia F. "Unsupervised texture segmentation using Gabor filters," *Pattern Recognition*, Vol. 24(12), pp. 1167–1186, 1991.

[Jone 87] Jones M.C., Sibson R. "What is projection pursuit?" *Journal of the Royal Statistical Society, Ser. A*, Vol. 150, pp. 1–36, 1987.

[Jutt 91] Jutten C., Herault J. "Blind separation of sources, Part I: An adaptive algorithm based on neuromimetic architecture," *Signal Processing*, Vol. 24, pp. 1–10, 1991.

[Kapo 96] Kapogiannopoulos G., Papadakis M. "Character recognition using biorthogonal discrete wavelet transform," Proceedings of the 41st annual SPIE meeting, Vol. 2825, August 1996.

[Koho 89] Kohonen T. *Self-Organization and Associative Memory*, 3rd ed., Springer Verlag, 1989.

[Lain 93] Laine A., Fan J. "Texture classification by wavelet packet signatures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(11), pp. 1186–1191, 1993.

[Lain 96] Laine A., Fan J. "Frame representations for texture segmentation," *IEEE Transcaction on Image Processing*, Vol. 5(5), pp. 771–780, 1996.

[Lee 98] Lee T.-W. *Independent Component Analysis*, Kluwer Academic Publishers, 1998.

[Levi 85] Levine M.D. *Vision in Man and Machine*, McGraw-Hill, New York, 1985.

[Lim 90] Lim J.S. *Two-Dimensional Signal Processing*, Prentice Hall, 1990.

[Mall 89] Mallat S. "Multifrequency channel decompositions of images and wavelet models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37(12), pp. 2091–2110, 1989.

[Mall 97] Mallet Y., Coomans D., Kautsky J., De Vel O. "Classification using adaptive wavelets for feature extraction," *IEEE Transactions for Pattern Analysis and Machine Intelligence*, Vol. 19(10), pp. 1058–1067, 1997.

[Marco 95] Marco S.D., Heller P.N., Weiss J. "An M-band two dimensional translation-invariant wavelet transform and its applivations," *Proceedings of the IEEE Conference on Acoustics Speech and Signal Processing*, pp. 1077–1080, 1995.

[Meye 93] Meyer Y. *Wavelets, Algorithms and Applications*, SIAM, Philadelphia, 1993.

[Mojs 00] Mojsilovic A., Popovic M.V., Rackov D.M. "On the selection of an optimal wavelet basis for texture characterization," *IEEE Transactions Image Processing*, Vol. 9(12), 2000.

[Nach 75] Nachmais J., Weber A. "Discrimination of simple and complex gratings," *Vision Research*, Vol. 15, pp. 217–223, 1975.

[Oja 83] Oja E. *Subspace Methods for Pattern Recognition*, Letchworth, U.K.: Res. Studies Press, 1983.

[Papo 91] Papoulis A. *Probability, Random Variables, and Stochastic Processes*, 3rd ed., McGraw-Hill, 1991.

[Pich 96] Pichler O., Teuner A., Hosticka, B. "A comparison of texture feature extraction using adaptive Gabor filtering, pyramidal and tree structured wavelet transforms," *Pattern Recognition*, Vol. 29(5), pp. 733–742, 1996.

[Pita 92] Pitas I. *Digital Image Processing Algorithms*, Prentice Hall, 1992.

[Prak 97] Prakash M., Murty M.N. "Growing subspace pattern recognition methods and their neural network models," *IEEE Transactions on Neural Networks*, Vol. 8(1), pp. 161–168, 1997.

[Proa 92] Proakis J., Manolakis D. *Digital Signal Processing*, 2nd ed., Macmillan, 1992.

[Stef 93] Steffen P., Heller P.N., Gopinath R.A., Burrus C.S. "Theory of regular M-band wavelet bases," *IEEE Tansactions on Signal Processing*, Vol. 41(12), pp. 3497–3511, 1993.

[Stra 80] Strang G. *Linear Algebra and its Applications*, 2nd ed., Harcourt Brace Jovanovich, 1980.

[Szu 92] Szu H.H., Telfer B.A., Katambe S. "Neural network adaptive wavelets for signal representation and classification," *Optical Eng.*, Vol. 31, pp. 1907–1916, 1992.

[Turn 86] Turner M.R. "Texture discrimination by Gabor functions," *Biol. Cybern.*, Vol. 55, pp. 71–82, 1986.

[Unse 86] Unser M. "Local linear transforms for texture measurements," *Signal Processing*, Vol. 11(1), pp. 61–79, 1986.

[Unse 89] Unser M., Eden M. "Multiresolution feature extraction and selection for texture segmentation," *IEEE Transations on Pattern Analysis and Machine Intelligence*, Vol. 11(7), pp. 717–728, 1989.

[Unse 95] Unser M. "Texture classification and segmentation using wavelet frames," *IEEE Transactions on Image Processing*, Vol. 4(11), pp. 1549–1560, 1995.

[Vaid 93] Vaidyanathan P.P. *Multirate Systems and Filter Banks*, Prentice Hall, 1993.

[Vett 92] Vetterli M., Herley C. "Wavelets and filter banks: Theory and design," *IEEE Transactions on Signal Processing*, Vol. 40(9), pp. 2207–2232, 1992.

[Vett 95] Vetterli M., Kovacevic J. *Wavelets and Subband Coding*, Prentice Hall, 1995.

[Wata 73] Watanabe S., Pakvasa N. "Subspace method in pattern recognition," *Proceedings of the International Joint Conference on Pattern Recognition*, pp. 25–32, 1973.

[Weld 96] Weldon T., Higgins W., Dunn D. "Efficient Gabor filter design for texture segmentation," *Pattern Recognition*, Vol. 29(2), pp. 2005–2025, 1996.

[Wuns 95] Wuncsh P., Laine A. "Wavelet descriptors for multiresolution recognition of handwritten characters," *Pattern Recognition*, Vol. 28(8), pp. 1237–1249, 1995.

CHAPTER 7

FEATURE GENERATION II

7.1 INTRODUCTION

In the previous chapter we dealt with the task of feature generation via linear transformation techniques. This is just one of the possibilities available to the designer. There is a number of alternatives which, however, are very much application dependent. Although similarities among various applications do exist, there are also big differences. We will approach the problem by focusing on one major application area, that of *image analysis*. Clearly, we cannot review all the techniques that have been suggested and used. The number is really large. Instead, we will focus on basic directions, with a wide range of applications in mind, such as medical imaging, remote sensing, robot vision, and character recognition. Finally, some of these techniques are also suitable to other applications of major interest to us, such as speech recognition.

Our major goal in this chapter may be summarized as follows: *given an image, or a region within an image, generate the features that will subsequently be fed to a classifier in order to classify the image in one of the possible classes.*

A digital image is usually the result of a discretization process (sampling) of a continuous image function $I(x, y)$ and it is stored in the computer as a two-dimensional array $I(m, n)$ with $m = 0, 1, \ldots, N_x - 1$ and $n = 0, 1, \ldots, N_y - 1$—that is, it is stored as an $N_x \times N_y$ array.[1] Every (m, n) element of the array corresponds to a *pixel* (picture element or image element) of the image, whose brightness or intensity is equal to $I(m, n)$. Furthermore, when the intensity $I(m, n)$ is quantized in N_g discrete (gray) levels, N_g is known as the depth of the image. Then, the gray level sequence $I(m, n)$ can take one of the integer values $0, 1, \ldots, N_g - 1$. The depth N_g is usually a power of 2 and can take large values (e.g., 64, 256) when the image is stored in the computer. However, for the human eye it is difficult to discern detailed intensity differences, and in practice $N_g = 32$ or 16 is a sufficient choice for image representation.

[1] Only monochrome images will be considered.

The need for feature generation stems from our inability to use the raw data. Even for a small 64×64 image the number of pixels is 4096. For most classification tasks this number is too large, raising computational as well as generalization problems, as discussed in earlier chapters. Feature generation is a procedure that computes new variables that in one way or another originate from the stored values of the image array $I(m, n)$. *The goal is to generate features that exhibit high information packing properties*, from the class separability point of view. Because we cannot use the raw data $I(m, n)$ directly, the features should encode efficiently the relevant information residing in the original data.

7.2 REGIONAL FEATURES

7.2.1 Features for Texture Characterization

The texture of an image region is determined by the way the gray levels are distributed over the pixels in this region. Although there is no clear definition of "texture," we are all in a position to describe an image by the look of it as *fine or coarse, smooth or irregular, homogeneous or inhomogeneous*, and so forth. Our goal in this subsection is to generate appropriate features that, somehow, quantify these properties of an image region. These features will emerge by exploiting space relations underlying the gray level distribution.

First-Order Statistics Features

Let I be the random variable representing the gray levels in the region of interest. The first-order histogram $P(I)$ is defined as

$$P(I) = \frac{\textit{number of pixels with gray level } I}{\textit{total number of pixels in the region}} \tag{7.1}$$

That is, $P(I)$ is the fraction of pixels with gray level I. Let N_g be the number of possible gray levels. Based on (7.1), the following quantities are defined.

Moments:

$$m_i = E[I^i] = \sum_{I=0}^{N_g-1} I^i P(I), \quad i = 1, 2, \ldots \tag{7.2}$$

Obviously $m_0 = 1$ and $m_1 = E[I]$, the mean value of I.

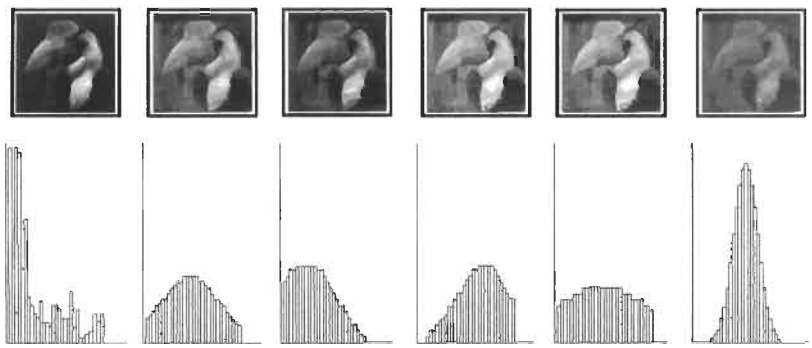


FIGURE 7.1: Examples of images and corresponding histograms.

Central moments:

$$\mu_i = E[(I - E[I])^i] = \sum_{I=0}^{N_g-1} (I - m_1)^i P(I) \tag{7.3}$$

The most frequently used central moments are μ_2, μ_3, and μ_4. $\mu_2 = \sigma^2$ is the variance, and μ_3 is known as the *skewness* (sometimes is normalized by σ^3) and μ_4 as the *kurtosis* (sometimes is normalized by σ^4) of the histogram. The variance is a measure of the histogram width, that is, a measure of how much the gray levels differ from the mean. Skewness is a measure of the degree of histogram asymmetry around the mean, and μ_4 is a measure of the histogram sharpness. Depending on the value of μ_4, the resulting histogram is called platykurtic, for large values, leptokurtic, for small values, and mesokurtic otherwise. The normal distribution is a mesokurtic one. Figure 7.1 shows six variations of the same image (with 16 gray levels) with their corresponding histograms. We can observe the difference between the platykurtic and the leptokurtic one. In the latter, only the middle gray levels are present (with no $I = 0$ or $I = 15$), in contrast to the platykurtic one, where all gray levels are present. For the two asymmetric cases, one corresponds to a majority of low gray levels and the other to a majority of high levels. The resulting values of μ_3 and μ_4 from left to right are

μ_3:	587	0	−169	169	0	0
μ_4:	16609	7365	7450	7450	9774	1007

Other quantities that result from the first-order histogram are:

Absolute moments:

$$\hat{\mu}_i = E[\mid I - E[I] \mid^i] = \sum_{I=0}^{N_g-1} \mid I - E[I] \mid^i P(I) \tag{7.4}$$

Entropy:

$$H = -E[\log_2 P(I)] = -\sum_{I=0}^{N_g-1} P(I) \log_2 P(I) \tag{7.5}$$

Entropy is a measure of histogram uniformity. The closer to the uniform distribution ($P(I) = \text{constant}$), the higher the H. For the six images of Figure 7.1 the corresponding values are

$$H: \quad 4.61 \quad 4.89 \quad 4.81 \quad 4.81 \quad 4.96 \quad 4.12$$

Second-Order Statistics Features—Co-occurrence Matrices

The features resulting from the first-order statistics provide information related to the gray level distribution of the image, but they do not give any information about the relative positions of the various gray levels within the image. Are all low-value gray levels positioned together, or are they interchanged with the high-value ones? This type of information can be extracted from the second-order histograms, where the pixels are considered in pairs. Two more parameters now enter into the scene. These are the relative distance among the pixels and their relative orientation. Let d be the relative distance measured in pixel numbers ($d = 1$ for neighboring pixels, etc.). The orientation ϕ is quantized in four directions: horizontal, diagonal, vertical, and antidiagonal (0°, 45°, 90°, 135°), as shown in Figure 7.2. For each combination of d and ϕ a two-dimensional histogram is defined

$$0°: \quad P\big(I(m,n) = I_1,\ I(m \pm d, n) = I_2\big) \tag{7.6}$$
$$= \frac{\textit{number of pairs of pixels at distance } d \textit{ with values } (I_1, I_2)}{\textit{total number of possible pairs}}$$

In a similar way

$$\begin{aligned} 45° &: P\big(I(m,n) = I_1, I(m \pm d, n \mp d) = I_2\big) \\ 90° &: P\big(I(m,n) = I_1, I(m, n \mp d) = I_2\big) \\ 135° &: P\big(I(m,n) = I_1, I(m \pm d, n \pm d) = I_2\big) \end{aligned}$$

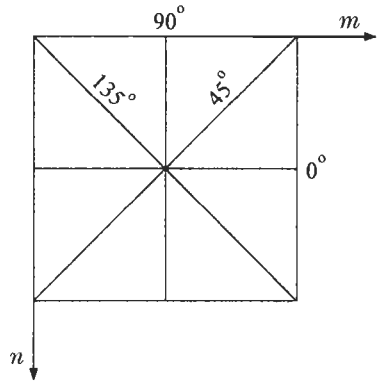


FIGURE 7.2: The four orientations used to construct co-occurrence matrices.

For each of these histograms an array is defined, known as the *co-occurrence or spatial dependence matrix*. Let, for example, an image array $I(m, n)$ be

$$I = \begin{bmatrix} 0 & 0 & 2 & 2 \\ 1 & 1 & 0 & 0 \\ 3 & 2 & 3 & 3 \\ 3 & 2 & 2 & 2 \end{bmatrix} \tag{7.7}$$

which corresponds to a 4×4 image. We have also assumed that $N_g = 4$ ($I(m,n) \in \{0, 1, 2, 3\}$). The co-occurrence matrix for a pair (d, ϕ) is defined as the $N_g \times N_g$ matrix

$$A = \frac{1}{R} \begin{bmatrix} \eta(0,0) & \eta(0,1) & \eta(0,2) & \eta(0,3) \\ \eta(1,0) & \eta(1,1) & \eta(1,2) & \eta(1,3) \\ \eta(2,0) & \eta(2,1) & \eta(2,2) & \eta(2,3) \\ \eta(3,0) & \eta(3,1) & \eta(3,2) & \eta(3,3) \end{bmatrix}$$

where $\eta(I_1, I_2)$ is the number of pixel pairs, at relative position (d, ϕ), which have gray-level values I_1, I_2, respectively. R is the total number of possible pixel pairs. Hence $\frac{1}{R}\eta(I_1, I_2) = P(I_1, I_2)$. For the image of (7.7) and relative pixel position $(1, 0°)$ we have

$$A^0(d = 1) = \frac{1}{24} \begin{bmatrix} 4 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 0 & 6 & 3 \\ 0 & 0 & 3 & 2 \end{bmatrix}$$

In words, for each of the intensity pairs, such as (0, 0), we count the number of pixel pairs at relative distance $d = 1$ and orientation $\phi = 0°$ that take these values. For our example this is 4. Two of them result from searching in the positive direction and two in the negative. According to the definition (7.6), these pixel pairs have coordinates (m, n) and $(m \pm 1, n)$ and gray levels $I_1 = 0$, $I_2 = 0$. The total number of pixel pairs for this case is 24. Indeed, for each row there are $N_x - 1$ pairs and there are N_y rows. Thus, the total number *for both positive and negative directions* is $2(N_x - 1)N_y = 2(3 \times 4) = 24$. For the diagonal direction 45° and $d = 1$ for each row we have $2(N_x - 1)$ pairs, except the first (or last) one, for which no pairs exist. Thus, the total number is $2(N_x - 1)(N_y - 1) = 2(3 \times 3) = 18$. For $d = 1$ and 90° we have $2(N_y - 1)N_x$ pairs, and finally for $d = 1$ and 135° $2(N_x - 1)(N_y - 1)$. For our example image and $(d = 1, \phi = 45°)$ we obtain

$$A^{45}(d = 1) = \frac{1}{18}\begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 3 \\ 1 & 1 & 3 & 0 \end{bmatrix}$$

From the definition of the co-occurrence matrix, it is apparent that it is a symmetric one, something that can be used to reduce subsequent computations.

Having defined probabilities of occurrence of gray levels with respect to relative spatial pixel position, we will go ahead to define the corresponding features. Some of them have a direct physical interpretation with respect to texture, for example, to quantify coarseness, smoothness, and so on. On the other hand, others do not possess such a property, but they still encode texture-related information with high discriminatory power.

- *Angular second moment*

$$ASM = \sum_{i=0}^{N_g-1}\sum_{j=0}^{N_g-1}(P(i, j))^2 \tag{7.8}$$

  This feature is a measure of the smoothness of the image. Indeed, if all pixels are of the same gray level $I = k$, then $P(k, k) = 1$ and $P(i, j) = 0$, $i \neq k$ or $j \neq k$, and $ASM = 1$. At the other extreme, if we could have all possible pairs of gray levels with equal probability $\frac{1}{R}$, then $ASM = \frac{R}{R^2} = \frac{1}{R}$. The less smooth the region is, the more uniformly distributed $P(i, j)$ and the lower the ASM (Problem 7.5).

- *Contrast*

$$CON = \sum_{n=0}^{N_g-1} n^2 \left\{ \sum_{i=0}^{N_g-1} \sum_{\substack{j=0 \\ |i-j|=n}}^{N_g-1} P(i, j) \right\} \tag{7.9}$$

This is a measure of the image contrast—that is, a measure of local gray level variations. Indeed, $\sum_i \sum_j P(i, j)$ is the percentage of pixel pairs whose intensity differs by n. The n^2 dependence weighs the big differences more; thus, *CON* takes high values for images of high contrast.

- *Inverse difference moment*

$$IDF = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} \frac{P(i, j)}{1 + (i - j)^2} \tag{7.10}$$

This feature takes high values for low-contrast images due to the inverse $(i - j)^2$ dependence.

- *Entropy*

$$H_{xy} = - \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} P(i, j) \log_2 P(i, j) \tag{7.11}$$

Entropy is a measure of randomness and takes low values for smooth images.

These features are only a few from a larger set that can be derived. In the classical [Hara 73] paper fourteen of those are summarized. They are repeated in Table 7.1. $P_x(P_y)$ (and related quantities) refer to the statistics with respect to the $x(y)$ axis. All features in the table are functions of the distance d and the orientation ϕ. Thus, if an image is rotated, the values of the features will be different. In practice, for each d the resulting values for the four directions are averaged out. In this way, we make these textural features *rotation tolerant*.

Besides the previous list of features, a number of other statistics-related features have been proposed. For example, in [Tamu 78] textural features are generated with an emphasis on the human visual perception. A set of features is suggested corresponding to texture coarseness, contrast, regularity, etc. In [Davi 79] features based on a generalized definition of co-occurrence matrices are suggested, that is more appropriate for textures with long scale variations (macrotextures).

Example 7.1. Figure 7.3 shows two texture images, one coarse, known as grass [Brod 66], and the other smooth one. Table 7.2 summarizes the values of some of the features for both of them.

Features Using Gray Level Run Lengths

A gray level *run* is a set of *consecutive* pixels having the *same gray level value*. The *length of the run* is the number of pixels in the run [Gall 75, Tang 98]. Run length features encode textural information related to the number of times each gray level, for example, "1," appears in the image by itself, the number of times it

Table 7.1: Features for texture characterization

Angular Second Moment:

$$f_1 = \sum_i \sum_j (P(i,j))^2$$

Contrast:

$$f_2 = \sum_{n=0}^{N_g-1} n^2 \left\{ \sum_{|i-j|=n} \sum_{j} P(i,j) \right\}$$

Correlation:

$$f_3 = \frac{\{\sum_i \sum_j (ij) P(i,j)\} - \mu_x \mu_y}{\sigma_x \sigma_y}$$

Variance:

$$f_4 = \sum_i \sum_j (i-\mu)^2 P(i,j)$$

Inverse Difference Moment:

$$f_5 = \sum_i \sum_j \frac{P(i,j)}{1+(i-j)^2}$$

Sum (Difference) Average:

$$f_6(\hat{f}_6) = \sum_{i=0}^{2N_g-2\ (N_g-1)} i P_{x+(-)y}(i)$$

Sum Variance:

$$f_7 = \sum_{i=0}^{2N_g-2} (i - f_6)^2 P_{x+y}(i)$$

Sum Entropy:

$$f_8 = -\sum_{i=0}^{2N_g-2} P_{x+y}(i) \log P_{x+y}(i)$$

Entropy:

$$f_9 = -\sum_i \sum_j P(i,j) \log P(i,j) \equiv H_{xy}$$

Difference Variance:

$$f_{10} = \sum_{i=0}^{N_g-1} (i - \hat{f}_6)^2 P_{x-y}(i)$$

Difference Entropy:

$$f_{11} = -\sum_{i=0}^{N_g-1} P_{x-y}(i) \log P_{x-y}(i)$$

Information Measure I:

$$f_{12} = \frac{H_{xy} - H_{xy}^1}{max\{H_x, H_y\}}$$

Information Measure II:

$$f_{13} = \sqrt{1 - \exp(-2(H_{xy}^2 - H_{xy}))}$$

Maximal Correlation Coefficient:

$$f_{14} = (\text{2nd largest eigenvalue of } Q)^{\frac{1}{2}}$$

Definitions: $Q(i,j) = \sum_k \frac{P(i,k)P(j,k)}{P_x(i)P_y(k)}$

$$H_{xy}^1 = -\sum_i \sum_j P(i,j) \log(P_x(i)P_y(j))$$

$$H_{xy}^2 = -\sum_j \sum_i P_x(i)P_y(j) \log(P_x(i)P_y(j))$$

$$P_x(i) = \sum_j P(i,j) \qquad P_y(j) = \sum_i P(i,j)$$

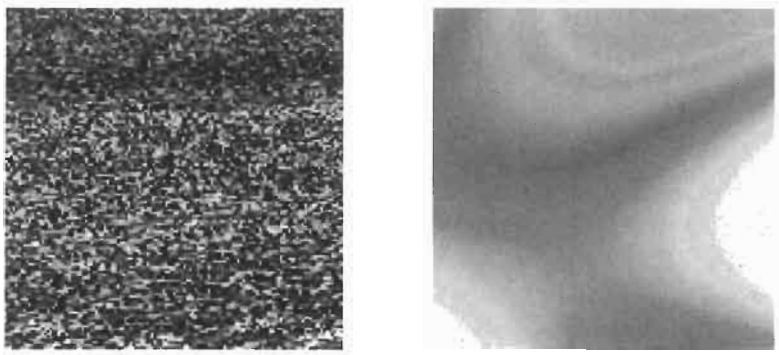
$$P_{x\pm y}(k) = \sum_i \sum_{j,|i\pm j|=k} P(i,j)$$

$\mu, \mu_x, \mu_y, \sigma_x, \sigma_y$; H_x, H_y means, st. deviations and entropies.

appears in pairs, and so on. Take for example the image

$$I = \begin{bmatrix} 0 & 0 & 2 & 2 \\ 1 & 1 & 0 & 0 \\ 3 & 2 & 3 & 3 \\ 3 & 2 & 2 & 2 \end{bmatrix}$$

with four possible levels of gray (N_g=4.) For each of the four directions (0°, 45°, 90°, 135°) we define the corresponding run length matrix Q_{RL}. Its (i, j)



(a) (b)

FIGURE 7.3: Examples of (a) coarse and (b) smooth images.

Table 7.2: Second-order histogram features for the two images of Figure 7.3

	Coarse	Smooth
ASM	0.0066	0.0272
CON	989.5	0.613
IDF	0.117	0.783
H_{xy}	8.352	5.884

element gives the number of times a gray level $i - 1, i = 1, \ldots, N_g$, appears in the image with run length $j, j = 1, 2, \ldots, N_r$. This is an $N_g \times N_r$ array, where N_r is the largest possible run length in the image. For 0° we obtain

$$Q_{RL}(0°) = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \tag{7.12}$$

The first element of the first row of the matrix is the number of times gray level "0" appears by itself (0 for our example), the second element is the number of times it appears in pairs (2 in the example), and so on. The second row provides the same information for gray level "1" and so on. For the 45° direction

we have

$$Q_{RL}(45°) = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 \end{bmatrix} \tag{7.13}$$

Based on the preceding definition of the run length matrix, the following features are defined.

- *Short run emphasis*

$$SRE = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} (Q_{RL}(i, j)/j^2)}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} Q_{RL}(i, j)} \tag{7.14}$$

  The denominator is the total number of run lengths in the matrix, 9 for (7.12) and 16 for (7.13). This feature emphasizes small run lengths, due to the division by j^2.
- *Long run emphasis*

$$LRE = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} (Q_{RL}(i, j) j^2)}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} Q_{RL}(i, j)} \tag{7.15}$$

  This gives emphasis to long run lengths. Thus, we expect *SRE* to be large for coarser and *LRE* to be large for smoother images.
- *Gray level nonuniformity*

$$GLNU = \frac{\sum_{i=1}^{N_g} \left[\sum_{j=1}^{N_r} Q_{RL}(i, j) \right]^2}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} Q_{RL}(i, j)} \tag{7.16}$$

  The term in the brackets is the total number of run lengths for each gray level. Large run length values contribute a great deal because of the square. When runs are uniformly distributed among the gray levels, *GNLU* takes small values.
- *Run length nonuniformity*

$$RLN = \frac{\sum_{j=1}^{N_r} \left[\sum_{i=1}^{N_g} Q_{RL}(i, j) \right]^2}{\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} Q_{RL}(i, j)} \tag{7.17}$$

  In a similar way, *RLN* is a measure of run length nonuniformity.

Table 7.3: Run length features for the images of Figure 7.3

	Coarse	Smooth
SRE	0.932	0.563
LRE	1.349	16.929
GLNU	255.6	71.6
RLN	3108	507
RP	0.906	0.4

- *Run percentage*

$$RP = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} Q_{RL}(i, j)}{L} \tag{7.18}$$

where L is the total possible number of runs in the image, if *all* runs had length equal to one, that is, the total number of pixels. RP takes low values for smooth images.

Example 7.2. For the two images of Figure 7.3 the values of Table 7.3 have resulted.

7.2.2 Local Linear Transforms for Texture Feature Extraction

Second-order statistics features were introduced in order to exploit the spatial dependencies that characterize the texture of an image region. We will now focus on an alternative possibility, which has been used extensively in practice. Let us consider a neighborhood of size $N \times N$ centered at pixel location (m, n). Let $\boldsymbol{x}_{mn}$ be the vector with elements the N^2 points within the area, arranged in a row-by-row mode. A *local linear transform or local feature extractor* is defined as

$$\boldsymbol{y}_{mn} = A^T \boldsymbol{x}_{mn} \equiv \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_{N^2}^T \end{bmatrix} \boldsymbol{x}_{mn} \tag{7.19}$$

The respective correlation matrices are related via the $N^2 \times N^2$ nonsingular transformation matrix A as

$$R_y \equiv E[\boldsymbol{y}_{mn} \boldsymbol{y}_{mn}^T] = A^T R_x A \tag{7.20}$$

From these definitions it is readily seen that each element of $\boldsymbol{y}$ contains information about *all* the elements of $\boldsymbol{x}$. This becomes clearer if we look more closely at the way the two correlation matrices are related. Indeed, the diagonal elements of R_y are the respective variances of the elements of $\boldsymbol{y}$. These are first-order statistics quantities, yet their values contain information about the spatial dependencies (second-order statistics) of the original image. *Here lies the essence of the technique. Texture-related spatial dependencies of an image can be accommodated in the first-order statistics of the transformed image.* Using appropriately defined local transform matrices, various aspects of texture properties can be extracted. Of course, the philosophy does not change if instead of transforming vectors we use 2-D (separable) transforms of the corresponding subimage region.

One way to look at (7.19) is to interpret it as a series of N^2 filtering operations (convolutions, Appendix D), with a common input vector, $\boldsymbol{x}_{mn}$, that is, the $N \times N$ subimage centered at (m, n). The elements of $\boldsymbol{y}_{mn}$ are the respective filter output samples. This is illustrated in Figure 7.4, where the $N \times N$ subimage ($N = 3$) is filtered through 9 equivalent 2-D filters, each characterized by a different coefficient matrix, known as *mask*. In [Laws 80] it is suggested that the corresponding masks be constructed from three basic vectors, namely $[1, 2, 1]^T$, $[-1, 0, 1]^T$, $[-1, 2, -1]^T$, for $N = 3$. The first corresponds to a local averaging operator, the second to an edge detection operator, and the third to a spot detector. These form a complete (nonorthogonal) set of vectors in the $\mathcal{R}^3$ space.

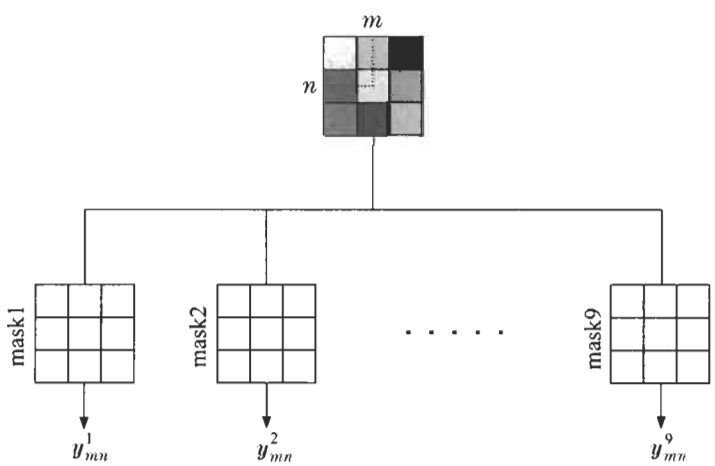


FIGURE 7.4: Filtering the image with each of the masks results in new transformed images/channels.

The respective nine masks are formed by their cross-products, that is,

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 2 & -1 \\ -2 & 4 & -2 \\ -1 & 2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ -2 & 0 & 2 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Each element of the vector $\mathbf{y}_{mn}$ is the result of filtering the local image neighborhood centered at (m, n) with each of the masks. By moving the masks around at the various (m, n) positions, nine different images, *channels*, will be obtained, *each encoding different aspects of the texture of the original image.* First-order-statistics quantities, such as variance and kurtosis, computed from each of these images, can then be used as features for texture classification. Masks larger than 3×3 have also been used. In some cases, an attempt to optimize the masks has been made, so that the resulting variances of the channels for the different classes are as different as possible [Unse 86]. This turns out to be an eigenvalue–eigenvector task, similar to the ones we have already met in Chapter 5. A comparative study of a number of optimal or suboptimal local linear transforms, including orthogonal ones, such as DCT, DST and Karhunen-Loève, is given in [Unse 86], [Unse 89], [Rand 99]. Finally, it must be pointed out that all these techniques are closely related to the Gabor filtering approach of the previous chapter.

7.2.3 Moments

Geometric Moments

Let $I(x, y)$ be a continuous image function. Its *geometric moment* of order $p + q$ is defined as

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q I(x, y)\, dx\, dy \tag{7.21}$$

Geometric moments provide rich information about the image and are popular features for pattern recognition. Their information content stems from the fact that

moments provide an equivalent representation of an image, in the sense that an image can be reconstructed from its moments (of all orders) [Papo 91, page 115]. Thus, each moment coefficient conveys a certain amount of the information residing in an image.

It is by now commonplace to state that a desirable property in pattern recognition is invariance in geometric transformations. Moments, as defined in (7.21), depend on the coordinates of the object of interest within an image; thus, they lack the invariance property. This problem can be circumvented by defining appropriate combinations of normalized versions of the moments. Specifically, our goal will be to define moments that are invariant to:

Translations:

$$x' = x + a, \quad y' = y + b$$

Scaling:

$$x' = \alpha x, \quad y' = \alpha y$$

Rotations:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

To this end, let us define

Central moments:

$$\mu_{pq} = \int\int I(x, y)(x - \bar{x})^p (y - \bar{y})^q \, dx \, dy \tag{7.22}$$

where

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

Central moments are invariant to translations.

Normalized central moments:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \quad \gamma = \frac{p + q + 2}{2} \tag{7.23}$$

These are easily shown to be invariant to both translation and scaling (Problem 7.6)

The seven moments of Hu: Hu [Hu 62] has defined a set of seven moments that are invariant under the actions of translation, scaling, and rotation. These are

$$
\begin{aligned}
p+q &= 2 \\
\phi_1 &= \eta_{20} + \eta_{02} \\
\phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
p+q &= 3 \\
\phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (\eta_{03} - 3\eta_{21})^2 \\
\phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{03} + \eta_{21})^2 \\
\phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (\eta_{03} - 3\eta_{21})(\eta_{03} + \eta_{21})[(\eta_{03} + \eta_{21})^2 - 3(\eta_{12} + \eta_{30})^2] \\
\phi_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21}) \\
\phi_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[(\eta_{03} + \eta_{21})^2 - 3(\eta_{30} + \eta_{12})^2]
\end{aligned}
$$

The first six of these are also invariant under the action of reflection, while ϕ_7 changes sign. The values of these quantities can be quite different. In practice, in order to avoid precision problems, the logarithms of their absolute values are usually used as features. A number of other moment-based features that are invariant to more general transformations have also been proposed [Reis 91, Flus 93, Flus 94]. The case of moment invariants in the general l-dimensional space is treated in [Mami 98].

For a digital image $I(i, j)$, with $i = 0, 1, \ldots, N_x - 1$, $j = 0, 1, \ldots, N_y - 1$, the preceding moments can be *approximated* by replacing integrals by summations,

$$m_{pq} = \sum_i \sum_j I(i, j) i^p j^q \tag{7.24}$$

In order to keep the dynamic range of the moment values consistent for different-sized images, a normalization of the $x - y$ axis can be performed, prior to the computation of the moments. The moments are then approximated by

$$m_{pq} = \sum_i I(x_i, y_i) x_i^p y_i^q \tag{7.25}$$

where the sum is over all image pixels. Then x_i, y_i are the coordinates of the center point of the ith pixel and are no longer integers but real numbers in the interval

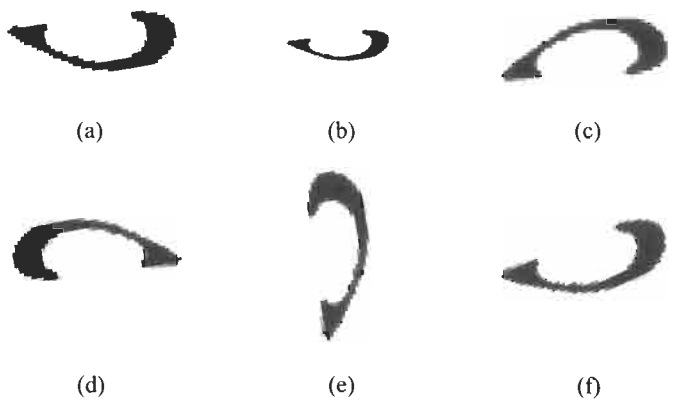


FIGURE 7.5: The Byzantine symbol "petasti" in various scaled and rotated versions, from (a) to (f).

$x_i \in [-1, +1]$, $y_i \in [-1, +1]$. *For digital images, the invariance properties of the moments we have defined are only approximately true.* An analysis in [Liao 96] reveals that the approximation error increases with the coarseness of the sampling grid as well as with the order of the moments.

Example 7.3. Figure 7.5 shows the Byzantine music symbol known as "petasti," resulting from a scanner, in scaled and various rotated versions. From left to right in the clockwise sense we have the original version, the scaled, the mirrored, and the rotated by 15°, 90°, and 180° versions, respectively.

Table 7.4 shows the resulting Hu moments for each of the versions. The (approximate) invariance of the moments is apparent. Note the minus sign in ϕ_7 for the reflected (mirror) version.

Zernike Moments

The geometric moments defined in (7.21) can also be viewed as projections (Chapter 6) of $I(x, y)$ on the basis functions formed by the monomials $x^p y^q$. These monomials are not orthogonal; thus, the resulting geometric moment features are not optimal from an information redundancy point of view. In this subsection we will derive moments based on alternative complex polynomial functions, known as *Zernike polynomials*. These form a *complete orthogonal set over the interior of the unit circle* $x^2 + y^2 \leq 1$ (Problem 7.7) and are defined as

$$V_{pq}(x, y) = V_{pq}(\rho, \theta) = R_{pq}(\rho) \exp(jq\theta)$$

Table 7.4: The invariant moments of Hu for the versions of the "petasti" symbol

Moments	0°	Scaled	180°	15°	Mirror	90°
ϕ_1	93.13	91.76	93.13	94.28	93.13	93.13
ϕ_2	58.13	56.60	58.13	58.59	58.13	58.13
ϕ_3	26.70	25.06	26.70	27.00	26.70	26.70
ϕ_4	15.92	14.78	15.92	15.83	15.92	15.92
ϕ_5	3.24	2.80	3.24	3.22	3.24	3.24
ϕ_6	10.70	9.71	10.70	10.57	10.70	10.70
ϕ_7	0.53	0.46	0.53	0.56	−0.53	0.53

where:
p is a nonnegative integer
q is an integer subject to the constraint $p - |q|$ even, $|q| \leq p$
$\rho = \sqrt{x^2 + y^2}$
$\theta = \tan^{-1} \frac{y}{x}$

$$R_{pq}(\rho) = \sum_{s=0}^{(p-|q|)/2} \frac{(-1)^s[(p-s)!]\rho^{p-2s}}{s!\left(\frac{p+|q|}{2} - s\right)!\left(\frac{p-|q|}{2} - s\right)!}$$

The Zernike moments of a function $I(x, y)$ are given by

$$A_{pq} = \frac{p+1}{\pi} \int\int_{x^2+y^2 \leq 1} I(x, y) V^*(\rho, \theta)\, dx\, dy$$

where the $*$ denotes complex conjugation. For a digital image, the respective Zernike moments are computed as

$$A_{pq} = \frac{p+1}{\pi} \sum_i I(x_i, y_i) V^*(\rho_i, \theta_i), x_i^2 + y_i^2 \leq 1$$

where i runs over all the image pixels. The computation of the corresponding moments of an image considers the center of the image as the origin and pixels are mapped into the unit circle, that is, $x_i^2 + y_i^2 \leq 1$. The pixels falling outside the unit circle are not taken into consideration. The magnitude of the Zernike moments is invariant to rotations [Teag 80](Problem 7.8). Translation and scaling invariance is treated in [Khot 90a]. Computational aspects of the Zernike moments are examined in [Muku 95]. Comparative studies of the performance of the Zernike moments against the moments of Hu, in the context of character recognition, have demonstrated that the former behave better, especially in noisy environments [Khot 90b]. In [Wang 98], Zernike moments are used to cope both

with geometry as well as illumination invariance, in the context of multispectral texture classification. Variants of the Zernike moments, called pseudo-Zernike moments, have also been proposed and used. Comparative studies can be found in [Teh 88, Heyw 95].

7.2.4 Parametric Models

So far, in various parts of the book, we have treated the gray levels as random variables and looked at aspects of their first- and second-order statistics. In this subsection, their randomness will be approached from a different perspective. We will assume that $I(m, n)$ is a real *nondiscrete* random variable, and we will try *to model* its underlying generation mechanism by adopting an appropriate *parametric model*. The parameters of the resulting models encode useful information and lend themselves as powerful feature candidates for a number of pattern recognition tasks.

We will move in two directions. One is to treat an image as a successive sequence of rows or columns. That is, our random variables will be considered as successive realization samples from a one-dimensional random process $I(n)$. The alternative looks at the image as a two-dimensional random process $I(m, n)$, also known as *random field*.

One-Dimensional Parametric Models

Let $I(n)$ denote the random sequence. We will assume that it is stationary in the wide sense. This means that its autocorrelation sequence $r(k)$ exists and is of the form

$$r(k) = E[I(n)I(n-k)]$$

and the Fourier transform of $r(k)$ also exists and is a *positive* function (power spectral density)

$$I(\omega) = \sum_{k=-\infty}^{+\infty} r(k) \exp(-j\omega k)$$

Under certain assumptions, which are met in practice most of the time [Papo 91, Theo 93], it can be shown that such a random sequence can be generated at the output of a linear, causal, stable, time-invariant system with impulse response $h(n)$, whose input is excited by a white noise sequence, as shown in Figure 7.6. In simple terms, this means that we can write

$$I(n) = \sum_{k=0}^{\infty} h(k)\eta(n-k)$$

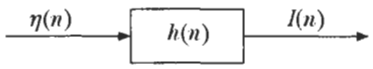


FIGURE 7.6: Generation model of a stationary random process at the output of a stable, linear, time-invariant system excited by a white noise sequence.

where $h(n)$ satisfies the stability condition $\sum_n |h(n)| < \infty$. The sequence $\eta(n)$ is a white noise sequence, that is, $E[\eta(n)] = 0$ and $E[\eta(n)\eta(n-l)] = \sigma^2\delta(l)(\delta(l) = 1$ for $l = 0$ and zero otherwise). Such processes of a special type are the so-called *autoregressive processes (AR)*, which are generated by systems of the form

$$I(n) = \sum_{k=1}^{p} a(k)I(n-k) + \eta(n) \tag{7.26}$$

In words, the random sequence $I(n)$ is given as a linear combination of previous samples $I(n-k)$ and the current input sample $\eta(n)$ (Figure 7.7). Here p is the order of the AR model, and we write AR(p). The coefficients $a(k)$, $k = 1, 2, \ldots, p$ are the AR model parameters. AR models are a special case of a more general class of models, known as *autoregressive–moving average (ARMA(p,m))*, for which

$$I(n) = \sum_{k=1}^{p} a(k)I(n-k) + \sum_{l=0}^{m} b(l)\eta(n-l) \tag{7.27}$$

That is, the model is regressive with respect to both input and output sequences. The major advantage of the AR models, compared with their ARMA relatives, is that the former lead to *linear* systems of equations for the estimation of the model parameters.

Estimation of the AR Parameters

Another way to look at (7.26) is to interpret the coefficients $a(k)$, $k = 1, \ldots, p$, as the predictor parameters of the sequence $I(n)$. That is, the parameters weigh

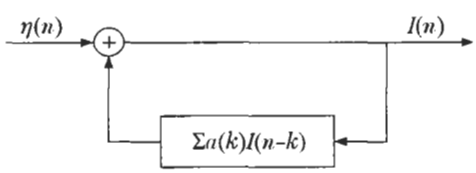


FIGURE 7.7: Generation model of an AR stationary random process.

previous samples, $I(n-1), \ldots, I(n-p)$, in order to predict the value of the current sample $I(n)$, and $\eta(n)$ is the prediction error,

$$\hat{I}_n = \sum_{k=1}^{p} a(k) I(n-k) \equiv \boldsymbol{a}_p^T \boldsymbol{I}_p(n-1) \tag{7.28}$$

where $\boldsymbol{I}_p^T(n-1) \equiv [I(n-1), \ldots, I(n-p)]$. The unknown parameter vector $\boldsymbol{a}_p^T = [a(1), a(2), \ldots, a(p)]$ is optimally estimated, for example, by minimizing the mean square prediction error,

$$E[\eta^2(n)] = E[(I(n) - \hat{I}(n))^2] = E[(I(n) - \boldsymbol{a}_p^T \boldsymbol{I}_p(n-1))^2] \tag{7.29}$$

The problem is exactly the same as that of the mean square linear classifier estimation of Chapter 3, and the unknown parameters result from the solution of

$$E[\boldsymbol{I}_p(n-1)\boldsymbol{I}_p^T(n-1)]\boldsymbol{a}_p = E[I(n)\boldsymbol{I}_p(n-1)] \tag{7.30}$$

or

$$\begin{bmatrix} r(0) & r(-1) & \ldots & r(-p+1) \\ r(1) & r(0) & \ldots & r(-p+2) \\ \vdots & \vdots & \vdots & \vdots \\ r(p-2) & r(p-3) & \ldots & r(-1) \\ r(p-1) & r(p-2) & \ldots & r(0) \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \\ \vdots \\ a(p-1) \\ a(p) \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ \vdots \\ r(p-1) \\ r(p) \end{bmatrix}$$

or

$$R\boldsymbol{a}_p = \boldsymbol{r}_p \tag{7.31}$$

with $\boldsymbol{r}_p \equiv [r(1), \ldots, r(p)]^T$. The relation of the optimal parameters $a(k)$ with the mean square error (variance of generating noise) is obtained from (7.29) and (7.31) and is given by

$$\sigma_\eta^2 = E[\eta^2(n)] = r(0) - \sum_{k=1}^{p} a(k) r(k) \tag{7.32}$$

The autocorrelation matrix has a computationally rich structure. It is symmetric ($r(k) = r(-k)$) and Toeplitz; that is, all the elements across any of its diagonals are the same. Exploitation of these properties leads to the development of a computationally efficient scheme for the solution of (7.31). This is *Levinson's algorithm*, which solves the linear system of equations in $O(p^2)$ multiplications and additions, as opposed to $O(p^3)$ required by more classical algorithmic schemes [Theo 93, Hayk 96]. In Chapter 3, we saw that when the autocorrelation sequence

is not known, it is often preferable to adopt the least sum of squares instead of the mean square criterion. Then the AR parameters are still provided by a linear system of equations, but the associated matrix is no longer Toeplitz. However, it is still computationally rich, and Levinson-type $O(p^2)$ algorithms for the efficient solution of such systems have also been derived [Theo 93].

Besides images, AR (ARMA) models have been used extensively to model other type of random sequences, such as those resulting from digitizing speech signals and electroencephalographic signals. *For all these cases the resulting AR parameters can be used as features to classify one type of signal from another.*

Example 7.4. Let the AR random sequence of order $p = 2$ be

$$I(n) = \sum_{k=1}^{2} a(k) I(n-k) + \eta(k)$$

with $r(0) = 1$, $r(1) = 0.5$, $r(2) = 0.85$. Computing the mean square estimates of $a(k)$, $k = 1, 2$, we obtain

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.85 \end{bmatrix}$$

and its solution gives $a(1) = 0.1$, $a(2) = 0.8$.

Two-Dimensional AR Models

A two-dimensional AR random sequence $I(m, n)$ is defined as

$$\hat{I}(m, n) = \sum_{k} \sum_{l} a(k, l) I(m-k, n-l), \quad (k, l) \in W \tag{7.33}$$

$$I(m, n) = \hat{I}(m, n) + \eta(m, n) \tag{7.34}$$

Figure 7.8 shows the region W of the pixels that contribute to the prediction of $\hat{I}(m, n)$, for a number of possible choices. The case in Figure 7.8a corresponds to what is known as a *strongly causal predictor model.* This is because all pixels in the contributing area have coordinates smaller than the coordinates m, n of the predicted pixel, which is represented by the unshaded node in the figure. The corresponding window is $W_1 = \{0 \leq k \leq p, 0 \leq l \leq q, (k, l) \neq (0, 0)\}$. However, the notions of past and present have no real meaning for an image, and

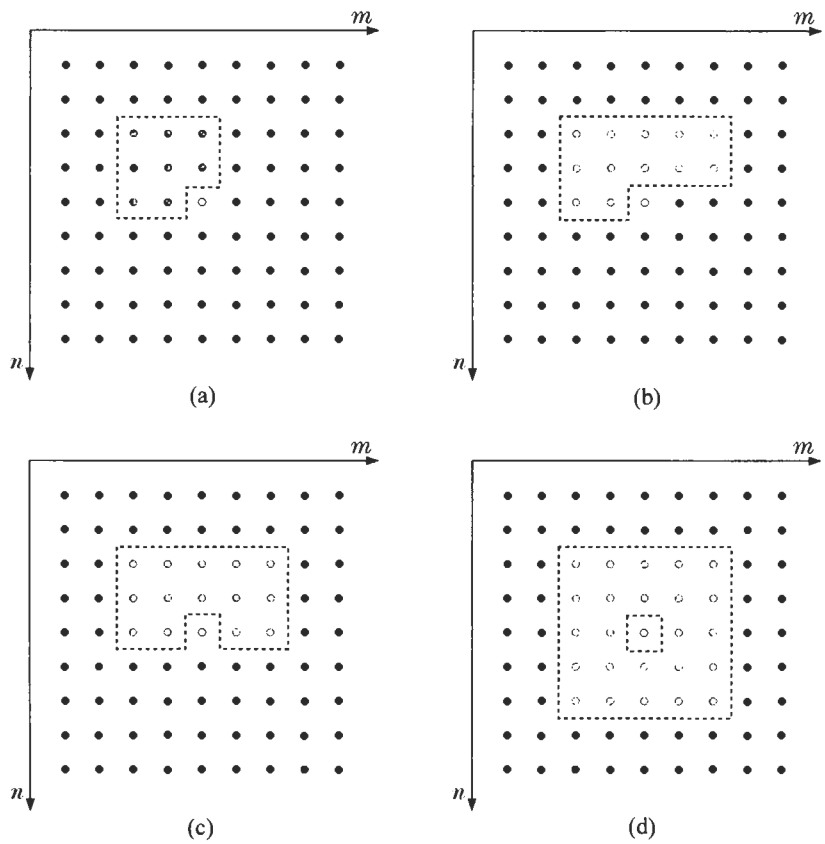


FIGURE 7.8: Different types of 2-D prediction models. The predicted pixel is represented by the unshaded node. The pixels that take part in the prediction are lightly shaded and the corresponding window W is the area enclosed by the dotted line. (a) Strictly causal, (b) causal, (c) semicausal, and (d) noncausal.

alternative windows can also be used. A *noncausal predictor* is defined as

$$I(m,n) = \sum_{k=-p}^{p} \sum_{l=-q}^{q} a(k,l) I(m-k, n-l) + \eta(m,n)$$

In Figure 7.8d the corresponding window is shown for the case of $p = q = 2$. Figure 7.8c shows a third possibility, which is known as a *semicausal predictor*, and Figure 7.8b shows the case of a *causal* predictor. Next we summarize the last

three cases, which are the most common in practice:

$$\begin{aligned} \textit{Causal}: W_2 &= \{(-p \le k \le p, 1 \le l \le q) \cup (1 \le k \le p, l = 0)\} \\ \textit{Semicausal}: W_3 &= \{-p \le k \le p, 0 \le l \le q\ ,\ (k,l) \ne (0,0)\} \\ \textit{Noncausal}: W_4 &= \{-p \le k \le p, -q \le l \le q\ ,\ (k,l) \ne (0,0)\} \end{aligned}$$

AR Parameter Estimation

We have

$$\hat{I}(m,n) = \sum_k \sum_l a(k,l) I(m-k, n-l)$$

Recalling the orthogonality condition from Chapter 3, in its 2-D generalization, we obtain that the minimum mean square error solution satisfies

$$E\left[I(m-i,n-j)\left(I(m,n) - \sum_k \sum_l a(k,l) I(m-k,n-l) \right) \right] = 0, \quad (i,j) \in W \tag{7.35}$$

or

$$r(i,j) = \sum_k \sum_l a(k,l) r(i-k, j-l), \quad (i,j) \in W \tag{7.36}$$

where $r(i,j) \equiv E[I(m,n)I(m-i,n-j)]$ is the 2-D autocorrelation sequence of the random field $I(m,n)$. The set of equations in (7.36) constitute a linear system of equations leading to the estimates of $a(k,l)$. The associated matrix also has a computationally rich structure, which can be exploited to develop efficient schemes to compute the solution. Let us take for example the noncausal window for $p = q$. This is a symmetric window, in the sense that for each index pair (i,j), the $(-i,-j)$ is also present. Combining (7.36) with the equation of the variance of the minimum error, which is given by (Problem 7.11)

$$\sigma_\eta^2 = r(0,0) - \sum_k \sum_l a(k,l) r(k,l) \tag{7.37}$$

the following system results:

$$\boldsymbol{R}\boldsymbol{a} = -\begin{bmatrix} \mathbf{0} \\ \sigma_n^2 \\ \mathbf{0} \end{bmatrix} \tag{7.38}$$

where $\mathbf{0}$ is the zero vector of appropriate dimension and

$$\boldsymbol{a}^T = [a(p,p), \ldots, a(p,-p), \ldots, a(0,0), \ldots, a(-p,p), \ldots, a(-p,-p)]$$

where $a(0,0) \equiv 1$ and R is the corresponding autocorrelation matrix. The dimension of $\boldsymbol{a}$ is $(2p+1)^2$. The correlation of a homogeneous (i.e., $E[I(m,n)I(m-k, n-l)] = r(k,l)$) and isotropic (no direction dependence) image depends only on the relative distance between pixels,

$$r(k,l) = r\left(\sqrt{k^2+l^2}\right)$$

and the resulting autocorrelation matrix is easily shown to be *symmetric and block Toeplitz with each block being itself a Toeplitz matrix*,

$$R = \begin{bmatrix} R_0 & R_1 & \dots & R_{2p} \\ R_1 & R_0 & \dots & R_{2p-1} \\ \vdots & \vdots & \vdots & \vdots \\ R_{2p} & R_{2p-1} & \dots & R_0 \end{bmatrix} \tag{7.39}$$

where

$$R_i = \begin{bmatrix} r(i,0) & \dots & r(i,2p) \\ \vdots & \vdots & \vdots \\ r(i,2p) & \dots & r(i,0) \end{bmatrix} \tag{7.40}$$

For homogeneous images and symmetric windows it is easy to show that the AR parameters are symmetric $a(k,l) = a(-k,-l)$ and the system can be solved efficiently by a Levinson-type algorithm [Kalo 89]. If the image is homogeneous but anisotropic, the resulting system's associated matrix is block Toeplitz, but the elements are no longer Toeplitz. Furthermore, more general windows than the ones introduced in this section have also been suggested and used. Efficient Levinson-type algorithms for such cases have also been developed (e.g., [Glen 94]). Finally, besides the squared error criteria, maximum likelihood techniques can be employed for the estimation of the unknown parameters, which can lead to more accurate estimates. Of course, in such cases assumptions about the underlying statistics have to be adopted (e.g., [Kash 82]).

Remarks

- The AR modeling of images has been used in the classification context in a number of cases [Chel 85, Cros 83, Kash 82, Sark 97]. In [Kash 86, Mao 92] extensions have been proposed for rotation-invariant models.
- The AR random field models are related to a class of models known as *Markov random fields*. The essence of these fields is that for each pixel (m,n) the image is divided into three areas: Ω^+ ("future"), Ω ("present"), and Ω^- ("past"). It is then assumed that the random variable $I(m,n)$, $(m,n) \in \Omega^+$,

is independent of its values in Ω^- and depends only on the values in Ω; thus, the conditional density function satisfies

$$p\left(I(m,n),(m,n)\in\Omega^+|I(m,n),(m,n)\in\Omega^-\cup\Omega\right)$$
$$=p\left(I(m,n),(m,n)\in\Omega^+|I(m,n),(m,n)\in\Omega\right)$$

In words, the "future" depends only on the "present" and not on the "past"; that is, the value of the random variable at a pixel depends on the values that the random variable takes in a specific (neighboring) area only, and it does not depend on the values in the remaining regions of the image.

- It can be shown that every Gaussian AR model is a Markov random field. [Wood 72, Chel 85].

Example 7.5. For an image whose autocorrelation sequence obeys

$$r(k,l)=0.8^{\sqrt{k^2+l^2}}$$

estimate the AR parameters for a noncausal $p=q=1$ window.

From the definition we have

$$\begin{aligned}\hat{I}(m,n)=&\,a(1,1)I(m-1,n-1)+a(1,0)I(m-1,n)\\&+a(1,-1)I(m-1,n+1)+a(0,1)I(m,n-1)\\&+a(0,-1)I(m,n+1)+a(-1,1)I(m+1,n-1)\\&+a(-1,0)I(m+1,n)+a(-1,-1)I(m+1,n+1)\end{aligned}$$

The resulting matrix R is a block $(2p+1)\times(2p+1)=3\times3$ matrix with elements the 3×3 matrices

$$R=\begin{bmatrix}R_0&R_1&R_2\\R_1&R_0&R_1\\R_2&R_1&R_0\end{bmatrix}$$

where

$$R_0=\begin{bmatrix}r(0,0)&r(0,1)&r(0,2)\\r(0,1)&r(0,0)&r(0,1)\\r(0,2)&r(0,1)&r(0,0)\end{bmatrix}$$

$$R_1=\begin{bmatrix}r(1,0)&r(1,1)&r(1,2)\\r(1,1)&r(1,0)&r(1,1)\\r(1,2)&r(1,1)&r(1,0)\end{bmatrix}$$

$$R_2 = \begin{bmatrix} r(2,0) & r(2,1) & r(2,2) \\ r(2,1) & r(2,0) & r(2,1) \\ r(2,2) & r(2,1) & r(2,0) \end{bmatrix}$$

For this specific model the linear system in (7.38) has nine unknowns and the solution gives

$$\begin{aligned} a(1,1) = a(-1,-1) &= -0.011, \quad a(1,0) = a(-1,0) = -0.25 \\ a(1,-1) = a(-1,1) &= -0.011, \quad a(0,1) = a(0,-1) = -0.25 \\ \sigma_\eta^2 &= 0.17 \end{aligned}$$

7.3 FEATURES FOR SHAPE AND SIZE CHARACTERIZATION

In a number of image analysis applications, an important piece of information is the shape and size of an object of interest within the image. For example, in medical applications the shape and size of nodules are crucial in classifying them as malignant or benign. Nodules with an irregular boundary have a high probability of being malignant and those with a more regular boundary are usually benign. Also, it has been observed that in certain cases nodules with a perimeter of more than 3 cm are usually malignant [Cavo 92], [Vinci 02].

Another example in which the shape of the object is of major importance is the automatic character recognition in an *optical character recognition (OCR)* system [Mori 92], [Plam 00]. Although there are OCR systems employing our already familiar regional features, there is a large class of techniques that use the shape information residing in the *boundary curve* of the characters.

Figure 7.9a shows the character "5" seen from the scanner of an OCR system. An appropriate image segmentation algorithm (e.g., [Pita 94]) has first been applied to separate the character from the rest of the image. The character in Figure 7.9b is in binary form. This is the result of the binarization phase, in which all gray levels

(a) (b) (c)

FIGURE 7.9: The character "5" after (a) the segmentation of the scanned image and then (b) the application of a binarization algorithm and (c) its boundary after the application of a boundary extraction algorithm in the binarized version.

of the character region below a certain threshold become 0 and all above it become 1 [Trie 95]. Figure 7.9c shows the resulting boundary, after the application of a boundary extraction algorithm (e.g., [Pita 94]) on the binary version. Thus, in the last version there is no texture of interest inside the character. What is of paramount importance in such systems is feature invariance in geometric transformations. The recognition of the character must be insensitive to its position, size, and orientation. A review of various methodologies for invariant pattern recognition techniques can be found in [Wood 96].

The shape characterization of a region or an object can be achieved in various ways. Two are the major directions along which we will proceed. One is to develop techniques that provide a full description of the boundary of the object in a regenerative manner. In words, the boundary can be reobtained from the description coefficients, such as by using a Fourier expansion of the boundary, which in turn can be reconstructed from its Fourier coefficients. The other direction is to use features that are descriptive of the characteristics of the shape of the region but are not regenerative. Examples of such features are the number of corners in the boundary and the perimeter. They provide useful information about the boundary, but they are not sufficient to reproduce it. In the following we will focus on some basic techniques, which have in turn given birth to a large number of variants shaped to fit specific application requirements (for example, see [Trie 96] for a review).

7.3.1 Fourier Features

Let (x_k, y_k), $k = 0, 1, \ldots, N-1$, be the coordinates of N samples on the boundary of an image region, Figure 7.10a. For each pair (x_k, y_k) we define the complex variable

$$u_k = x_k + jy_k$$

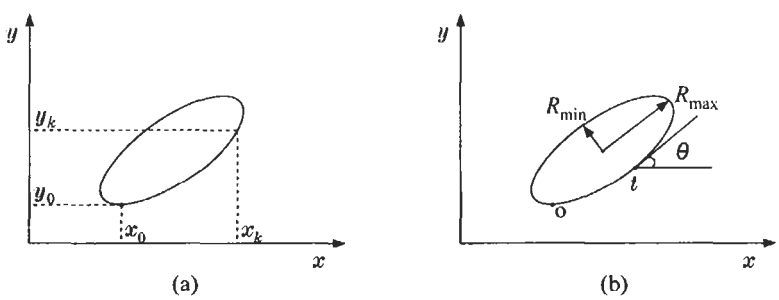


FIGURE 7.10: Boundary of an image region and associated parameters.

For the N u_k points we obtain the DFT f_l

$$f_l = \sum_{k=0}^{N-1} u_k \exp\left(-j\frac{2\pi}{N}lk\right), \quad l = 0, 1, \ldots, N-1$$

The coefficients f_l are also known as the *Fourier descriptors* of the boundary. Once the f_l are available, the u_k can be recovered and the boundary can be reconstructed. However, our goal in pattern recognition is not to reconstruct the boundary. Thus, a smaller number of coefficients (or descriptors) is usually used, enough to include sufficient discriminatory information. In the sequel, we will investigate how the Fourier descriptors are affected by the actions of translation, rotation, and scaling. For translation we have

$$x'_k = x_k + \Delta x$$
$$y'_k = y_k + \Delta y$$

Then

$$u'_k = u_k + (\Delta x + j\Delta y) \equiv u_k + \Delta u'$$

For rotation, it is not difficult to verify that in rotating all points of the region by θ, with respect to the origin, the rotated coordinates correspond to (Problem 7.13)

$$u'_k = u_k \exp(j\theta)$$

If f_l, f'_l are the DFTs of u_k, u'_k, respectively, then from the DFT definition we get

$$\textit{Translation} : u'_k = u_k + \Delta u' \Longrightarrow f'_l = f_l + \Delta u'\delta(l)$$
$$\textit{Rotation} : u'_k = u_k \exp(j\theta) \Longrightarrow f'_l = f_l \exp(j\theta)$$
$$\textit{Scaling} : u'_k = au_k \Longrightarrow f'_l = af_l$$
$$\textit{Translation of the sampling origin} : u'_k = u_{k-k_0} \Longrightarrow$$
$$f'_l = f_l \exp\left(-j2\pi k_0 \frac{l}{N}\right)$$

In words, translation affects only the f'_0 coefficient. Rotation affects *the phase* of all the coefficients by the *same factor* and it has *no effect on their magnitude*. Scaling affects all coefficients in the same way, and thus it has no effect on the ratios $\frac{f_i}{f_j}$. The sampling point origin, within the boundary, affects the phase but leaves invariant the magnitude $|f_l|$.

This *deterministic* manner, in which the three geometric transformations affect the Fourier coefficients, allows the development of appropriate normalized

versions that are invariant to these actions [Crim 82, Arbt 90, Gran 72]. Let us demonstrate the rationale of such approaches via an example, by considering the boundary of an object. The first decision to be taken, prior to the computation of the Fourier coefficients, is to define the first sampling point (x_0, y_0) on the boundary. In practice, the choice of this point for each character has a degree of randomness. The choice of a different sampling origin corresponds to a relative translation of, say, $k_0 < N$ samples (since the boundary is a closed curve, the relative translation will always be $(k - k_0)$ modulo $N < N$). As we have seen earlier, this affects the Fourier descriptors

$$u'_k = u_{k-k_0} \Longrightarrow f'_l = f_l \exp\left(-j2\pi k_0 \frac{l}{N}\right) \tag{7.41}$$

hence

$$f'_1 = f_1 \exp\left(-j2\pi \frac{k_0}{N}\right) \Longrightarrow f'_1 = |f_1| \exp(-j\phi_1) \exp\left(-j2\pi \frac{k_0}{N}\right)$$

where $|f_1|, \phi_1$ are the magnitude and phase of f_1, respectively. Hence, the phase of f'_1 is $\phi'_1 = \phi_1 + 2\pi \frac{k_0}{N}$. In the sequel we define the following normalized Fourier coefficients:

$$\hat{f}_l = f_l \exp(jl\phi_1) \tag{7.42}$$

The corresponding normalized coefficient with shifted origin will be

$$\hat{f}'_l = f'_l \exp(jl\phi'_1) = f'_l \exp\left(jl\phi_1 + j2\pi k_0 \frac{l}{N}\right) \tag{7.43}$$

Taking into account (7.41), we obtain

$$\hat{f}'_l = \hat{f}_l$$

Thus, the preceding normalization generates features that are *invariant to the choice of the sampling origin* (x_0, y_0).

This method of exploiting the power of the Fourier transform as a tool for boundary description is not the only possibility. An alternative is to express the coordinates of the boundary contour points as functions of the boundary length t, measured from an origin within the boundary, that is, $(x(t), y(t))$. Since the boundary is a closed curve, these are periodic functions and they can be expanded in their Fourier series. Invariant versions of the Fourier coefficients can then be computed and used as features for pattern recognition [Kuhl 82, Lin 87]. Comparative performance studies of a number of invariant Fourier-based features, in the context of handwritten character recognition, can be found in [Pers 77, Taxt 90].

Another way is to generate Fourier descriptors from the curvature $k(t)$ function of the boundary, defined as

$$k(t) = \frac{d\theta(t)}{dt}$$

where $\theta(t)$ is the tangent angle (Figure 7.10b) at a point a distance t from the origin, which is marked "o" in the figure. Such a description is justified by Gauss's theorem, stating that every curvature function corresponds to one and only one curve in space (with the exception of its position in space). The advantage of such a description stems from its obvious scale invariance property. If we measure the length of the boundary at a point by the number of pixels n between this point and the origin of the curve, the curvature of the boundary is approximated by

$$\begin{aligned} \theta_n &= \tan^{-1} \frac{y_{n+1} - y_n}{x_{n+1} - x_n}, \quad n = 0, 1, \ldots N-1 \\ k_n &= \theta_{n+1} - \theta_n, \qquad\qquad n = 0, 1, \ldots N-1 \end{aligned} \tag{7.44}$$

In the previous chapter we have seen that an alternative to Fourier descriptors is to use wavelet coefficients. However, as we pointed out there, the definition of invariant wavelet descriptors is not a straightforward task, and invariance is attempted via indirect methods.

7.3.2 Chain Codes

Chain coding is among the most widely used techniques for boundary shape description. In [Free 61], the boundary curve is approximated via a sequence of connected straight line segments of preselected direction and length. Every line segment is coded with a specific coding number depending on its direction.

In Figure 7.11 two possible choices, usually encountered in practice, are shown. In this way a *chain code* $[d_i]$ is created, where d_i is the coding number of the direction of the line segment that connects boundary pixel (x_i, y_i) with the next one (x_{i+1}, y_{i+1}), sweeping the boundary in, say, the clockwise sense. A disadvantage of this description is that the resulting chain codes are usually long and at the same time are very sensitive in the presence of noise. This leads to chain codes with variations due to noise and not necessarily to the boundary curve. A way out is to resample the boundary curve by selecting a grid of larger dimensions. For each of the boxes of the grid all points inside a box are assigned the value of the respective box center. In Figure 7.12a the original samples are shown alongside the larger sampling grid. Figure 7.12b is the resulting resampled version. The chain code is formed from the sequential connection of these pixels, Figure 7.12c. If we consider the length of the grid side as the basic measurement unit, then for even-coded

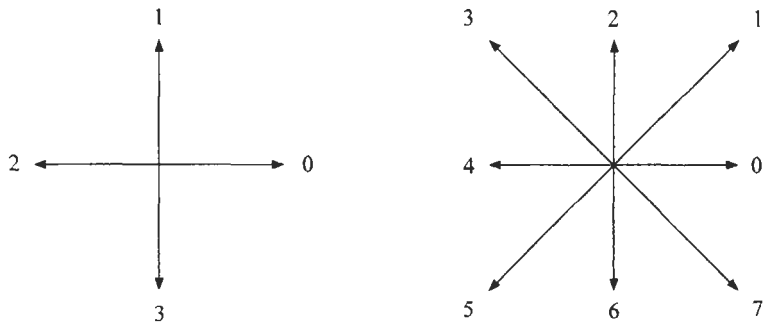


FIGURE 7.11: Directions for a (a) 4-directional chain code and (b) an 8-directional chain code.

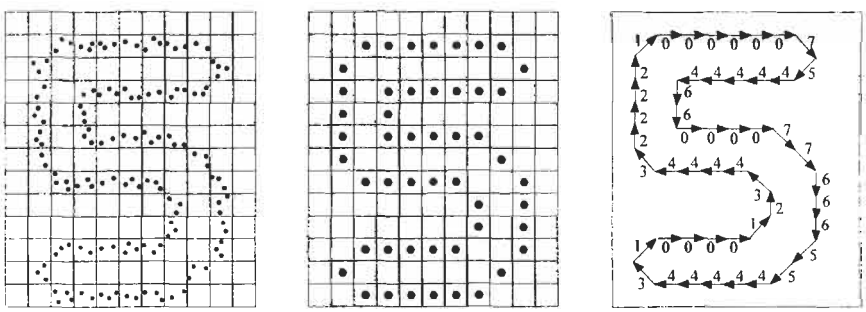
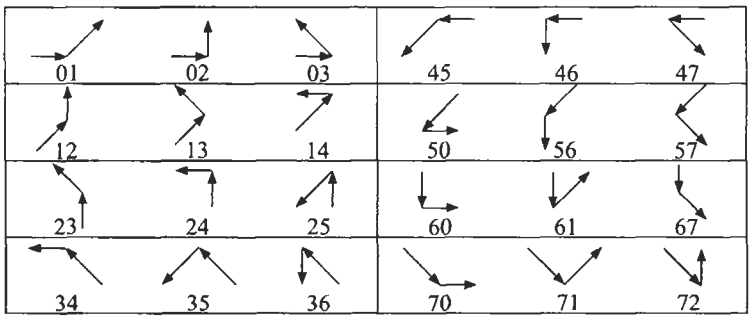


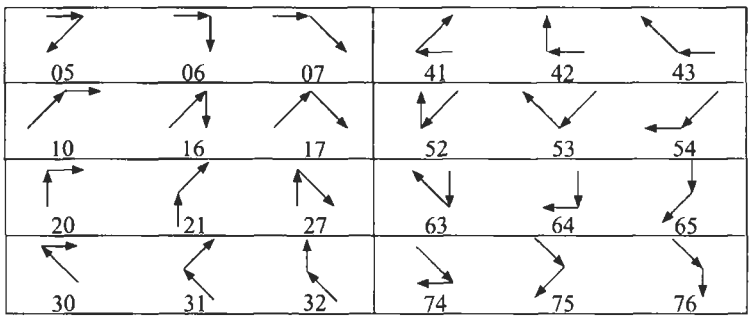
FIGURE 7.12: The character "5" and (a) its original sampled image, (b) its resampled version on a coarser grid, and (c) the resulting chain code.

directions, 0, 2, 4, 6, the length of the corresponding straight line segment is 1, and for the odd-coded directions, 1, 3, 5, 7, it is $\sqrt{2}$ (from Pythagoras' theorem). For the case of Figure 7.12b and for a coding with eight possible directions, the resulting chain code is shown in the Figure 7.12c. This sequence of numbers constitutes the spine on which a number of shape-related features are built. Two possibilities, for example, are the following [Lai 81, Mahm 94].

Direction and direction length features. For each direction we count the number of times a specific chain code number appears in the chain. Then this number is divided by the total number of chain codes that appear in the chain description of



(a) Concave features



(b) Convex features

FIGURE 7.13: Curvature features characterizing the boundary polygon that results from an eight-directional chain code description of the boundary.

the boundary. For an eight-code scheme this procedure gives rise to eight features (one for each direction). Another way that also provides eight features is to divide the total length of the line segments in each direction by the total length of the boundary line.

Curvature features. These features quantify concave (smaller than 180°) and convex (larger than 180°) external angles between adjacent edges at the corners of the polygon that is formed by the line segments when the boundary curve is scanned in the *clockwise sense*. Figure 7.13 shows the possible cases. For example, successive directions 01, 02, 23, 71 correspond to concave external angles, and the pairs 06, 41, 64, 75 to convex angles. The occurrence percentage of each of these cases in the chain code defines a respective feature. Sometimes chain code pairs are grouped according to whether the first chain code is even or odd. Thus, a total of 16 features are generated, 8 for the convex and 8 for the concave case.

7.3.3 Moment-Based Features

In (7.21) and (7.25) the geometric moments and central moments were defined. If in the place of $I(i, j)$ we consider the sequence

$$I(i, j) = \begin{cases} 1 & (i, j) \in C \\ 0 & (i, j) \text{ otherwise} \end{cases}$$

where C is the set of points (i, j) *inside* the object of interest, then we obtain a way to describe the shape of the object through the moments. Indeed, in such a case only the limits in the summations (hence the object's shape) are taken into account, whereas the details inside the object (i.e., texture) do not participate. Hence

$$m_{pq} = \sum_i \sum_j i^p j^q, \qquad (i, j) \in C$$

with $m_{00} = N$, the total number of pixels inside the region. The features

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

define the *center of mass* $(\bar{x}, \bar{y})$. The respective central moments become

$$\mu_{pq} = \sum_i \sum_j (i - \bar{x})^p (j - \bar{y})^q, \qquad (i, j) \in C$$

The invariant moments can in turn be computed and used, whenever appropriate. Two useful quantities that are related to these moments and provide useful discriminatory information are:

1. *Orientation*

$$\theta = \frac{1}{2} \tan^{-1} \left[\frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \right]$$

   which is the angle between the axis with the least moment of inertia and the x-coordinate axis (Problem 7.18)
2. *Eccentricity*

$$\epsilon = \frac{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}}{area}$$

   Another representation of the eccentricity is via the ratio $\frac{R_{max}}{R_{min}}$ of the maximum to the minimum distance of the center of mass $(\bar{x}, \bar{y})$ from the object's boundary (Figure 7.10b)

7.3.4 Geometric Features

The features of this subsection are derived directly from the geometry of the object's shape. The *perimeter* P of the object and its *area* A are two widely used features. If $\boldsymbol{x}_i, i = 1, 2, \ldots, N$, are the samples of the boundary, then the perimeter is given by

$$P = \sum_{i=1}^{N-1} \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\| + \|\boldsymbol{x}_N - \boldsymbol{x}_1\|$$

If we consider the area of a pixel as the measuring unit, a straightforward way to compute the area enclosed by a boundary is by counting the number of pixels inside the region of the object. The *roundness ratio* is a third quantity, defined as

$$\gamma = \frac{P^2}{4\pi A}$$

A useful feature that is related to the curvature of the boundary, as defined in (7.44), is the so-called *bending energy* at a point n, given by

$$E(n) = \frac{1}{P} \sum_{i=0}^{n-1} |k_i|^2$$

Another popular feature is the *number of corners* in the boundary contour. These correspond to points where the curvature k_i takes large values (infinity in theory). In [Ghos 97] corners as well as other topological features are detected via the use of Zernike moments and appropriate parametric modeling of the respective topological image intensity profile.

The *number of holes* inside the region of an object is another useful quantity. For example, a large error percentage in handwriting character recognition tasks is related to the difficulty of the classifiers in distinguishing "8" from "0," because their boundaries look alike. The detection of the presence of holes inside the object, using appropriate algorithms, is extra information that can be beneficially used for recognition [Lai 81, Mahm 94].

In our study so far, we have demonstrated how to derive geometric features from the boundary curve. However, this is not the only possibility. For example in [Wang 98] geometric features are extracted directly from the gray level variation within the image region. In this way, the binarization phase is avoided, which in some cases can become tricky. Another direction that has been used extensively in OCR is to work on the thinned version of the binarized character.

Figure 7.14 illustrates the procedure via an example. Figure 7.14b is the result of the application of a thinning algorithm (e.g., [Pita 94]) on the binary version of the character "5" of Figure 7.14a. Also in Figure 7.14b the so-called

(a) (b) (c)

FIGURE 7.14: (a) Binarized version of 5 (b) the thinned version with the key points and (c) version with edges connecting key points.

key points are denoted. These can be node points where one or more lines (strokes) of the character are crossed or corner points or end points. These can be computed by processing neighboring pixels. For example, in order to identify an end point, we look at its eight neighboring pixels. An end point has only one neighbor at gray level 1 and the rest are 0. In the sequel, the thinned version of the character is simplified as a set of line segments (edges) connecting the key points, Figure 7.14c. Each edge can then be characterized by its direction, for example, using the chain code; its length, for example, long or short; and its relation to its neighboring edges. In the sequel each character is described by an array providing this information in a coded form. Classification is then based on these coded matrices by defining appropriate costs. The interested reader may consult, for example, [Lu 91, Alem 90] for more details.

7.4 A GLIMPSE AT FRACTALS

We have already seen that the 1980s was the decade in which two major tools were introduced into the realm of pattern recognition applications (among others): neural networks and wavelets. The same decade was also the time when another tool was adopted in many application areas to offer its potential power. *Fractals and fractal dimension* have become the focus of considerable research effort. This section aims at giving the basic definitions and outlining the basic concepts behind the use of fractals in pattern recognition. A deeper study of the area is beyond the goals of a short section, and the interested reader may refer to a number of books and articles available [Mand 77, Tson 92, Falc 90].

7.4.1 Self-Similarity and Fractal Dimension

Let us consider the straight line segment of length L in Figure 7.15a. Divide L into N (two for the example of the figure) equal parts of length l. Each of the resulting parts is still a straight line segment and its length has been scaled down

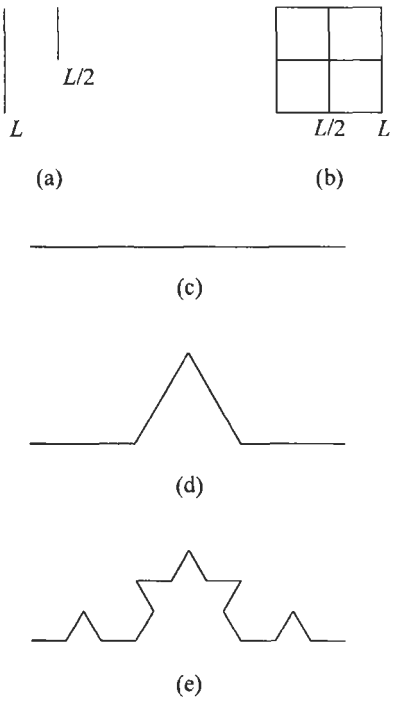


FIGURE 7.15: Self similar structures (a) line segment (b) square (c)–(e) three stages in the generation of Koch's curve.

by a factor $m = \frac{l}{L} = \frac{1}{N}$. Magnification of any of these parts by the same factor will reproduce the original line segment. We refer to such types of structures as *self-similar*. If instead of a straight line segment we had a square of side L (Figure 7.15b), then scaling down the side by $m = \frac{1}{N^{1/2}}$ would result into N square parts. Each part looks like the original square, which can be reobtained from the parts after magnification. The same is true for all dimensions, where N similar parts result after scaling the sides of the D-dimensional (hyper)cube by $m = \frac{1}{N^{1/D}}$, $D = 1, 2, \ldots$. That is, the Euclidean dimension D is directly related to the scaling and the number N of the resulting self-similar parts. We can write

$$N = \left(\frac{L}{l}\right)^D \equiv m^{-D} \tag{7.45}$$

If we now want to measure the length (area, volume) of the original segment (hypercube) using as a measurement unit a scaled element of length l (l^2, l^3, etc.),

then the result is independent of the size l of the measuring unit. Indeed, the resulting metric property (length, area, etc.) is given by

$$M = N(l)l^D \tag{7.46}$$

where $N(l)$ is the number of parts that cover the curve (area, etc.) to be measured and l is the size of the measuring unit. Combination of (7.45) and (7.46) leads to a metric M that is always constant (L^D) for the same structure and it is independent of the size l of the chosen unit, as expected.

Let us now turn our attention to some more interesting structures, such as the one in Figure 7.15d. The curve in Figure 7.15d results from the straight line segment of Figure 7.15c, known as the *initiator,* by the following strategy: a) divide the segment into three equal parts and (b) replace the central one by the two sides of an equilateral triangle, with sides of size equal to the size of the scaled parts. The procedure is then repeated for each of the line segments in Figure 7.15d and this results in the structure of Figure 7.15e. This process can go on indefinitely and the limit curve is the so-called Koch curve [Mand 77]. Such a curve is everywhere continuous but nowhere differentiable. It is readily observed that at each step of the iteration, the resulting structure is part of the structure that will result in the next iteration, after a scaling by 3. The curve therefore has a self-similar structure. In the sequel we will try to measure the length of the curve. Using as a (measuring) unit a segment of length $l = \frac{L}{3}$ (i.e., Figure 7.15d), the resulting length is 4. For a unit segment $l = \frac{L}{3^2}$ (i.e., Figure 7.15e), the measured length is 4^2. It is not difficult to see that the length keeps increasing with decreasing unit size and tends to infinity as the size of the measuring unit tends to zero! That is, the length of the curve depends not only on the curve itself but also on the adopted measurement unit! This strange result is the outcome of an "unfair" measurement process. Indeed, in the case of the Koch curve, scaling by 3 results in four similar parts. In contrast, in the case of a straight line segment scaling by $m = \frac{1}{N}$ results in the same number N of similar parts. In higher dimensional Euclidean space, scaling by $m = N^{-1/D}$ results in N parts. The measurement process involves this number N, the scaled side length l, and the *Euclidean dimension* D, as (7.46) suggests. From this discussion, the Euclidean dimension can also be seen as the ratio $\frac{\ln N}{-\ln m} = D$. Starting from this observation, let us now define the *similarity dimension* of a general self-similar structure as

$$D = \frac{\ln N}{-\ln m} \tag{7.47}$$

where N is the number of the resulting similar parts, when scaling by a factor m. For hypercube structures the similarity dimension is the respective Euclidean dimension, which is an *integer.* In contrast the corresponding similarity dimension of the Koch curve $D = \frac{\ln 4}{-\ln(\frac{1}{3})}$ is a *fraction* and not an integer. Such structures

are called *fractals*, and the corresponding similarity dimension is called a *fractal dimension*. Measuring a fractal structure, we can adopt (7.46) with the corresponding fractal dimension in the place of D. The result of the measurement process now becomes independent of the measuring tool l. Indeed, it is easy to see that using the definition in (7.47), (7.46) results in a constant $M = L^D$ for $m = \frac{l}{L}$. *The use of similarity dimension, therefore, results in a consistent description of the metric properties of such self-similar structures*. For a deeper treatment and other definitions of the dimension the interested reader may consult more specialized texts (e.g., [Falc 90]).

7.4.2 Fractional Brownian Motion

A major part of our effort in this chapter was dedicated to the description of statistical properties of signals and images and to the ways these can be exploited to extract information-rich features for classification (e.g., co-occurrence matrices, AR models). In this section we will focus our attention on whether the notion of self-similarity is extendable to stochastic processes and, if it is, how useful it can be for our interests. In the previous section "similarity" referred to the shape of a curve. For statistics such a view would be of no interest. From a statistical point of view it would be more reasonable and justifiable to interpret similarity from the perspective of "similar statistical properties," that is, mean, standard deviation, and so forth. Indeed, it can be shown that stochastic processes that are self-similar under scaling do exist. Furthermore, such processes can model adequately a number of processes met in practice.

Let $\eta(n)$ be a white (Gaussian) noise sequence with variance $\sigma_\eta^2 = 1$. The process defined as

$$x(n) = \sum_{i=1}^{n} \eta(i)$$

is known as a *random walk* sequence, and it belongs to a more general class of stochastic processes known as *Brownian motion* processes [Papo 91, page 350]. It is straightforward to see that

$$E[x(n)] = 0$$

and that its variance is given by

$$E[x^2(n)] = n\sigma_\eta^2$$

Thus, the process is a nonstationary one, because its variance is time dependent. A direct generalization of this is that

$$E[\Delta^2 x(n)] \equiv E[(x(n + n_0) - x(n_0))^2] = n\sigma_\eta^2 \tag{7.48}$$

where by definition $\Delta x(n)$ is the sequence of increments. Scaling the time axis by m results in

$$E[\Delta^2 x(mn)] \equiv E[(x(mn + n_0) - x(n_0))^2] = mn\sigma_\eta^2 \tag{7.49}$$

Hence, if the sequence of increments is to retain the same variance after scaling, it should be scaled by $\sqrt{m}$. Furthermore, it is easy to see that the sequence of increments, as well as the scaled versions, follow a Gaussian distribution (e.g., [Falc 90]). Recalling that Gaussian processes are completely specified by their mean and variance, we conclude that the increments $\Delta x(n)$ of $x(n)$ are *statistically self-similar in the sense that*

$$\Delta x(n) \quad \text{and} \quad \frac{1}{\sqrt{m}} \Delta x(mn) \tag{7.50}$$

are described by the same probability density functions, for any n_0 and m. Figure 7.16 shows three curves of the scaled random walk increments for $m = 1, 3, 6$. It is readily observed that they indeed "look" alike. Such curves, for which different scaling has been used for the coordinates $(\Delta x, n)$, are also known as *statistically self-affine*.

The random walk Brownian motion is a special case of a more general class of processes known as *fractional Brownian motion sequences* (fBm), introduced

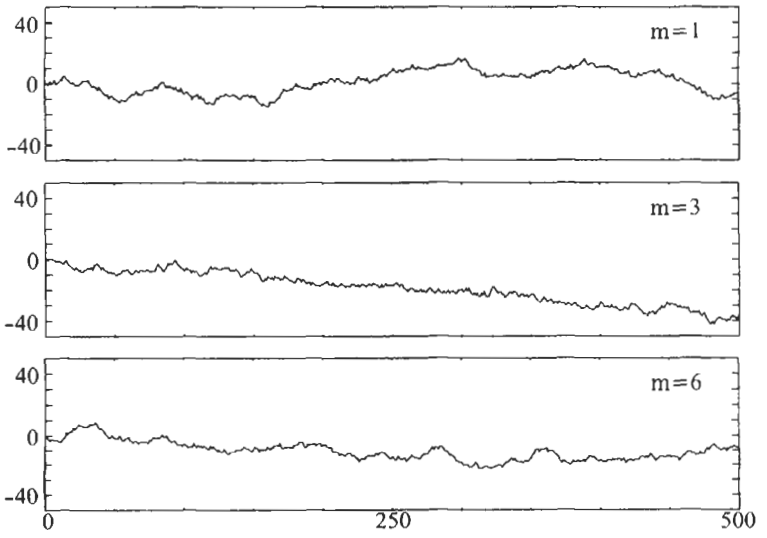


FIGURE 7.16: Time evolution of the random walk ($m = 1$) sequence and two of its self-affine versions. They all do look alike.

in [Mand 68]. The increments of this type of processes have variance of the general form

$$E[\Delta^2 x] \propto (\Delta n)^{2H} \tag{7.51}$$

with $0 < H < 1$, $\Delta n \equiv n - n_0$ and $\propto$ denoting proportionality. The parameter H is also known as the *Hurst parameter*. As in the case of Brownian motion, the increments of such processes are *statistically self-affine* in the sense that the processes

$$\Delta x(n) \quad \text{and} \quad \frac{1}{m^H}\Delta x(mn)$$

are described by the same probability density functions. The parameter H relates to the visual smoothness or coarseness of the respective graph of the increments versus time. This is an implication of (7.51). Let us start from a maximum interval Δn, corresponding to an incremental variance σ^2. In the sequel we halve the interval to $\Delta n/2$. The respective variance will be reduced by the factor $(1/2)^{2H}$. This process can go on. Each time we reduce the interval Δn by half, we look at increments between points located closer in time. The higher the value of H, the greater the reduction of the variance of the increments between these points, indicating smoother curves. For $H = 0$, the variance of the increments remains constant and independent of Δn. This process is not an fBm and it corresponds to a white noise stationary process, with no dependence between adjacent time instants. Hence, it exhibits the most erratic behavior, and its graph has the most coarse appearance. *This observation indicates that the parameter H could be used as a measure of the "smoothness" of such curves*. By varying H one can get curves of varying degree of smoothness [Saup 91]. Figure 7.17 indeed verifies that the curve for $H = 0.8$ is smoother than the one for $H = 0.2$ and both are smoother than the top one, which corresponds to a white noise sequence. As was the case with the fractal curves of the previous subsection, a dimension can also be defined for curves resulting from fBm processes. It can be shown [Falc 90, page 246] *that an fBm process with parameter H corresponds to a curve with fractal dimension* $2 - H$. In general, if l is the number of free parameters of the graph, the corresponding fractal dimension is $l + 1 - H$. For a graph as in Figure 7.17, $l = 1$ and for an image $l = 2$.

The question now is how all these no doubt mind-stimulating points can be of use to us in the context of pattern recognition. The terms smoothness and coarseness have been used in association with the parameter H and equivalently with the dimension D of an fBm process. On the other hand, the terms "smoothness," "roughness," were in a central position when dealing with feature generation for texture classification. We have now reached our crucial point. Provided that we can describe the sequence of gray levels of an image as an fBm process,

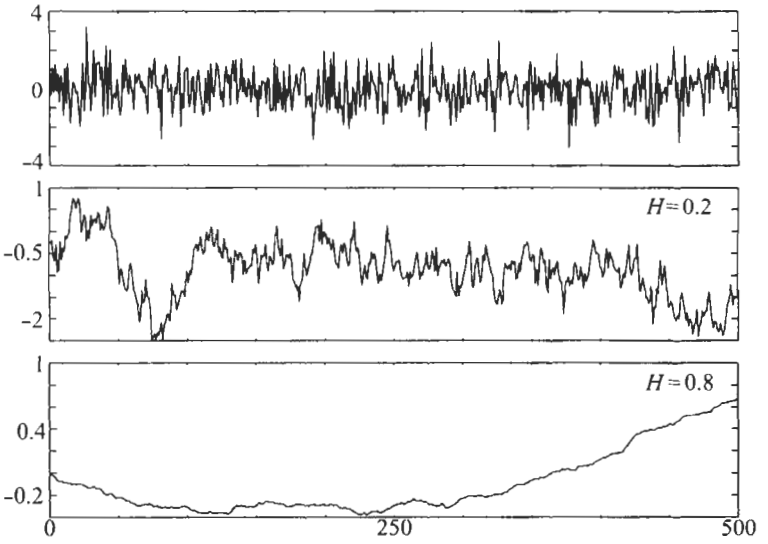


FIGURE 7.17: Time evolution of a white noise sequence (top) and two fBm processes of different Hurst parameters H. The lower the value of H, the coarser the appearance of the graph.

the corresponding fractal dimension can be used as a potential feature for texture classification. In [Pent 84] it is reported that this is indeed true for a number of cases. Using a number of textured images from [Brod 66], as well as images from natural scenes, it was found that a large percentage of them exhibited fBm behavior. This was easily verified by constructing the histogram of differences (increments) of the gray level intensities for various relative pixel distances Δn. It turned out that for each value of Δn the corresponding histogram was close to a Gaussian pdf centered at zero. Furthermore, the widths of the Gaussian-like histograms were different for the different relative pixel distances Δn. The larger the Δn, the wider the resulting histogram. However, we know that the width of a Gaussian is directly related to its variance. The plot of the variance as a function of relative pixel distance revealed an underlying fBm nature of the intensity processes, at least over a 10:1 range of relative distances measured. The parameter H, or equivalently the fractal dimension D, was then used successfully to distinguish a number of different textured regions in an image. The estimation of the H can take place via its definition in (7.51). Taking the logarithm, we get

$$\ln E[\Delta^2 x] = \ln c + 2H \ln \Delta n$$

where c is the proportionality constant and Δx is now the difference in the gray level intensities between pixels at relative distance Δn, that is, $\Delta n = 1, 2,$

and so on. Obviously, $c = E[\Delta^2 x]$ for $\Delta n = 1$. For each pixel distance Δn the corresponding average $\Delta^2 x$ is computed over the image window of interest. The resulting points $(E[\Delta^2 x], \Delta n)$ are plotted in a two-dimensional logarithmic plot. A straight line is then fitted through the points using a least squares linear regression technique. The parameter H is provided by the slope of the line. This is also a test of the fractal nature of the underlying process. If the resulting points do not lie approximately on a straight line, the fractal model assumption will not be valid. Figure 7.18 demonstrates the procedure for two images. The one on the right is an artificially produced fractal image with $H = 0.76$. The least squares fit in the logarithmic plot of the standard deviation against Δn results in a straight line of slope 0.76. The image on the left is from [Brod 66]. We observe that the resulting least squares fit is reasonable, suggesting that the image is approximately fractal in nature. The slope is now $H = 0.27$. The lower value of H reflects the fact that the latter image is coarser than the former.

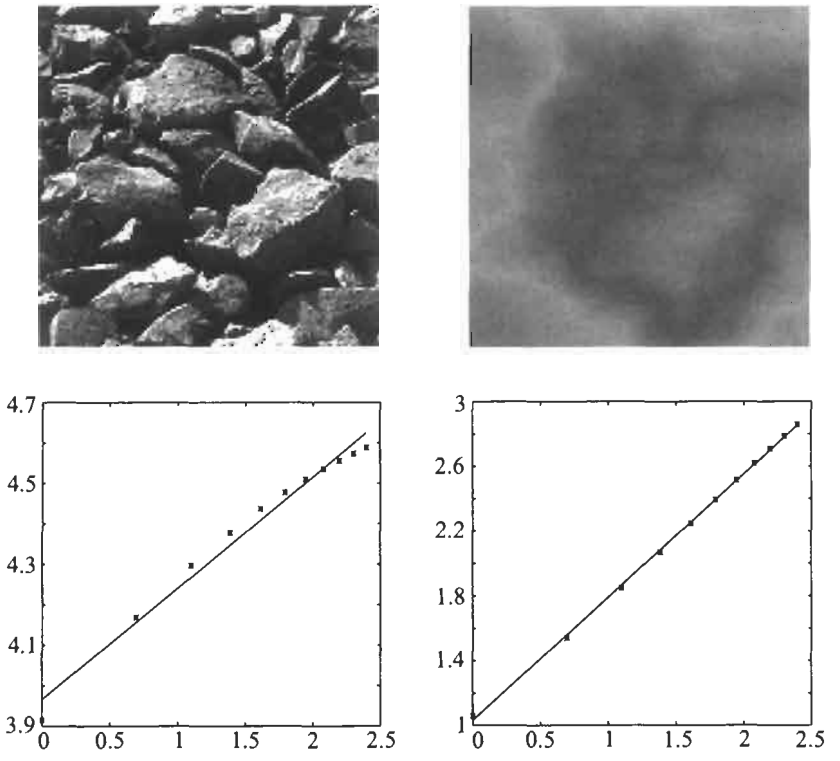


FIGURE 7.18: Examples of images with corresponding logarithmic plots of the standard deviation of increments (vertical axis) versus relative distance (horizontal axis).

Fractional modeling and the use of fractal dimension D as a feature for classification have been demonstrated in a number of applications [Chen 89, Lund 86, Rich 95]. However, the method is not beyond drawbacks. Indeed, it may happen that different textures result in the same fractal dimension, thus limiting the classification potential of the method. Another shortcoming is that, in practice, physical processes retain their fractal characteristics over a range of distances but not over all ranges. Thus, the fractal dimension may change as we pass from one range of scales to another [Pent 84, Pele 84], and the use of a single Hurst parameter may not lead to sufficient modeling. To overcome these drawbacks a number of possible solutions have been suggested [Flan 92, Bass 92, Ardu 92, Kara 95, Kapl 95, Deri 93, Kapl 00].

A comparative study of various textural features, including fractal modeling techniques, can be found in [Ohan 92, Ojal 96].

Problems

7.1 Consider the image array

$$I = \begin{bmatrix} 0 & 1 & 2 & 2 & 3 \\ 1 & 2 & 0 & 2 & 0 \\ 3 & 0 & 3 & 2 & 1 \\ 1 & 2 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 & 2 \end{bmatrix}$$

Compute the co-occurrence matrix for $d = 1$ and the four directions. Then compute the ASM and CON features.

7.2 For the image array of the previous problem compute the run length matrix for the four directions and then the features SRE, LRE.

7.3 Construct a 4×4 array that will have a high contrast (CON) value in the $0°$ direction and low CON value in the $45°$ direction.

7.4 For two of the test images provided in the web site of the book, develop a program that computes the co-occurrence and run length matrices for the four directions and $d = 1$. Then compute the $ASM, CON, IDF, H_{xy}, SRE, LRE, GLNU, RLN$ features and average their respective values over the four directions. Justify your findings.

7.5 Show that if

$$\sum_{i=1}^{N} P_i = 1$$

then

$$S = \sum_{i=1}^{N} P_i^2$$

becomes minimum if $P_i = \frac{1}{N}, i = 1, 2, \ldots, N$.

7.6 Show that the central moments defined in (7.22) are invariant to translations and the normalized central moments in (7.23) are invariant to both translation and scaling.

7.7 Show that the Zernike polynomials are orthogonal on the unit circle disk, that is,

$$\int\int_{x^2+y^2\le 1} V_{nm}^*(x, y)V_{pq}(x, y)\,dx\,dy = \frac{\pi}{n+1}\delta_{np}\delta_{mq}$$

7.8 Show that if a region in an image is rotated by an angle θ with respect to the origin, the Zernike moments of the rotated region are related to the unrotated ones by

$$A'_{pq} = A_{pq}\exp(-jq\theta)$$

7.9 Write a program to compute the moments of Hu. Then apply it to two of the test images involving objects, available from the web, and compute the respective moments.

7.10 Repeat Problem 7.9 for the Zernike moments of order A_{11}, A_{20}, A_{02}.

7.11 Show Eq. (7.37).

7.12 Write a program that computes the AR parameters for a noncausal prediction model. Apply it to a homogeneous and isotropic image whose autocorrelation sequence is given by

$$r(k, l) = \exp(-\sqrt{k^2 + l^2})$$

for a window W of order $p = q = 1$.

7.13 Let $u_k = x_k + jy_k$, where (x_k, y_k) are the coordinates of the points on the boundary of an object in an image. Show that if the object is rotated by an angle θ with respect to the origin of the axis, the new complex sequence is

$$u'_k = u_k\exp(j\theta)$$

7.14 Let t denote the length along a closed boundary curve measured from an origin within the curve, and $x(t)$, $y(t)$ the coordinates as functions of t. If T is the total length of the curve, then the following Fourier series expansion holds.

$$x(t) = a_0 + \sum_{n=1}^{\infty}\left[a_n\cos\frac{2\pi nt}{T} + b_n\sin\frac{2\pi nt}{T}\right]$$

$$y(t) = c_0 + \sum_{n=1}^{\infty}\left[c_n\cos\frac{2\pi nt}{T} + d_n\sin\frac{2\pi nt}{T}\right]$$

Prove that if $x(t)$, $y(t)$ are piecewise linear functions between the sampled points $(x(t), y(t))$, $t = 0, 1, \ldots, m-1$, the Fourier coefficients a_n, b_n, c_n, d_n are given by

$$a_n = \frac{T}{2\pi^2 n^2}\sum_{i=1}^{m}\frac{\Delta x_i}{\Delta t_i}[\cos\phi_i - \cos\phi_{i-1}]$$

$$b_n = \frac{T}{2\pi^2 n^2} \sum_{i=1}^{m} \frac{\Delta x_i}{\Delta t_i} [\sin \phi_i - \sin \phi_{i-1}]$$

$$c_n = \frac{T}{2\pi^2 n^2} \sum_{i=1}^{m} \frac{\Delta y_i}{\Delta t_i} [\cos \phi_i - \cos \phi_{i-1}]$$

$$d_n = \frac{T}{2\pi^2 n^2} \sum_{i=1}^{m} \frac{\Delta y_i}{\Delta t_i} [\sin \phi_i - \sin \phi_{i-1}]$$

where

$$\Delta x_i = x_i - x_{i-1}, \qquad \Delta y_i = y_i - y_{i-1}$$

$$\Delta t_i = \sqrt{\Delta x_i^2 + \Delta y_i^2}, \qquad t_i = \sum_{j=1}^{i} \Delta t_j$$

$$T = t_m, \qquad \phi_i = \frac{2n\pi t_i}{T}$$

7.15 For the Fourier coefficients in Problem 7.14 prove that the following parameters are rotation invariant:

$$I_n = a_n^2 + b_n^2 + c_n^2 + d_n^2$$

$$J_n = a_n d_n - b_n c_n$$

$$K_{1,n} = (a_1^2 + b_1^2)(a_n^2 + b_n^2) + (c_1^2 + d_1^2)(c_n^2 + d_n^2) + 2(a_1 c_1 + b_1 d_1)(a_n c_n + b_n d_n)$$

7.16 If $(x(t), y(t))$ are defined as in Problem 7.14 and

$$z(t) = x(t) + jy(t)$$

the respective complex exponential Fourier series is given as

$$z(t) = \sum_{n=-\infty}^{\infty} a_n \exp(j2\pi nt/T)$$

$$a_n = \frac{1}{T} \int_0^T z(t) \exp(-j2\pi nt/T)\, dt$$

Prove that the following parameters are scale and rotation invariant [Gran 72]:

$$b_n = \frac{a_{1+n} a_{1-n}}{a_1^2}, \quad d_{mn} = \frac{a_{1+m}^n a_{1-n}^m}{a_1^{(m+n)}}$$

where $n \neq 1$.

7.17 Prove that if $x(t), y(t)$ of the previous problem are piecewise linear functions between the points $(x(t), y(t)), t = 0, 1, \ldots, m-1$, then the Fourier coefficients a_n are given by [Lai 81]

$$a_n = \frac{T}{(2\pi n)^2} \sum_{i=1}^{m} (b_{i-1} - b_i) \exp(-jn2\pi t_i / T)$$

where

$$b_i = \frac{V_{i+1} - V_i}{|V_{i+1} - V_i|}, \quad t_i = \sum_{k=1}^{i} |V_k - V_{k-1}|, \quad i > 0, t_0 = 0$$

and $V_i, i = 1, 2, \ldots, m$, the phasors at the respective points.

7.18 Show that the orientation θ in section 7.3.3 results from minimizing

$$I(\theta) = \sum_i \sum_j [(i - \bar{x}) \cos\theta - (j - \bar{y}) \sin\theta]^2$$

7.19 Show that the power spectrum of an fdm process with Hurst parameter H is given by

$$S(f) \propto \frac{1}{f^{(2H+1)}}$$

7.20 Show that the definition of M in (7.46) results in a consistent metric for the Koch curve.

7.21 Assuming that $x(0) = 0$, show that

$$E[x(n)(x(n + n_0) - x(n))] = \frac{1}{2}\left\{(n + n_0)^{2H} - n^{2H} - n_0^{2H}\right\}$$

For the case of a Brownian motion ($H = \frac{1}{2}$) this suggests that $x(n)$ is uncorrelated to the increment. This is not true for $H \neq \frac{1}{2}$, where a nonzero correlation exists, *positive* for $H > 1/2$ and *negative* for $H < 1/2$. To prove this use the definition in (7.51). This can be generalized. That is, if $n_1 \leq n_2 \leq n_3 \leq n_4$ and the process is Brownian then

$$E[(x(n_2) - x(n_1))(x(n_4) - x(n_3))] = 0$$

References

[Alem 90] Al-Emami S., Usher M. "On-line recognition of handwritten Arabic characters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12(7), pp. 704–710, 1990.

[Arbt 89] Arbter K. "Affine-invariant Fourier descriptors," in *From Pixel to Features* (Simon J. C., ed.), pp. 153–164, Elsevier, 1989.

[Arbt 90] Arbter K., Snyder W.E., Burkhardt H., Hirzinger G. "Application of affine-invariant Fourier descriptors to recognition of 3-D objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 640–647, 1990.

[Ardu 92] Ardunini F., Fioravanti S., Giusto D.D., Inzirillo F. "Multifractals and texture classification," *IEEE International Conference on Image Processing*, pp. 454–457, 1992.

[Bass 92] Bassevile M., Benveniste A., Chou K., Golden S. A., Nikoukhah R., Willsky A. S. "Modelling and estimation of multiresolution stochastic processes," *IEEE Transactions on Information Theory*, Vol. 38, pp. 766–784, 1992.

[Brod 66] Brodatz P. *Textures — A Photographic Album for Artists and Designers*, Dover, 1966.

[Cavo 92] Cavouras D., Prassopoulos P., Pantelidis N. "Image analysis methods for solitary pulmonary nodule characterization by CT," *European Journal of Radiology*, Vol. 14, pp. 169–172, 1992.

[Chel 85] Chellapa R. "Two dimensional discrete Gaussian Markov random field models for image processing," in *Progress in Pattern Recognition* (Kanal L.N., Rosenfeld A., eds.), Vol. 2, pp. 79–112, North Holland, 1985.

[Chen 89] Chen C.C., Daponee J.S., Fox M.D. "Fractal feature analysis and classification in medical imaging," *IEEE Transactions on Medical Imaging*, Vol. 8, pp. 133–142, 1989.

[Crim 82] Crimmins T.R. "A complete set of Fourier descriptors," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 12, pp. 236–258, 1982.

[Cros 83] Cross G.R., Jain A.K. "Markov random field texture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 5(1), pp. 25–39, 1983.

[Davi 79] Davis L., Johns S., Aggrawal J.K. "Texture analysis using generalized co-occurrence matrices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 1(3), pp. 251–259, 1979.

[Deri 93] Deriche M., Tewfik A.H. "Signal modeling with filtered discrete fractional noise processes," *IEEE Transactions on Signal Processing*, Vol. 41, pp. 2839–2850, 1993.

[Falc 90] Falconer K. *Fractal Geometry: Mathematical Foundations and Applications*, John Wiley, 1990.

[Flan 92] Flandrin P. "Wavelet analysis and synthesis of fractional Brownian motion," *IEEE Transactions on Information Theory*, Vol. 38, pp. 910–917, 1992.

[Flus 93] Flusser J., Suk T. "Pattern recognition by affine moment invariants," *Pattern Recognition*, Vol. 26(1), pp. 167–174, 1993.

[Flus 94] Flusser J., Suk T. "Affine moment invariants: A new tool for character recognition," *Pattern Recognition*, Vol. 15(4), pp. 433–436, 1994.

[Free 61] Freeman H. "On the encoding of arbitrary geometric configurations," *IRE Transactions on Electronic Computers*, Vol. 10(2), pp. 260–268, 1961.

[Gall 75] Galloway M. "Texture analysis using gray level run lengths," *Computer Graphics and Image Processing*, Vol. 4, pp. 172–179, 1975.

[Ghos 97] Ghosal S., Mehrotra R. "A moment based unified approach to image feature detection," *IEEE Transactions on Image Processing*, Vol. 6(6), pp. 781–794, 1997.

[Glen 94] Glentis G., Slump C., Herrmann O. "An efficient algorithm for two-dimensional FIR filtering and system identification," *SPIE Proceedings*, VCIP, pp. 220–232, Chicago, 1994.

[Gran 72] Granlund G.H. "Fourier preprocessing for hand print character recognition," *IEEE Transactions on Computers*, Vol. 21, pp. 195–201, 1972.

[Hara 73] Haralick R., Shanmugam K., Distein I. "Textural features for image classification," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 3(6), pp. 610–621, 1973.

[Hayk 96] Haykin S. *Adaptive Filter Theory*, 3nd ed., Prentice Hall, 1996.

[Heyw 95] Heywodd M.I., Noakes P.D. "Fractional central moment method for movement-invariant object classification," *IEE Proceedings Vision, Image and Signal Processing*, Vol. 142(4), pp. 213–219, 1995.

[Hu 62] Hu M.K. "Visual pattern recognition by moment invariants," *IRE Transactions on Information Theory*, Vol. 8(2), pp. 179–187, 1962.

[Kalo 89] Kalouptsidis N., Thedoridis S., "Concurrent algorithms for a class of 1-D and 2-D Wiener filters with symmetric impulse response," *IEEE Transactions on Signal Processing*, Vol. ASSP-37, pp. 1780–1782, 1989.

[Kapl 95] Kaplan L.M., Kuo C.C.J. "Texture roughness analysis and synthesis via extended self-similar model," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17(11), pp. 1043–1056, 1995.

[Kapl 00] Kaplan L.M. "Extended fractal analysis for the texture classification and segmentation," *IEEE Transactions on Image Processing*, Vol. 8(11), pp. 1572–1585, 2000.

[Kara 95] Karayannis Y.A., Stouraitis T. "Texture classification using fractal dimension as computed in a wavelet decomposed image," *IEEE workshop on nonlinear Signal and Image Processing*, pp. 186–189, Neos Marmaras, Halkidiki, June 95.

[Kash 82] Kashyap R.L., Chellapa R., Khotanzad A. "Texture classification using features derived from random field models," *Pattern Recognition Letters*, Vol. I, pp. 43–50, 1982.

[Kash 86] Kashyap R.L., Khotanzad A. "A model based method for rotation invariant texture classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8(4), pp. 472–481, 1986.

[Khot 90a] Khotanzad A., Hong Y.H. "Invariant image recognition by Zernike moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12(5), pp. 489–497, 1990.

[Khot 90b] Khotanzad A., Lu J.H. "Classification of invariant image representations using a neural network," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 38(6), pp. 1028–1038, 1990.

[Kuhl 82] Kuhl F.P., Giardina C.R. "Elliptic Fourier features of a closed contour," *Comput. Vis. Graphics Image Processing*, Vol. 18, pp. 236–258, 1982.

[Lai 81] Lai M., Suen Y.C. "Automatic recognition of characters by Fourier descriptors and boundary line encoding," *Pattern Recognition*, Vol. 14, pp. 383–393, 1981.

[Laws 80] Laws K.I. "Texture image segmentation" Ph.D. Thesis, University of Southern California, 1980.

[Liao 96] Liao S., Pawlak M. "On image analysis by moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18(3), pp. 254–266, March 1996.

[Lin 87] Lin C.S., Hwang C.L. "New forms of shape invariants from elliptic Fourier descriptors," *Pattern Recognition*, Vol. 20(5), pp. 535–545, 1987.

[Lu 91] Lu S.Y., Ren Y., Suen C.Y. "Hierarchical attributed graph representation and recognition of handwritten Chinese characters," *Pattern Recognition*, Vol. 24(7), pp. 617–632, 1991.

[Lund 86] Lundahl T., Ohley W.J., Kay S.M., Siffer R. "Fractional Brownian motion: A maximum likelihood estimator and its application to image texture," *IEEE Transactions on Medical Imaging"*, Vol. 5, pp. 152–161, 1986.

[Mao 92] Mao J., Jain A.K. "Texture classification and segmentation using multiresolution simultaneous autoregressive models," *Pattern Recognition*, Vol. 25(2), pp. 173–188, 1992.

[Mahm 94] Mahmoud S. "Arabic character recognition using Fourier descriptors and character contour encoding," *Pattern Recognition*, Vol. 27(6), pp. 815–824, 1994.

[Mami 98] Mamistvalov A.G. "n-Dimensional moment invariants and the conceptual mathematical theory of recognition n-dimensional objects,"*IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20(8), pp. 819–831, 1998.

[Mand 68] Mandelbrot B.B, Van Ness J.W. "Fractional Brownian motion, fractional noises and applications," *SIAM Review*, Vol. 10, pp. 422–437, 1968.

[Mand 77] Manderbrot B.B. *The Fractal Geometry of Nature*, W.H. Freeman, New York, 1982.

[Mori 92] Mori S., Suen C. "Historical review of OCR research and development," *Proceedings of IEEE*, Vol. 80(7), pp. 1029–1057, 1992.

[Muku 95] Mukundan R., Ramakrshnan J. "Fast computation of Legendre and Zernike moments," *Pattern Recognition*, Vol. 28(9), pp. 1433–1442, 1995.

[Ohan 92] Ohanian P., Dubes R. "Performance evaluation for four classes of textural features," *Pattern Recognition*, Vol. 25(8), pp. 819–833, 1992.

[Ozal 96] Ojala T., Pietikainen M., Harwood D. "A comparative study of texture measures with classification based on feature distributions," *Pattern Recognition*, Vol. 29(1), pp. 51–59, 1996.

[Papo 91] Papoulis A. *Probability, Random Variables, and Stochastic Processes*, 3nd ed., McGraw-Hill, 1991.

[Pele 84] Peleg S., Naor J., Hartley R., Anvir D. "Multiple resolution texture analysis and classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 818–523, 1984.

[Pent 84] Pentland A. "Fractal based decomposition of natural scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6(6), pp. 661–674, 1984.

[Pers 77] Persoon E., Fu K.S. "Shape discrimination using Fourier descriptors," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 7, pp. 170–179, 1977.

[Pita 94] Pitas I. *Image Processing Algorithms*, Prentice Hall, 1994.

[Plam 00] Plamondon R., Srihari S.N. "On-line and off-line handwriting recognition: A comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22(1), pp. 63–84, 2000.

[Rand 99] Randen T., Husoy H.H "Filtering for texture classification: A comparative study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21(4), pp. 291–310, 1999.

[Reis 91] Reiss T.H. "The revised fundamental theorem of moment invariants," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, pp. 830–834, 1991.

[Rich 95] Richardson W. "Applying wavelets to mammograms," *IEEE Engineering in Medicine and Biology*, Vol. 14 pp. 551–560, 1995.

[Sark 97] Sarkar A., Sharma K.M.S., Sonak R.V. "A new approach for subset 2-D AR model identification for describing textures," *IEEE Transactions on Image Processing*, Vol. 6(3), pp. 407–414, 1997.

[Saup 91] Saupe D. "Random fractals in image processing," in *Fractals and Chaos* (Crilly A. J., Earnshaw R.A., Jones H., eds.), pp. 89–118, Springer Verlag, 1991.

[Tam 78] Tamura H., Mori S., Yamawaki T. "Textural features corresponding to visual Perception," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 8(6), pp. 460–473, 1978.

[Taxt 90] Taxt T., Olafsdottir J.B., Daechlen M. "Recognition of hand written symbols," *Pattern Recognition*, Vol. 23(11), pp. 1155–1166, 1990.

[Teh 88] Teh C.H., Chin R.T. "On image analysis by the method of moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10(4), pp. 496–512, 1988.

[Theo 93] Theodoridis S., Kalouptsidis N. "Spectral analysis," in *Adaptive System Identification and Signal Processing Algorithms* (Kalouptsidis N., Theodoridis S., eds.), Prentice Hall, 1993.

[Tang 98] Tang X. "Texture information in run-length matrices," *IEEE Transactions on Image Processing*, Vol. 7(11), pp. 1602–1609, 1998.

[Teag 80] Teague M. "Image analysis via the general theory of moments," *Journal of Optical Society of America*, Vol. 70(8), pp. 920–930, 1980.

[Trie 95] Trier O.D., Jain A.K. "Goal-directed evaluation of binarization methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17(12), pp. 1191–1201, 1995.

[Trie 96] Trier O.D., Jain A.K., Taxt T. "Feature extraction methods for character recognition — A survey," *Pattern Recognition*, Vol. 29(4), pp. 641–661, 1996.

[Tson 92] Tsonis A. *Chaos: From Theory to Applications*, Plenum Press, 1992.

[Unse 86] Unser M. "Local linear transforms for texture measurements," *Signal Processing*, Vol. 11, pp. 61–79, 1986.

[Unse 89] Unser M., Eden M. "Multiresolution feature extraction and selection for texture segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11(7), pp. 717–728, 1989.

[Vinci 02] Vinciarelli A. "A survey on off-line cursive word recognition," *Pattern Recognition*, Vol. 35, pp. 1433–1446, 2002.

[Wang 93] Wang L., Pavlidis T. "Direct gray-scale extraction of features for character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(10), pp. 1053–1067, 1993.

[Wang 98] Wang L., Healey G. "Using Zernike moments for the illumination and geometry invariant classification of multispectral textures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 7(2), pp. 196–203, 1998.

[Wood 72] Woods J. "Markov image modelling," *IEEE Transactions on Information Theory*, Vol. 18(3), pp. 232–240, 1972.

[Wood 96] Wood J. "Invariant pattern recognition," *Pattern Recognition*, Vol. 29(1), pp. 1–17, 1996.

CHAPTER 8

TEMPLATE MATCHING

8.1 INTRODUCTION

In all previous chapters, the task of major concern was to appoint an unknown pattern to one of the possible classes. The problem that will accompany us throughout this chapter is of a slightly different nature. We will assume that a set of reference patterns (*templates*) are available to us and we have to decide which one of these reference patterns an unknown one (the *test pattern*) matches best. These templates may be certain objects in a scene or can be strings of patterns, such as letters forming words in a written text or words or phrases in a spoken text. Typically, such problems arise in speech recognition, in automation using robot vision, in motion estimation for video coding, and in image database retrieval systems, to name but a few. A reasonable first step to approaching such a task is to define a *measure or* a *cost* measuring the "distance" or the "similarity" between the (known) reference patterns and the (unknown) test pattern, in order to perform the matching operation known as *template matching*. We know by now that each pattern is expressed in terms of a vector or a matrix with elements the set of the selected features. Then why not use one of the already known distance measures, that is, Euclidean or Frobenius norms, and perform the matching operation based on the minimum distance? A little more thinking reveals that such a straightforward approach is not enough and something more is needed. This is the crucial point that makes template matching different and at the same time interesting.

To understand this better, let us consider a written text matching problem, that is, to identify which one from a set of written words is the word, say, "beauty." However, because of errors in the reading sensors the specific test pattern may appear, for example, as "beety" or "beaut." In a speech recognition task, if a specific word is spoken by the same speaker a number of times, it will be spoken differently every time. Sometimes it will be spoken quickly and the resulting pattern will be of short duration in time, sometimes slowly and the pattern will be longer. Yet in all cases it is the "same" word spoken by the same person. In a scene analysis application the object to be identified may be present in an image but its location within the image is not known. In content-based image database retrieval systems, queries often include the shape of an object. However, the shape provided

by the user, most often, does not match exactly the shape of the object residing in the database images. The major goal of this chapter is to define measures that accommodate the distinct characteristics for each category of these problems. As is always the case with a textbook, only general directions and typical cases will be treated.

We will begin with the problem of string pattern matching and will come to the scene analysis and shape recognition problems later on. The tasks, although they share the same goal, require different tools because of their different nature.

8.2 MEASURES BASED ON OPTIMAL PATH SEARCHING TECHNIQUES

We will first focus on a category of template matching, where the involved patterns consist of strings of identified symbols or feature vectors (string patterns). That is, each of the reference and test patterns is represented as a sequence (string) of measured parameters and one has to decide which reference sequence the test one matches best.

Let $\boldsymbol{r}(i), i = 1, 2, \ldots, I$, and $\boldsymbol{t}(j), j = 1, 2, \ldots, J$, be the respective feature vector sequences for a specific pair of reference and test patterns, where in general $I \neq J$. The objective is to develop an appropriate distance measure between the two sequences. To this end, we form a two-dimensional grid with the elements of the two sequences as points on the respective axes, that is, the reference string at the abscissa (i-axis) and the test one at the ordinate (j-axis). Figure 8.1 is an example for $I = 6$, $J = 5$. Each point of the grid (node) marks a correspondence between the respective elements of the two sequences. For example node $(3, 2)$ maps the element $\boldsymbol{r}(3)$ to $\boldsymbol{t}(2)$. Each node (i, j) of the grid is associated with a *cost*, which is an appropriately defined function $d(i, j)$ measuring the "distance" between the respective elements of the strings, $\boldsymbol{t}(j)$ and $\boldsymbol{r}(i)$. A path through the grid from an initial node (i_0, j_0) to a final one (i_f, j_f) is an *ordered* set of nodes of the form

$$(i_0, j_0), (i_1, j_1), (i_2, j_2), \ldots, (i_f, j_f)$$

Each path is associated with an overall cost D defined as

$$D = \sum_{k=0}^{K-1} d(i_k, j_k)$$

where K is the number of nodes along the path. For the example of Figure 8.1, $K = 8$. The overall cost up to node (i_k, j_k) will be denoted by $D(i_k, j_k)$ and by

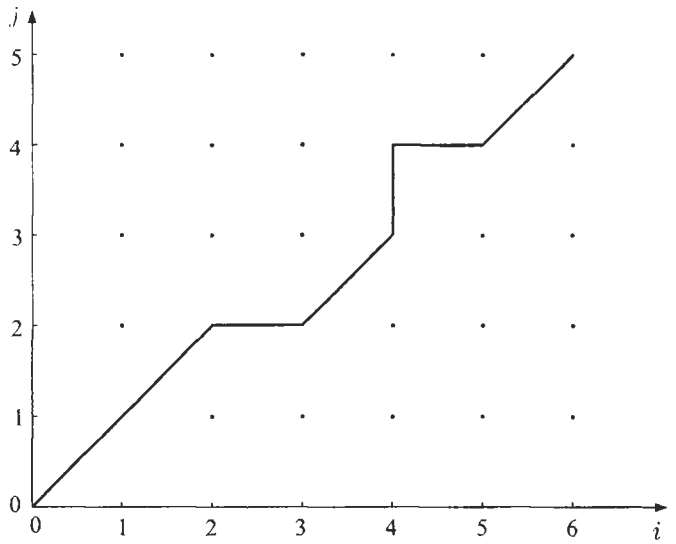


FIGURE 8.1: Each point along the path marks a correspondence between the respective elements of the test and reference patterns.

convention we assume $D(0, 0) = 0$ and also $d(0, 0) = 0$. The path is said to be *complete* if

$$(i_0, j_0) = (0, 0), \ (i_f, j_f) = (I, J)$$

The distance[1] *between the two sequences is defined as the minimum D over all possible paths.* At the same time the minimum cost path unravels the optimal pairwise correspondence between the elements of the two sequences, which is the crucial part, since the two sequences are of different lengths. In other words, the optimal path procedure makes the *alignment or warping* of the elements of the test string to the elements of the reference string, corresponding to the best matching score. Before we talk about the optimization procedure, we must point out that there are a number of variations of this scheme. For example, one may not impose the constraint of having necessarily a complete path but may adopt more relaxed constraints known as *end point constraints*. Furthermore, one could associate a cost not only with each node but also with each transition between nodes, making certain transitions more costly than others. In such cases, the cost at a node (i_k, j_k) also depends on the specific transition, that is, from which node (i_{k-1}, j_{k-1}) the

[1] The term distance here must not be interpreted with its strict mathematical definition.

(i_k, j_k) node was reached. Thus, the cost d is now of the form $d(i_k, j_k|i_{k-1}, j_{k-1})$ and the overall path cost is

$$D = \sum_k d(i_k, j_k|i_{k-1}, j_{k-1})$$

In some cases the overall path cost is defined as the product

$$D = \prod_k d(i_k, j_k|i_{k-1}, j_{k-1})$$

Finally, there are cases where d is chosen so that maximization instead of minimization is required. Obviously, in all these variations appropriate initial conditions have to be adopted. Let us now come back to the optimization problem itself. To obtain the best path, one has to search all possible combinations of paths. However, this is a computationally costly procedure. *Dynamic programming algorithms* based on Bellman's principle are powerful tools that we will adopt to reduce the computational complexity.

8.2.1 Bellman's Optimality Principle and Dynamic Programming

Let the optimal path between an initial node (i_0, j_0) and a final one (i_f, j_f) be denoted as

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

If (i, j) is an intermediate node between (i_0, j_0) and (i_f, j_f), we will denote the optimal path constrained to pass through (i, j) as

$$(i_0, j_0) \xrightarrow[(i,j)]{opt} (i_f, j_f)$$

Bellman's principle states that [Bell 57]

$$(i_0, j_0) \xrightarrow[(i,j)]{opt} (i_f, j_f) = (i_0, j_0) \xrightarrow{opt} (i, j) \oplus (i, j) \xrightarrow{opt} (i_f, j_f)$$

where $\oplus$ denotes concatenation of paths. In other words, Bellman's principle states that the overall optimal path from (i_0, j_0) to (i_f, j_f) through (i, j) is the concatenation of the optimal path from (i_0, j_0) to (i, j) and the optimal path from (i, j) to (i_f, j_f). The consequence of this principle is that once we are at (i, j) through the optimal path, then to reach (i_f, j_f) optimally we need *to search only* for the optimal path from (i, j) to (i_f, j_f).

Let us now express this in a way that will be useful to us later on. Assume that we have departed from (i_0, j_0) and let the kth node of the path be (i_k, j_k). The goal

is to compute the minimum cost required to reach the latter node. The transition to (i_k, j_k) has to take place from one of the possible nodes that are allowed to be in the $(k-1)$th position of the path (i.e., the (i_{k-1}, j_{k-1}) node). This is important. For *each node* of the grid we assume that there is a set of allowed *predecessors*, defining the so-called *local constraints*. Bellman's principle readily leads to

$$D_{min}(i_k, j_k) = \min_{i_{k-1}, j_{k-1}} [D_{min}(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})] \tag{8.1}$$

Indeed, the overall minimum cost to reach node (i_k, j_k) is the minimum cost up to node (i_{k-1}, j_{k-1}) plus the extra cost of the transition from (i_{k-1}, j_{k-1}) to (i_k, j_k). Furthermore, the search for the minimum is constrained only within the set of allowable predecessors for the (i_k, j_k) node. This procedure is carried out for all the nodes of the grid. However, in many cases not all nodes of the grid are involved, and the optimal path searching takes place among a subset of them, which are defined via the so-called *global constraints*. The resulting algorithm is known as *dynamic programming*. Equation (8.1) has to be modified accordingly if the cost D is given in its multiplicative form and/or if maximization is required.

Let us now focus on our string pattern matching task and see how the *recursive* equation (8.1) is used to construct the optimal complete path.

Figure 8.2 illustrates the procedure. The set of nodes involved in the optimization (global constraints) are denoted as dark dots, and the local constraints, defining the allowable transitions among these nodes are shown in the figure by the thin lines. Having decided to search for the complete path and assuming $D(0, 0) = 0$, the respective costs $D(i_1, j_1)$ for all the allowed nodes involved in step $k = 1$ are computed via (8.1) (in our case there are only two allowable nodes, $(1, 1)$ and $(1, 2)$). Subsequently, the costs of the (three) nodes at step $k = 2$ are computed and the procedure is repeated until we arrive at the final node (I, J). The sequence of transitions leading to the minimum $D(I, J)$ of the final node define the minimum cost path, denoted by the thick line. The optimal node correspondence, between the test and reference patterns, can then be unravelled by backtracking the optimal path. In the example of Figure 8.2 each step k of the recursion involves only nodes with the same abscissa coordinate, which reflects the local constraints adopted. In general, this is not necessary and more involved topologies may be used. However, the philosophy of the search for the minimum remains the same. In the sequel, we will apply the procedure in two different popular application areas.

8.2.2 The Edit Distance

In this section we will be concerned with patterns that consist of sets of *ordered symbols*. For example, if these symbols are letters, then the patterns are words from a written text. Such problems arise in automatic editing and text retrieval applications. Other examples of symbol strings occur in structural pattern recognition.

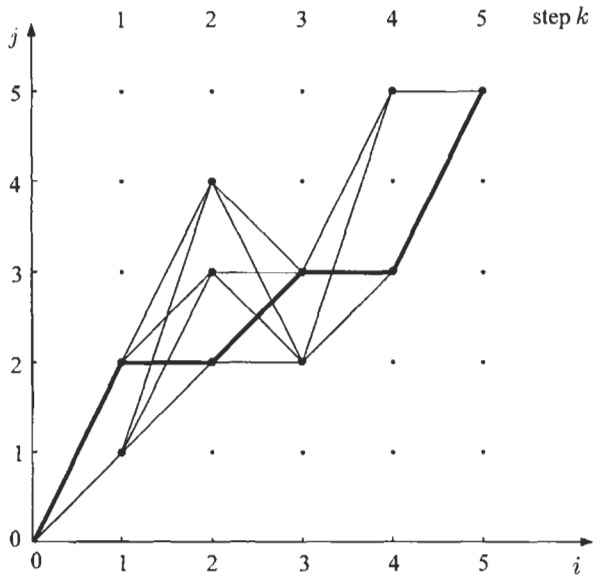


FIGURE 8.2: The optimal path is constructed by searching among all allowable paths, as defined by the global and local constraints.

Once the symbols of a (test) pattern have been identified, for example, via a reading device, the task is to recognize the pattern, searching for the best match of it against a set of reference patterns. The measure to be adopted for the matching procedure should take into account the following errors, which may occur during the symbol identification phase.

- Wrongly identified symbol (e.g., "befuty" instead of "beauty")
- Insertion error (e.g., "bearuty")
- Deletion error (e.g., "beuty")

Obviously, a combination of these errors may also occur. For the matching procedure we will adopt the philosophy behind the so-called *variational similarity*. In other words, the similarity between two patterns is based on the "cost" associated with converting one pattern to the other. If the patterns are of the same length, then the cost is directly related to the number of symbols that have to be changed in one of them so that the other pattern results. More interest arises when the two patterns are not of equal length. In such cases symbols have to be either deleted or inserted at certain places of the test string. The location where deletions or insertions are to be done presupposes an optimal alignment (warping) among the symbols of the two patterns. The Edit distance [Dome 64, Leven 66] between two

string patterns A and B, denoted $D(A, B)$, is defined as *the minimum total number of changes C, insertions I, and deletions R* required to change pattern A into pattern B,

$$D(A, B) = \min_{j}[C(j) + I(j) + R(j)] \tag{8.2}$$

where j runs over all possible combinations of symbol variations in order to obtain B from A. To elaborate a bit, note that there is more than one way to change, say, "beuty" to "beauty." For example, one can either insert "a" after "e" or change "u" to "a" and then insert "u."

We will employ the dynamic programming methodology to compute the required minimum in (8.2). To this end, we form the grid by placing the symbols of the reference pattern in the abscissa axis and the test pattern in the ordinate one. Figure 8.3 demonstrates the procedure via four examples. As already pointed out, the first step in an optimal path searching procedure via dynamic programming techniques is to state the node transition constraints imposed by the problem. For our case of interest the following constraints are adopted.

- The cost $D(0, 0)$ of the $(0, 0)$ node is zero.
- A complete path is searched.
- Each node (i, j) can be reached only through three allowable predecessors, that is,

$$(i - 1, j),\ (i - 1, j - 1),\ (i, j - 1)$$

as indicated at the bottom of Figure 8.3.

The costs associated with the above three transitions are:

1. Diagonal transitions:

$$d(i, j|i - 1, j - 1) = \begin{cases} 0 & \text{if } r(i) = t(j) \\ 1 & \text{if } r(i) \neq t(j) \end{cases}$$

That is, the cost of a transition is zero if the symbols corresponding to the (i, j) node are the same and unity if they are different, hence a symbol change has to take place.

2. Horizontal and vertical transitions:

$$d(i, j|i - 1, j) = d(i, j|i, j - 1) = 1$$

The meaning of horizontal transitions is that they attempt alignment of the two strings by insertion of a symbol; see Figure 8.3a. Thus, they add to the cost, because they imply local mismatch. Similarly, vertical transitions add to the cost because they imply deletions, Figure 8.3c.

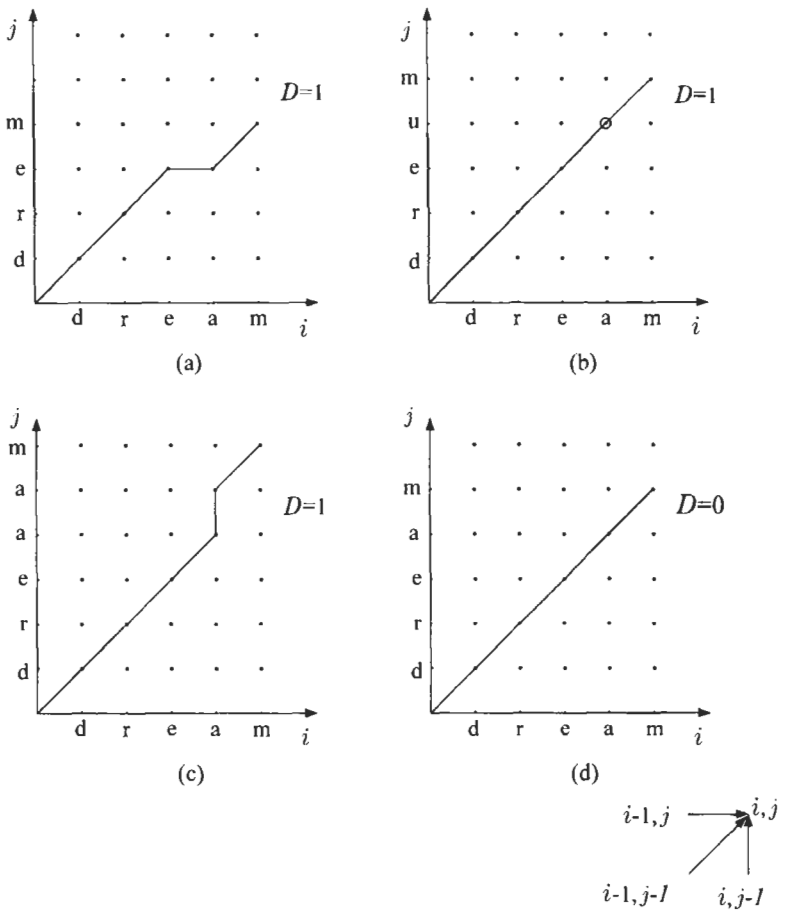


FIGURE 8.3: Computation of the Edit distance with (a) an insertion, (b) a change, (c) a deletion, and (d) an equality. The local constraints are shown at the bottom right corner.

Incorporating these constraints and the distance (8.2) in a dynamic programming procedure, the following algorithm results.

Algorithm for Computing the Edit Distance

- $D(0,0) = 0$
- For $i = 1$ to I

  —$D(i,0) = D(i-1,0) + 1$

- End { For }
- For $j = 1$ to J

  —$D(0, j) = D(0, j-1) + 1$
- End { For }
- For $i = 1$ to I

  —For $j = 1$ to J

  * $c1 = D(i-1, j-1) + d(i, j|i-1, j-1)$
  * $c2 = D(i-1, j) + 1$
  * $c3 = D(i, j-1) + 1$
  * $D(i, j) = \min(c1, c2, c3)$

  —End { For }
- End { For }
- $D(A, B) = D(I, J)$

In other words, we first compute the minimum cost for reaching *each node* of the grid, starting at (0.0), and the optimal (complete) path is subsequently constructed. Figure 8.3 shows the respective minimum cost paths and the resulting Edit distances for each of the cases. Verify that any other path for the examples of Figure 8.3 results in higher costs. A generalization of the Edit distance to allow for merges, splits and two-letter substitutions in the context of handwriting recognition is given in [Seni 96].

8.2.3 Dynamic Time Warping in Speech Recognition

In this section, we highlight the application of dynamic programming techniques in speech recognition. We will focus on the simpler form of the task, known as *discrete or isolated word recognition* (IWR). That is, we will assume that the spoken text consists of discrete words, well isolated with sufficient silent periods between them. In tasks of this type, it is fairly straightforward to decide where, in time, one word finishes and where another one starts. This is not, however, the case in the more complex *continuous speech recognition* (CSR) systems, where the speaker speaks in a natural way and temporal boundaries between words are not well defined. In the latter case, more elaborate schemes are required (e.g., [Silv 90, Desh 99, Neg 99]). When words are spoken by a single speaker and the purpose of the recognition system is to recognize words spoken by this person, then the recognition task is known as *speaker-dependent recognition*. A more complex task is *speaker-independent recognition*. In the latter case, the system must be trained using a number of speakers and the system must be able to generalize and recognize words spoken by people outside the training population.

At the heart of any IWR system are a set of known reference patterns and a distance measure (recall the footnote in Section 8.2). Recognition of an unknown

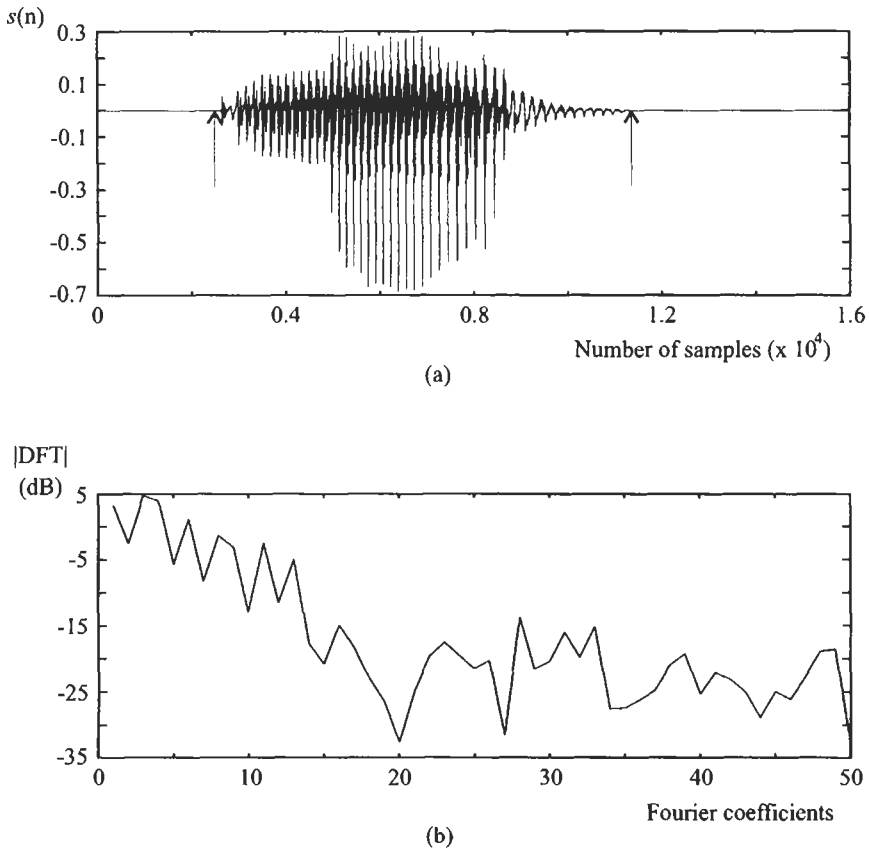


FIGURE 8.4: Plots of (a) the time sequence corresponding to the word "love" and (b) the magnitude of the DFT, in dB, for one of its frames.

test pattern is achieved by searching for the best match between the test and each of the reference patterns, on the basis of the adopted measure.

Figures 8.4a and 8.5a show the plots of two time sequences resulting from the sampling of the word "love," spoken twice by the same speaker. The samples were taken at the output of a microphone at a sampling rate of 22050 Hz. Although it is difficult to describe the differences, these are readily noticeable. Moreover, the two spoken words are of different duration. The arrows indicate (approximately) the intervals in which the spoken segments lie. The intervals outside the arrows correspond to silent periods. Specifically, the sequence in Figure 8.5a is 0.4 seconds long, and the sequence in Figure 8.4a is 0.45 seconds long. Furthermore, it is

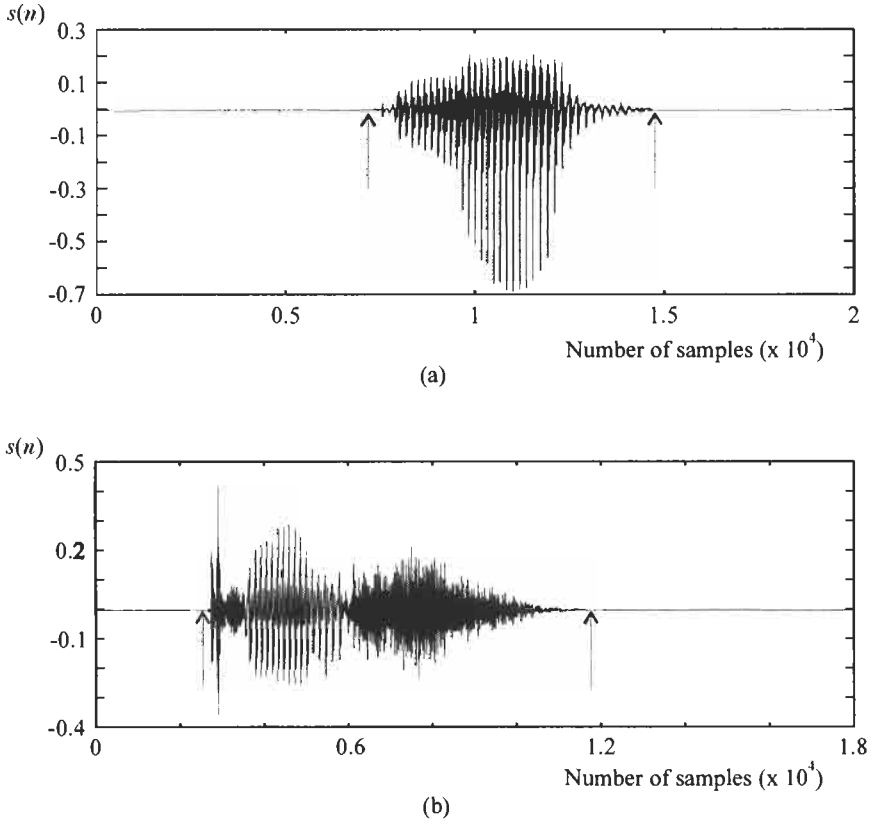


FIGURE 8.5: Plots of the time sequences resulting from the words (a) "love" and (b) "kiss," spoken by the same speaker.

important to say that this is not the result of a simple linear time scaling. On the contrary, a *highly nonlinear mapping* is required to obtain a match between these two "*same*" words spoken by the *same* person. For comparison, Figure 8.5b shows the plot of the time sequence corresponding to another word, "kiss," spoken by the same speaker.

We will resort to dynamic programming techniques to unravel the nonlinear mapping (warping) required to achieve the optimal match between a test and a reference pattern. To this end, we must first express the spoken words as sequences (strings) of appropriate feature vectors, $\boldsymbol{r}(i), i = 1, \ldots, I$, for the reference pattern and $\boldsymbol{t}(j), j = 1, \ldots, J$, for the test one. Obviously, there is more than one way to choose the feature vectors. We will focus on Fourier transform features.

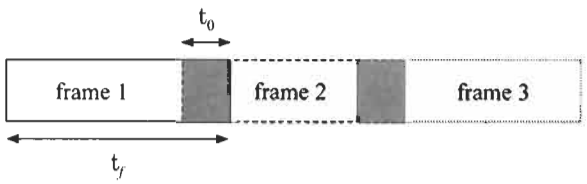


FIGURE 8.6: Successive overlapping frames for the computation of the DFT feature vectors.

Each of the time sequences involved is divided into successive overlapped time frames. In our case each frame is chosen to be $t_f = 512$ samples long and the overlap between successive frames is $t_0 = 100$ samples, as shown in Figure 8.6. The resulting number of frames for the speech segment shown in Figure 8.4a is $I = 24$, and for the other two they are $J = 21$ (Figure 8.5a) and $J = 23$ (Figure 8.5b), respectively. We assume that the former is the reference pattern and the latter two the test patterns. Let $x_i(n), n = 0, \ldots, 511$, be the samples for the ith frame of the reference pattern, with $i = 1, \ldots, I$. The corresponding DFT is given as

$$X_i(m) = \frac{1}{\sqrt{512}} \sum_{n=0}^{n=511} x_i(n) \exp\left(-j\frac{2\pi}{512}mn\right), \quad m = 0, \ldots, 511$$

Figure 8.4b shows the magnitude of the DFT coefficients for one of the I frames of the reference pattern. The plot is a typical one for speech segments. The magnitude of the higher DFT coefficients is very small with little contribution to the signal. This justifies the use of the first l DFT coefficients as features, where usually $l \ll t_f$. In our case $l = 50$ was considered to be sufficient. Thus, the vector sequence becomes

$$\boldsymbol{r}(i) = \begin{bmatrix} X_i(0) \\ X_i(1) \\ \vdots \\ X_i(l-1) \end{bmatrix}, \quad i = 1, \ldots, I \tag{8.3}$$

The feature vectors $\boldsymbol{t}(j)$, for each of the test patterns, are formed in a similar way. The choice of the DFT coefficients as features is just one of various possibilities that have been suggested and used over the years. Other popular alternatives include the parameters from an AR modeling of the speech segment, the *ceptral coefficients* (inverse DFT of the logarithm of the magnitude of the DFT coefficients), and so on (e.g., [Davi 80, Dell 93]). Having completed the preprocessing and feature selection, the reference and test patterns are expressed as (ordered) sequences

of feature vectors $\boldsymbol{r}(i)$ and $\boldsymbol{t}(j)$. Our goal becomes to compute the best match among the frames of the test and reference patterns. That is, the test pattern will be *stretched in time* (one test frame corresponds to more than one frame of the reference patterns) or *compressed in time* (more than one test frame corresponds to one frame of the reference pattern). This optimal alignment of the vectors in the two string patterns will take place via the dynamic programming procedure. To this end, we first locate the vectors of the reference string along the abscissa and those of the test pattern along the ordinate. Then, the following need to be determined:

- Global constraints
- Local constraints
- End point constraints
- The cost d for the transitions.

Various assumptions can be adopted for each of these, leading to different results with relative merits. In the sequel, we will focus on some widely used cases.

End Point Constraints. In our example we will look for the optimal complete path that starts at $(0, 0)$ and ends at (I, J) and whose first transition is to the node $(1, 1)$. Thus, it is implicitly assumed that the end points of the speech segments (i.e., $\boldsymbol{r}(1), \boldsymbol{t}(1)$ and $\boldsymbol{r}(I), \boldsymbol{t}(J)$) match to a fair degree. These can be the vectors resulting from the silent periods just before and just after the speech segments, respectively. A simple variation of the complete path constraints results if the end points of the path are not specified a priori and are assumed to be located within a distance ϵ from points $(1, 1)$ and (I, J), and it is left to the optimizing algorithm to locate them.

Global Constraints. The global constraints define the region of nodes that are searched for the optimal path. Nodes outside this region are not searched. Basically, the global constraints define the overall stretching or compression allowed for the matching procedure. An example is shown in Figure 8.7. They are known as *Itakura constraints* and impose a maximum factor of 2 for any expansion or compression of the test with respect to the reference pattern. The allowable nodes are then located within the parallelogram shown in Figure 8.7 by the solid line. The dashed line corresponds to the same global constraints when the relaxed end point constraints, mentioned before, are adopted. Observe from the figure that paths across the sides of the parallelogram compress or expand corresponding frame intervals by a factor of 2, and this is the maximum possible factor attained. This is usually a reasonable constraint, and at the same time it reduces the number of nodes to be searched for the optimal path substantially. If $I \approx J$, then it is not difficult to show that the number of grid points to be searched is reduced by approximately one third.

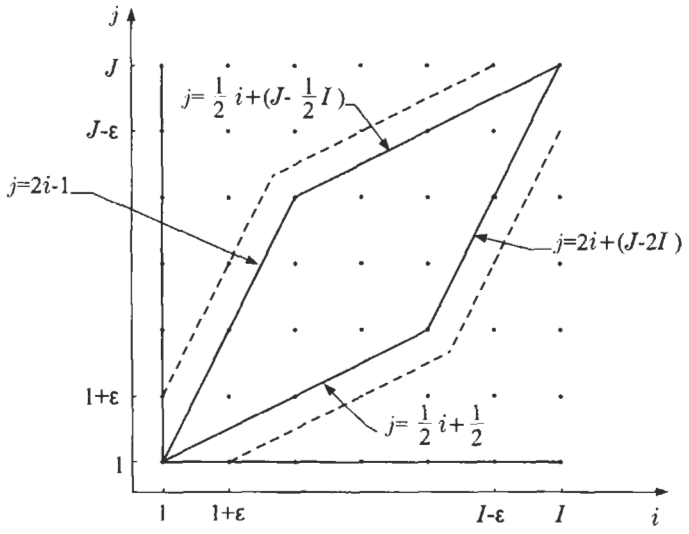


FIGURE 8.7: Itakura global constraints. The maximum compression/expansion factor is 2, and it determines the slope of the boundary line segments.

Local Constraints. These constraints define the set of predecessors and the allowable transitions to a given node of the grid. Basically, they impose limits for the maximum expansion/compression rates that successive transitions can achieve. A property that any local constraint must satisfy is *monotonicity*. That is,

$$i_{k-1} \leq i_k \quad \text{and} \quad j_{k-1} \leq j_k$$

In other words, all predecessors of a node are located to its left and south. This guarantees that the matching operation follows the natural time evolution and avoids confusing, for example, the word "from" with the word "form."

Two examples of nonmonotonic paths are shown in Figure 8.8. A popular set of local constraints, known as the Itakura constraints [Itak 75], is shown in Figure 8.9. The maximum achievable expansion (compression) rate over a local path is measured by the associated *slope*, which is defined as the *maximum* ratio of the total change Δi, in the ith direction, to the total change Δj, in the jth direction, over the local path. The slope for the Itakura constraints is 2, and it is the result of a (repetitive) transition of the type $(i - 1, j - 2)$ to (i, j). Another notable characteristic of the Itakura constraints is that horizontal transitions are allowed, but *not successively*, and this is indicated by the cross over the arrow. Thus, Itakura constraints do not allow long horizontal paths, corresponding to ∞ slopes. Finally, these constraints allow the path to skip at *most one* feature vector in the test pattern

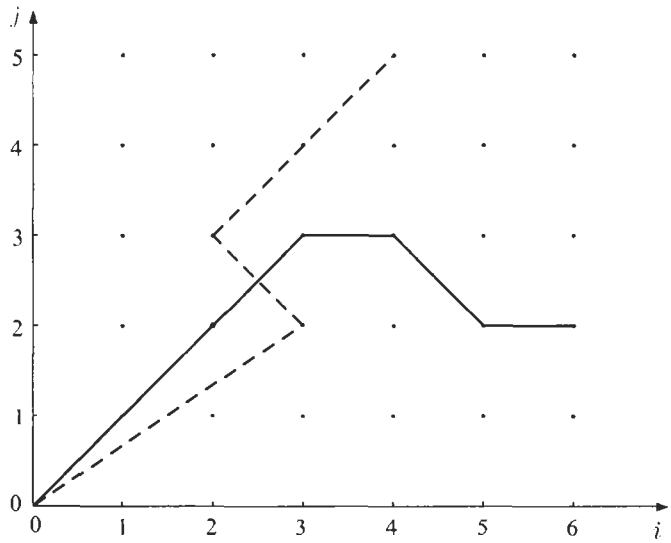


FIGURE 8.8: Examples of nonmonotonic paths. Such paths are not allowed and are not considered in the search for the optimum.

string, that is, the one at the $j-1$ position of the ordinate axis, and the path jumps from $(i-1, j-2)$ to (i, j). In contrast, feature vectors in the reference string are not skipped and all take part in the optimal path. Such constraints are known as *asymmetrical*.

A number of alternative local constraints have also been suggested and used in practice. Figure 8.10 shows four different types of constraints considered by Sakoe and Chiba [Sako 78]. For the type in Figure 8.10a there is no limit in the rate of expansion/compression, since successive horizontal or vertical transitions can take

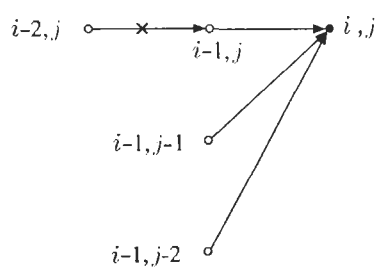


FIGURE 8.9: The Itakura local constraints. Two successive horizontal transitions are not allowed.

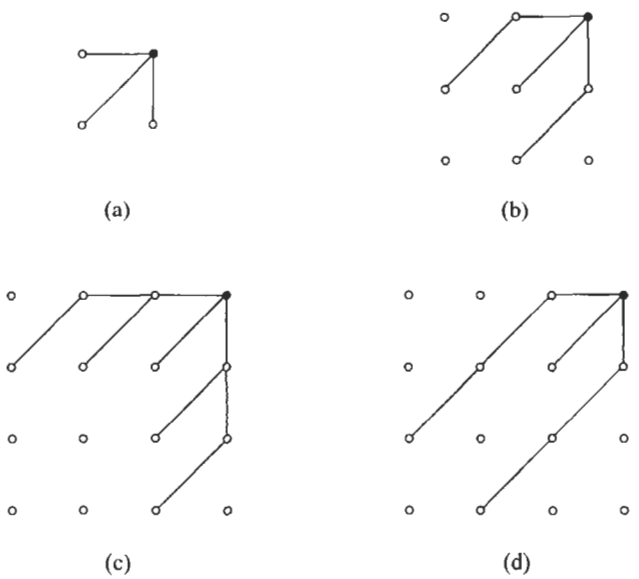


FIGURE 8.10: The Sakoe and Chiba local constraints.

place, until of course one falls outside the region defined by the global constraints. In contrast, in Figure 8.10b horizontal (vertical) transitions are allowed *only after* a diagonal transition and in Figure 8.10d after two successive diagonal transitions. In Figure 8.10c, at most two successive horizontal (vertical) transitions are allowed only after a diagonal one. The slopes for each of the constraints in Figure 8.10a, b, c and d are ∞, 2, 3, and 3/2, respectively (Problem 8.2). For a more detailed treatment of the topic the interested reader may consult [Dell 93, Silv 90, Myer 80].

The Cost. A commonly used cost, which we will also adopt here, is the Euclidean distance between $\boldsymbol{r}(i_k)$ and $\boldsymbol{t}(j_k)$, corresponding to node (i_k, j_k), that is,

$$d(i_k, j_k | i_{k-1}, j_{k-1}) = \|\boldsymbol{r}(i_k) - \boldsymbol{t}(j_k)\| \equiv d(i_k, j_k)$$

In this, we assume that no cost is associated with the transitions to a specific node, and the cost depends entirely on the feature vectors corresponding to the respective node. Other costs have also been suggested and used [Gray 76, Gray 80]. More recently ([Pikr 02], [Pikr 03]) used a cost that accounts for the most commonly encountered errors (e.g., different player's style) in the context of music recognition. Finally, it must be stated that often a normalization of the overall cost D is carried out. This is to compensate for the difference in the path lengths, offering "equal" opportunities to all of them. A logical choice is to divide the overall cost D by the length of each path [Myer 80].

The resulting overall costs for the two test patterns of Figure 8.5, against the reference pattern of Figure 8.4, using the Itakura constraints, were $D = 11.473$, $D = 25.155$, respectively. Thus, the overall cost for the word "love" is lower than the overall cost for the word "kiss," and our procedure has recognized the spoken word correctly. The resulting normalized overall costs after dividing by the number of nodes along each path were 0.221 and 0.559, respectively.

8.3 MEASURES BASED ON CORRELATIONS

The major task to be addressed in this section can be summarized as follows: "Given a block of recorded data, find *whether* a specific known (reference) pattern is contained within the block and *where* it is located." A typical application of this is found in scene analysis, when we want to search for specific objects within the image. Such problems arise in many applications, such as target detection, robot vision, and video coding. For example, in video coding a major step is that of *motion estimation*, that is, the process of locating *corresponding pixels* (i.e., the same moved object) among successive image frames at different time instants. This step is then followed by the *motion compensation* stage, which compensates for the displacement of moving objects from one frame to another. One then codes the frame difference

$$e(i, j, t) = r(i, j, t) - r(i - m, j - n, t - 1)$$

where $r(i, j, t)$ are the pixel gray levels of the image frame at time t and $r(i - m, j - n, t - 1)$ the *corresponding* pixel values at spatial locations $i - m$, $j - n$ of the previous frame at time $t - 1$. In this way, we code only the *new information* contained at the latest frame, avoiding redundancies.

Let us assume that we are given a reference pattern expressed as an $M \times N$ image array $r(i, j)$, $i = 0, \ldots, M - 1$, $j = 0, \ldots, N - 1$, and an $I \times J$ image array $t(i, j)$, $i = 0, \ldots, I - 1$, $j = 0, \ldots, J - 1$, where $M \leq I$, $N \leq J$. The goal is to develop a measure for detecting an $M \times N$ subimage within $t(i, j)$ that matches best the reference pattern $r(i, j)$. To this end, the reference image $r(i, j)$ is superimposed on the test image and *it is translated* to all possible positions (m, n) within it. For each of the points (m, n), the mismatch between $r(i, j)$ and the $M \times N$ subimage of $t(i, j)$ is computed according to

$$D(m, n) = \sum_{i=m}^{m+M-1} \sum_{j=n}^{n+N-1} |t(i, j) - r(i - m, j - n)|^2 \tag{8.4}$$

Template matching is conducted by searching for the location (m, n) for which $D(m, n)$ is minimum. Let us now give this a computationally more attractive form.

Equation (8.4) is equivalent to

$$D(m,n) = \sum_i \sum_j |t(i,j)|^2 + \sum_i \sum_j |r(i,j)|^2 - 2\sum_i \sum_j t(i,j)r(i-m, j-n) \tag{8.5}$$

The second summand is constant for a given reference pattern. *Assuming that the first one does not change much across the image*, that is, there is not much variation of the pixel gray levels over the test image, the minimum of $D(m,n)$ is achieved when

$$c(m,n) = \sum_i \sum_j t(i,j)r(i-m, j-n) \tag{8.6}$$

is maximum for all possible locations (m,n). The quantity $c(m,n)$ is nothing other than a cross-correlation sequence between $t(i,j)$ and $r(i,j)$ computed at the point (m,n). In cases for which the assumption of little gray level variation is not valid, this measure is very sensitive to gray level variations within $t(i,j)$. In such cases the *cross-correlation coefficient*, defined as

$$c_N(m,n) = \frac{c(m,n)}{\sqrt{\sum_i \sum_j |t(i,j)|^2 \sum_i \sum_j |r(i,j)|^2}} \tag{8.7}$$

is a more appropriate measure. Here $c_N(m,n)$ is a normalized version of $c(m,n)$ and variations in $t(i,j)$ tend to cancel out. Recall now the Cauchy–Schwarz inequality

$$\left| \sum_i \sum_j t(i,j)r(i-m, j-n) \right| \leq \sqrt{\sum_i \sum_j |t(i,j)|^2 \sum_i \sum_j |r(i,j)|^2}$$

Equality holds *if and only if*

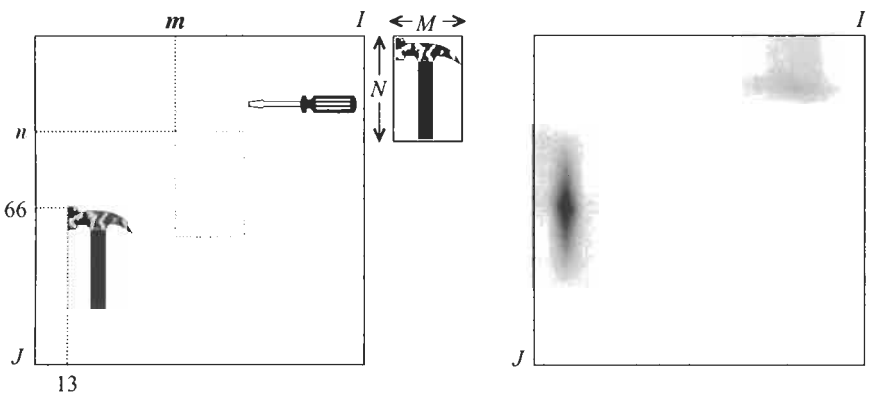
$$t(i,j) = \alpha r(i-m, j-n), \quad i = m, \ldots, m+M-1 \text{ and } j = n, \ldots, n+N-1$$

with α being an arbitrary constant. Hence, $c_N(m,n)$ is always less than unity and achieves its maximum value of one only if the (test) subimage is the same (within a scaling factor) as the reference pattern.

In our discussion so far, we have assumed that the reference pattern has only been translated within $t(i,j)$ and no rotation or scaling has been involved. In applications such as video coding, this is a valid assumption and it has been adopted in

the video coding standards [Bhas 95]. However, this is not always the case and the technique has to be modified. One way is to describe the reference and test subimages in terms of invariant moments and measure the similarity using correlations involving these moments [Hall 79] (Problem 8.4). Another rotation- and scale-invariant technique, using a combination of the Fourier and Mellin transforms, is described in [Scha 89]. This technique tries to exploit the translation invariance of the magnitude of the Fourier transform (already discussed in Chapter 7) and the scale invariance of the Mellin transform ([Ravi 95], Problem 8.5). Another path, which of course demands high computational resources, is to have a set of distorted (e.g., rotated and scaled) reference templates to cover all possibilities. Correlation matching will then reveal the best match between a test pattern and one of the reference templates. A computationally more attractive technique is to employ the Karhunen–Loève transform [Ueno 97]. The main idea is that rotated templates are highly correlated, and each of them can be approximated by its projection onto a lower dimension eigenspace, using the most significant eigenvectors of their correlation matrix. Matching of an unknown pattern with the template of the right orientation is performed in the lower dimensional space, leading to substantial computational savings.

Example 8.1. The image $t(i, j)$ in Figure 8.11a contains two objects, a screwdriver and a hammer. The latter is the object that we want to search for in the image. The reference image is shown at the top right corner of Figure 8.11a. The dotted area represents the general (m, n) position of the reference image when it is superimposed on the test one. Figure 8.11b shows the cross-correlation $c(m, n)$ between the two images. We readily observe that the maximum (black) occurs at the position (13, 66), that is, where the hammer is located in $t(i, j)$.



(a) (b)

FIGURE 8.11: Example of (a) reference and test images and (b) their respective correlation.

Computational Considerations

- In some cases, it is more efficient to compute the cross-correlation via its Fourier transform. Recall that in the frequency domain (8.6) is written as

$$C(k, l) = T(k, l)R^*(k, l) \tag{8.8}$$

where $T(k, l)$, $R(k, l)$ are the DFT transforms of $t(i, j)$ and $r(i, j)$, respectively, with "$*$" denoting complex conjugation. Of course, in order to write (8.8) both images must be of the same size. If they are not, which is usually the case, a number of zeros must be appended to extend the smaller sized image. $c(m, n)$ is obtained via the inverse DFT of $C(k, l)$. Taking into account the computational efficiency of the FFT, this procedure may lead to savings depending, of course, on the relative size of M, N and I, J.

- A major computational load in correlation-based template matching is searching over the pixels of $t(i, j)$ in order to locate the maximum correlation. Usually, the search is restricted within a rectangle $[-p, p] \times [-p, p]$ centered at a point (x, y) in $t(i, j)$. For example, in video coding, if the $M \times N$ block is centered at a position (x, y) in the frame at time $t - 1$, then the current frame is searched within $(x \pm p, y \pm p)$. The value of p depends on the application. For broadcast TV $p = 15$ is sufficient. For sporting events (high motion) $p = 63$ is more appropriate. Thus, an exhaustive search for the maximum of $c(m, n)$, defined in (8.6), will require a number proportional to $(2p + 1)^2MN$ additions and multiplications. This leads to a huge number of operations indeed (Problem 8.6). Thus, in practice, suboptimal heuristic searching techniques are usually adopted, which, although they do not guarantee locating the maximum, reduce the required number of operations substantially. There are two major directions. One is to reduce the search points and the other is to reduce the size of the involved images.

Two-Dimensional Logarithmic Search

Logarithmic Search Figure 8.12 shows the rectangular $[-p, p] \times [-p, p]$ searching area for the case of $p = 7$. The center of the rectangle is assumed to be the point $(0, 0)$. The cross-correlation computation is first performed at the center as well as the eight points located on the perimeter of the inner $[-p/2, p/2] \times [-p/2, p/2]$ area ($p/2$ rounded to an integer). These points are denoted by a square. The spacing between these points is $d_1 = 4$ pixels, that is, $d_1 = 2^{k-1}$ and $k = \lceil \log_2 p \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the first larger integer. For $p = 7, k = 3$ and $d_1 = 4$. We will demonstrate the procedure via an example. Let us assume that the largest cross-correlation value resulted at the position $(-4, 0)$ (shaded square). Then we consider this point as the center of a rectangle of size $[-p/4, p/4] \times [-p/4, p/4]$ ($[-2, 2] \times [-2, 2]$ in our case) and compute the correlation at the eight points

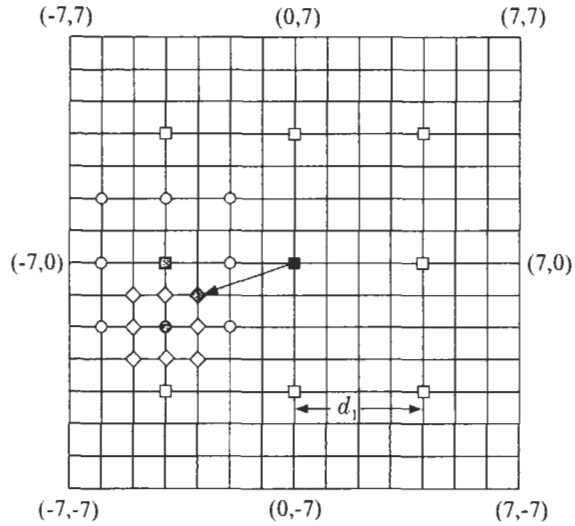


FIGURE 8.12: Logarithmic search to find the point of maximum cross-correlation.

of its perimeter. These points are denoted by a circle and the spacing between them is now $d_2 = d_1/2$ (2). The process is repeated and finally the computation is performed on the eight (diamond) points on the perimeter of the rectangle of size $[-1, 1] \times [-1, 1]$, which is centered at the optimum (of the previous step) shaded circle point. The spacing between the diamond points is $d_3 = 1$. The shaded diamond corresponds to the point with the maximum cross-correlation and the process is terminated. The number of computations has now been reduced to $MN(8k + 1)$ operations, which is a substantial saving compared with the exhaustive search.

A variant of the two-dimensional logarithmic search is to search the i and j directions independently. The point whose coordinates are the resulting best values of i and j becomes the new origin of the coordinate system, and the search is repeated in the new i, j directions, with smaller spacing d. The process is repeated until the spacing becomes unity.

Hierarchical Search

This technique springs from the multiresolution concept considered in Chapter 6. Let us again consider an example.

- Step 1: A reference block of, say, 16×16 is given and the search area is assumed to be the rectangle $[-p, p] \times [-p, p]$, centered at the point (x, y) in the test image. We refer to level 0 versions of the images. Both the reference block and the test image are low-pass filtered and subsampled by 2, resulting

in their level 1 versions. The total number of pixels in the level 1 versions has been reduced by 4. Level 1 versions are in turn low-pass filtered and subsampled, resulting in level 2 versions. In general, this process can continue.

- Step 2: At level 2 the search for the maximum takes place with the 4×4 low-pass version of the reference block. The search area in the level 2 low-pass version of the test image is the rectangle $[-p/4, p/4] \times [-p/4, p/4]$ centered at $(x/4, y/4)$. Either a full or a logarithmic search can be employed. Let (x_1, y_1) be the coordinates of the optimum, with respect to $(x/4, y/4)$.
- Step 3: At level 1, the search for the maximum is performed using the corresponding 8×8 version of the reference block. The search area, within the level 1 version of the test image, is the rectangle of size $[-1, 1] \times [-1, 1]$ centered at $(x/2+2x_1, y/2+2y_1)$, that is, nine pixels in total. This is because the eight pixels at the perimeter of this area were not involved at level 2, due to the subsampling (see Figure 6.16 of Chapter 6). The center point must also be included in order to have a fair comparison at this level for the search for the maximum. Let the maximum occur at (x_2, y_2) with respect $(x/2, y/2)$.
- Step 4: At level 0 the search is performed using the original reference template, within the area of size $[-1, 1] \times [-1, 1]$ centered at $(x+2x_2, y+2y_2)$ in the test image. The location of the maximum is the final one and the process terminates.

The computational saving with this method depends on the number of levels, as well as the type of search adopted at the highest level (Problem 8.5). In general, hierarchical methods are very efficient from a computational point of view. This is gained at the expense of increased memory requirements, due to the various image versions that must be kept. A drawback of the method is that if small objects are present in the templates, they may disappear at the lowest resolution images due to the subsampling. Furthermore, the method cannot guarantee to find the global best match. In [Alkh 01] an alternative philosophy is suggested that results in the global best match. Computational savings are achieved by pruning the number of candidates for the best match in a level, using the results in a higher level and an appropriately chosen threshold value.

Sequential Method

A number of other techniques have also been suggested. For example, the *sequential search* method computes a variant of (8.4) directly. Specifically, define

$$D_{pq}(m, n) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} |t(i+m, j+n) - r(i, j)| \tag{8.9}$$

Thus, the error is computed in a smaller and sequentially increasing window area, for $p, q = 1, 2, \ldots$ and $p \leq M, q \leq N$. The computations stop when $D_{pq}(m, n)$

becomes larger than a predetermined threshold. Then computations start in a different direction (m, n). Hence, saving is achieved, because for bad positions only a small number of computations need to be performed.

8.4 DEFORMABLE TEMPLATE MODELS

In the previous section we considered the problem of searching for a known reference pattern (template) within a test image. We assumed that the template and the object, residing in the image, were identical. The only differences that were allowed to enter into our discussion were those imposed by a different orientation and/or scaling. However, there are many problems where we know a priori that the available template and the object we search for in the image may not look exactly the same. This may be due to varying imaging conditions, occlusion, and imperfect image segmentation. Furthermore, in a content-based image data-base retrieval system, the user may provide the system with a sketch of the shape of the object to be retrieved. Obviously, the sketch will not match exactly the corresponding object in the database images. Our goal in this section is to allow the template matching procedure to account for deviations between the reference template and the corresponding test pattern in the image. In our discussion, we will assume that the reference template is available in the form of an image array containing the object's boundary information (contour). That is, we will focus on shape information only. Extensions incorporating more information, e.g., texture, are also possible.

Let us denote the reference template image array as $r(i, j)$. This is also known as *prototype*. The basic idea behind the *deformable template matching* procedure is simple: *Deform* the prototype and produce *deformed* variants of it. From a mathematical point of view a deformation consists on the application of a *parametric transform* T_{ξ} on $r(i, j)$ to produce a deformed version $T_{\xi}[r(i, j)]$. Different values of the vector parameter ξ lead to different versions. From the set of the deformed prototype variants that can be generated, there will be one that "best" matches the test pattern. The goodness of fit is measured via a cost, which we will call the *matching energy* $E_m(\xi)$. Obviously, the goal is to choose ξ so that $E_m(\xi)$ is *minimum*. However, this is not enough. If, for example, the optimal set of parameters is such that the corresponding deformed template has been deformed to such an extent that it bears little resemblance to the original prototype, the method would be meaningless. Thus, one more term has to be taken into account in the optimization process. This is the cost measuring the "deformation" which the prototype needs to undergo in order to fit the test pattern. We will call this term of the cost as *deformation energy* $E_d(\xi)$. Then the optimal vector parameter is computed so that

$$\xi : \min_{\xi}\{E_m(\xi) + E_d(\xi)\} \tag{8.10}$$

In words, one could think of the boundary curve of the prototype as made by rubber. Then with the help of a pencil we deform the shape of the rubber curve to match the test pattern. The more we deform the shape of the prototype, the higher the energy we have to spend for it. This energy, quantified by $E_d(\xi)$, depends on the shape of the prototype. That is, it is an intrinsic property of the prototype and this is the reason that it is also known as *internal energy*. The other energy term, $E_m(\xi)$, depends on the input data (test image) and we usually refer to it as the *external energy*. The optimal vector parameter, ξ, is chosen so that the best trade-off between these two energy terms is achieved. Sometimes, a weighting factor C is used to give preference to one of the two terms and ξ is computed so that

$$\xi : \min_{\xi}\{E_m(\xi) + C E_d(\xi)\} \tag{8.11}$$

Hence, in order to apply the above procedure in practice one must have at his/her disposal the following ingredients:

- A prototype
- A transformation procedure to deform the prototype
- The two energy function terms

Choice of the Prototype

This should be carefully chosen so that it is a (typical) representative of the various instances in which this object is expected to appear in practice. In a way, the prototype should encode the "mean shape" characteristics of the corresponding "shape class."

Deformation Transformation

This consists of a set of parametric operations. Let (x, y) be the (continuous) coordinates of a point in a 2D image. Without loss of generality, assume that the image is defined in a square $[0, 1] \times [0, 1]$. Then each point (x, y) is mapped using a continuous mapping function, as

$$(x, y) \longrightarrow (x, y) + (D^x(x, y), D^y(x, y)) \tag{8.12}$$

For discrete image arrays a quantization step is necessary after the transformation. Different functions can be used to perform the above mapping. A set which has successfully been used in practice is ([Amit 91])

$$D^x(x, y) = \sum_{m=1}^{M}\sum_{n=1}^{N} \xi_{mn}^x e_{mn}^x(x, y) \tag{8.13}$$

$$D^y(x, y) = \sum_{m=1}^{M}\sum_{n=1}^{N} \xi_{mn}^y e_{mn}^y(x, y) \tag{8.14}$$

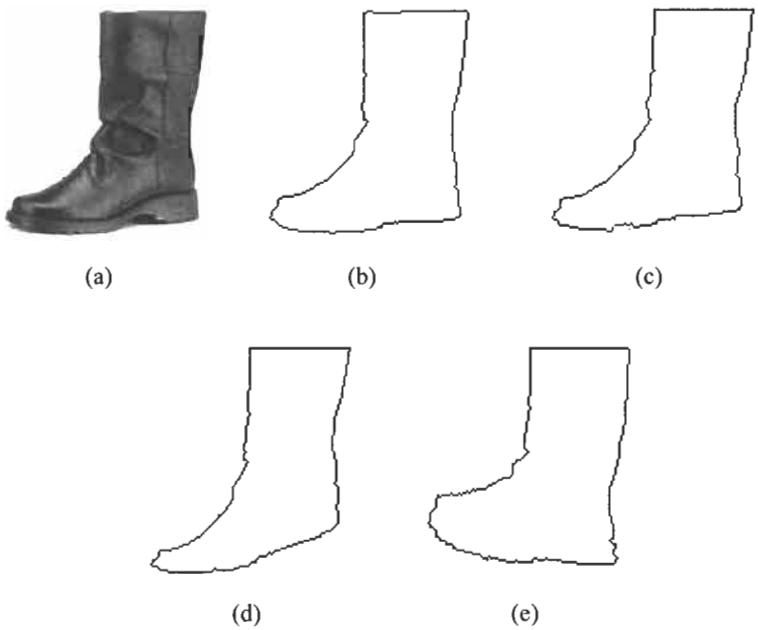


FIGURE 8.13: (a) A reference pattern, (b) its contour used as prototype, and (c), (d), (e) three of its deformed variants.

$$e^x_{mn}(x, y) = \alpha_{mn} \sin \pi n x \cos \pi m y \tag{8.15}$$

$$e^y_{mn}(x, y) = \alpha_{mn} \cos \pi m x \sin \pi n y \tag{8.16}$$

for appropriately chosen values of M, N. The normalizing constants α_{mn} can be taken as

$$\alpha_{mn} = \frac{1}{\pi^2(n^2 + m^2)}$$

Other basis functions can also be used, such as splines or wavelets. Figure 8.13 shows a prototype for an object and three deformed versions obtained for the simplest case of the transformation model in (8.13)–(8.16), i.e., $M = N = 1$.

Internal Energy

This should be minimum for no deformation, i.e., for $\boldsymbol{\xi} = \mathbf{0}$. A reasonable choice, associated with the transformation functions considered above, is

$$E_d(\boldsymbol{\xi}) = \sum_m \sum_n ((\xi^x_{mn})^2 + (\xi^y_{mn})^2) \tag{8.17}$$

External Energy

Here again a number of choices are possible, measuring the goodness of fit between the test pattern and each one of the deformed template variants. For example, for a specific position, orientation, and scale of a deformed template, this energy term can be measured in terms of the distance of each point in the contour of the deformed template from the nearest point in the contour of the test image, I. One way to achieve this is via the following energy function:

$$E_m(\boldsymbol{\xi}, \boldsymbol{\theta}, I) = \frac{1}{N_d} \sum_{i,j} (1 + \Phi(i, j)) \tag{8.18}$$

where $\boldsymbol{\theta}$ is the vector of the parameters defining the position, orientation, and scaling and N_d the number of contour pixels of the corresponding deformed template and

$$\Phi(i, j) = -\exp\left(-\rho(\delta_i^2 + \delta_j^2)^{1/2}\right) \tag{8.19}$$

where ρ a constant and (δ_i, δ_j) is the displacement of the (i, j) pixel of the deformed template from the nearest point in the test image. In [Jain 96] directional information is also incorporated into the cost.

Remarks

- One can arrive at (8.10) in a more systematic way via probabilistic arguments, that is, by seeking to maximize the aposteriori probability density of $(\boldsymbol{\xi}, \boldsymbol{\theta})$ given the test image array, i.e.,

$$p(\boldsymbol{\xi}, \boldsymbol{\theta} | I) = \frac{p(\boldsymbol{\xi}, \boldsymbol{\theta}) p(I | \boldsymbol{\xi}, \boldsymbol{\theta})}{p(I)} \tag{8.20}$$

  where the Bayes rule has been employed. In this framework, (8.17) results if one assumes that the various parameters ξ_{mn}^x, ξ_{mn}^y are statistically independent and are normally distributed, e.g., $p(\xi) = \mathcal{N}(0, \sigma^2)$. The higher the variance σ^2 the wider the range of the values that occur with high probability. To obtain (8.18)–(8.19) the model

$$p(I | \boldsymbol{\xi}, \boldsymbol{\theta}) = \alpha \exp\left(-E_m(\boldsymbol{\xi}, \boldsymbol{\theta}, I)\right) \tag{8.21}$$

  is adopted, where α is a normalizing constant [Jain 96].
- The cost in (8.10) is a nonlinear function and its minimization can be achieved via any nonlinear optimization scheme. Besides complexity, the major drawback is the omnipresent danger that the algorithm will be trapped in a local minimum. In [Jain 96] a procedure has been suggested that builds around

the gradient descent methodology (Appendix C.) The idea is to adopt a multiresolution iterative approach and use larger values of ρ in (8.19) for the coarser levels and smaller values for the finer ones. This procedure seems to help the algorithm to avoid local minima, at an affordable computing cost.

- The methodology we described in this section belongs to a more general class of deformable template matching techniques, which builds around an available prototype. This is not the only available path. Another class of deformable models stems from an analytic description of the prototype shape, using a set of parameterized geometrical shapes, e.g., ellipses or parabolic curves. To delve deeper into these issues the reader may refer to the review articles [Jain 98, McIn 96, Cheu 02] and the references therein. [Widr 73, Fisc 73] seem to be the first attempts to introduce the concept of deformable models in computer vision.
- In the pattern recognition context, given an unknown test pattern, we seek to see to which one from a known set of different prototypes this matches best. For each prototype, the best deformed variant is selected, based on the minimum energy cost. The prototype that wins is the one whose best deformed variant results in the overall minimum energy cost.

Problems

8.1 Find the Edit distance between the word "poem" and its misspelled version "poten." Draw the optimal path.

8.2 Derive the slopes for the Sakoe–Chiba constraints and draw paths that achieve the maximum expansion/compression rates as well as paths corresponding to intermediate rates.

8.3 Develop a computer program for dynamic time warping, for a complete optimum path and the Itakura constraints. Verify the algorithm, using speech segments available from the book's web site.

8.4 Let the seven Hu moments of the reference $M \times N$ image block be ϕ_i, $i = 1, 2, \ldots, 7$. Also, denote by $\psi_i(m, n)$, $i = 1, 2, \ldots, 7$, the respective moments resulting from the test subimage located at (m, n). Explain why

$$\mathcal{M}(m, n) = \frac{\sum_{i=1}^{7} \phi_i \psi_i(m, n)}{\left(\sum_{i=1}^{7} \phi_i^2 \sum_{i=1}^{7} \psi_i^2(m, n)\right)^{1/2}}$$

is a reasonable measure of similarity.

8.5 Show that the Mellin transform $M(u, v)$ of a function $f(x, y)$, defined as

$$M(u, v) = \iint f(x, y) x^{-ju-1} y^{-jv-1} \, dx \, dy$$

is scale invariant.

8.6 For the motion compensation step in a video coding application the image frame of size $I \times J$ is divided into subblocks of size $M \times N$. For each of the subblocks the search for its corresponding one, in the current frame, is restricted in an area of size $[-p, p] \times [-p, p]$. Find the required number of operations per second for the computation of the maximum cross-correlation for a full search, a two-dimensional and a one-dimensional (independent i, j) logarithmic search, and a hierarchical search of three levels. Typical values for broadcast TV are $M = N = 16$, $I = 720$, $J = 480$, and $p = 15$. The number of frames transmitted per second is $f = 30$.

References

[Alkh 01] Ghavari-Alkhansavi M. "A fast globally optimal algorithm for template matching using low resolution pruning," *IEEE Transactions on Image Processing*, Vol. 10(4), pp. 526–533, 2001.

[Amit 91] Amit Y., Grenander U., Piccioni M. "Structural image restoration through deformable template," *J. Amer. Statist. Association*, Vol. 86(414), pp. 376–387, 1991.

[Bell 57] Bellman R.E. *Dynamic Programming*, Princeton University Press, 1957.

[Bhas 95] Bhaskaran V., Konstantinides K. *Image and Video Compression Standards*, Kluwer Academic Publishers, 1995.

[Cheu 02] Cheung K.-W., Yeung D.-Y., Chin R.T. "On deformable models for visual pattern recognition," *Pattern Recognition*, Vol. 35, pp. 1507–1526, 2002.

[Dame 64] Damerau F.J. "A technique for computer detection and correction of spelling errors," *Commun. ACM*, Vol. 7(3), pp. 171–176, 1964.

[Davi 80] Davis S.B., Mermelstein P. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 28(4), pp. 357–366, 1980.

[Dell 93] Deller J., Proakis J., Hansen J.H.L. *Discrete-Time Processing of Speech Signals*, Macmillan, 1993.

[Desh 99] Deshmukh N., Ganapathirajn A., Picone J. "Hierarchical search for large vocabulary conversational speech recognition," *IEEE Signal Processing Magazine*, Vol. 16(5), pp. 84–107, 1999.

[Fisc 73] Fischler M., Elschlager R. "The representation and matching of pictorial structures," *IEEE Transations on Computers*, Vol. 22(1), pp. 67–92, 1973.

[Gray 76] Gray A.H., Markel J.D. "Distance measures for speech processing," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 24(5), pp. 380–391, 1976.

[Gray 80] Gray R.M., Buzo A., Gray A.H., Matsuyama Y. "Distortion measures for speech processing," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 28(4), pp. 367–376, 1980.

[Hall 79] Hall E. *Computer Image Processing and Recognition*, Academic Press, 1979.

[Itak 75] Itakura F. "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 23(2), pp. 67–72, 1975.

[Jain 96] Jain A.K., Zhong Y., Lakshmanan S. "Object matching using deformable templates," *IEEE Transactions on Pattern Analysis and Machine Itelligence*, Vol. 18(3), pp. 267–277, 1996.

[Jain 98] Jain A.K., Zhong Y., Dubuisson-Jolly M.P. "Deformable template models: A review," *Signal Processing*, Vol. 71, pp. 109–129, 1998.

[Leven 66] Levenshtein V.I. "Binary codes capable of correcting deletions, insertions and reversals," *Soviet phys. Dokl.*, Vol. 10(8), pp. 707–710, 1966.

[McIn 96] McInerney T., Terzopoulos D. "Deformable models in medical image analysis: A survey," *Med. Image Anal.*, Vol. 1(2), pp. 91–108, 1996.

[Myer 80] Myers C.S., Rabiner L.R., Rosenberg A.E. "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 28(6), pp. 622–635, 1980.

[Neg 99] Neg H., Ortmanns S. "Dynamic Programming search for continuous speech recognition," *IEEE Signal Processing Magazine*, Vol. 16(5), pp. 64–83, 1999.

[Pikr 02] Pikrakis A., Theodoridis S., Kamabrotos D. "Recognition of isolated musical patterns using context dependent dynamic time warping," EUSIPCO-2002, Toulouse, France, September 3–6, 2002.

[Pikr 03] Pikrakis A., Theodoridis S., Kamarotos D. "Recognition of isolated musical patterns using context dependent dynamic time warping," *IEEE Transactions on Speech and Audio Processing*, To appear 2003.

[Ravi 95] Ravichandran G., Trivedi M.M. "Circular-Mellin features for texture segmentation," *IEEE Transactions on Image Processing*, Vol. 2(12), pp. 1629–1641, 1995.

[Sako 78] Sakoe H., Chiba S. "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 26(2), pp. 43–49, 1978.

[Scha 89] Schalkoff R. *Digital Image Processing and Computer Vision*, John Wiley, 1989.

[Silv 90] Silverman H., Morgan D.P. "The application of the dynamic programming to connected speech recognition," *IEEE Signal Processing Magazine*, Vol. 7(3), pp. 7–25, 1990.

[Seni 96] Seni G., Kripasundar V., Srihari R. "Generalizing Edit distance to incorporate domain information: Handwritten text recognition as a case study," *Pattern Recognition*, Vol. 29(3), pp. 405–414, 1996.

[Ueno 97] Uenohara M., Kanade T. "Use of the Fourier and Karhunen–Loève decomposition for fast pattern matching with a large set of templates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(8), pp. 891–899, 1997.

[Widr 73] Widrow B. "The rubber mask technique, Parts I and II," *Pattern Recognition*, Vol. 5, pp. 175–211, 1973.

CHAPTER 9

CONTEXT-DEPENDENT CLASSIFICATION

9.1 INTRODUCTION

The classification tasks considered so far assumed that no relation exists among the various classes. In other words, having obtained a feature vector $\boldsymbol{x}$ from a class ω_i, the next feature vector could belong to any other class. In this chapter we will remove this assumption and we will assume that the various classes are closely related. That is, successive feature vectors are not independent. Under such an assumption, classifying each feature vector separately from the others obviously has no meaning. The class to which a feature vector is assigned depends (a) on its own value, (b) on the values of the other feature vectors, and (c) on the existing relation among the various classes. Such problems appear in various applications such as communications, speech recognition, and image processing.

In the context-free classification, our starting point was the Bayesian classifier. In other words, a feature vector was classified to a class ω_i if

$$P(\omega_i|\boldsymbol{x}) > P(\omega_j|\boldsymbol{x}), \quad \forall j \neq i$$

The Bayesian point of view will also be adopted here. However, the dependence among the various classes sets demands for a more general formulation of the problem. The mutual information that resides within the feature vectors requires the classification to be performed using *all* feature vectors simultaneously and also to be arranged in the same sequence in which they occurred from the experiments. For this reason, in this chapter we will refer to the feature vectors as *observations* occurring in sequence, one after the other, with $\boldsymbol{x}_1$ being the first and $\boldsymbol{x}_N$ the last from a set of N observations.

9.2 THE BAYES CLASSIFIER

Let $X : \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ be a sequence of N (feature vectors) observations and ω_i, $i = 1, 2, \ldots, M$, the classes in which these vectors must be classified. Let Ω_i : $\omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_N}$ be one of the possible sequences of these classes corresponding to

the observation sequence, with $i_k \in \{1, 2, \ldots, M\}$ for $k = 1, 2, \ldots, N$. The total number of these class sequences Ω_i is M^N, that is, the number of possible *ordered* combinations of M distinct objects taken in groups of N. *Our classification task is to decide to which class sequence Ω_i a sequence of observations X corresponds.* This is equivalent to appending $\boldsymbol{x}_1$ to class ω_{i_1}, $\boldsymbol{x}_2$ to ω_{i_2}, and so on. A way to approach the problem is to view each specific sequence X as an (extended) feature vector and $\Omega_i, i = 1, 2, \ldots, M^N$, as the available classes. Having observed a specific X, the Bayes rule assigns it to Ω_i for which

$$P(\Omega_i|X) > P(\Omega_j|X), \quad \forall i \neq j \tag{9.1}$$

and following our already familiar arguments, this is equivalent to

$$P(\Omega_i)p(X|\Omega_i) > P(\Omega_j)p(X|\Omega_j), \quad \forall i \neq j \tag{9.2}$$

In the following we will investigate the specific form that Eq. (9.2) takes for some typical class dependence models.

9.3 MARKOV CHAIN MODELS

One of the most widely used models describing the underlying class dependence is the Markov chain rule. If $\omega_{i_1}, \omega_{i_2}, \ldots$ is a sequence of classes, then the Markov model assumes that

$$P(\omega_{i_k}|\omega_{i_{k-1}}, \omega_{i_{k-2}}, \ldots, \omega_{i_1}) = P(\omega_{i_k}|\omega_{i_{k-1}}) \tag{9.3}$$

The meaning of this is that the class dependence is limited only within two successive classes. This type of model is also called a first-order Markov model, to distinguish it from obvious generalizations (second, third, etc.). In other words, given that the observations $\boldsymbol{x}_{k-1}, \boldsymbol{x}_{k-2}, \ldots, \boldsymbol{x}_1$ belong to classes $\omega_{i_{k-1}}, \omega_{i_{k-2}}, \ldots, \omega_{i_1}$, respectively, the probability of the observation $\boldsymbol{x}_k$, at stage k, belonging to class ω_{i_k} *depends only on the class from which observation $\boldsymbol{x}_{k-1}$, at stage $k-1$, has occurred.* Now combining (9.3) with the probability chain rule [Papo 91, page 192],

$$\begin{aligned} P(\Omega_i) &\equiv P(\omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_N}) \\ &= P(\omega_{i_N}|\omega_{i_{N-1}}, \ldots, \omega_{i_1})P(\omega_{i_{N-1}}|\omega_{i_{N-2}}, \ldots, \omega_{i_1}) \ldots P(\omega_{i_1}) \end{aligned}$$

we obtain

$$P(\Omega_i) = P(\omega_{i_1}) \prod_{k=2}^{N} P(\omega_{i_k}|\omega_{i_{k-1}}) \tag{9.4}$$

where $P(\omega_{i_1})$ is the prior probability for class $\omega_{i_1}, i_1 \in \{1, 2, \ldots, M\}$, to occur. Furthermore, two commonly adopted assumptions are that (a) given the sequence

of classes, the observations are statistically independent, and (b) the probability density function in one class does not depend on the other classes. *That is, dependence exists only on the sequence in which classes occur, but within a class observations "obey" the class' own rules.* This assumption implies that

$$p(X|\Omega_i) = \prod_{k=1}^{N} p(\boldsymbol{x}_k|\omega_{i_k}) \tag{9.5}$$

Combining Eqs. (9.4) and (9.5), the Bayes rule for Markovian models becomes equivalent to the statement:

Statement. Having observed the sequence of feature vectors $X: \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, classify them in the respective sequence of classes $\Omega_i: \omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_N}$, so that the quantity

$$p(X|\Omega_i)P(\Omega_i) = P(\omega_{i_1})p(\boldsymbol{x}_1|\omega_{i_1}) \prod_{k=2}^{N} P(\omega_{i_k}|\omega_{i_{k-1}})p(\boldsymbol{x}_k|\omega_{i_k}) \tag{9.6}$$

becomes maximum.

As we have already stated, searching for this maximum requires the computation of Eq. (9.6) for each of the Ω_i, $i = 1, 2, \ldots, M^N$. This amounts to a total of $O(NM^N)$ multiplications, which is usually a large number indeed. However, this direct computation is a brute-force task. Let us take for example two sequences Ω_i and Ω_j, which we assume differ only in the last class, that is, $\omega_{i_k} = \omega_{j_k}$, $k = 1, 2, \ldots, N-1$ and $\omega_{i_N} \neq \omega_{j_N}$. It does not take much "scientific thought" to realize that the computation of (9.6) for these two sequences shares all multiplications (which need not be repeated) but one. Furthermore, closer observation of (9.6) reveals that it has a rich computational structure, which can be exploited in order to maximize it in a much more efficient way. This now becomes our next concern.

9.4 THE VITERBI ALGORITHM

Figure 9.1 shows a diagram with N dot columns, where each dot in a column represents one of the M possible classes, $\omega_1, \omega_2, \ldots, \omega_M$. Successive columns correspond to successive observations $\boldsymbol{x}_k$, $k = 1, 2, \ldots, N$. The arrows determine transitions from one class to another, as observation vectors are obtained in sequence. Thus, *each of the class sequences Ω_i corresponds to a specific path of successive transitions.* Each transition from one class ω_i to another

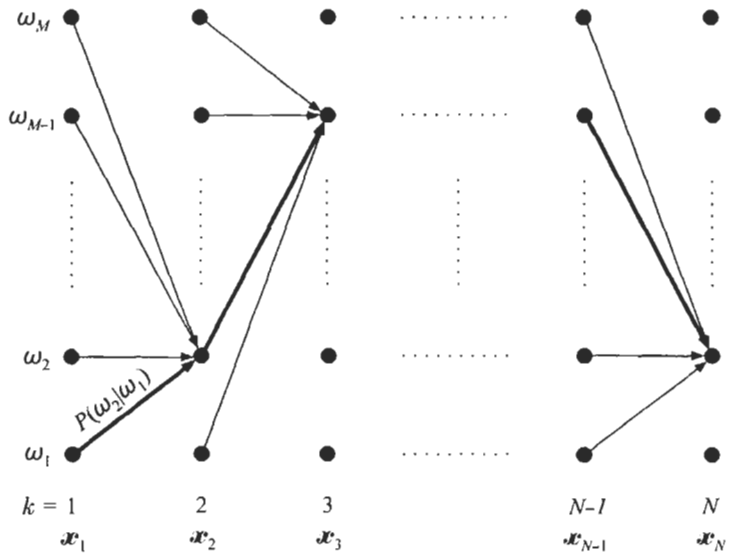


FIGURE 9.1: Trellis diagram for the Viterbi algorithm.

ω_j is characterized by a *fixed* probability $P(\omega_j|\omega_i)$, which is assumed to be known for the adopted model of class dependence. Furthermore, we assume that these probabilities are the same for all successive stages k. That is, the probabilities depend only on the respective class transitions and not on the stage at which they occur. We will further assume that the conditional probability densities $p(\boldsymbol{x}|\omega_i), i = 1, 2, \ldots, M$, are also known to the model. The task of maximizing (9.6) can now be stated as follows. Given a sequence of observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, find the path of successive (class) transitions that maximizes (9.6) (e.g., thick line in the figure). *The classes along this optimal path will be the ones in which the respective observations are classified.* To search for the optimal path we have to associate a cost with each of the transitions, in agreement with the cost function given in (9.6). A careful look at (9.6) suggests adopting

$$\hat{d}(\omega_{i_k}, \omega_{i_{k-1}}) = P(\omega_{i_k}|\omega_{i_{k-1}})\, p(\boldsymbol{x}_k|\omega_{i_k}) \tag{9.7}$$

as the cost associated with a transition of a path i from node (class) $\omega_{i_{k-1}}$, at stage $k-1$, to node ω_{i_k}, at stage k *and* at the same time observation $\boldsymbol{x}_k$ occurring. The initial condition for $k = 1$ is given by

$$\hat{d}(\omega_{i_1}, \omega_{i_0}) = P(\omega_{i_1})\, p(\boldsymbol{x}_1|\omega_{i_1})$$

Using this notation, the overall cost to be optimized becomes

$$\hat{D} \equiv \prod_{k=1}^{N} \hat{d}(\omega_{i_k}, \omega_{i_{k-1}}) \tag{9.8}$$

It will not come as a surprise to us if instead of (9.8) one chooses to maximize

$$\begin{aligned} \ln(\hat{D}) &= \sum_{k=1}^{N} \ln \hat{d}(\omega_{i_k}, \omega_{i_{k-1}}) \\ &\equiv \sum_{k=1}^{N} d(\omega_{i_k}, \omega_{i_{k-1}}) \equiv D \end{aligned} \tag{9.9}$$

Looking carefully at (9.9) or (9.8), it will not take us long to realize that Bellman's principle can again offer us the means for efficient optimization. Indeed, let us define, in accordance with D, the cost for reaching class ω_{i_k} at stage k via a path i as

$$D(\omega_{i_k}) = \sum_{r=1}^{k} d(\omega_{i_r}, \omega_{i_{r-1}})$$

Then, Bellman's principle states that

$$D_{max}(\omega_{i_k}) = \max_{i_{k-1}}[D_{max}(\omega_{i_{k-1}}) + d(\omega_{i_k}, \omega_{i_{k-1}})], \quad i_k, i_{k-1} = 1, 2, \ldots, M \tag{9.10}$$

with

$$D_{max}(\omega_{i_0}) = 0 \tag{9.11}$$

It is now straightforward to see that the optimal path, which leads to the maximum D in (9.9), is the one that ends at the final stage N in the class $\omega_{i_N}^*$ for which

$$\omega_{i_N}^* = arg \max_{\omega_{i_N}} D_{max}(\omega_{i_N}) \tag{9.12}$$

Going back to Figure 9.1, we see that at each stage $k, k = 1, 2, \ldots, N$, there are M possible transitions to each of the nodes ω_{i_k}. The recursive relation in (9.10) suggests that in searching for the maximum we need only keep one of these transitions for every node, *the one that leads to the maximum respective cost* $D_{max}(\omega_{i_k})$ (thick lines in the figure). *Hence, at each stage there are only* M *surviving paths.* Therefore, the number of operations is M for each node, thus M^2 for all the nodes at each stage and NM^2 in total. The latter number compares

very favorably with the NM^N of the "brute-force task." The resulting algorithm is known as the Viterbi algorithm. This dynamic programming algorithm was first suggested for optimal decoding in convolutional codes in communications [Vite 67].

In previous chapters, we stated that for a number of reasons an alternative to the optimal Bayes classifier can be used. No doubt context-dependent classification can also be emancipated from its Bayesian root. This can easily be achieved by adopting different transition costs $d(\omega_{i_k}, \omega_{i_{k-1}})$, which are not necessarily related to probability densities. In the sequel we will present two typical application areas of the Viterbi algorithm.

9.5 CHANNEL EQUALIZATION

Channel equalization is the task of recovering a transmitted sequence of information bits I_k (e.g., 1 or 0) after they have been corrupted by the transmission channel and noise sources. The samples received at the receiver end are, thus, given by

$$x_k = f(I_k, I_{k-1}, \ldots, I_{k-n+1}) + \eta_k \tag{9.13}$$

where the function $f(\cdot)$ represents the action of the channel and η_k the noise sequence. The channel contribution to the overall corruption is called the *intersymbol interference (ISI)* and it spans n successive information bits. The equalizer is the *inverse* system whose task is to provide decisions $\hat{I}_k$ about the transmitted information bits I_k, based on l successively received samples $[x_k, x_{k-1}, \ldots, x_{k-l+1}] \equiv \boldsymbol{x}_k^T$. Usually, a delay r must be used in order to accommodate the (possible) noncausal nature of the inverse system. In such cases the decisions made at time k correspond to the I_{k-r} transmitted information bit (Figure 9.2). A simple example will reveal to us how the equalization problem comes under the umbrella of a Markovian context-dependent classification task. Assume a simple linear channel

$$x_k = 0.5I_k + I_{k-1} + \eta_k \tag{9.14}$$

For $l = 2$, successively received samples are combined in vectors in the two-dimensional space, that is,

$$\boldsymbol{x}_k^T = [x_k, x_{k-1}]$$

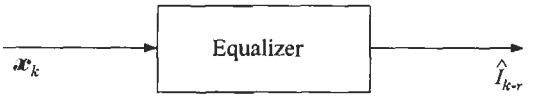
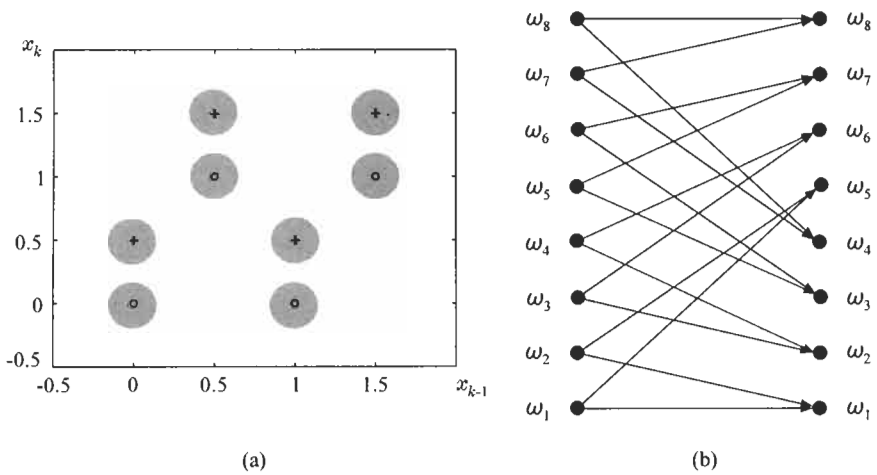


FIGURE 9.2: Block diagram of an equalizer.

Table 9.1: Received samples and respective information bits for the channel of Eq. (9.14)

I_k	I_{k-1}	I_{k-2}	x_k	x_{k-1}	Cluster
0	0	0	0	0	ω_1
0	0	1	0	1	ω_2
0	1	0	1	0.5	ω_3
0	1	1	1	1.5	ω_4
1	0	0	0.5	0	ω_5
1	0	1	0.5	1	ω_6
1	1	0	1.5	0.5	ω_7
1	1	1	1.5	1.5	ω_8



(a) (b)

FIGURE 9.3: Plot (a) of the eight possible clusters associated with the channel of Eq. (9.14) and (b) the allowable transitions among them.

Let us further assume that there are N such observation vectors available. From (9.14) it is readily seen that each $\boldsymbol{x}_k, k = 1, 2, \ldots, N$, depends on the values of three successive information bits, namely I_k, I_{k-1}, I_{k-2}. Neglecting the effects of noise, the possible values of the received samples x_k are given in Table 9.1, together with the respective information bits. Figure 9.3a shows the geometry in the two-dimensional space with (x_k, x_{k-1}) on its axis. When the effect of noise is taken into account, the received vectors are clustered around these points

(for example, shaded area). For the specific channel of (9.14) there are eight possible clusters, ω_i, $i = 1, 2, \ldots, 8$ (Table 9.1). Clusters around "+" correspond to $I_k = 1$ and those around "o" to $I_k = 0$. In general, the total number of clusters for a binary information sequence is $m = 2^{n+l-1}$. On the reception of each $\boldsymbol{x}_k = [x_k, x_{k-1}]^T$, the equalizer has to decide whether the corresponding transmitted information bit I_k was either a "1" (i.e., class "A") or a "0" (class "B"). In other words, this is nothing other than a two-class classification problem (M class for the M-ary case), and each class consists of a *union of clusters*. Thus, various techniques, from those we have already studied in previous chapters, can be used. A simple way, which was followed in [Theo 95], consists of two steps. During the training period a sequence of known information bits is transmitted and the representative center $\boldsymbol{\mu}_i$ for each of the clusters is computed by a simple averaging of all the vectors $\boldsymbol{x}_k$ belonging to the respective cluster. This is possible during the *training period*, since the transmitted information bits are known and thus we know to which of the clusters each received $\boldsymbol{x}_k$ belongs. For example, in the case of Table 9.1 if the sequence of transmitted bits is ($I_k = 1$, $I_{k-1} = 0$, $I_{k-2} = 1$), then the received $\boldsymbol{x}_k$ belongs to ω_6. At the same time the clusters are labeled as "1" or "0" depending on the value of the I_k bit. At the so-called *decision directed mode* the transmitted information bits are unknown, and the decision about the transmitted I_k is based on which cluster ("1" or "0" label) the received vector $\boldsymbol{x}_k$ is closest to. For this purpose a metric is adopted to define distance. The Euclidean distance of the received vector $\boldsymbol{x}_k$ from the representatives $\boldsymbol{\mu}_i$ of the clusters is an obvious candidate. Although such an equalizer can result in reasonable performance, measured in *bit error rate (BER)* (the percentage of information bits wrongly identified), there is still a great deal of information that has not been exploited. Let us search for it!

From the definition of ISI in (9.13) we know that the channel spans a number of successive information bits. *Thus, only certain transitions among clusters are possible*. Indeed, let us assume for example that at time k the transmitted information bits (I_k, I_{k-1}, I_{k-2}) were (0, 0, 1); hence, the corresponding observation vector $\boldsymbol{x}_k$ belongs to cluster ω_2. The next received observation vector $\boldsymbol{x}_{k+1}$ will depend on the triple (I_{k+1}, I_k, I_{k-1}) bits, which can be either (1, 0, 0) or (0, 0, 0). Thus, $\boldsymbol{x}_{k+1}$ will belong either to ω_5 or to ω_1. That is, there are only two possible transitions from ω_2. Figure 9.3b shows the possible transitions among the various clusters. Assuming equiprobable information bits, then from Figure 9.3b we can easily conclude that all *allowable* transitions have probability 0.5, that is,

$$P(\omega_1|\omega_1) = 0.5 = P(\omega_5|\omega_1)$$

and the rest are zero (not allowable). We now have all the ingredients to define the equalization problem as a context-dependent classification task.

Given N observation vectors $\boldsymbol{x}_k$, $k = 1, 2, \ldots, N$, classify them in a sequence of clusters $\omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_N}$. This automatically classifies each $\boldsymbol{x}_k$ in one of the two

classes "A" and "B," which is equivalent to deciding that I_k is 1 or 0. For this goal a cost function has to be adopted. In [Theo 95, Geor 97] the cost $d(\omega_{i_k}, \omega_{i_{k-1}})$ in (9.9) used in the Viterbi algorithm, for the allowable transitions, was taken to be

$$d(\omega_{i_k}, \omega_{i_{k-1}}) = d_{\omega_{i_k}}(\boldsymbol{x}_k) \tag{9.15}$$

where $d_{\omega_{i_k}}(\boldsymbol{x}_k)$ is the distance of $\boldsymbol{x}_k$ from the representative of the ω_{i_k} cluster. This can be either the Euclidean

$$d_{\omega_{i_k}}(\boldsymbol{x}_k) = \|\boldsymbol{x}_k - \boldsymbol{\mu}_{i_k}\| \tag{9.16}$$

or the Mahalanobis distance

$$d_{\omega_{i_k}}(\boldsymbol{x}_k) = \left((\boldsymbol{x}_k - \boldsymbol{\mu}_{i_k})^T \Sigma_{i_k}^{-1} (\boldsymbol{x}_k - \boldsymbol{\mu}_{i_k})\right)^{1/2} \tag{9.17}$$

The covariance matrices Σ_{i_k}, describing the distribution of the observation vectors around the respective cluster representatives, are learned during the training period together with the cluster representatives $\boldsymbol{\mu}_i$. The Viterbi algorithm is then used to obtain the optimal overall minimum distance sequence and recursion (9.10) is modified for the search of the minimum.

An alternative way to define the cost function is to consider observations as scalars x_k, that is, the received samples ($l = 1$), and make the following assumptions:

- Successive noise samples are statistically independent and Gaussian.
- The channel impulse response is known or can be estimated. In practice, a specific channel model is assumed, that is, $\hat{f}(\cdot)$, and its parameters are estimated via an optimization method, for example, least squares [Proa 89].

Under the preceding assumptions the cost for the allowable state transitions in (9.9) becomes

$$\begin{aligned} d(\omega_{i_k}, \omega_{i_{k-1}}) &= \ln p(x_k|\omega_{i_k}) \equiv \ln(p(\eta_k)) \\ &= -(x_k - \hat{f}(I_k, \ldots, I_{k-n+1}))^2 \end{aligned} \tag{9.18}$$

where η_k is the respective Gaussian distributed noise sample. Obviously, in (9.18) the constants in the Gaussian density function have been omitted. If the Gaussian and independence assumptions are valid, this is obviously a Bayesian optimal classification to the clusters (from which the "0" and "1" classes result). However, if this is not true, the cost in (9.18) is no longer the optimal choice. This is, for example, the case when the so-called cochannel (nonwhite) interference is present. In such cases, the cluster-based approach is justifiable and

indeed it leads to equalizers with more robust performance [Geor 97]. Furthermore, the fact that in the clustering approach no channel estimation is required can be very attractive in a number of cases, where nonlinear channels are involved and their estimation is not a straightforward task [Theo 95]. In [Kops 02] equalization is performed in the one-dimensional space, i.e., $l = 1$. Although this increases the probability of having clusters with different labels to overlap, this is not crucial to the performance, since the Viterbi algorithm has the power to detect the correct label, by exploiting the history in the path. Furthermore, it is pointed out that one needs not determine directly all the 2^n cluster representatives; it suffices to learn, during the training phase, only n of the clusters, and the rest can be obtained by simple arithmetic operations. This is achieved by exploiting the mechanism underlying the cluster formation and the associated symmetries. Both of these observations have a substantial impact on reducing the computational complexity as well as the required length of the training sequence.

The discussion so far was based on the development of a trellis diagram associated with the transitions among clusters, and the goal was to unravel the optimal path, using the Viterbi algorithm. However, although this came as a natural consequence of the context dependent Bayesian classifier, it turns out this it is not the most efficient way from a computational point of view. From Figure 9.3b one can easily observe that pairs of clusters jump to the same clusters after transition. For example, the allowable transitions from ω_1 and ω_2 are the same and lead to either ω_1 or ω_5. The same is true for ω_3 and ω_4, etc. This is because the allowable transitions are determined by the $n + l - 2$ most recent bits. For the example of Figure 9.3, transitions are determined by the pair (I_k, I_{k-1}), which, however, is shared by two clusters, i.e., (I_k, I_{k-1}, I_{k-2}), depending on the value of I_{k-2} if it is 0 or 1. The pair (I_k, I_{k-1}) is known as the *state* at time k. This is because, knowing the state at time k and the transmitted bit I_{k+1} at time $k + 1$ we can determine the next state (I_{k+1}, I_k) at time $k + 1$, as is the case in the finite state machines. Since transitions are determined by the states, one can construct a trellis diagram based on the states instead of on clusters. For the example of Figure 9.3, where eight clusters are present, there is a total of four states, i.e., $s_1 : (0, 0)$, $s_2 : (0, 1)$, $s_3 : (1, 0)$, $s_4 : (1, 1)$. Figure 9.4 shows these states and the allowable transitions among them. Each transition is associated with one bit, which is the current transmitted bit. Obviously there is a close relationship between states and clusters. If we know the state transition, i.e., $(I_k, I_{k-1}) \rightarrow (I_{k+1}, I_k)$, then the current cluster at time $k + 1$ will be determined by the corresponding values of (I_{k+1}, I_k, I_{k-1}), and this automatically determines the cost of the respective transition, e.g., (9.16, 9.17). Hence the estimates of the transmitted bits are obtained from the sequence of bits along the optimal path, that is, the one with the minimum total cost, in the state trellis diagram instead of the larger cluster trellis diagram.

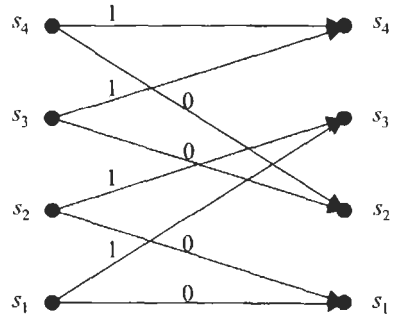


FIGURE 9.4: Plot of the four states associated with the channel of Eq. (9.14) and a two-dimensional equalizer showing the allowable transitions among them.

9.6 HIDDEN MARKOV MODELS

In the channel equalization application of the previous section, the states (clusters) of the Markov chain were *observable*. That is, given the $l + n - 1$ most recent information bits (i.e., I_k, I_{k-1}, I_{k-2}, in the given example), the state to which the corresponding observation vector $\boldsymbol{x}_k$ belongs is readily known. Thus, during the training period these states can be "labeled" and we can estimate their associated parameters. In this section we will be concerned with systems where the states are not directly observable and can only be inferred from the sequence of the observations via some optimizing technique. These types of Markov models are known as *hidden Markov models (HMMs)*. An HMM is a type of stochastic modeling appropriate for nonstationary stochastic sequences, with statistical properties that undergo distinct random transitions among a set of different stationary processes. In other words, an HMM models a sequence of observations as a *piecewise stationary process*. Such models have been used extensively in speech recognition to model speech utterances [Bake 75, Jeli 76]. An utterance may be a spoken word, part of a word, or even a complete sentence or a paragraph. The statistical properties of the speech signal within an utterance undergo a series of transitions. For example, a word consists of subword portions of *voiced* (vowels) and *unvoiced* (consonants) sounds. These are characterized by distinctly different statistical properties, which are in turn reflected in transitions of the speech signal from one statistic to another. Handwriting recognition [Chen 95, Vlon 92, Agaz 93, ElYa 99], texture classification [Chen 95a, Wu 96], blind equalization [Anto 97, Kale 94, Geor 98] and musical pattern recognition [Pikr 02] are some other example applications in which the power of HMM modeling has been successfully exploited.

An HMM model is basically a *stochastic finite state automaton*, which generates an observation string, that is, the sequence of observation vectors, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$. Thus, an HMM model consists of a number, say K, states and the observation string

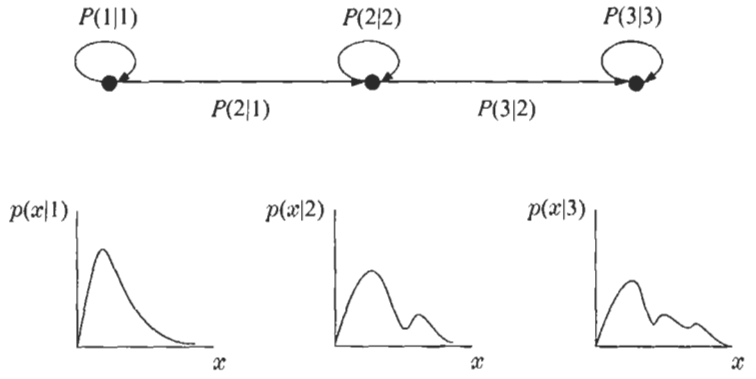


FIGURE 9.5: Model parameters describing a three-state hidden Markov model.

is produced as a result of successive transitions from one state i to another j. We will adopt the so-called Moore machine model, according to which observations are produced as emissions from the states upon *arrival* (of the transition) at each state.

Figure 9.5 shows a typical diagram of an HMM of three states, where arrows indicate transitions. Such a model could correspond to a short word with three different stationary parts. The model provides information about the successive transitions between the states ($P(i|j)$, $i, j = 1, 2, 3$—temporal modeling of the spoken word) and also about the stationary statistics underlying each state ($p(\boldsymbol{x}|i)$, $i = 1, 2, 3$). This type of HMM model is also known as left to right, because transitions to states with a smaller index are not allowed. Other models also do exist [Rabi 89]. In practice, the states correspond to certain physical characteristics, such as distinct sounds. In speech recognition the number of states depends on the expected number of such sound phenomena (phonemes[1]) within one word. Actually, a number of states (typically three or four) are used for each phoneme. The average number of observations, resulting from various versions of a spoken word, can also be used as an indication of the number of required states. However, the exact number of states is often the result of experimentation and cannot be determined accurately a priori. In blind equalization, the states are associated with the number of clusters formed by the received data [Geor 98]. In character recognition tasks the states may correspond to line or arc segments in the character [Vlon 92].

In the sequel we will assume that we are given a set of M known reference patterns, and our goal is to *recognize* which one of them an unknown test pattern

[1] A phoneme is a basic sound unit corresponding to a unique set of articulatory gestures, characterizing the vocal tract articulators for speech sound production [Dell 93].

matches best. This problem was studied in the previous chapter from a template matching (deterministic) perspective. A different (stochastic) path will be taken here. Specifically, we will assume that each known (reference) pattern is described via an HMM model. That is, each of the M patterns is characterized by the following set of parameters:

- The number K_s of the states, $s = 1, 2, \ldots, M$.
- The probability densities $p(\boldsymbol{x}|j)$, $j = 1, 2, \ldots, K_s$, describing the distribution of the observations emitted from state j.
- The transition probabilities $P(i|j)$, $i, j = 1, 2, \ldots, K_s$, among the various states. Some of them can be zero.
- The probabilities $P(i)$, $i = 1, 2, \ldots, K_s$, of the initial state.

Although this is quite a general description of an HMM model, it is worth pointing out that variations are also possible. For example, sometimes the place of the self-transition probability ($P(i|i)$) is taken by the state duration probability distribution, which describes the number of successive stages for which the model stays in state i ([Rabi 89], Problem 9.2). In some other cases, a model for the generation mechanism of the observations is adopted, for example, an autoregressive model [Pori 82] or even a time-varying one that models nonstationary state statistics [Deng 94]. In the sequel we will adhere to the foregoing model. Our problem now consists of two major tasks. One is the *training* for each of the HMM models, that is, the computation of the parameters just listed. The other is the task of *recognition*. That is, once the HMM parameters of the reference models are known, how do we decide which reference model the unknown pattern matches best? We will start with the latter task.

Recognition

Any path method: To start with, we will treat each of the M reference HMM models as a distinct class. The sequence of observations X is *the result of emissions, due to transitions among the different states of the respective model.* The problem then becomes a typical classification task. Given the sequence of N observations X:$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, resulting from the unknown pattern, decide to which class it belongs. The Bayesian classifier decides in favor of pattern $\mathcal{S}^*$ for which

$$\mathcal{S}^* = arg \max_{\mathcal{S}} P(\mathcal{S}|X), \quad \text{that is, over all the models} \tag{9.19}$$

and for equiprobable reference models (classes) this is equivalent to

$$\mathcal{S}^* = arg \max_{\mathcal{S}} p(X|\mathcal{S}) \tag{9.20}$$

where for convenience we have used $\mathcal{S}$ to denote the set of parameters describing each HMM model, that is,

$$\mathcal{S} = \{P(i|j),\ p(\boldsymbol{x}|i),\ P(i),\ K_s\}$$

For each model $\mathcal{S}$ there is more than one possible set of successive state transitions Ω_i, each having probability of occurrence $P(\Omega_i|\mathcal{S})$. Thus, recalling the known rule for probabilities we can write [Papo 91]

$$p(X|\mathcal{S}) = \sum_i p(X, \Omega_i|\mathcal{S}) = \sum_i p(X|\Omega_i, \mathcal{S})P(\Omega_i|\mathcal{S}) \tag{9.21}$$

In order to find the maximum $p(X|\mathcal{S})$ over all possible models, the quantity in (9.21) has to be computed for each of the M reference models. Its efficient computation can be achieved via an approach similar to the Viterbi algorithm. Indeed, the only difference between (9.21) and (9.6) is that, instead of simply searching for the maximum over all possible state sequences Ω_i, (9.21) requires summing up the respective values for each of them. To this end, let us define as $\alpha(i_k)$ the probability density of the joint event: (a) a path is at state $i_k (i_k \in \{1, 2, \ldots, K_s\})$ at stage k *and* (b) observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{k-1}$ have been emitted at the previous stages *and* (c) observation $\boldsymbol{x}_k$ is emitted from the state i_k at stage k. From the definition of $\alpha(i_k)$ the following recursive relation is easily understood:

$$\begin{aligned}\alpha(i_{k+1}) &\equiv p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k+1}, i_{k+1}|\mathcal{S}) \\ &= \sum_{i_k} \alpha(i_k)P(i_{k+1}|i_k)p(\boldsymbol{x}_{k+1}|i_{k+1}), \quad k = 1, 2, \ldots, N-1\end{aligned} \tag{9.22}$$

with

$$\alpha(i_1) = P(i_1)p(\boldsymbol{x}_1|i_1)$$

The product $P(i_{k+1}|i_k)p(\boldsymbol{x}_{k+1}|i_{k+1})$ in (9.22) provides the local information for the last transition and $\alpha(i_k)$ is the information accumulated from the path history up to stage k. As is apparent from its definition, $\alpha(i_k)$ does not depend on the subsequent observations $\boldsymbol{x}_{k+1}, \ldots, \boldsymbol{x}_N$. The definition of a joint probability density function, which depends on *all* available observations, is also possible and will be used later on. To this end, let us define $\beta(i_k)$ as the probability density function of the event: observations $\boldsymbol{x}_{k+1}, \ldots, \boldsymbol{x}_N$ occur at stages $k+1, \ldots, N$, given that at stage k the path is at state i_k. Then after a little thought we conclude that $\beta(i_k)$ obeys the following recursion:

$$\begin{aligned}\beta(i_k) &\equiv p(\boldsymbol{x}_{k+1}, \boldsymbol{x}_{k+2}, \ldots, \boldsymbol{x}_N|i_k, \mathcal{S}) \\ &= \sum_{i_{k+1}} \beta(i_{k+1})P(i_{k+1}|i_k)p(\boldsymbol{x}_{k+1}|i_{k+1}), \quad k = N-1, \ldots, 1\end{aligned} \tag{9.23}$$

where by definition

$$\beta(i_N) = 1, i_N \in \{1, 2, \ldots, K_s\} \tag{9.24}$$

Thus, the probability density of the joint event: (a) a path is at state i_k at stage k *and* (b) $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ have been observed, is given by

$$\begin{aligned} \gamma(i_k) &\equiv p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, i_k | \mathcal{S}) \\ &= p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, i_k | \mathcal{S}) p(\boldsymbol{x}_{k+1}, \ldots, \boldsymbol{x}_N | i_k, \mathcal{S}) \\ &= \alpha(i_k)\beta(i_k) \end{aligned} \tag{9.25}$$

where the assumption about the observations' independence has been employed. Equation (9.25) also justifies the choice $\beta(i_N) = 1, i_N \in \{1, 2, \ldots, K_s\}$.

Let us now return to our original goal of computing the maximum $p(X|\mathcal{S})$. Equation (9.22) suggests that we can write Equation (9.21) as

$$p(X|\mathcal{S}) = \sum_{i_N=1}^{K_s} \alpha(i_N) \tag{9.26}$$

For the computation of (9.26), we need to compute all $\alpha(i_k)$, for $k = 1, 2, \ldots, N$. This is efficiently achieved using the diagram of Figure 9.6. Each node corresponds to a stage k and a state $i_k, i_k = 1, 2, \ldots, K_s$. For each of the nodes the density $\alpha(i_k)$ from Eq. (9.22) is computed. Thus, the number of computations is of the order of NK_s^2. The resulting algorithm is known as the *any path* method, since all paths participate in the final cost. The computation of (9.26) is performed for each of the M models and the unknown string pattern of the observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ is classified to the reference model $\mathcal{S}$ for which $p(X|\mathcal{S})$ becomes maximum.

Best path method: An alternative, suboptimal, approach is the so-called *best path* method. According to this method, for a given observation sequence X we compute the most probable (best) path of states sequence *for each* of the reference models. The task now becomes that of (9.1) and the search for each of the optima can be achieved efficiently via the Viterbi algorithm, with the cost D given as in (9.9),

$$\begin{aligned} D &= \sum_{k=1}^{N} d(i_k, i_{k-1}) \\ d(i_k, i_{k-1}) &= \ln P(i_k | i_{k-1}) + \ln p(\boldsymbol{x}_k | i_k) \end{aligned} \tag{9.27}$$

In other words, for each of the models we compute the maximum of $P(\Omega_i)p(X|\Omega_i) = p(\Omega_i, X)$. Hence, the summation in (9.21) is replaced by a

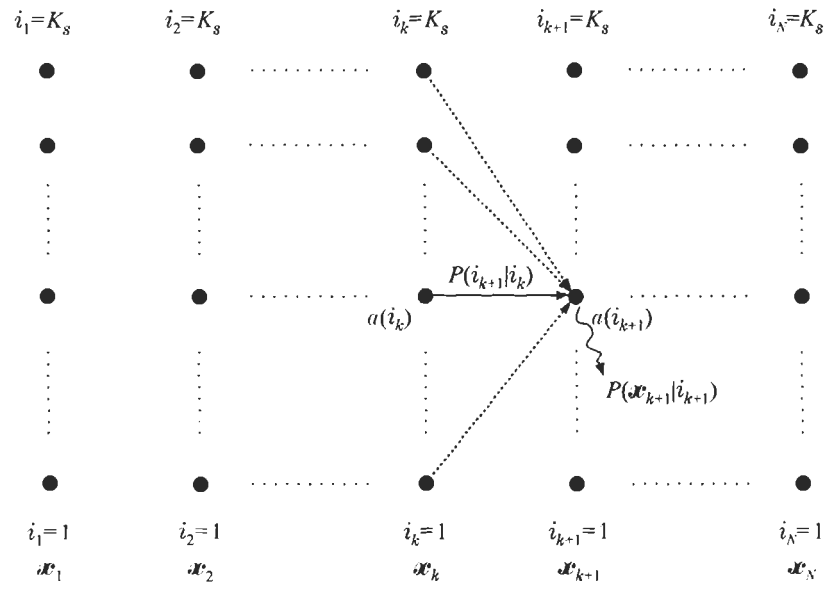


FIGURE 9.6: Diagram showing the computational flow for the any path method.

maximum operation. The unknown pattern is classified to that reference model S for which the resulting optimal cost D is maximum.

Training

This is a more difficult task. The states now are not observable and a direct training approach, such as the one used in Section 9.5, cannot be adopted. The parameters that define each HMM model S can *only* be inferred from the available observations.

One way to achieve this goal is to estimate the unknown parameters, so that the output for each model, (9.27) or (9.26), becomes maximum *for a training set of observations known to belong to the model.* This is an optimization task with a nonlinear cost function and it is carried out iteratively. To this end, assumptions about the probability density functions $p(\boldsymbol{x}|i)$ are required. The procedure can be simplified if one assumes that the observations $\boldsymbol{x}_k$ can take only discrete values. In such cases probability density functions $p(\boldsymbol{x}|i)$ become probabilities, $P(\boldsymbol{x}|i)$.

Discrete Observation HMM Models

We will assume that the training observation string $\boldsymbol{x}_k$, $k = 1, 2, \ldots, N$, consists of quantized vectors. In practice, this is achieved via vector quantization techniques,

to be discussed later on in Chapter 14. Hence, each observation vector can take one only out of L possible distinct values in the l-dimensional space. Thus, observations can be described as integers r, $r = 1, 2, \ldots, L$. The steps for each of the two methods, that is, any path and best path, are as following:

Baum–Welch reestimation: The "output" quantity in the any path procedure is $p(X|\mathcal{S})$. Thus, estimating the parameters of the model $\mathcal{S}$ so that $p(X|\mathcal{S})$ is a maximum is nothing but a *maximum likelihood* parameter estimation procedure. Before going into the discussion of the iteration steps some definitions are needed.

Definitions:

- $\xi_k(i, j, X|\mathcal{S}) \equiv$ the probability of the joint event: (a) a path passes through state i at stage k *and* (b) through state j at the next stage $k+1$ *and* (c) the model generates the available sequence of observations X, given the parameters of the model $\mathcal{S}$.
- $\gamma_k(i, X|\mathcal{S}) \equiv$ the probability of the joint event: (a) a path passes through state i at stage k *and* (b) the model generates the available observation sequence, given the parameters of the model $\mathcal{S}$.

From these definitions it is not difficult to show that

$$\xi_k(i, j) \equiv \xi_k(i, j|X, \mathcal{S}) = \frac{\xi_k(i, j, X|\mathcal{S})}{P(X|\mathcal{S})} \tag{9.28}$$

Mobilizing the definitions in Eqs. (9.22) and (9.23), Eq. (9.28) becomes

$$\xi_k(i, j) = \frac{\alpha(i_k = i)P(j|i)P(\boldsymbol{x}_{k+1}|j)\beta(i_{k+1} = j)}{P(X|\mathcal{S})} \tag{9.29}$$

where $\alpha(i_k = i)$ accounts for the path history terminating at stage k and state i. $\beta(i_{k+1} = j)$ accounts for the future of the path, which at stage $k+1$ is at state j and then evolves *unconstrained* until the end. The product $P(j|i)P(\boldsymbol{x}_{k+1}|j)$ accounts for the local activity at stage k. The other quantity of interest is the probability of being at stage k at state i, given the model and the observation sequence, that is

$$\gamma_k(i) \equiv \gamma_k(i|X, \mathcal{S}) = \frac{\alpha(i_k = i)\beta(i_k = i)}{P(X|\mathcal{S})} \tag{9.30}$$

From the foregoing it is not difficult to see that

- $\sum_{k=1}^{N} \gamma_k(i)$ can be regarded as the expected (over the number of stages) number of times state i occurs, given the model $\mathcal{S}$ and the observation sequence X. When the upper index in the summation is $N-1$, this quantity is the expected number of transitions from state i.

- $\sum_{k=1}^{N-1} \xi_k(i, j)$ can be regarded as the expected number of transitions from state i to state j, given the model and the observation sequence.

The preceding definitions lead us to adopt the following (re)estimation formulas as reasonable estimates of the unknown model parameters.

$$\bar{P}(j|i) = \frac{\sum_{k=1}^{N-1} \xi_k(i, j)}{\sum_{k=1}^{N-1} \gamma_k(i)} \tag{9.31}$$

$$\bar{P}_{\boldsymbol{x}}(r|i) = \frac{\sum_{(k=1 \text{ and } x \to r)}^{N} \gamma_k(i)}{\sum_{k=1}^{N} \gamma_k(i)} \tag{9.32}$$

$$\bar{P}(i) = \gamma_1(i) \tag{9.33}$$

The numerator in (9.32) sums only those of $\gamma_k(i)$ for which the corresponding observation $\boldsymbol{x}_k$ takes the rth discrete value. The iterative algorithm can now be expressed in terms of the following steps:

Iterations:

- Initial conditions: Assume initial conditions for the unknown quantities. Compute $P(X|\mathcal{S})$.
- Step 1: From the current estimates of the model parameters reestimate the new model $\bar{\mathcal{S}}$ via Eqs. (9.31) to (9.33).
- Step 2: Compute $P(X \mid \bar{\mathcal{S}})$. If $P(X \mid \bar{\mathcal{S}}) - P(X \mid \mathcal{S}) > \epsilon$ set $\mathcal{S} = \bar{\mathcal{S}}$ and go to step 1. Otherwise stop.

Remarks

- Each iteration improves the model $\bar{\mathcal{S}}$; that is, it is true that $P(X|\bar{\mathcal{S}}) \geq P(X|\mathcal{S})$.
- The algorithm may lead to local maxima; see for example [Baum 67, Baum 68, Baum 70]. This is why the algorithm in practice runs a number of times, starting from different initial conditions, in order to find a favorable local maximum for $P(X|\mathcal{S})$. Other computational issues, including parallelism and memory requirements, are treated in [Turi 98].
- The Baum–Welch algorithm is basically an implementation of the EM algorithm, which was introduced in Chapter 2. Indeed, a little thought reveals that the ML estimation of the HMM parameters is a typical ML problem with an incomplete data set, that is, the unobserved states (e.g., [Moon 96, Diga 99]). A generalization of the method that allows multiple observation training sequences is given in [Li 00]. Other, gradient-based, optimizing techniques for the estimation of the unknown parameters have also been suggested and used [Levi 83].

- Practical implementation issues:
  1. *Scaling:* The probabilities $\alpha(i_k)$, $\beta(i_k)$ are obviously less than one, and their values tend to zero very fast as the number of terms in the products (9.22) and (9.23) increases. In practice, the dynamic range of their computed values may exceed the precision range of the computer, so appropriate scaling is required. A basic procedure is to scale $\alpha(i_k)$ in proportion to the number of stages. If the same scaling factor is used for the $\beta(i_k)$, then on taking their product in the recursions the effect of the scaling cancels out [Levi 83, Rabi 89] (Problem 9.4).
  2. *Initial conditions:* This is an omnipresent problem in all iterative optimization algorithms. Usually, the unknown parameters are initialized randomly, subject, of course, to the constraints of the problem. That is, if some transitions are not allowed, the corresponding probabilities are set to zero, and also probabilities must add to one.
  3. *Insufficient training data:* Generally a large amount of training data is necessary to learn the HMM parameters. The observation sequence must be sufficiently long with respect to the number of states of the HMM model. This will guarantee that all state transitions will appear a sufficient number of times, so that the reestimation algorithm learns their respective parameters. If this not the case, a number of techniques have been devised to cope with the issue. For a more detailed treatment the reader may consult [Rabi 89, Dell 93] and the references therein.

Viterbi reestimation: In the speech literature the algorithm is also known as the *segmental k-means training* algorithm [Rabi 89]. It is related to the best path method.

Definitions:

- $n_{i|j} \equiv$ number of transitions from state j to state i.
- $n_{|j} \equiv$ number of transitions originated from state j.
- $n_{i|} \equiv$ number of transitions terminated at state i.
- $n(r|i) \equiv$ number of times observation $r \in \{1, 2, \ldots, L\}$ occurs jointly with state i.

Iterations:

- Initial conditions: Assume the initial estimates of the unknown parameters. Obtain the best path and compute D.
- Step 1: From the available best path, reestimate the new model parameters as:

$$\bar{P}(i|j) = \frac{n_{i|j}}{n_{|j}}$$

$$\bar{P}\boldsymbol{x}(r|i) = \frac{n(r|i)}{n_{i|}}$$

- Step 2: For the new model parameters obtain the best path and compute the corresponding overall cost $\bar{D}$. Compare it with the cost D of the previous iteration. If $\bar{D} - D > \epsilon$ set $D = \bar{D}$ and go to step 1. Otherwise stop.

Symbol $\tilde{P}\boldsymbol{x}(r|i)$ is the current iteration estimate of the probability of emitting from state i the rth value from the available palette of the L possible vectors. The preceding algorithm has assumed that the initial state is known; thus no estimation of the respective probabilities is needed. This is, for example, true for left-to-right models, such as the one shown in Figure 9.5. The Viterbi reestimation algorithm can be shown to converge to a proper characterization of the underlying observations [Fu 82, Lee 72].

Continuous Observation HMM

The discrete observation modeling of originally continuous variables suffers from a serious drawback. During the (vector) quantization stage of the signal (e.g., speech segment), a severe loss of information about the original waveform may occur, which can seriously degrade the performance of the recognizer. The alternative is to work with continuous observation modeling, albeit at the expense of higher complexity. This approach requires modeling of the probability densities $p(\boldsymbol{x}|i)$, prior to estimation. Once these have been estimated, the recognition problem evolves along the same lines as with the discrete observation case. The difference exists only in the training task. One way to approach the problem is to assume a parametric model for the probability density function and then use reestimation procedures to compute the unknown model parameters. As we have already discussed in Chapter 2, a very general parameterization of the probability density function is via mixture modeling, that is,

$$p(\boldsymbol{x}|i) = \sum_{m=1}^{L} c_{im} F(\boldsymbol{x}, \boldsymbol{\mu}_{im}, \Sigma_{im}) \tag{9.34}$$

where $F(\cdot, \cdot, \cdot)$ is a density function and $\boldsymbol{\mu}_{im}$, Σ_{im} are the mean vector and the covariance matrix of the mth mixture. We will adhere to Gaussian functions, which are usually employed in practice. The mixture coefficients c_{im} have to satisfy the constraint

$$\sum_{m=1}^{L} c_{im} = 1, \quad 1 \leq i \leq K_s$$

so that

$$\int_{-\infty}^{+\infty} p(\boldsymbol{x}|i)\, d\boldsymbol{x} = 1, \quad 1 \leq i \leq K_s$$

Following arguments similar to those used to reestimate the parameters in the discrete observation HMM case, the following reestimation formulas are obtained [Lipo 82, Juan 85, Juan 86].

$$\bar{c}_{im} = \frac{\sum_{k=1}^{N} \gamma_k(i, m)}{\sum_{k=1}^{N} \sum_{r=1}^{L} \gamma_k(i, r)} \tag{9.35}$$

$$\bar{\boldsymbol{\mu}}_{im} = \frac{\sum_{k=1}^{N} \gamma_k(i, m) \boldsymbol{x}_k}{\sum_{k=1}^{N} \gamma_k(i, m)} \tag{9.36}$$

$$\bar{\Sigma}_{im} = \frac{\sum_{k=1}^{N} \gamma_k(i, m)(\boldsymbol{x}_k - \bar{\boldsymbol{\mu}}_{im})(\boldsymbol{x}_k - \bar{\boldsymbol{\mu}}_{im})^T}{\sum_{k=1}^{N} \gamma_k(i, m)} \tag{9.37}$$

The term $\gamma_k(i, m)$ is the probability density of being at state i and stage k with the mth mixture component accounting for $\boldsymbol{x}_k$, that is,

$$\gamma_k(i, m) = \frac{\alpha(i_k = i)\beta(i_k = i)}{\sum_{i_k=1}^{K_s} \alpha(i_k)\beta(i_k)} \times \frac{c_{im} F(\boldsymbol{x}_k, \boldsymbol{\mu}_{im}, \Sigma_{im})}{\sum_{r=1}^{L} c_{ir} F(\boldsymbol{x}_k, \boldsymbol{\mu}_{ir}, \Sigma_{ir})} \tag{9.38}$$

where c_{im} is the ratio of the expected number of times the system is at state i using the mth mixture component to the overall expected number of times the system is at state i. Similar interpretations can be made for the other formulas too.

When the Viterbi method is employed, reestimation of the parameters is based on averages computed across the best path. For example, for mixture modeling using a single Gaussian ($L = 1$) we get

$$\boldsymbol{\mu}_i = \frac{1}{N_i} \sum_{k=1}^{N} \boldsymbol{x}_k \delta_{ik}$$

$$\Sigma_i = \frac{1}{N_i} \sum_{k=1}^{N} (\boldsymbol{x}_k - \boldsymbol{\mu}_i)(\boldsymbol{x}_k - \boldsymbol{\mu}_i)^T \delta_{ik}$$

where $\delta_{ik} = 1$ for the stages where the path goes through state i and is zero otherwise, and N_i is the respective number of times the path passes through state i.

Remarks

- The algorithms just described estimate the unknown parameters using all the available observations simultaneously. An alternative path, of major practical importance, is to employ adaptive techniques in which new information can be incorporated to adapt an already trained model, without it being necessary to retrain it with all previously used data. It is generally accepted that speaker-dependent recognizers outperform speaker-independent systems, as

long as sufficient training data are available. Thus, a long-standing idea is to use speaker-independent recognizers, trained with enough data on a multispeaker platform, and then adapt the model parameters to fit a specific speaker (and/or acoustic environment). This can be achieved by using the minimum number of data from the new speaker. Both batch and sequential schemes have been suggested. Some examples of such learning procedures are given in [Lee 91, Diga 95, Legg 95, Huo 95, Huo 97, Diga 99, Wang 01].

- A drawback of the modeling in (9.34) is that a mixture model is adopted for each of the states. This makes the number of parameters to be estimated rather high. Thus, for a given size of training data it affects the robustness of the parameter estimation. To alleviate such problems and decrease the number of unknown parameters, so-called tied-mixture densities modeling has been suggested, where the same Gaussian densities are shared across the mixtures of all the states [Bell 90] or groups of states [Diga 96, Kim 95, Gale 99, Gu 02].
- The Baum–Welch algorithm is an iterative procedure to maximize the likelihood function with respect to the unknown parameters. MAP procedures incorporating prior statistical information have also been proposed and enhanced performance has been reported [Gauv 94]. An alternative is to optimize with respect to all the unknown parameters, instead of optimizing each HMM model separately, as was the case with ML earlier. The goal of such an optimization approach is to enhance the discrimination capabilities of the models. Maximizing the mutual information [Bahl 86] or minimizing the cross-entropy [Ephr 89] and, more recently, the classification error rate [Juan 97], or controlling the influence of the outliers [Arsl 99] are examples of approaches that enhance performance at the expense of complexity.

Segment Modeling

Although HMM modeling is one of the most powerful and widely used techniques in recognition, it is not without shortcomings. One of its principal limitations is the required assumption of independence among the observations (conditioned on the state sequence). In fact, this is not true for most of the cases. Another limitation is the rather weak state duration modeling achieved by standard HMM modeling. To overcome these limitations, a number of schemes have been suggested. Recently, an effort was made to present a variety of such schemes in a unified framework, under the notion of *segment modeling*. Here only the basic definitions will be reviewed. In HMM modeling, on the arrival of a transition at a state, *a single* observation (corresponding to a single frame of the original speech samples) is assumed to be emitted and the fundamental observation distribution is *on the frame level*, that is, $p(\boldsymbol{x}|i)$. In contrast, in segment modeling a *segment* X_r^m consisting

FIGURE 9.7: HMM and segment modeling (SM) for the emission of observations upon arrival of a transition at a state.

of m frames, $X_r^m = [\boldsymbol{x}_r, \ldots, \boldsymbol{x}_{r+m-1}]$, is assumed to be emitted upon the arrival at a state. Here m is a random variable itself. The fundamental distribution is now at the segment level, that is, $p(X_r^m|i, m)$. A schematic representation is given in Figure 9.7. The parameters describing a segment model are (a) the number of states, (b) their transition modeling parameters, (c) the joint probability density function for the segment distribution, given the duration m, and (d) the duration probability $P(m|i)$. Training of these parameters follows generalizations of the Baum–Welch and Viterbi schemes. A more detailed treatment of the topic is beyond our scope and the interested reader may consult, for example, [Oste 96, Russ 97, Gold 99] and the references therein.

9.7 TRAINING MARKOV MODELS VIA NEURAL NETWORKS

The training phase of an HMM-based recognition system is entirely dedicated to the learning of probabilities (and densities). In Chapter 3 we have seen that a *supervised* classifier optimized via certain criteria, such as least squares, can approximate posterior class probabilities. This is the kickoff point for our current concern. States can be treated as classes and a multilayer perceptron can be used as a nonlinear classifier. The observations feed the input nodes and the network has as many outputs as the states. Training can be done via the backpropagation algorithm, and the desired responses will be 1 at the true state output and 0 at the others (see Chapter 3). It is now straightforward to see that the outputs of the NN will sufficiently approximate the posterior probabilities $P(i|\boldsymbol{x})$. These can then be changed to $p(\boldsymbol{x}|i)$, as required in the recognition phase of a Markov model–based recognizer, via the Bayes rule

$$p(\boldsymbol{x}|i) = \frac{P(i|\boldsymbol{x})p(\boldsymbol{x})}{P(i)}$$

where the state priors $P(i)$ are determined from their relative occurrence frequencies and $p(\boldsymbol{x})$ is constant for all states during recognition.

In following this procedure we have made a crucial assumption. That is, the states are treated here as *being observable*, and during training we know the specific

state from which each observation originates. For example, in speech recognition this is possible by associating each phoneme in a spoken word with a state. Here lies a disadvantage of this approach, since accurate segmentation of the speech signal is required and the boundaries are not always well defined. This is not the case in the HMM approach, where it is left to the algorithm to decide optimally about the state boundaries. A scheme for unified training of HMM/MLP that avoids the segmentation problem has been suggested [Koni 96]. Let us now turn to the benefits of bringing neural networks into the scene.

We have already commented that a major disadvantage of the standard HMM is the assumption of independence among the observations. Using a multilayer perceptron, the underlying statistical dependence can easily be accommodated. Figure 9.8 shows a possible way. Together with the "current" observation vector $\boldsymbol{x}_k$, p "past" as well as p "future" ones appear simultaneously at the input nodes. Thus, the input nodes amount to $(2p + 1)l$, with l being the dimension of the observation vectors. During training, the desired response will be 1 at the output node corresponding to the state that "gives birth" to the "current" vector. We say that the network is trained with *contextual input* information. Further data dependence can also be accommodated by providing the input with information about the previous state in the sequence. During recognition this is provided via an output feedback, shown in the figure by the dotted lines [Bourl 90]. Obviously, in such a configuration the output nodes of the network compute the conditional state probabilities $P(i_k|X_{k-p}^{k+p}, i_{k-1})$, where X_{k-p}^{k+p} denotes the contextual input information ranging from $\boldsymbol{x}_{k-p}$ to $\boldsymbol{x}_{k+p}$. Having these probabilities at our disposal,

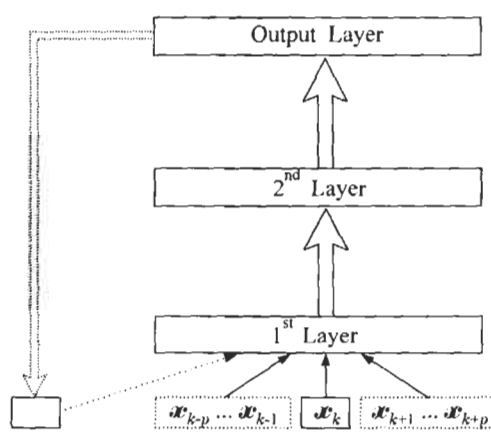


FIGURE 9.8: A multilayer perceptron architecture for training the parameters of a Markov model.

a number of new "opportunities" open to us. Let us, for example, return to our original goal in (9.1) and treat states as classes. By the chain rule we have

$$\begin{aligned} P(\Omega_i|X) &\equiv P(i_1, i_2, \ldots, i_N|X) \\ &= P(i_N|i_{N-1}, \ldots, i_1, X) \ldots P(i_{N-1}|i_{N-2}, \ldots, i_1, X)P(i_1|X) \end{aligned} \tag{9.39}$$

Taking into account the Markovian property of the state dependence and relaxing a bit the conditional constraint on the observations, this can be written as

$$P(\Omega_i|X) = \prod_k P(i_k|X_{k-p}^{k+p}, i_{k-1}) \tag{9.40}$$

with some appropriate initial conditions. Computing its maximum can easily be done via dynamic programming arguments. A number of other alternatives are also possible; see for example [Bourl 90, Bourl 94, Morg 95] for a more detailed discussion of the topic.

Finally, it must be emphasized that in order to obtain good probability estimates the size of the multilayer perceptron must be large enough to have good approximating capabilities. This, of course, requires increased computational resources for the training. Furthermore, the incorporation of the contextual information imposes its own demands on large networks. Another point is that the approximation of probabilities by the network is valid at the global minimum of the minimized cost function, at least in theory. All these practical issues affecting the overall performance of such an approach are still ongoing research topics [Spec 94].

9.8 A DISCUSSION OF MARKOV RANDOM FIELDS

So far, our concern with context-dependent classification was limited to the one-dimensional case. The current subsection is focused on the related two-dimensional generalizations. That is, observations will be treated as two-dimensional sequences $X(i, j)$. Such problems result in image processing and observations can be, for example, the gray levels of the image array pixels. No doubt, complications arise, and our aim here is to provide the basic definitions and directions and not a detailed treatment of the topic.

Let us assume that we are given an array of observations $X: X(i, j), i = 0, 1, \ldots, N_x - 1, j = 0, 1, \ldots, N_y - 1$, and a corresponding array of classes/states Ω: ω_{ij}, where each ω_{ij} can take one of M values. Once more our objective is, given the array of the observations, to estimate the corresponding values of the state array Ω so that

$$p(X|\Omega)P(\Omega) \text{ is maximum} \tag{9.41}$$

Within the scope of context-dependent classification the values of the elements of Ω will be assumed to be mutually dependent. Furthermore, we will assume that the range of this dependence is limited within a neighborhood. This brings us to the notion of Markov random fields (MRFs) defined in Chapter 7. Thus, for each (i, j) element of the array Ω a respective *neighborhood* $\mathcal{N}_{ij}$ is defined so that

- $\omega_{ij} \notin \mathcal{N}_{ij}$
- $\omega_{ij} \in \mathcal{N}_{kl} \Longleftrightarrow \omega_{kl} \in \mathcal{N}_{ij}$

In words, the (i, j) element does not belong to its own set of neighbors, and if ω_{ij} is a neighbor of ω_{kl} then ω_{kl} is also a neighbor of ω_{ij}. The Markov property is then defined as

$$P(\omega_{ij}|\bar{\Omega}_{ij}) = P(\omega_{ij}|\mathcal{N}_{ij}) \tag{9.42}$$

where $\bar{\Omega}_{ij}$ includes all the elements of Ω except the (i, j) one. Figure 9.9 gives a typical example of a neighborhood with eight neighbor pixels. Equation (9.42) is a generalization of (9.3). In the one-dimensional case the ordering of the sequence led to the relation (9.4). Unfortunately, this *sequence ordering does not generalize in a natural way* to the two-dimensional case and imposes limitations on the involvement of the computationally elegant dynamic programming techniques [Hans 82].

A seminal paper that had an impact on the use of MRF modeling in image processing and analysis was that of Geman and Geman [Gema 84]. They built upon

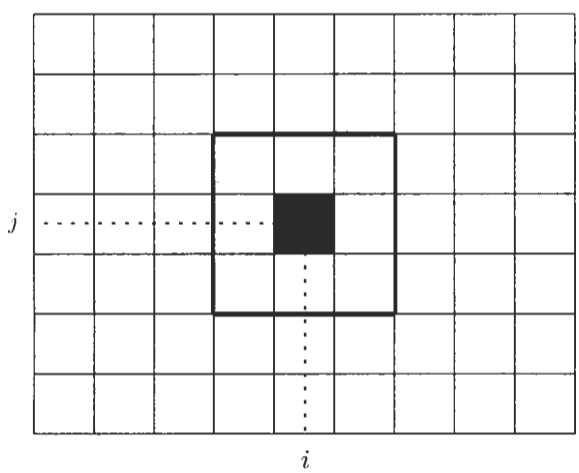


FIGURE 9.9: Example of a neighborhood involving eight neighbors of the (i, j) element.

the important Hammersley–Clifford theorem, which establishes the *equivalence* between Markov random fields and Gibbs distributions [Besa 74]. Thus, we can talk of *Gibbs random fields* (GRFs). A Gibbs conditional probability is of the form

$$P(\omega_{ij}|\mathcal{N}_{ij}) = \frac{1}{Z}\exp\left(-\frac{1}{T}\sum_{k} F_k(\mathcal{C}_k(i,j))\right) \tag{9.43}$$

where Z is a normalizing constant so that probabilities sum up to 1, T is a parameter, and $F_k(\cdot)$ are functions of the states of the pixels in the *cliques* $\mathcal{C}_k(i,j)$. A clique consists of either a single pixel or a set of pixels, which are neighbors of each other, with respect to the type of the chosen neighborhood. Figure 9.10 shows two cases of neighborhoods and the corresponding sets of cliques. A typical example of the exponent function in (9.43) for the four neighbors case is

$$-\frac{1}{T}\omega_{ij}\left(\alpha_1 + \alpha_2(\omega_{i-1,j} + \omega_{i+1,j}) + \alpha_2(\omega_{i,j-1} + \omega_{i,j+1})\right)$$

where the a_i's are constants.

It turns out that the joint probability $P(\Omega)$ for the Gibbsian model is

$$P(\Omega) = \exp\left(-\frac{U(\Omega)}{T}\right) \tag{9.44}$$

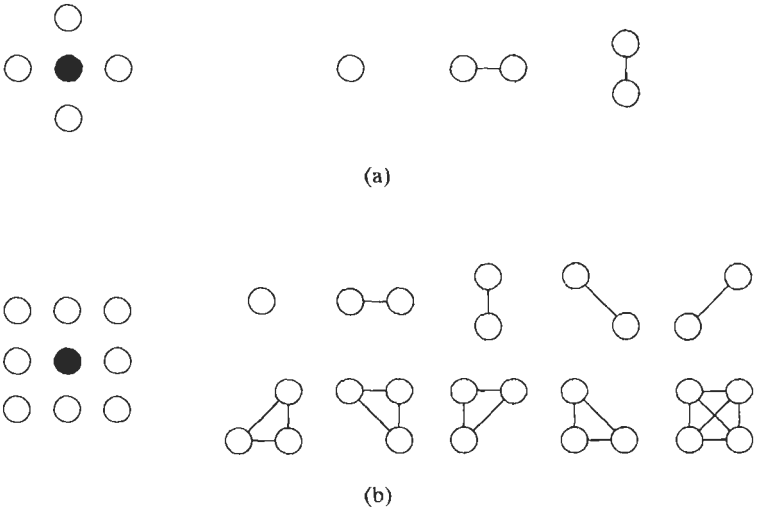


FIGURE 9.10: Two examples of neighborhoods with the corresponding cliques.

where

$$U(\Omega) = \sum_{i,j} \sum_{k} F_k(\mathcal{C}_k(i, j)) \tag{9.45}$$

that is, the sum of the functions over all possible cliques associated with the neighborhood. In many cases, the posterior probability $P(\Omega|X)$, which is to be maximized (i.e., 9.41) also turns out to be Gibbsian. Such cases result, for example, if the regions in the image are themselves generated by Markov (e.g., Gaussian 2-D AR) processes [Deri 86, Chel 85]. *Simulating annealing* techniques can then be employed to obtain the required maximum [Gema 84].

Hidden Markov generalizations to the two-dimensional plane have also been considered [Povl 95]. The idea here is to build a *pseudolikelihood function* starting from the local state transition probabilities, using Besag's method for coding the image in mutually independent pixel sets [Besa 74]. An alternative EM formulation of the problem was given in [Zhan 94]. Finally, another direction, which is gaining in popularity, is the combination of Markov random fields and multiresolution analysis. At the subsampling stage the Markov property is lost and suitable models are derived for the coarser resolutions. For more details the reader may consult, for example, [Laks 93, Bell 94, Kris 97] and the references therein.

Problems

9.1 Assume an HMM model with K states and an observation string X of N continuous observations. Assume that the pdf in each state is described by a Gaussian with known diagonal covariance matrix and unknown mean values. Using the EM algorithm, derive the reestimation recursions.

Hint: Form the complete data set as $Y = (X, \Omega)$, where Ω is the set of the states.

9.2 Show that if the self-transition probability of a state is $P(i|i)$, then the probability of the model being at state i for d successive stages is given by $P_i(d) = (P(i|i))^{d-1}(1 - P(i|i))$. Show also that the average duration for staying in state i is equal to $\bar{d} = \frac{1}{1-P(i|i)}$. In practice, this may be inappropriate, thus the need for a duration distribution HMM modeling.

9.3 In practice, a number Q of different versions of the spoken word are used for training, each resulting in a sequence of observations X_m of length N_m, $m = 1, 2, \ldots, Q$. Then comment on the following reestimation formulas:

$$\hat{P}(j|i) = \frac{\sum_{m=1}^{Q} \frac{1}{P(X_m|\mathcal{S})} \sum_{k=1}^{N_m - 1} \xi_k(i, j, X_m|S)}{\sum_{m=1}^{Q} \frac{1}{P(X_m|\mathcal{S})} \sum_{k=1}^{N_m - 1} \alpha^m(i_k = i)\beta^m(i_k = i)}$$

$$\hat{P}_{\boldsymbol{x}}(r|i) = \frac{\sum_{m=1}^{Q} \frac{1}{P(X_m|\mathcal{S})} \sum_{(k=1 \text{ and } x \to r)}^{N_m} \alpha^m(i_k = i)\beta^m(i_k = i)}{\sum_{m=1}^{Q} \frac{1}{P(X_m|\mathcal{S})} \sum_{k=1}^{N_m} \alpha^m(i_k = i)\beta^m(i_k = i)}$$

where superscript m refers to the mth observation sequence, and $\xi(i, j, X|\mathcal{S})$ is defined in (9.28).

9.4 Rederive recursions (9.31) and (9.32) in terms of the scaled versions of α, β

$$\hat{\alpha}(i_k = i) = \frac{1}{c_k}\alpha(i_k = i), \quad \hat{\beta}(i_{k+1} = i) = c_k\beta(i_{k+1} = i)$$

where $c_k = \sum_{i_k=1}^{K_s} \alpha(i_k)$.

9.5 Assume that the HMM models are not equiprobable and let Λ be the set of all the unknown parameters for the M available models. Assume now that a training string X is known to correspond to the model $\mathcal{S}_r$. However, during training, maximization of $P(\mathcal{S}_r|X, \Lambda)$ is done with respect to all the parameters and not only those of the specific model. Show that this optimization leads to a maximum ratio between the contribution $p(X|\mathcal{S}_r, \Lambda)P(\mathcal{S}_r)$ of the correct model and $\sum_{s\neq r} p(X|\mathcal{S}_s, \Lambda)P(\mathcal{S}_s)$ of the incorrect models. That is, optimization with respect to all the parameters offers maximum discrimination power.

References

[Agaz 93] Agazi O.E., Kuo S.S. "Hidden Markov model based optical character recognition in the presence of deterministic transformations," *Pattern Recognition*, Vol. 26, pp. 1813–1826, 1993.

[Anto 97] Anton-Haro C., Fonollosa J.A.R., Fonollosaa J.R. "Blind channel estimation and data detection using HMM," *IEEE Transactions on Signal Processing*, Vol. 45(1), pp. 241–247, 1997.

[Arsl 99] Arslan L. Hansen J.H.L. "Selective training for hidden Markov models with applications to speech classification," *IEEE Transactions on Speech and Audio Processing*, Vol. 7(1), pp. 46–54, 1999.

[Bahl 86] Bahl L.R., Brown B.F., Desouza P.V. "Maximum mutual information estimation of hidden Markov model parameters for speech recognition," *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, Vol. 1, pp. 872–875, Japan, 1986.

[Bake 75] Baker J. "The DRAGON system—an overview," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 23(1), pp. 24–29, 1975.

[Baum 67] Baum L.E., Eagon J.A. "An inequality with applications to statistical prediction for functions of Markov processes and to a model for ecology," *Bulletin of the American Mathematical Society*, Vol. 73, pp. 360–362, 1967.

[Baum 68] Baum L.E., Sell G.R. "Growth functions for transformations of manifolds," *Pacific Journal of Mathematics*, Vol. 27, pp. 211–227, 1968.

[Baum 70] Baum L.E., Petrie T., Soules G., Weiss N. "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Annals of Mathematical Statistics*, Vol. 41, pp. 164–171, 1970.

[Bell 90] Bellegarda J.R., Nahamoo D. "Tied mixture continuous parameter modeling for speech recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 38(12), pp. 2033–2045, 1990.

[Bell 94] Bello M.G. "A combined Markov random field and wave-packet approach to image segmentation," *IEEE Transactions on Image Processing*, Vol. 3(6), pp. 834–847, 1994.

[Besa 74] Besag J. "Spatial interaction and the statistical analysis of lattice systems," *J. Royal Stat. Soc. B*, Vol. 36(2), pp.192–236, 1974.

[Bourl 90] Bourland H., Wellekens C.J. "Links between Markov models and the multilayer perceptrons," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12(12), pp. 1167–1178, 1990.

[Bourl 94] Bourland H., Morgan N. *Connectionist Speech Recognition*. Kluwer Academic Publishers, 1994.

[Chel 85] Chellapa R., Kashyap R.L. "Texture synthesis using 2-D noncausal autoregressive models," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 33(1), pp. 194–203, 1985.

[Chen 95] Chen M.Y., Kundu A., Srihari S.N. "Variable duration HMM and morphological segmentation for handwritten word recognition," *IEEE Transactions on Image Processing*, Vol. 4(12), pp. 1675–1689, 1995.

[Chen 95a] Chen J.-L., Kundu A. "Unsupervised texture segmentation using multichannel decomposition and hidden Markov models," *IEEE Transactions on Image Processing*, Vol. 4(5), pp. 603–620, 1995.

[Dell 93] Deller J., Proakis J., Hansen J. *Discrete Time Processing of Speech Signals*. Macmillan, 1993.

[Deng 94] Deng L., Aksmanovic M. "Speaker-independent phonetic classification using HMM with mixtures of trend functions," *IEEE Transactions on Speech and Audio Processing*, Vol. 5(4), pp. 319–324, 1997.

[Deri 86] Derin H. "Segmentation of textured images using Gibb's random fields," *Computer Vision, Graphics, and Image Processing*, Vol. 35, pp. 72–98, 1986.

[Diga 93] Digalakis V., Rohlicek J.R., Ostendorf M. "ML estimation of a stochastic linear system with the EM algorithm and its application to speech recognition," *IEEE Transactions on Speech and Audio Processing*, Vol. 1(4), pp. 431–441, 1993.

[Diga 95] Digalakis V., Rtischef D., Neumeyer L.G. "Speaker adaptation using constrained estimation of Gaussian mixtures," *IEEE Transaction on Speech and Audio Processing*, Vol. 3(5), pp. 357–366, 1995.

[Diga 96] Digalakis V., Monaco P., Murveit H. "Genones: Generalized mixture tying in continuous HMM model-based speech recognizers," *IEEE Transactions on Speech and Audio Processing*, Vol. 4(4), pp. 281–289, 1996.

[Diga 99] Digalakis V. "Online adaptation of hidden Markov models using incremental estimation algorithms," *IEEE Transactions on Speech and Audio Processing*, Vol. 7(3), pp. 253–261, 1999.

[ElYa 99] El-Yacoubi A., Gilloux M., Sabourin R., Suen C.Y. "An HHM-based approach for off-line unconstrained handwritten word modeling and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21(8), pp. 752–760, 1999.

[Ephr 89] Ephraim Y., Dembo A., Rabiner L.R. "A minimum discrimination information approach to hidden Markov modelling," *IEEE Transactions on Information Theory*, Vol. 35, pp. 1001–1023, September 1989.

[Fu 82] Fu K.S. *Syntactic Pattern Recognition and Applications*, Prentice Hall, 1982.

[Gale 99] Gales M.J.F. "Semitied covariance matrices for hidden Markov models," *IEEE Transactions on Speech and Audio Processing*, Vol. 7(3), pp. 272–281, 1999.

[Gauv 94] Gauvain J.L., Lee C.H. "Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains," *IEEE Transactions on Speech and Audio Processing*, Vol. 2(2), pp. 291–299, 1994.

[Gema 84] Geman S., Geman D. "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6(6), pp. 721–741, 1984.

[Geor 97] Georgoulakis C., Theodoridis S. "Efficient clustering techniques for channel equalization in hostile environments," *Signal Processing*, Vol. 58, pp. 153–164, 1997.

[Geor 98] Georgoulakis C., Theodoridis S. "Blind equalization for nonlinear channels via hidden Markov modeling," *Proceedings EUSIPCO-98*, Rhodes, Greece, 1998.

[Gold 99] Goldberger J., Burshtein D., Franco H. "Segmental modeling using a continuous mixture of noparametric models," *IEEE Transactions on Speech and Audio Processing*, Vol. 7(3), pp. 262–271, 1999.

[Gu 02] Gu L., Rose K. "Substate tying with combined parameter training and reduction in tied-mixture HMM design," *IEEE Transactions on Speech and Audio Processing*, Vol. 10(3), 2002.

[Hans 82] Hansen F.R., Elliot H. "Image segmentation using simple Markov field models," *Computer Graphics and Image Processing*, Vol. 20, pp. 101–132, 1982.

[Huo 95] Huo Q., Chan C., Lee C.H. "Bayesian adaptive learning of the parameters of hidden Markov model for speech recognition," *IEEE Transactions on Speech and Audio Processing*, Vol. 3(5), pp. 334–345, 1995.

[Huo 97] Huo Q., Lee C.H. "On-line adaptive learning of the continuous density HMM based on approximate recursive Bayes estimate," *IEEE Transactions on Speech and Audio Processing*, Vol. 5(2), pp. 161–173, 1997.

[Jeli 76] Jelinek F. "Continuous speech recognition by statistical methods," *Proceedings of the IEEE*, Vol. 64(4), pp. 532–555, 1976.

[Juan 85] Juang B.H. "Maximum likelihood estimation for mixture multivariate stochastic observations of Markov chains," *AT&T System Technical Journal*, Vol. 64, pp. 1235–1249, July–August 1985.

[Juan 86] Juang B.H., Levinson S.E., Sondhi M.M. "Maximum likelihood estimation for multivariate mixture observations of Markov chains," *IEEE Transactions on Information Theory*, Vol. IT-32, pp. 307–309, March 1986.

[Juan 97] Juang B.H., Chou W., Lee C.H. "Minimum classification error rate methods for speech recognition," *IEEE Transactions on Speech and Audio Processing*, Vol. 5(3), pp. 257–266, 1997.

[Kale 94] Kaleh G.K., Vallet R. "Joint parameter estimation and symbol detection for linear and nonlinear unknown channels," *IEEE Transactions on Communications*, Vol. 42(7), pp. 2406–2414, 1994.

[Kim 95] Kim N.S., Un C.K. "On estimating robust probability distribution in HMM-based speech recognition," *IEEE Transactions on Speech and Audio Processing*, Vol. 3(4), pp. 279–286, 1995.

[Koni 96] Konig Y. "REMAP: Recursive estimation and maximization of a-posteriori probabilities in transition-based speech recognition," *Ph.D. thesis*, University of California at Berkeley, 1996.

[Kops 02] Kopsinis Y., Theodoridis S. "A novel cluster-based MLSE equalizer for 2-PAM systems," *Proceedings of EUSIPCO-2002*, EUSIPCO, Toulouse, France, 2002.

[Kris 97] Krishnamachari S., Chellappa R. "Multiresolution Gauss–Markov random field models for texture segmentation," *IEEE Transactions on Image Processing*, Vol. 6(2), pp. 251–268, 1997.

[Laks 93] Lakshmanan S., Derin H. "Gaussian Markov random fields at multiple resolutions," in *Markov Random Fields: Theory and Applications* (R. Chellappa, ed.), Academic Press, 1993.

[Lee 72] Lee C.H., Fu K.S. "A stochastic syntax analysis procedure and its application to pattern recognition," *IEEE Transactions on Computers*, Vol. 21, pp. 660–666, 1972.

[Lee 91] Lee C.H., Lin C.H., Juang B.H. "A study on speaker adaptation of the parameters of continuous density hidden Markov models," *IEEE Transactions on Signal Processing*, Vol. 39(4), pp. 806–815, 1991.

[Legg 95] Leggetter C.J., Woodland P.C. "Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models," *Comput. Speech Lang.*, Vol. 9, pp. 171–185, 1995.

[Levi 83] Levinson S.E., Rabiner L.R., Sondhi M.M. "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," *Bell System Technical Journal*, Vol. 62(4), pp. 1035–1074, April 1983.

[Li 00] Li X., Parizeau M., Plamondon R. "Training hidden Markov models with multiple observations-A combinatorial method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22(4), pp. 371–377, 2000.

[Lipo 82] Liporace L.A. "Maximum likelihood estimation for multivariate observations of Markov sources," *IEEE Transactions on Information Theory*, Vol. IT-28(5), pp. 729–734, 1982.

[Moon 96] Moon T. "The expectation maximization algorithm," *Signal Processing Magazine*, Vol. 13(6), pp. 47–60, 1996.

[Morg 95] Morgan N. Boulard H. "Continuous speech recognition," *Signal Processing Magazine*, Vol. 12(3), pp. 25–42, 1995.

[Oste 96] Ostendorf M., Digalakis V., Kimball O. "From HMM's to segment models: A unified view of stochastic modeling for speech," *IEEE Transactions on Audio and Speech Processing*, Vol. 4(5), pp. 360–378, 1996.

[Papo 91] Papoulis A. *Probability Random Variables and Stochastic Processes*, 3rd ed., McGraw-Hill 1991.

[Pikr 02] Pikrakis A., Theodoridis., Kamarotos D. "Recognition of isolated musical patterns using HMM," Proceedings of the *2nd International Conference on Music and Artificial Intelligence (ICMAI)*, Edinburg-Scotland, September 12–14, 2002. Reprinted in *Lecture Notes in Computer Science and AI*, Vol. 2445, Springer Verlag.

[Pori 82] Poritz A.B. "Linear predictive HMM and the speech signal," *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pp. 1291–1294, Paris, 1982.

[Povl 95] Povlow B., Dunn S. "Texture classificarion using noncausal hidden Markov models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17(10), pp. 1010–1014, 1995.

[Proa 89] Proakis J. *Digital Communications*, 2nd ed., McGraw-Hill, 1989.

[Rabi 89] Rabiner L. "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of IEEE*, Vol. 77, pp. 257–285, Feb. 1989.

[Russ 97] Russell M., Holmes W. "Linear trajectory segmental HMM's," *IEEE Signal Processing Letters*, Vol. 4(3), pp. 72–75, 1997.

[Spec 94] Special issue on neural networks for speech in *IEEE Transactions on Speech and Audio Processing*, Vol. 2(1), 1994.

[Theo 95] Theodoridis S., Cowan C.F.N., See C.M.S. "Schemes for equalization of communication channels with nonlinear impairments," *IEE Proceedings on Communications*, Vol. 142(3), pp. 165–171, 1995.

[Turi 98] Turin W. "Unidirectional and parallel Baum-Welch algorithms," *IEEE Transactions on Speech and Audio Processing*, Vol. 6(6), pp. 516–523, 1998.

[Vite 67] Viterbi A.J. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, Vol. 13, pp. 260–269, 1967.

[Vlon 92] Vlontzos J.A., Kung S.Y. "Hidden Markov models for character recognition," *IEEE Transactions on Image Processing*, Vol. 1(4), pp. 539–543, 1992.

[Wang 01] Wang S., Zhao Y. "Online Bayesian tree-structured transformation of HMM's with optimal model selection for speaker adaptation," *IEEE Transactions on Speech and Audio Processing*, Vol. 9(6), pp. 663–677, 2001.

[Wu 96] Wu W.R., Wei S.C. "Rotational and gray scale transform invariant texture classification using spiral resampling, subband decomposition, and hidden Markov model," *IEEE Transactions on Image Processing*, Vol. 5(10), pp. 1423–1435, 1996.

[Zhan 94] Zhang J., Modestino J.W., Langan D.A. "Maximum likelihood parameter estimation for unsupervised stochastic model based image segmentation," *IEEE Transactions on Image Processing*, Vol. 3(4), pp. 404–421, 1994.

CHAPTER 10

SYSTEM EVALUATION

10.1 INTRODUCTION

This chapter is focused on the last stage of the design procedure of a classification system. In other words, we assume that an optimal classifier has been designed, based on a selected set of training feature vectors. Our goal now is to evaluate its performance with respect to the probability of classification error associated with the designed system. Once the estimated error is considered satisfactory, full evaluation of the system performance is carried out in the real environment for which the system has been designed, such as a hospital for a medical diagnosis system or a factory for an industrial production–oriented system.

It is important to note that the evaluation stage is not cut off from the previous stages of the design procedure. On the contrary, it is an integral part of the procedure. The evaluation of the system's performance will determine whether the designed system complies with the requirements imposed by the specific application and intended use of the system. If this is not the case, the designer may have to reconsider and redesign parts of the system. Furthermore, the misclassification probability can also be used as a performance index, in the feature selection stage, to choose the best features associated with a specific classifier.

This chapter will evolve in two directions. First, methodologies will be developed for the estimation of the classification error probability, using the available, hence finite, set of data. Then we will adopt a case study and we will design a classification system, following the steps and techniques that have accompanied us up to now.

10.2 ERROR COUNTING APPROACH

Let us consider an M-class classification task. Our objective is to *estimate* the classification error probability, by testing the "correct/false" response of an independently designed classifier using a *finite set* of N test feature vectors. Let N_i be the vectors in each class, with $\sum_{i=1}^{M} N_i = N$ and P_i the corresponding error probability for class ω_i. *Assuming independence among the feature vectors*, the probability of k_i vectors from class ω_i being misclassified is given by the

binomial distribution

$$\text{prob}\{k_i \textit{ misclassified}\} = \begin{pmatrix} N_i \\ k_i \end{pmatrix} P_i^{k_i}(1 - P_i)^{N_i - k_i} \tag{10.1}$$

In our case the probabilities P_i are not known. An estimate $\hat{P}_i$ results if we maximize (10.1) with respect to P_i. Differentiating and equating to zero result in our familiar estimate

$$\hat{P}_i = \frac{k_i}{N_i} \tag{10.2}$$

Thus, the total error probability estimate is given by

$$\hat{P} = \sum_{i=1}^{M} P(\omega_i)\frac{k_i}{N_i} \tag{10.3}$$

where $P(\omega_i)$ is the occurrence probability of class ω_i. We will now show that $\hat{P}$ is an unbiased estimate of the true error probability. Indeed, from the properties of the binomial distribution (Problem 10.1) we have

$$E[k_i] = N_i P_i \tag{10.4}$$

which leads to

$$E[\hat{P}] = \sum_{i=1}^{M} P(\omega_i) P_i \equiv P \tag{10.5}$$

that is, the true error probability. To compute the respective variance of the estimator, we recall from Problem 10.1 that

$$\sigma_{k_i}^2 = N_i P_i (1 - P_i) \tag{10.6}$$

leading to

$$\sigma_{\hat{P}}^2 = \sum_{i=1}^{M} P^2(\omega_i)\frac{P_i(1 - P_i)}{N_i} \tag{10.7}$$

Thus, the error probability estimator in (10.3), which results from simply counting the errors, is unbiased but only asymptotically consistent as $N_i \to \infty$. *Thus, if small data sets are used for testing the performance of a classifier, the resulting estimate may not be reliable.*

In [Guyo 98] the minimum size of the test data set, N, is derived in terms of the true error probability P of the already designed classifier. The goal is to estimate

N so that to guarantee, with probability $1-a$, $0 \leq a \leq 1$, that P does not exceed the estimated from the test set, $\hat{P}$, by an amount larger than $\epsilon(N, a)$, that is

$$\text{prob}\{P \geq \hat{P} + \epsilon(N, a)\} \leq a \tag{10.8}$$

Let $\epsilon(N, a)$ be expressed as a function of P, that is, $\epsilon(N, a) = \beta P$. An analytical solution for Eq. (10.8) with respect to N is not possible. However, after some approximations certain bounds can be derived. For our purposes, it suffices to consider a simplified formula, which is valid for typical values of a and β ($a = 0.05, \beta = 0.2$),

$$N \approx \frac{100}{P} \tag{10.9}$$

In words, if we want to guarantee, with a risk a of being wrong, that the error probability P will not exceed $\frac{\hat{P}}{1-\beta}$, then N must be of the order given in Eq. (10.9). For $P = 0.01$, $N = 10000$ and for $P = 0.03$, $N = 3000$. Note that this result is independent of the number of classes. Furthermore, if the samples in the test data set are not independent this number must be further increased. Such bounds are also of particular importance, if the objective is to determine the size N of the test data set that provides good confidence in the results, when comparing different classification systems with relatively small differences in their error probabilities.

Although the error counting approach is by far the most popular one, other techniques have also been suggested in the literature. These techniques estimate the error probability by using smoother versions of the discriminant function(s) realized by the classifier. The error counting approach can be thought of as an extreme case of a hard limiter, where a 1 or 0 is produced and counted, depending on the discriminant function's response, that is, whether it is false or true, respectively; see, for example, [Raud 91].

10.3 EXPLOITING THE FINITE SIZE OF THE DATA SET

The estimation of the classification error probability presupposes that one has decided upon the data set to which the error counting will be applied. This is not a straightforward task. The set of samples that we have at our disposal is finite and it has to be utilized for both training and testing. Can we use the same samples for training and testing? If not, what are the alternatives? Depending on the answer to the question, the following methods have been suggested:

- *Resubstitution Method:* The same data set is used, first for training and then for testing. One need not go into mathematical details in order to see that such a procedure is not very fair. Indeed, this is justified by the

mathematical analysis. In [Fole 72] the performance of this method was analyzed using normal distributions. The analysis results show that this method provides an *optimistic* estimate of the true error probability. The amount of bias of the resubstitution estimate is a function of the ratio N/l, that is, the data set size and the dimension of the feature space. Furthermore, the variance of the estimate is inversely proportional to the data set size N. In words, *in order to obtain a reasonably good estimate, N as well as the ratio N/l must be large enough.* The results from the analysis and the related simulations show that N/l should be at least three and that an upper bound of the variance is $1/8N$. Of course, if this technique is to be used in practice, where the assumptions of the analysis are not valid, experience suggests that the suggested ratio must be even larger [Kana 74]. Once more, the larger the ratio N/l, the more comfortable one feels.

- *Holdout Method:* The available data set is divided into two subsets, one for training and one for testing. The major drawback of this technique is that it reduces the size for both the training and the testing data. Another problem is to decide how many of the N available data will be allocated to the training set and how many to the test set. This is an important issue. In Section 3.5.3 of Chapter 3, we saw that designing a classifier using a finite data set introduces an excess mean error and a variance around it, as different data sets, of the same size, are used for the design. Both these quantities depend on the size of the training set. In [Raud 91], it is shown that the classification error probability of a classifier, designed using a finite training data set, N, is always higher than the corresponding asymptotic error probability ($N \rightarrow \infty$). This excess error decreases as N increases. On the other hand, in our discussion in the previous section we saw that the variance of the error counting depends on the size of the test set, and for small test data sets the estimates can be *unreliable*. Efforts made to optimize the respective sizes of the two sets have not yet led to practical results.
- *Leave-One-Out Method:* This method [Lach 68] alleviates the lack of independence between the training and test sets in the resubstitution method and at the same time frees itself from the dilemma associated with the holdout method. The training is performed using $N - 1$ samples, and the test is carried out using the excluded sample. If this is misclassified an error is counted. This is repeated N times, each time excluding a *different* sample. The total number of errors leads to the estimation of the classification error probability. Thus, training is achieved using, basically, all samples, and at the same time independence between training and test sets is maintained. The major drawback of the technique is its high computational complexity. For certain types of classifiers (i.e., linear or quadratic) it turns out that a simple relation exists between the leave-one-out and the resubstitution

method ([Fuku 90], Problem 10.2). Thus, in such cases the former estimate is obtained using the latter method with some computationally simple modifications.

The estimates resulting from the holdout and leave-one-out methods turn out to be very similar, for comparable sizes of the test and training sets. Furthermore, it can be shown (Problem 10.3, [Fuku 90]) that the holdout error estimate, for a Bayesian classifier, is an upper bound of the true Bayesian error. In contrast, the resubstitution error estimate is a lower bound of the Bayesian error, confirming our previous comment that it is an optimistic estimate. To gain further insight into these estimates and their relation, let us make the following definitions:

- P_e^N denotes the classification error probability for a classifier designed using a finite set of N training samples.
- $\bar{P}_e^N$ denotes the average $E[P_e^N]$ over all possible training sets of size N.
- P_e is the average asymptotic error as $N \to \infty$.

It turns out that the holdout and leave-one-out methods (for statistically independent samples) provide an *unbiased* estimate of $\bar{P}_e^N$. In contrast, the resubstitution method provides a biased (underestimated) estimate of $\bar{P}_e^N$. Figure 10.1 shows the trend of a typical plot of $\bar{P}_e^N$ and the average (over all possible sets of size N) resubstitution error as functions of N [Fole 72, Raud 91]. It is readily observed that as the data size N increases both curves tend to approach the asymptotic P_e.

A number of variations and combinations of these basic schemes have also been suggested in the literature. For example, a variation of the leave-one-out method is

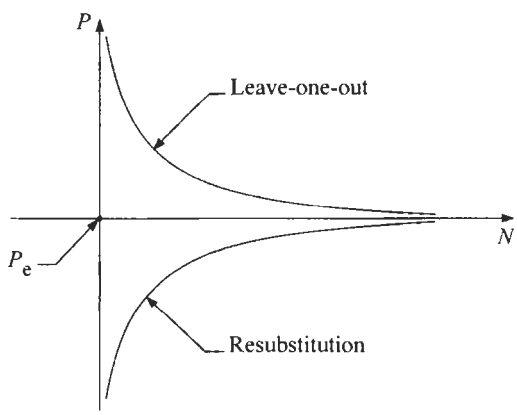


FIGURE 10.1: Plots indicating the general trend of the average resubstitution and leave-one-out error probabilities as functions of the number of training points.

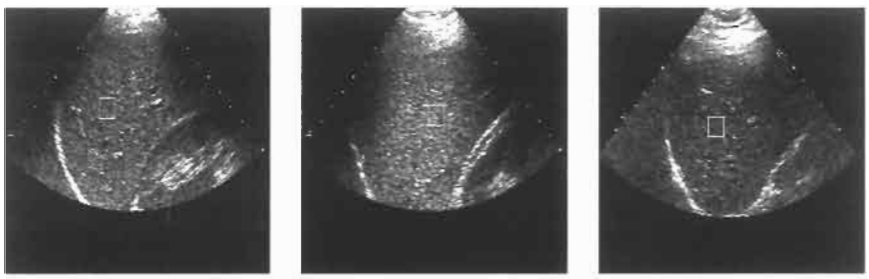
to leave $k > 1$, instead of one, samples out. The design and test process is repeated for all distinct choices of k samples. References [Kana 74, Raud 91] are two good examples of works discussing various aspects of the topic.

In [Leis 98] a method called *cross-validation with active pattern selection* is proposed, with the goal to reduce the high computational burden required by the leave-one-out method. It is suggested not to leave out (one at a time) all N feature vectors, but only $k < N$. To this end the "good" points of the data set (expected to contribute a 0 to the error) are not tested. Only the k "worst" points are considered. The choice between "good" and "bad" is based on the respective values of the cost function after an initial training. This method exploits the fact that the outputs of the classifier, trained according to the least squares cost function, approximate posterior probabilities, as discussed in Chapter 3. Thus, those feature vectors whose outputs have a large deviation from the desired value of 1 (for the true class) are expected to be the ones that contribute to the classification error.

Another set of techniques have been developed around the *bootstrap method* [Efro 79, Hand 86, Jain 87]. A major incentive for the development of these techniques is the variance of the leave-one-out method estimate for small data sets [Efro 83]. According to the "bootstrap" philosophy, new data sets are artificially generated. This is a way to overcome the limited number of available data and create more data in order to better assess the statistical properties of an estimator. Let X be the set of the available data of size N. A *bootstrap design sample set of size* N, X^*, is formed by *random* sampling with *replacement* of the set X. Replacement means that when a sample, say $\boldsymbol{x}_i$, is "copied" to the set X^*, it is not removed from X but is reconsidered in the next sampling. A number of variants have been built upon the bootstrap method. A straightforward one is to design the classifier using a bootstrap sample set and count the errors using the samples from X that do not appear in this bootstrap sample set. This is repeated for different bootstrap sample sets. The error rate estimate is computed by counting all the errors and dividing the sum by the total number of test samples used. However, in [Raud 91] it is pointed out that the bootstrap techniques improve on the leave-one-out method only when the classification error is large.

10.4 A CASE STUDY FROM MEDICAL IMAGING

Our goal in this section is to demonstrate the various design stages, discussed in the previous chapters, via a case study borrowed from a real application. It will not come as a surprise to say that focusing on a single example cannot cover all possible design approaches that are followed in practice. However, our aim is to provide a flavor for the newcomer. After all, "perfection is the enemy of the good."



(a) (b) (c)

FIGURE 10.2: Ultrasound images corresponding to (a) normal liver, (b) liver with fatty infiltration, and (c) liver with cirrhosis. The square shows the image area on which the analysis was carried out.

Our chosen application comes from the medical imaging discipline. Our task is to develop a pattern recognition system for the diagnosis of certain liver diseases. Specifically, the system will be presented with *ultrasound* images of the liver and it must be able to recognize *normal* from *abnormal* cases. Abnormal cases correspond to two types of liver diseases, namely *cirrhosis* and *fatty liver infiltration*. For each case, two different gratings must be recognized, depending on the degree of the disease development [Cavo 97]. Figure 10.2 shows three examples of ultrasound images corresponding to (a) a normal liver, (b) an abnormal liver with fatty infiltration, and (c) an abnormal liver with cirrhosis. It is readily realized that the visual differences between the images are not great. This makes the clinical diagnosis and the diagnostic accuracy very much dependent on the skill of the doctor. Thus, the development of a pattern recognition system can assist the doctor in assessing the case and, together with other clinical findings, reduce the need for invasive techniques (biopsy).

The first stage in the design process involves the close cooperation of the system designer with the specialist, that is, the doctor, in order to establish a "common language" and have the designer *understand* the task and define, in common with the doctor, the goals and requirements of the pattern recognition system. Besides the acceptable error rate, other performance issues come into play, such as complexity, computational time, and cost of the system. The next stage involves various image processing steps, such as image enhancement, in order to assist the system by presenting it only useful information as much as possible. Then things are ripe to begin with the design of the pattern recognition system.

Figure 10.3 outlines the task. There are five possible classes. The pattern recognition system can be designed either around a single classifier, which assigns an unknown image directly to one of the five classes, or around a number of

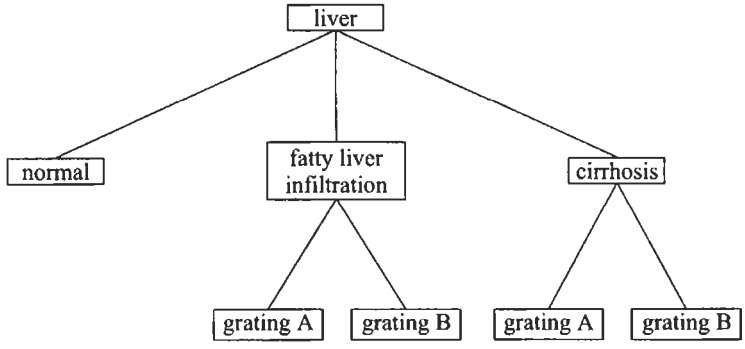


FIGURE 10.3: The classification task.

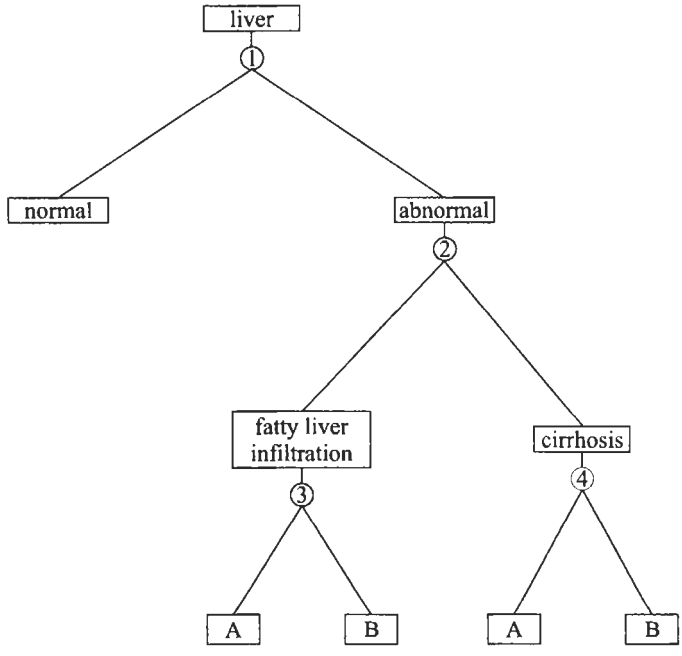


FIGURE 10.4: A tree-structured hierarchy of classifiers.

classifiers built upon a tree structure philosophy. The latter approach was adopted here. Figure 10.4 illustrates the procedure. A separate classifier was used at each node, and each of them performs a two-class decision. At the first node the respective classifier decides between normal and abnormal cases. At the second node, images, classified as abnormal, are tested and classified in either the cirrhosis or

the fatty liver infiltration class, and so on. The advantage of such a procedure is that we break the problem into a number of simpler ones. It must be stressed, however, that in other applications such a procedure may not be applicable. For the design of the classification system, 150 ultrasound liver images were obtained from a medical center. Fifty of them correspond to normal cases, 55 of them to patients suffering from cirrhosis, and 45 of them to patients suffering from fatty liver infiltration. Three classifiers were adopted for comparison, namely the least squares linear classifier, the minimum Euclidean distance classifier, and the kNN for different values of k. Each time, the same type of classifier was used for all nodes. *From the discussions with the specialists*, we concluded that what was of interest here was the texture of the respective images. The methods described in Section 7.2.1 of Chapter 7 were used and a total of 38 features were generated for each of the images. This is a large number, and a feature selection procedure was "mobilized" to reduce this number. Let us first concentrate on the first node classification task and the LS linear classifier.

- For each of the 38 features the t test was applied and only 19 of them passed the test at a significance level of 0.001. The latter is chosen so that "enough" features pass the test. Taking into account the size of the problem, enough was considered to be around 15 for our problem. However, 19 is still a large number, and a further reduction was considered necessary. For example, 19 is of the same order as 50 (the number of normal patterns), which would lead to poor generalization.
- The 19 features were considered in pairs, in triples, up to groups of seven, in all possible combinations. For each combination, the optimal LS classifier was designed and each time the corresponding classification error rate was estimated, using the leave-one-out method. It turned out that taking the features in groups larger than two did not improve the error rate significantly. Thus, it was decided that $l = 2$ was satisfactory and the best combination consisted of the *kurtosis* and the ASM. The percentage of correct classification with this combination was 92.5%.

For the design of the linear classifier of "node 2" the same procedure was followed, using, of course, only the images originated from abnormal cases. Of the 38 originally produced features, only 15 passed the t test. The optimal combination of features turned out to be the mean, the variance, and the correlation. It may be worth pointing out that the variance was rejected from the t test during the design of the "node 1" classifier. The percentage of correct classification for node 2 was 90.1%. The optimal combination for the "node 3" LS classifier was the variance, entropy, the sum entropy, and the difference entropy corresponding to a correct classification rate of 92.2%. Finally, the optimization procedure for the "node 4" classifier resulted in the mean value, the ASM, and the contrast with a correct classification rate estimate of 83.8%.

Having completed the design with the LS linear classifiers, the same procedure was followed for the Euclidean minimum distance classifier and the kNN classifier. However, in both of these cases the resulting error rate estimates were always higher than the ones obtained with the LS classifier. Thus, the latter one was finally adopted.

Once more, it must be stated that this case study does not and cannot represent the wealth of classification tasks encountered in practice, each with its own specific requirements. We could state, with a touch of exaggeration, of course, that each classification task is like a human being. *Each one has its own personality*! For example, the dimension of our problem was such that it was computationally feasible, with today's technology to follow the procedure described. The feature selection, classifier design, and classification error stages were combined to compute the best combination. This was also a motivation for choosing the specific case study, that is, to demonstrate that the various stages in the design of a classification system are not independent but they can be closely interdependent. However, this may not be possible for a large number of tasks, as, for example, the case of a large multilayer neural network in a high-dimensional feature space. Then the feature selection stage cannot be easily integrated with that of classifier design, and techniques such as those presented in Chapter 5 must be employed. Ideally, what one should aim at is to have a procedure to design the classifiers by minimizing the error probability directly (not the LS, etc.), and at the same time this procedure should be computationally simple (!) to allow also for a search for the optimal feature combination. However, this "utopia" is still quite distant.

Problems

10.1 Let P be the probability that event A occurs. The probability that event A occurs k times in a sequence of N independent experiments is given by the binomial distribution

$$\binom{N}{k} P^k (1-P)^{N-k}$$

Show that $E[k] = NP$ and $\sigma_k^2 = NP(1-P)$.

10.2 In a two-class problem the classifier to be used is the minimum Euclidean distance one. Assume N_1 samples from class ω_1 and N_2 from class ω_2. Show that the leave-one-out method estimate can be obtained from the resubstitution method, if the distance of $\boldsymbol{x}$ from the class means $d_i(\boldsymbol{x}), i = 1, 2$, are modified as $d_i'(\boldsymbol{x}) = (\frac{N_i}{N_i - 1})^2 d_i(\boldsymbol{x})$, if $\boldsymbol{x}$ belongs to class i. Furthermore, show that in this case the leave-one-out method always results in larger error estimates than the resubstitution method.

10.3 Show that for the Bayesian classifier, the estimate provided by the resubstitution method is a lower bound of the true error and that computed from the holdout method is an upper bound.

References

[Cavo 97] Cavouras D., et al. "Computer image analysis of ultrasound images for discriminating and grating liver parenchyma disease employing a hierarchical decision tree scheme and the multilayer perceptron classifier," *Proceedings of Medical Informatics Europe '97*, pp. 517–521, 1997.

[Efro 79] Efron B. "Bootstrap methods: Another look at the jackknife," *Annals of Statistics*, Vol. 7, pp. 1–26, 1979.

[Efro 83] Efron B. "Estimating the error rate of a prediction rule: Improvement on cross-validation," *Journal of the American Statistical Association*, Vol. 78, pp. 316–331, 1983.

[Fole 72] Foley D. "Consideration of sample and feature size," *IEEE Transactions on Information Theory*, Vol. 18(5), pp. 618–626, 1972.

[Fuku 90] Fukunaga K. *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press, 1990.

[Guyo 98] Guyon I., Makhoul J., Schwartz R., Vapnik U. "What size test set gives good error rate estimates?" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20(1), pp. 52–64, 1998.

[Hand 86] Hand D.J. "Recent advances in error rate estimation," *Pattern Recognition Letters*, Vol. 5, pp. 335–346, 1986.

[Jain 87] Jain A.K., Dubes R.C., Chen C.C. "Bootstrap techniques for error estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 9(9), pp. 628–636, 1987.

[Kana 74] Kanal L. "Patterns in Pattern Recognition," *IEEE Transactions on Information Theory*, Vol. 20(6), pp. 697–722, 1974.

[Lach 68] Lachenbruch P.A., Mickey R.M. "Estimation of error rates in discriminant analysis," *Technometrics*, Vol. 10, pp. 1–11, 1968.

[Leis 98] Leisch F., Jain L.C., Hornik K. "Cross-validation with active pattern selection for neural network classifiers," *IEEE Transactions on Neural Networks*, Vol. 9(1), pp. 35–41, 1998.

[Raud 91] Raudys S.J., Jain A.K. "Small size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13(3), pp. 252–264, 1991.

CHAPTER 11

CLUSTERING: BASIC CONCEPTS

11.1 INTRODUCTION

All the previous chapters were concerned with supervised classification. In the current and following chapters the focus of interest is turned to the unsupervised case, where class labelling of the training patterns is not available. Thus, our major concern now becomes to "reveal" the organization of patterns into "*sensible*" clusters (groups), which will allow us to discover similarities and differences among patterns and to derive useful conclusions about them. This idea is met in many fields, such as life sciences (biology, zoology), medical sciences (psychiatry, pathology), social sciences (sociology, archeology), earth sciences (geography, geology), and engineering [Ande 73]. Clustering may be found under different names in different contexts, such as unsupervised learning and learning without a teacher (in pattern recognition), numerical taxonomy (in biology, ecology), typology (in social sciences), and partition (in graph theory). The following example is inspired by biology and gives us a flavor of the problem.

Consider the following animals: sheep, dog, cat (mammals), sparrow, seagull (birds), viper, lizard (reptiles), goldfish, red mullet, blue shark (fish), and frog (amphibians). In order to organize these animals into clusters, we need to define a *clustering criterion*. Thus, if we employ the way these animals bear their progeny as a clustering criterion, the sheep, the dog, the cat, and the blue shark will be assigned to the same cluster, while all the rest will form a second cluster (Figure 11.1a). If the clustering criterion is the existence of lungs, the goldfish, the red mullet, and the blue shark are assigned to the same cluster, while all the other animals are assigned to a second cluster (Figure 11.1b). On the other hand, if the clustering criterion is the environment where the animals live, the sheep, the dog, the cat, the sparrow, the seagull, the viper, and the lizard will form one cluster (animals living outside water), the goldfish, the red mullet and the blue shark will form a second cluster (animals living only in water), and the frog will form a third cluster by itself, since it may live in the water or out of it (Figure 11.1c). It is worth pointing out that if the existence of a vertebral column is the clustering criterion, all the animals will lie in the same cluster. Finally, we may use composite clustering criteria as well. For example, if the clustering criterion is the way these animals

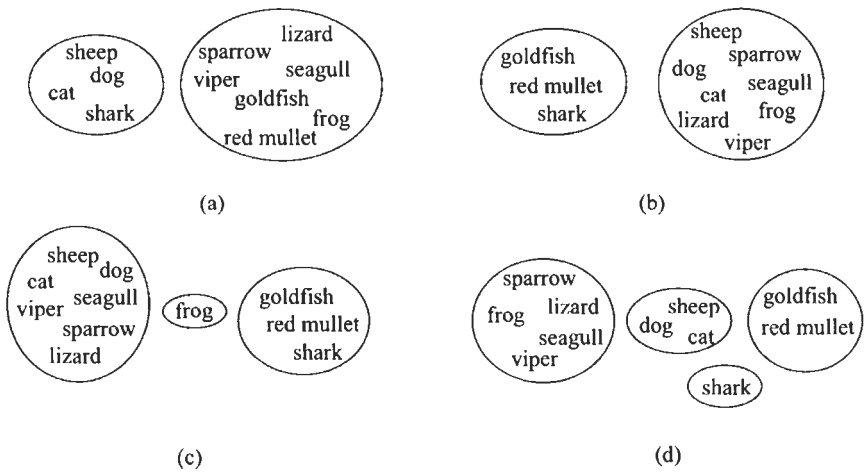


FIGURE 11.1: Resulting clusters if the clustering criterion is (a) the way the animals bear their progeny, (b) the existence of lungs, (c) the environment where the animals live, and (d) the way these animals bear their progeny and the existence of lungs.

bear their progeny *and* the existence of lungs, we end up with four clusters as shown in Figure 11.1d.

This example shows that the process of assigning objects to clusters may lead to very different results, depending on the specific criterion used for clustering.

Clustering is one of the most primitive mental activities of humans, used to handle the huge amount of information they receive every day. Processing every piece of information as a single entity would be impossible. Thus, humans tend to categorize entities (i.e., objects, persons, events) into clusters. Each cluster is then characterized by the common attributes of the entities it contains. For example, most humans "possess" a cluster "dog." If someone sees a dog sleeping on the grass, he or she will identify it as an entity of the cluster "dog." Thus, the individual will infer that this entity barks even though he or she has never heard this specific entity bark before.

As was the case with supervised learning, we will assume that all patterns are represented in terms of *features*, which form l-dimensional feature vectors.

The basic steps that an expert must follow in order to develop a clustering task are the following:

- *Feature selection.* Features must be properly selected so as to encode as much information as possible concerning the task of interest. Once more, parsimony and, thus, minimum information redundancy among the features

is a major goal. As in supervised classification, *preprocessing* of features may be necessary prior to their utilization in subsequent stages. The techniques discussed there are applicable here, too.

- *Proximity measure*. This is a measure that quantifies how "similar" or "dissimilar" two feature vectors are. It is natural to ensure that all selected features contribute equally to the computation of the proximity measure and there are no features that dominate others. This must be taken care of during preprocessing.
- *Clustering criterion*. This depends on the interpretation the expert gives to the term "sensible," based on the type of clusters that are expected to underlie the data set. For example, a compact cluster of feature vectors in the l-dimensional space, may be sensible according to one criterion, whereas an elongated cluster may be sensible according to another. The clustering criterion may be expressed via a cost function or some other types of rules.
- *Clustering algorithms*. Having adopted a proximity measure and a clustering criterion, this step refers to the choice of a specific algorithmic scheme that unravels the clustering structure of the data set.
- *Validation of the results*. Once the results of the clustering algorithm have been obtained, we have to verify their correctness. This is usually carried out using appropriate tests.
- *Interpretation of the results*. In many cases, the expert in the application field must integrate the results of clustering with other experimental evidence and analysis in order to draw the right conclusions.

In a number of cases, a step known as *clustering tendency* should be involved. This includes various tests that indicate whether or not the available data possess a clustering structure. For example, the data set may be of completely random nature, thus, trying to unravel clusters would be meaningless.

As one may have already suspected, different choices of features, proximity measures, clustering criteria and clustering algorithms may lead to totally different clustering results. *Subjectivity is a reality we have to live with from now on.* To demonstrate this, let us consider the following example. Consider Figure 11.2. How many "sensible" ways of clustering can we obtain for these points? The most "logical" answer seems to be two. The first clustering contains four clusters (surrounded by solid circles). The second clustering contains two clusters (surrounded by dashed lines). Which clustering is "correct"? It seems that there is no definite answer. Both clusterings are valid. The best thing to do is give the results to an expert and let the expert decide about the most sensible one. Thus, the final answer to these questions will be influenced by the knowledge of the expert.

The rest of the chapter is devoted to presenting some basic concepts and definitions related to clustering, and it discusses proximity measures that are commonly encountered in various applications.

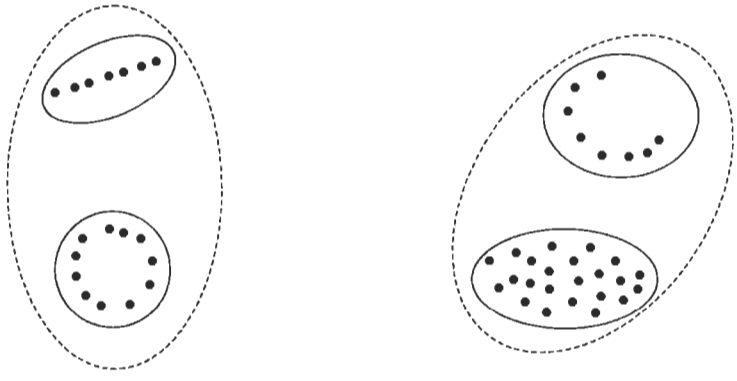


FIGURE 11.2: A coarse clustering of the data results in two clusters, whereas a finer one results in four clusters.

11.1.1 Applications of Cluster Analysis

Clustering is a major tool used in a number of applications. To enrich the list of examples already presented in the introductory chapter of the book, we summarize here four basic directions in which clustering is of use [Ball 71, Ever 81]:

- *Data reduction.* Many times, the amount of the available data, N, is very large and, as a consequence, its processing becomes very demanding. Cluster analysis can be used in order to group the data into a number of "sensible" clusters, m ($\ll N$), and to process each cluster as a single entity. For example, in data transmission, a representative for each cluster is defined. Then, instead of transmitting the data samples, we transmit a code number corresponding to the representative of the cluster in which each specific sample lies. Thus, data compression is achieved.
- *Hypothesis generation.* In this case we apply cluster analysis to a data set in order to infer some hypotheses concerning the nature of the data. Thus, clustering is used here as a vehicle to suggest hypotheses. These hypotheses must then be verified using other data sets.
- *Hypothesis testing.* In this context, cluster analysis is used for the verification of the validity of a specific hypothesis. Consider, for example, the following hypothesis: "Big companies invest abroad." One way to verify whether this is true is to apply cluster analysis to a large and representative set of companies. Suppose that each company is represented by its size, its activities abroad, and its ability to complete successfully projects on applied research. If, after applying cluster analysis, a cluster is formed that corresponds to companies that are large and have investments abroad (regardless

of their ability to complete successfully projects on applied research), then the hypothesis is supported by the cluster analysis.

- *Prediction based on groups.* In this case we apply cluster analysis to the available data set, and the resulting clusters are characterized based on the characteristics of the patterns by which they are formed. In the sequel, if we are given an unknown pattern, we can determine the cluster to which it is more likely to belong and we characterize it based on the characterization of the respective cluster. Suppose, for example, that cluster analysis is applied to a data set concerning patients infected by the same disease. This results in a number of clusters of patients, according to their reaction to specific drugs. Then for a new patient, we identify the most appropriate cluster for the patient and, based on it, we decide on his or her medication (e.g., see [Payk 72]).

11.1.2 Types of Features

A feature may take values from a continuous range (subset of $\mathcal{R}$) or from a finite discrete set. If the finite discrete set has only two elements, then the feature is called *binary* or *dichotomous*.

A different categorization of the features is based on the relative significance of the values they take [Jain 88, Spat 80]. We have four categories of features, the *nominal*, the *ordinal*, the *interval-scaled*, and the *ratio-scaled*.

The first category includes features whose possible values code states. Consider for example a feature that corresponds to the sex of an individual. Its possible values may be 1 for a male and 0 for a female. Clearly, any quantitative comparison between these values is meaningless. The next category includes features whose values can be *meaningfully ordered*. Consider for example a feature that characterizes the performance of a student in the pattern recognition course. Suppose that its possible values are 4, 3, 2, 1 and these correspond to the ratings "excellent," "very good," "good," "not good." Obviously, these values are arranged in a meaningful order. However, the difference between two successive values is of no meaningful quantitative importance.

If, for a specific feature, the difference between two values is meaningful while their ratio is meaningless, then it is an interval-scaled feature. A typical example is the measure of temperature in degrees Celsius. If the temperatures in London and Paris are 5 and 10 degrees Celsius, respectively, then it is meaningful to say that the temperature in Paris is 5 degrees higher than that in London. However, it is meaningless to say that Paris is twice as hot as London.

Finally, if the ratio between two values of a specific feature is meaningful, then this is a ratio-scaled feature. An example of such a feature is the weight, since it is meaningful to say that a person who weighs 100 kg is twice as fat as a person whose weight is 50 kg.

By ordering the types of features as nominal, ordinal, interval-scaled, and ratio scaled, one can easily notice that each type of feature possesses all the properties of the types that are before it. For example, an interval-scaled feature has all the properties of the ordinal and nominal types. This will be of use later on in Section 11.2.2.

Example 11.1. Suppose that we want to group companies according to their prospects of progress. To this end, we may take into account whether a company is private or public, whether or not the company has activities abroad, its annual budgets for the last, say, 3 years, its investments, and its rates of change of the budgets and investments. Therefore, each company is represented by a 10×1 vector. The first component of the vector corresponds to a nominal feature, which codes the state "public" or "private." The second component indicates whether or not there are activities abroad. Its possible values are 0, 1, and 2 (discrete range of values), which correspond to "no investments," "poor investments," and "large investments." Clearly, this component corresponds to an ordinal feature. All the remaining features are ratio scaled.

11.1.3 Definitions of Clustering

The definition of clustering leads directly to the definition of a single "cluster." Many definitions have been proposed over the years (e.g., [John 67, Wall 68, Ever 81]). However, most of these definitions are based on loosely defined terms, such as "similar", "alike", etc., or they are oriented to a specific kind of clusters. As it is pointed out in [Ever 81], most of these definitions are of vague and of circular nature. This fact reveals the difficulty of having a universally acceptable definition for the term cluster.

In [Ever 81], the vectors are viewed as points in the l-dimensional space and the clusters are described as "continuous regions of this space containing a relatively high density of points, separated from other high density regions by regions of relatively low density of points." Clusters described in this way are sometimes referred to as *natural clusters*. This definition is closer to our visual perception of clusters in the two- and three-dimensional spaces.

Let us now try to give some definitions for "clustering," which, although they may not be universal, give us an idea of what clustering is. Let X be our data set, that is,

$$X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}. \tag{11.1}$$

We define as an *m-clustering* of X, $\mathfrak{R}$, the partition of X into m sets (*clusters*), $C_1, \ldots, C_m$, so that the following three conditions are met:

- $C_i \neq \emptyset, i = 1, \ldots, m$

(a) (b) (c)

FIGURE 11.3: (a) Compact clusters. (b) Elongated clusters. (c) Spherical and ellipsoidal clusters.

- $\cup_{i=1}^{m} C_i = X$
- $C_i \cap C_j = \emptyset, i \neq j, i, j = 1, \ldots, m$

In addition, the vectors contained in a cluster C_i are "more similar" to each other and "less similar" to the feature vectors of the other clusters. Quantifying the terms "similar" and "dissimilar" depends very much on the types of clusters involved. For example, other *measures* (measuring similarity) are required for compact clusters (e.g., Figure 11.3a), others for elongated clusters (e.g., Figure 11.3b), and different ones for shell-shaped clusters (e.g., Figure 11.3c).

Note that, under the preceding definitions of clustering, each vector belongs to a single cluster. For reasons that will become clear later on, this type of clustering is sometimes called *hard* or *crisp*. An alternative definition is in terms of the *fuzzy sets*, introduced by Zadeh [Zade 65]. A fuzzy clustering of X into m clusters is characterized by m functions u_j where

$$u_j : X \rightarrow [0, 1], j = 1, \ldots, m \tag{11.2}$$

and

$$\sum_{j=1}^{m} u_j(\boldsymbol{x}_i) = 1, \quad i = 1, 2, \ldots, N, \qquad 0 < \sum_{i=1}^{N} u_j(\boldsymbol{x}_i) < N, \quad j = 1, 2, \ldots, m \tag{11.3}$$

These are called *membership functions*. The value of a fuzzy membership function is a mathematical characterization of a set, that is, a cluster in our case, which may not be precisely defined. That is, each vector $\boldsymbol{x}$ belongs to more than one cluster simultaneously "up to some degree," which is quantified by the corresponding value of u_j in the interval $[0,1]$. Values close to unity show a high "grade of membership" in the corresponding cluster and values close to zero a low grade

of membership. The values of these membership functions are indicative of the structure of the data set, in the sense that if a membership function has close to unity values for two vectors of X, that is, $\boldsymbol{x}_k$, $\boldsymbol{x}_n$, they are considered similar to each other [Wind 82].

The right condition in (11.3) guarantees that there are not trivial cases where clusters exist that do not share any vectors. This is analogous to the condition $C_i \neq \emptyset$ of the aforementioned definition.

The definition of clustering into m distinct sets C_i, given before, can be recovered as a special case of the fuzzy clustering if we define the fuzzy membership functions u_j to take values in $\{0, 1\}$, that is, to be either 1 or 0. In this sense, each data vector belongs exclusively to one cluster and the membership functions are now called *characteristic functions* ([Klir 95]).

11.2 PROXIMITY MEASURES

11.2.1 Definitions

We begin with definitions concerning measures between vectors, and we will extend them later on to include measures between subsets of the data set X.

A *dissimilarity measure* (DM) d on X is a function.

$$d : X \times X \rightarrow \mathcal{R}$$

where $\mathcal{R}$ is the set of real numbers, such that

$$\exists d_0 \in \mathcal{R} : -\infty < d_0 \leq d(\boldsymbol{x}, \boldsymbol{y}) < +\infty, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in X \tag{11.4}$$

$$d(\boldsymbol{x}, \boldsymbol{x}) = d_0, \quad \forall \boldsymbol{x} \in X \tag{11.5}$$

and

$$d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x}), \quad \forall \boldsymbol{x}, \boldsymbol{y} \in X \tag{11.6}$$

If in addition

$$d(\boldsymbol{x}, \boldsymbol{y}) = d_0 \quad \text{if and only if} \quad \boldsymbol{x} = \boldsymbol{y} \tag{11.7}$$

and

$$d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z}), \quad \forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in X \tag{11.8}$$

d is called a *metric DM*. Inequality (11.8) is also known as the *triangular inequality*. Finally, equivalence (11.7) indicates that the minimum possible dissimilarity level value d_0 between any two vectors in X is achieved when they

are identical. Sometimes we will refer to the dissimilarity level as distance, where the term is not used in its strict mathematical sense.

A *similarity measure* (SM) s on X is defined as

$$s : X \times X \rightarrow \mathcal{R}$$

such that

$$\exists s_0 \in \mathcal{R} : -\infty < s(\boldsymbol{x}, \boldsymbol{y}) \leq s_0 < +\infty, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in X \tag{11.9}$$

$$s(\boldsymbol{x}, \boldsymbol{x}) = s_0, \quad \forall \boldsymbol{x} \in X \tag{11.10}$$

and

$$s(\boldsymbol{x}, \boldsymbol{y}) = s(\boldsymbol{y}, \boldsymbol{x}), \quad \forall \boldsymbol{x}, \boldsymbol{y} \in X \tag{11.11}$$

If in addition

$$s(\boldsymbol{x}, \boldsymbol{y}) = s_0 \quad \text{if and only if} \quad \boldsymbol{x} = \boldsymbol{y} \tag{11.12}$$

and

$$s(\boldsymbol{x}, \boldsymbol{y})s(\boldsymbol{y}, \boldsymbol{z}) \leq [s(\boldsymbol{x}, \boldsymbol{y}) + s(\boldsymbol{y}, \boldsymbol{z})]s(\boldsymbol{x}, \boldsymbol{z}), \quad \forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in X \tag{11.13}$$

s is called a *metric SM*.

Example 11.2. Let us consider the well-known Euclidean distance, d_2

$$d_2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{l} (x_i - y_i)^2}$$

where $\boldsymbol{x}, \boldsymbol{y} \in X$ and x_i, y_i are the ith coordinates of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. This is a dissimilarity measure on X, with $d_0 = 0$; that is, the minimum possible distance between two vectors of X is 0. Moreover, the distance of a vector from itself is equal to 0. Also, it is easy to observe that $d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$.

The preceding arguments show that the *Euclidean distance is a dissimilarity measure*. In addition, the Euclidean distance between two vectors takes its minimum value $d_0 = 0$, when the vectors coincide. Finally, it is not difficult to show that the triangular inequality holds for the Euclidean distance (see Problem 11.2). Therefore, the Euclidean distance is a metric dissimilarity measure.

It is worth pointing out that for other measures the values d_0 (s_0) may be positive or negative.

However, not all clustering algorithms are based on proximity measures between vectors. For example, in the hierarchical clustering algorithms[1] one has to compute distances between pairs of sets of vectors of X. In the sequel, we extend the preceding definitions in order to measure "proximity" between subsets of X. Let U be a set containing subsets of X. That is, $D_i \subset X$, $i = 1, \ldots, k$, and $U = \{D_1, \ldots, D_k\}$. A *proximity measure* $\wp$ *on* U is a function

$$\wp : U \times U \rightarrow \mathcal{R}$$

Equations (11.4)–(11.8) for dissimilarity measures and Eqs. (11.9)–(11.13) for similarity measures can now be repeated with D_i, D_j in the place of $\boldsymbol{x}$ and $\boldsymbol{y}$ and U in the place of X.

Usually, the proximity measures between two sets D_i and D_j are defined in terms of proximity measures between elements of D_i and D_j.

Example 11.3. Let $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\}$ and $U = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \{\boldsymbol{x}_1, \boldsymbol{x}_4\}, \{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5\}, \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5\}\}$. Let us define the following dissimilarity function:

$$d_{min}^{ss}(D_i, D_j) = \min_{\boldsymbol{x} \in D_i,\ \boldsymbol{y} \in D_j} d_2(\boldsymbol{x}, \boldsymbol{y})$$

where d_2 is the Euclidean distance between two vectors and $D_i, D_j \in U$.

The minimum possible value of d_{min}^{ss} is $d_{min,0}^{ss} = 0$. Also, $d_{min}^{ss}(D_i, D_i) = 0$, since the Euclidean distance between a vector in D_i and itself is 0. In addition, it is easy to see that the commutative property holds. Thus, this dissimilarity function is a measure. It is not difficult to see that d_{min}^{ss} is not a metric. Indeed, Eq. (11.7) for subsets of X does not hold in general, since the two sets D_i and D_j may have an element in common. Consider for example the two sets $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ and $\{\boldsymbol{x}_1, \boldsymbol{x}_4\}$ of U. Although they are different their distance d_{min}^{ss} is 0, since they both contain $\boldsymbol{x}_1$.

Intuitively speaking, the preceding definitions show that the DMs are "opposite" to SMs. For example, it is easy to show that if d is a (metric) DM, with $d(\boldsymbol{x}, \boldsymbol{y}) > 0$, $\forall \boldsymbol{x}, \boldsymbol{y} \in X$, then $s = a/d$ with $a > 0$ is a (metric) SM (see Problem 11.1). Also, $d_{max} - d$ is a (metric) SM, where d_{max} denotes the maximum value of d among all pairs of elements of X. It is also easy to show that if d is a (metric) DM on a finite set X, such that $d(\boldsymbol{x}, \boldsymbol{y}) > 0, \forall \boldsymbol{x}, \boldsymbol{y} \in X$, then so are $-\ln(d_{max} + k - d)$ and $kd/(1 + d)$, where k is an arbitrary positive constant. On the other hand, if s is a (metric) SM with $s_0 = 1 - \varepsilon$, where ε is a small positive constant, then $1/(1 - s)$ is also a (metric) SM. Similar comments are valid for the similarity and dissimilarity measures between sets $D_i, D_j \in U$.

[1] These algorithms are treated in detail in Chapter 13.

In the sequel, we will review the most commonly used proximity measures between two points. For each measure of similarity we give a corresponding measure of dissimilarity. We will denote by b_{min} and b_{max} the corresponding minimum and maximum values that they take for a finite data set X.

11.2.2 Proximity Measures between Two Points

Real-Valued Vectors

A. Dissimilarity Measures

The most common DMs between real-valued vectors used in practice are:

- The *weighted* l_p metric DMs, that is,

$$d_p(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{i=1}^{l} w_i |x_i - y_i|^p \right)^{1/p} \tag{11.14}$$

where x_i, y_i are the ith coordinates of $\boldsymbol{x}$ and $\boldsymbol{y}$, $i = 1, \ldots, l$, and $w_i \geq 0$ is the ith *weight coefficient*. They are used mainly on real-valued vectors. If $w_i = 1$, $i = 1, \ldots, l$, we obtain the *unweighted* l_p metric DMs. A well-known representative of the latter category of measures is the *Euclidean distance*, which was introduced in Example 11.2 and is obtained by setting $p = 2$.

The weighted l_2 metric DM can be further generalized as follows:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T B (\boldsymbol{x} - \boldsymbol{y})} \tag{11.15}$$

where B is a symmetric, positive definite matrix (Appendix B).

This includes the Mahalanobis distance as a special case and it is also a metric DM.

Special l_p metric DMs that are also encountered in practice are the (weighted) l_1 or *Manhattan norm*,

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{l} w_i |x_i - y_i| \tag{11.16}$$

and the (weighted) l_∞ *norm*,

$$d_\infty(\boldsymbol{x}, \boldsymbol{y}) = \max_{1 \leq i \leq l} w_i |x_i - y_i| \tag{11.17}$$

The l_1 and l_∞ norms may be viewed as overestimation and underestimation of the l_2 norm, respectively. Indeed, it can be shown that $d_\infty(\boldsymbol{x}, \boldsymbol{y}) \leq d_2(\boldsymbol{x}, \boldsymbol{y}) \leq d_1(\boldsymbol{x}, \boldsymbol{y})$ (see Problem 11.6). When $l = 1$ all l_p norms coincide.

Based on these DMs, we can define corresponding SMs as $s_p(\boldsymbol{x}, \boldsymbol{y}) = b_{\max} - d_p(\boldsymbol{x}, \boldsymbol{y})$.

- Some additional DMs are the following [Spat 80]:

$$d_G(\boldsymbol{x}, \boldsymbol{y}) = -\log_{10}\left(1 - \frac{1}{l}\sum_{j=1}^{l}\frac{|x_j - y_j|}{b_j - a_j}\right) \tag{11.18}$$

where b_j and a_j are the maximum and the minimum values among the jth features of the N vectors of X, respectively. It can easily be shown that $d_G(\boldsymbol{x}, \boldsymbol{y})$ is a metric DM. Notice that the value of $d_G(\boldsymbol{x}, \boldsymbol{y})$ depends not only on $\boldsymbol{x}$ and $\boldsymbol{y}$ but also on the whole of X. Thus, if $d_G(\boldsymbol{x}, \boldsymbol{y})$ is the distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ that belong to a set X and $d'_G(\boldsymbol{x}, \boldsymbol{y})$ is the distance between the same two vectors when they belong to a different set X', then, in general, $d_G(\boldsymbol{x}, \boldsymbol{y}) \neq d'_G(\boldsymbol{x}, \boldsymbol{y})$. Another DM is [Spat 80]

$$d_Q(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\frac{1}{l}\sum_{j=1}^{l}\left(\frac{x_j - y_j}{x_j + y_j}\right)^2} \tag{11.19}$$

Example 11.4. Consider the three-dimensional vectors $\boldsymbol{x} = [0, 1, 2]^T$, $\boldsymbol{y} = [4, 3, 2]^T$. Then, assuming that all w_i's are equal to 1, $d_1(\boldsymbol{x}, \boldsymbol{y}) = 6, d_2(\boldsymbol{x}, \boldsymbol{y}) = 2\sqrt{5}$, and $d_\infty(\boldsymbol{x}, \boldsymbol{y}) = 4$. Notice that $d_\infty(\boldsymbol{x}, \boldsymbol{y}) < d_2(\boldsymbol{x}, \boldsymbol{y}) < d_1(\boldsymbol{x}, \boldsymbol{y})$.

Assume now that these vectors belong to a data set X that contains N vectors with maximum values per feature 10, 12, 13 and minimum values per feature 0, 0.5, 1, respectively. Then $d_G(\boldsymbol{x}, \boldsymbol{y}) = 0.0922$. If, on the other hand, $\boldsymbol{x}$ and $\boldsymbol{y}$ belong to an X' with the maximum (minimum) values per feature being 20, 22, 23 (-10, -9.5, -9), respectively, then $d_G(\boldsymbol{x}, \boldsymbol{y}) = 0.0295$.

Finally, $d_Q(\boldsymbol{x}, \boldsymbol{y}) = 0.6455$.

B. Similarity Measures

The most common similarity measures for real-valued vectors used in practice are:

- *The inner product.* It is defined as $s_{inner}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{y} = \sum_{i=1}^{l} x_i y_i$. In most cases, the inner product is used when the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are normalized, so that they have the same length a. In these cases, the upper and the lower bounds of s_{inner} are $+a^2$ and $-a^2$, respectively, and $s_{inner}(\boldsymbol{x}, \boldsymbol{y})$ depends exclusively on the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$.

  A corresponding dissimilarity measure for the inner product is $d_{inner}(\boldsymbol{x}, \boldsymbol{y}) = b_{\max} - s_{inner}(\boldsymbol{x}, \boldsymbol{y})$.
- Another commonly used SM is the Tanimoto measure, which is also known as Tanimoto distance [Tani 58]. It may be used for real- as well as for

discrete-valued vectors. It is defined as

$$s_T(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - \boldsymbol{x}^T\boldsymbol{y}} \tag{11.20}$$

By adding and subtracting the term $\boldsymbol{x}^T\boldsymbol{y}$ in the denominator of (11.20) and after some algebraic manipulations, we obtain

$$s_T(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{1 + \frac{(\boldsymbol{x}-\boldsymbol{y})^T(\boldsymbol{x}-\boldsymbol{y})}{\boldsymbol{x}^T\boldsymbol{y}}}$$

That is, the Tanimoto measure between $\boldsymbol{x}$ and $\boldsymbol{y}$ is inversely proportional to the squared Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ divided by their inner product. Intuitively speaking, since the inner product may be considered as a measure of the correlation between $\boldsymbol{x}$ and $\boldsymbol{y}$, $s_T(\boldsymbol{x}, \boldsymbol{y})$ is inversely proportional to the squared Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{y}$, divided by their correlation.

In the case in which the vectors of X have been normalized to the same length a, the last equation leads to

$$s_T(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{-1 + 2\frac{a^2}{\boldsymbol{x}^T\boldsymbol{y}}}$$

In this case, s_T is inversely proportional to $a^2/\boldsymbol{x}^T\boldsymbol{y}$. Thus, the more correlated $\boldsymbol{x}$ and $\boldsymbol{y}$ are, the larger the value of s_T.

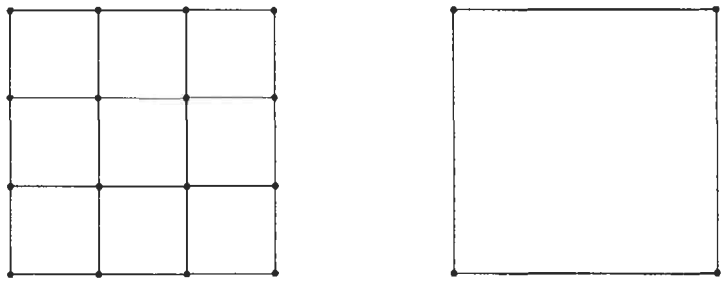
- Finally, another similarity measure that has been proved useful in certain applications [Fu 93] is the following:

$$s_c(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{d_2(\boldsymbol{x}, \boldsymbol{y})}{\|\boldsymbol{x}\| + \|\boldsymbol{y}\|} \tag{11.21}$$

$s_c(\boldsymbol{x}, \boldsymbol{y})$ takes its maximum value (1) when $\boldsymbol{x} = \boldsymbol{y}$ and its minimum (0) when $\boldsymbol{x} = -\boldsymbol{y}$.

Discrete-Valued Vectors

We will now consider vectors $\boldsymbol{x}$ whose coordinates belong to the finite set $F = \{0, 1, \ldots, k-1\}$, where k is a positive integer. It is clear that there are exactly k^l vectors $\boldsymbol{x} \in F^l$. One can imagine these vectors as vertices in an l-dimensional grid as depicted in Figure 11.4. When $k = 2$, the grid collapses to the H_l (unit) hypercube.



(a) (b)

FIGURE 11.4: (a) The $l = 2$ dimensional grid for $k = 4$. (b) The H_2 hypercube (square).

Consider $\boldsymbol{x}, \boldsymbol{y} \in F^l$ and let

$$A(\boldsymbol{x}, \boldsymbol{y}) = [a_{ij}] \quad i, j = 0, 1, \ldots, k-1 \tag{11.22}$$

be a $k \times k$ matrix, where the element a_{ij} is the number of places where the first vector has the i symbol and the corresponding element of the second vector has the j symbol, $i, j \in F$. This matrix is also known as a *contingency table*. For example, if $l = 6$, $k = 3$ and $\boldsymbol{x} = [0, 1, 2, 1, 2, 1]^T$, $\boldsymbol{y} = [1, 0, 2, 1, 0, 1]^T$, then matrix $A(\boldsymbol{x}, \boldsymbol{y})$ is equal to

$$A(\boldsymbol{x}, \boldsymbol{y}) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

It is easy to verify that

$$\sum_{i=0}^{k-1} \sum_{j=0}^{k-1} a_{ij} = l$$

Most of the proximity measures between two discrete-valued vectors may be expressed as combinations of elements of matrix $A(\boldsymbol{x}, \boldsymbol{y})$.

A. Dissimilarity Measures

- *The Hamming distance* (e.g., [Lipp 87, Gers 92]). It is defined as the number of places where two vectors differ. Using the matrix A, we can define the

Hamming distance $d_H(\boldsymbol{x}, \boldsymbol{y})$ as

$$d_H(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{k-1} \sum_{j=0, j\neq i}^{k-1} a_{ij} \tag{11.23}$$

that is, the summation of all the off-diagonal elements of A, which indicate the positions where $\boldsymbol{x}$ and $\boldsymbol{y}$ differ.

In the special case in which $k = 2$, the vectors $\boldsymbol{x} \in F^l$ are binary valued and the Hamming distance becomes

$$d_H(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{l} (x_i + y_i - 2x_i y_i) = \sum_{i=1}^{l} (x_i - y_i)^2 \tag{11.24}$$

In the case where $\boldsymbol{x} \in F_1^l$, where $F_1 = \{-1, 1\}$, $\boldsymbol{x}$ is known as bipolar vector and the Hamming distance is given as

$$d_H(\boldsymbol{x}, \boldsymbol{y}) = 0.5 \left(l - \sum_{i=1}^{l} x_i y_i \right) \tag{11.25}$$

Obviously, a corresponding similarity measure of d_H is $s_H(\boldsymbol{x}, \boldsymbol{y}) = b_{\max} - d_H(\boldsymbol{x}, \boldsymbol{y})$.

- *The l_1 distance.* It is defined as in the case of the continuous-valued vectors, that is,

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{l} |x_i - y_i| \tag{11.26}$$

The l_1 distance and the Hamming distance coincide when binary-valued vectors are considered.

B. Similarity Measures

A widely used similarity measure for discrete-valued vectors is *the Tanimoto measure. It is inspired by the comparison of sets.* If X and Y are two sets and n_X, n_Y, $n_{X\cap Y}$ are the cardinalities (number of elements) of X, Y, and $X \cap Y$, respectively, the Tanimoto measure between two sets X and Y is defined as

$$\frac{n_{X\cap Y}}{n_X + n_Y - n_{X\cap Y}} = \frac{n_{X\cap Y}}{n_{X\cup Y}}$$

In other words, the Tanimoto measure between two sets is the ratio of the number of elements they have in common to the number of all different elements.

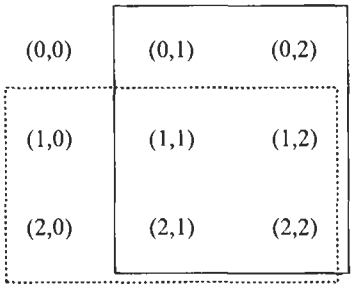


FIGURE 11.5: The elements of a contingency table taken into account for the computation of the Tanimoto measure.

We turn now to the Tanimoto measure between two discrete-valued vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. The measure takes into account all pairs of corresponding coordinates of $\boldsymbol{x}$ and $\boldsymbol{y}$, except those whose corresponding coordinates (x_i, y_i) are both 0. This is justified if we have ordinal features and interpret the value of the ith coordinate of, say, $\boldsymbol{y}$ as the degree to which the vector $\boldsymbol{y}$ possesses the ith feature. According to this interpretation, the pairs $(x_i, y_i) = (0, 0)$ are less important than the others. We now define $n_x = \sum_{i=1}^{k-1} \sum_{j=0}^{k-1} a_{ij}$ and $n_y = \sum_{i=0}^{k-1} \sum_{j=1}^{k-1} a_{ij}$, where a_{ij} are elements of the $A(\boldsymbol{x}, \boldsymbol{y})$ matrix (see Figure 11.5). In words, n_x (n_y) denotes the number of the nonzero coordinates of $\boldsymbol{x}$ ($\boldsymbol{y}$). Then, the Tanimoto measure is defined as

$$s_T(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{k-1} a_{ii}}{n_{\boldsymbol{x}} + n_{\boldsymbol{y}} - \sum_{i=1}^{k-1} \sum_{j=1}^{k-1} a_{ij}} \tag{11.27}$$

In the special case $k = 2$, this equation results in [Tani 58, Spat 80]

$$s_T(\boldsymbol{x}, \boldsymbol{y}) = \frac{a_{11}}{a_{11} + a_{01} + a_{10}} \tag{11.28}$$

Other similarity functions between $\boldsymbol{x}, \boldsymbol{y} \in F^l$ can be defined using elements of $A(\boldsymbol{x}, \boldsymbol{y})$. Some of them consider only the number of places where the two vectors agree and the corresponding value is not 0, whereas others consider all the places where the two vectors agree. Similarity functions that belong to the first category are

$$\frac{\sum_{i=1}^{k-1} a_{ii}}{l} \quad \text{and} \quad \frac{\sum_{i=1}^{k-1} a_{ii}}{l - a_{00}} \tag{11.29}$$

A representative of the second category is

$$\frac{\sum_{i=0}^{k-1} a_{ii}}{l} \tag{11.30}$$

When dealing with binary-valued vectors (i.e., $k = 2$), probabilistic similarity measures have also been proposed [Good 66, Li 85, Broc 81]. For two binary-valued vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, a measure of this kind, s, is based on the number of positions where $\boldsymbol{x}$ and $\boldsymbol{y}$ agree. The value of $s(\boldsymbol{x}, \boldsymbol{y})$ is then compared with the distances of pairs of randomly chosen vectors, in order to conclude whether $\boldsymbol{x}$ and $\boldsymbol{y}$ are "close" to each other. This task is carried out using statistical tests (see also Chapter 16).

Dynamic Similarity Measures

The proximity measures discussed so far apply to vectors with the same dimension, l. However, in certain applications, such as the comparison of two strings st_1 and st_2 stemming from two different texts, this is not the case. For example, one of the two strings may be shifted with respect to the other. In these cases the preceding proximity measures fail. In such cases, dynamic similarity measures, such as the Edit distance, discussed in Chapter 8, can be used.

Mixed Valued Vectors

An interesting case, which often arises in practice, is when the features of the feature vectors are not all real or all discrete valued. In terms of Example 11.1, the third to the tenth features are real valued and the second feature is discrete valued. A naive way to attack this problem is to adopt proximity measures (PMs) suitable for real-valued vectors. The reason is that discrete-valued vectors can be accurately compared in terms of PMs for real-valued vectors, whereas the opposite does not lead, in general, to reasonable results. A good PM candidate for such cases is the l_1 distance.

Example 11.5. Consider the vectors $\boldsymbol{x} = [4, 1, 0.8]^T$ and $\boldsymbol{y} = [1, 0, 0.4]^T$. Their (unweighted) l_1 and l_2 distances are

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = |4 - 1| + |1 - 0| + |0.8 - 0.4| = 3 + 1 + 0.4 = 4.4$$

and

$$d_2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{|4 - 1|^2 + |1 - 0|^2 + |0.8 - 0.4|^2} = \sqrt{9 + 1 + 0.16} = 3.187$$

respectively. Notice that in the second case, the difference between the first coordinates of $\boldsymbol{x}$ and $\boldsymbol{y}$ specifies almost exclusively the difference between the two vectors. This is not the case with l_1 distance (see also related comments in Chapter 5, Section 5.2).

Another method that may be employed is to convert the real-valued features to discrete-valued ones, that is, to discretize the real-valued data. To this end, if a feature x_i takes values in the interval $[a, b]$, we may divide this interval

into k subintervals. If the value of x_i lies in the rth subinterval, the value $r - 1$ will be assigned to it. This strategy leads to discrete-valued vectors and, as a consequence, we may use any of the measures discussed in the previous section.

In [Ande 73] the types "nominal," "ordinal," and "interval scaled" are considered and methods for converting features from one type to another are discussed. This is achieved keeping in mind (see Section 11.1.2) that as we move from "nominal" to "interval scaled" we have to impose information on the specific feature, and when we move along the opposite direction, we have to give up information.

A similarity function that deals with mixed valued vectors, without making any conversions to the type of features, is proposed in [Gowe 71]. Let us consider two l-dimensional mixed valued vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Then, the similarity function between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is defined as

$$s(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\sum_{q=1}^{l} s_q(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sum_{q=1}^{l} w_q} \tag{11.31}$$

where $s_q(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the similarity between the qth coordinates of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and w_q is a weight factor corresponding to the qth coordinate. Specifically, if at least one of the qth coordinates of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is undefined, then $w_q = 0$. Also, if the qth coordinate is a binary variable and it is 0 for both vectors, then $w_q = 0$. In all other cases, w_q is set equal to 1. Finally, if all w_q's are equal to 0 then $s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is undefined. If the qth coordinates of the two vectors are binary then

$$s_q(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{cases} 1, & \text{if } x_{iq} = x_{jq} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{11.32}$$

If the qth coordinates of the two vectors correspond to nominal or ordinal variables, then $s_q(\boldsymbol{x}_i, \boldsymbol{x}_j) = 1$ if x_{iq} and x_{jq} have the same values. Otherwise, $s_q(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$. Finally, if the qth coordinates correspond to interval or ratio scaled variables, then

$$s_q(\boldsymbol{x}_i, \boldsymbol{x}_j) = 1 - \frac{|x_{iq} - x_{jq}|}{r_q} \tag{11.33}$$

where r_q is the length of the interval where the values of the qth coordinates lie. One can easily observe that for the case of interval or ratio scaled variables, when x_{ik} and x_{jk} coincide, $s_q(\boldsymbol{x}_i, \boldsymbol{x}_j)$ takes its maximum value, which equals 1. On the other hand, if the absolute difference between x_{iq} and x_{jq} equals r_q, then $s_q(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$. For any other value of $|x_{iq} - x_{jq}|$, $s_q(\boldsymbol{x}_i, \boldsymbol{x}_j)$ lies between 0 and 1.

Example 11.6. Let us consider the following four 5-dimensional feature vectors, each representing a specific company. More specifically, the first three coordinates (features)

correspond to their annual budget for the last 3 years (in millions of dollars), the fourth indicates whether or not there is any activity abroad, and the fifth coordinate corresponds to the number of employees of each company. The last feature is ordinal scaled and takes the values 0 (small number of employees), 1 (medium number of employees), and 2 (large number of employees). The four vectors are

Company	1st bud.	2nd bud.	3rd bud.	Act. abr.	Empl.
1 (x_1)	1.2	1.5	1.9	0	1
2 (x_2)	0.3	0.4	0.6	0	0
3 (x_3)	10	13	15	1	2
4 (x_4)	6	6	7	1	1

(11.34)

For the first three coordinates, which are ratio scaled, we have $r_1 = 9.7$, $r_2 = 12.6$, and $r_3 = 14.4$. Let us first compute the similarity between the first two vectors. It is

$$s_1(\boldsymbol{x}_1, \boldsymbol{x}_2) = 1 - |1.2 - 0.3|/9.7 = 0.9072$$

$$s_2(\boldsymbol{x}_1, \boldsymbol{x}_2) = 1 - |1.5 - 0.4|/12.6 = 0.9127$$

$$s_3(\boldsymbol{x}_1, \boldsymbol{x}_2) = 1 - |1.9 - 0.6|/14.4 = 0.9097$$

$$s_4(\boldsymbol{x}_1, \boldsymbol{x}_2) = 0$$

and

$$s_5(\boldsymbol{x}_1, \boldsymbol{x}_2) = 0$$

Also, $w_4 = 0$, while all the other weight factors are equal to 1. Using Eq. (11.31), we finally obtain $s(\boldsymbol{x}_1, \boldsymbol{x}_2) = 0.6824$.

Working in the same way, we find that $s(\boldsymbol{x}_1, \boldsymbol{x}_3) = 0.0541$, $s(\boldsymbol{x}_1, \boldsymbol{x}_4) = 0.5588$, $s(\boldsymbol{x}_2, \boldsymbol{x}_3) = 0$, $s(\boldsymbol{x}_2, \boldsymbol{x}_4) = 0.3047$, $s(\boldsymbol{x}_3, \boldsymbol{x}_4) = 0.4953$.

Fuzzy Measures

In this section, we consider real-valued vectors $\boldsymbol{x}$, $\boldsymbol{y}$ whose components x_i and y_i belong to the interval [0, 1], $i = 1, \ldots, l$. In contrast to what we have said so far, *the values of x_i are not the outcome of a measuring device*. The closer the x_i to 1 (0), the more likely $\boldsymbol{x}$ possesses (does not possess) the ith feature (characteristic).[2] As x_i approaches 1/2, we become less certain about the possession or not of the ith feature from $\boldsymbol{x}$. When $x_i = 1/2$ we have absolutely no clue whether $\boldsymbol{x}$ possesses the ith feature or not. It is easy to observe that this situation is a generalization of binary logic, where x_i can take only the value 0 or 1 ($\boldsymbol{x}$ possesses a feature or not).

[2]The ideas of this section follow [Zade 73].

In binary logic, there is a certainty about the occurrence of a fact (for example, it will rain or it will not rain). The idea of fuzzy logic is that nothing is happening or not happening with absolute certainty. This is reflected in the values that x_i takes. The binary logic can be viewed as a special case of fuzzy logic where x_i takes only the value 0 or 1.

Next, we will define the similarity between two real-valued variables in [0, 1]. We will approach it as a generalization of the equivalence between two binary variables. The equivalence of two binary variables a and b is given by the following relation:

$$(a \equiv b) = ((NOT\ a)\ AND\ (NOT\ b))\ OR\ (a\ AND\ b) \tag{11.35}$$

Indeed, if $a = b = 0$ (1), the first (second) argument of the *OR* operator is 1. On the other hand if $a = 0$ (1) and $b = 1$ (0), then none of the arguments of the *OR* operator becomes 1.

An interesting observation is that the *AND* (*OR*) operator between two binary variables may be seen as the min (max) operator on them. Also, the *NOT* operation of a binary variable a may be written as $1 - a$. In the fuzzy logic context and based on this observation, the logical *AND* is replaced by the operator min, while the logical *OR* is replaced by the operator max. Also, the logical *NOT* on x_i is replaced by $1 - x_i$ [Klir 95]. This suggests that the degree of similarity between two real-valued variables x_i and y_i in [0, 1] may be defined as

$$s(x_i, y_i) = \max(\min(1 - x_i, 1 - y_i), \min(x_i, y_i)) \tag{11.36}$$

Note that this definition includes the special case where x_i and y_i take binary values and results in (11.35).

When we now deal with vectors in the l-dimensional space ($l > 1$), the vector space is the H_l hypercube. In this context, the closer a vector $\boldsymbol{x}$ lies to the center of H_l $(1/2, \ldots, 1/2)$, the greater the amount of uncertainty. That is, in this case we have almost no clue whether $\boldsymbol{x}$ possesses any of the l features. On the other hand, the closer $\boldsymbol{x}$ lies to a vertex of H_l, the less the uncertainty.

Based on similarity s between two variables in [0, 1] given in (11.36), we are now able to define a similarity measure between two vectors. A common similarity measure between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as

$$s_F^q(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{i=1}^{l} s(x_i, y_i)^q \right)^{1/q} \tag{11.37}$$

It is easy to verify that the maximum and minimum values of s_F are $l^{1/q}$ and $0.5l^{1/q}$, respectively. As $q \to +\infty$, we get $s_F(\boldsymbol{x}, \boldsymbol{y}) = \max_{1 \le i \le l} s(x_i, y_i)$. Also, when $q = 1$, $s_F(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{l} s(x_i, y_i)$ (Problem 11.7).

Example 11.7. In this example we consider the case where $l = 3$ and $q = 1$. Under these circumstances, the maximum possible value of s_F is 3. Let us consider the vectors $\boldsymbol{x}_1 = [1, 1, 1]^T$, $\boldsymbol{x}_2 = [0, 0, 1]^T$, $\boldsymbol{x}_3 = [1/2, 1/3, 1/4]^T$, and $\boldsymbol{x}_4 = [1/2, 1/2, 1/2]^T$. If we compute the similarities of these vectors with themselves, we obtain

$$s_F^1(\boldsymbol{x}_1, \boldsymbol{x}_1) = 3 \max(\min(1 - 1, 1 - 1), \min(1, 1)) = 3$$

and similarly, $s_F^1(\boldsymbol{x}_2, \boldsymbol{x}_2) = 3$, $s_F^1(\boldsymbol{x}_3, \boldsymbol{x}_3) = 1.92$, and $s_F^1(\boldsymbol{x}_4, \boldsymbol{x}_4) = 1.5$. *This is very interesting. The similarity measure of a vector with itself depends not only on the vector but also on its position in the H_l hypercube.* Furthermore, we observe that the greatest similarity value is obtained at the vertices of H_l. As we move toward the center of H_l, the similarity measure between a vector and itself decreases, attaining its minimum value at the center of H_l.

Let us now consider the vectors $\boldsymbol{y}_1 = [3/4, 3/4, 3/4]^T$, $\boldsymbol{y}_2 = [1, 1, 1]^T$, $\boldsymbol{y}_3 = [1/4, 1/4, 1/4]^T$, $\boldsymbol{y}_4 = [1/2, 1/2, 1/2]^T$. Notice that in terms of the Euclidean distance $d_2(\boldsymbol{y}_1, \boldsymbol{y}_2) = d_2(\boldsymbol{y}_3, \boldsymbol{y}_4)$. However, $s_F^1(\boldsymbol{y}_1, \boldsymbol{y}_2) = 2.25$ and $s_F^1(\boldsymbol{y}_3, \boldsymbol{y}_4) = 1.5$. These results suggest that the closer the two vectors to the center of H_l the less their similarity. On the other hand, the closer the two vectors to a vertex of H_l, the greater their similarity. *That is, the value of $s_F^q(\boldsymbol{x}, \boldsymbol{y})$ depends not only on the relative position of $\boldsymbol{x}$ and $\boldsymbol{y}$ in H_l but also on their closeness to the center of H_l.*

Missing Data

A problem that is commonly met in real-life applications is that of missing data. This means that for some feature vectors we do not know all of their components. This may be a consequence of a failure of the measuring device. Also, in cases such as the one mentioned in Example 11.1, missing data may be the result of a recording error. The following are some commonly used techniques that handle this situation [Snea 73, Dixo 79, Jain 88].

1. Discard all feature vectors that have missing features. This approach may be used when the number of vectors with missing features is small compared with the total number of available feature vectors. If this is not the case, the nature of the problem may be affected.
2. For the ith feature, find its mean value based on the corresponding available values of all feature vectors of X. Then, substitute this value for the vectors where their ith coordinate is not available.
3. For all the pairs of components x_i and y_i of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ define b_i as

$$b_i = \begin{cases} 0, & \text{if both } x_i \text{ and } y_i \text{ are available} \\ 1, & \text{otherwise} \end{cases} \tag{11.38}$$

Then, the proximity between $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as

$$\wp(\boldsymbol{x}, \boldsymbol{y}) = \frac{l}{l - \sum_{i=1}^{l} b_i} \sum_{\text{all } i:b_i=0} \phi(x_i, y_i) \tag{11.39}$$

where $\phi(x_i, y_i)$ denotes the proximity between the two scalars x_i and y_i. A common choice of ϕ when a dissimilarity measure is involved, is $\phi(x_i, y_i) = |x_i - y_i|$. The rationale behind this approach is simple. Let $[a, b]$ be the interval of the allowable values of $\wp(\boldsymbol{x}, \boldsymbol{y})$. The preceding definition ensures that the proximity measure between $\boldsymbol{x}$ and $\boldsymbol{y}$ spans all $[a, b]$, regardless of the number of unavailable features in both vectors.

4. Find the average proximities $\phi_{avg}(i)$ between all feature vectors in X along all components $i = 1, \ldots, l$. It is clear that for some vectors $\boldsymbol{x}$ the ith component is not available. In that case, the proximities that include x_i are excluded from the computation of $\phi_{avg}(i)$. We define the proximity $\psi(x_i, y_i)$ between the ith components of $\boldsymbol{x}$ and $\boldsymbol{y}$ as $\phi_{avg}(i)$ if at least one of the x_i and y_i is not available, and as $\phi(x_i, y_i)$ if both x_i and y_i are available ($\phi(x_i, y_i)$ may be defined as in the previous case). Then,

$$\wp(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{l} \psi(x_i, y_i) \tag{11.40}$$

Example 11.8. Consider the set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5\}$, where $\boldsymbol{x}_1 = [0, 0]^T$, $\boldsymbol{x}_2 = [1, *]^T$, $\boldsymbol{x}_3 = [0, *]^T$, $\boldsymbol{x}_4 = [2, 2]^T$, $\boldsymbol{x}_5 = [3, 1]^T$. The "$*$" means that the corresponding value is not available.

According to the second technique, we find the average value of the second feature, which is 1, and we substitute it for the "$*$"'s. Then, we may use any of the proximity measures defined in the previous sections.

Assume now that we wish to find the distance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ using the third technique. We use the absolute difference as the distance between two scalars. Then $d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{2}{2-1} 1 = 2$. Similarly, $d(\boldsymbol{x}_2, \boldsymbol{x}_3) = \frac{2}{2-1} 1 = 2$.

Finally, if we choose the fourth of the techniques we must first find the average of the distances between any two values of the second feature. We again use the absolute difference as the distance between two scalars. The distances between any two available values of the second feature are $|0 - 2| = 2$, $|0 - 1| = 1$, and $|2 - 1| = 1$, and the average is $4/3$. Thus, the distance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is $d(\boldsymbol{x}_1, \boldsymbol{x}_2) = 1 + 4/3 = 7/3$.

11.2.3 Proximity Functions between a Point and a Set

In many clustering schemes, a vector $\boldsymbol{x}$ is assigned to a cluster C taking into account the proximity between $\boldsymbol{x}$ and C, $\wp(\boldsymbol{x}, C)$. There are two general directions for the definition of $\wp(\boldsymbol{x}, C)$. According to the first one, all points of C contribute

to $\wp(\boldsymbol{x}, C)$. Typical examples of this case include:

- The *max proximity function:*

$$\wp_{max}^{ps}(\boldsymbol{x}, C) = \max_{\boldsymbol{y} \in C} \wp(\boldsymbol{x}, \boldsymbol{y}). \tag{11.41}$$

- The *min proximity function:*

$$\wp_{min}^{ps}(\boldsymbol{x}, C) = \min_{\boldsymbol{y} \in C} \wp(\boldsymbol{x}, \boldsymbol{y}). \tag{11.42}$$

- The *average proximity function:*

$$\wp_{avg}^{ps}(\boldsymbol{x}, C) = \frac{1}{n_C} \sum_{\boldsymbol{y} \in C} \wp(\boldsymbol{x}, \boldsymbol{y}) \tag{11.43}$$

  where n_C is the cardinality of C.

In these definitions, $\wp(\boldsymbol{x}, \boldsymbol{y})$ may be any proximity measure between two points.

Example 11.9. Let $C = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_8\}$, where $\boldsymbol{x}_1 = [1.5, 1.5]^T$, $\boldsymbol{x}_2 = [2, 1]^T$, $\boldsymbol{x}_3 = [2.5, 1.75]^T$, $\boldsymbol{x}_4 = [1.5, 2]^T$, $\boldsymbol{x}_5 = [3, 2]^T$, $\boldsymbol{x}_6 = [1, 3.5]^T$, $\boldsymbol{x}_7 = [2, 3]^T$, $\boldsymbol{x}_8 = [3.5, 3]^T$, and let $\boldsymbol{x} = [6, 4]^T$ (see Figure 11.6). Assume that

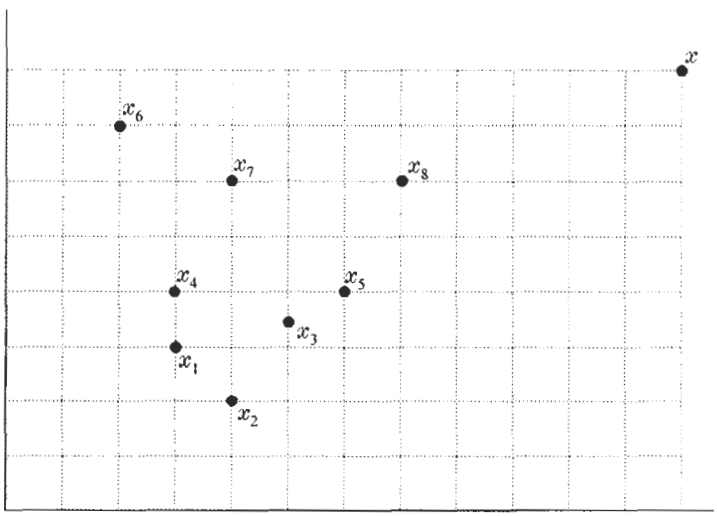
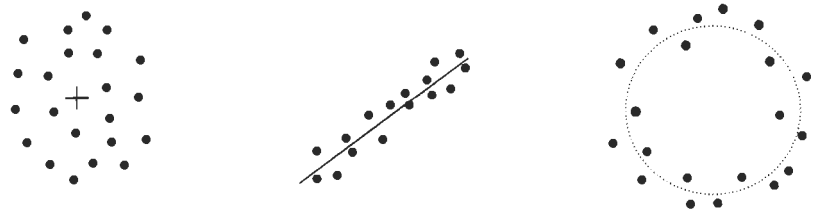


FIGURE 11.6: The setup of Example 11.9.



(a) (b) (c)

FIGURE 11.7: (a) Compact cluster. (b) Hyperplanar (linear) cluster. (c) Hyperspherical cluster.

the Euclidean distance is used to measure the dissimilarity between two points. Then $d^{ps}_{max}(\boldsymbol{x}, C) = \max_{\boldsymbol{y}\in C} d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{x}, \boldsymbol{x}_1) = 5.15$. For the other two distances we have $d^{ps}_{min}(\boldsymbol{x}, C) = \min_{\boldsymbol{y}\in C} d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{x}, \boldsymbol{x}_8) = 2.69$ and $d^{ps}_{avg}(\boldsymbol{x}, C) = \frac{1}{n_C}\sum_{\boldsymbol{y}\in C} d(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{8}\sum_{i=1}^{8} d(\boldsymbol{x}, \boldsymbol{x}_i) = 4.33$.

According to the second direction, C is equipped with a representative and the proximity between $\boldsymbol{x}$ and C is measured as the proximity between $\boldsymbol{x}$ and the representative of C. Many types of representatives have been used in the literature. Among them, the point, the hyperplane, and the hypersphere are most commonly used.[3] Point representatives are suitable for compact clusters (Figure 11.7a) and hyperplane (hyperspherical) representatives for clusters of linear shape (Figure 11.7b) (hyperspherical shape, Figure 11.7c).

Point Representatives

Typical choices for a point representative of a cluster are:

- The *mean vector* (or *mean point*)

$$\boldsymbol{m}_p = \frac{1}{n_C}\sum_{\boldsymbol{y}\in C} \boldsymbol{y} \tag{11.44}$$

where n_C is the cardinality of C. This is the most common choice when point representatives are employed and we deal with data of a continuous space. However, it may not work well when we deal with points of a discrete space F^l. This is because it is possible for $\boldsymbol{m}_p$ to lie outside F^l. To cope with this problem, we may use the mean center $\boldsymbol{m}_c$ of C, which is defined next.

[3] In Chapter 14 we discuss the more general family of hyperquadric representatives, which include hyperellipsoids, hyperparabolas, and pairs of hyperplanes.

- The *mean center* $\boldsymbol{m}_c \in C$ is defined as

$$\sum_{y \in C} d(\boldsymbol{m}_c, y) \leq \sum_{y \in C} d(z, y), \quad \forall z \in C \tag{11.45}$$

where d is a dissimilarity measure between two points. When similarity measures are involved, the inequality is reversed.

Another commonly used point representative is the median center. It is usually employed when the proximity measure between two points is not a metric.

- The *median center* $\boldsymbol{m}_{med} \in C$ is defined as

$$med(d(\boldsymbol{m}_{med}, y) | y \in C) \leq med(d(z, y) | y \in C), \quad \forall z \in C \tag{11.46}$$

where d is a dissimilarity measure between two points. Here $med(T)$, with T being a set of q scalars, is the minimum number in T that is greater than or equal to exactly $[(q + 1)/2]$ numbers of T. An algorithmic way to determine $med(T)$ is to list the elements of T in increasing order and to pick the $[(q + 1)/2]$ element of that list.

Example 11.10. Let $C = \{x_1, x_2, x_3, x_4, x_5\}$, where $x_1 = [1, 1]^T$, $x_2 = [3, 1]^T$, $x_3 = [1, 2]^T$, $x_4 = [1, 3]^T$, and $x_5 = [3, 3]^T$ (see Figure 11.8). All points lie in the discrete space $\{0, 1, 2, \ldots, 6\}^2$. We use the Euclidean distance to measure the dissimilarity between

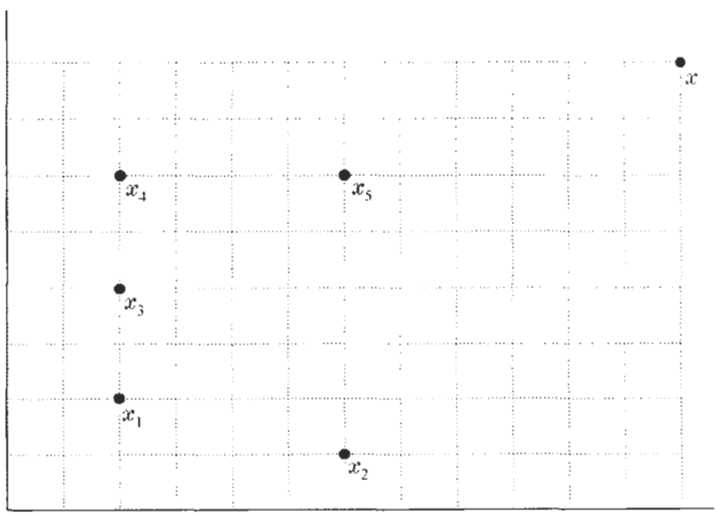


FIGURE 11.8: The setup of Example 11.10.

two vectors in C. The mean point of C is $\boldsymbol{m}_p = [1.8, 2]^T$. It is clear that $\boldsymbol{m}_p$ lies outside the space where the elements of C belong.

To find the mean center $\boldsymbol{m}_c$, we compute, for each point $\boldsymbol{x}_i \in C$, $i = 1, \ldots, 5$, the sum A_i of its distances from all other points of C. The resulting values are $A_1 = 7.83$, $A_2 = 9.06$, $A_3 = 6.47$, $A_4 = 7.83$, $A_5 = 9.06$. The minimum of these values is A_3. Thus, $\boldsymbol{x}_3$ is the mean center of C.

Finally, for the computation of the median center $\boldsymbol{m}_{med}$ we work as follows. For each vector $\boldsymbol{x}_i \in C$ we form the $n_C \times 1$ dimensional vector T_i of the distances between $\boldsymbol{x}_i$ and each of the vectors of C. Working as indicated, we identify $med(T_i)$, $i = 1, \ldots, 5$. Thus, $med(T_1) = med(T_2) = 2$, $med(T_3) = 1$, $med(T_4) = med(T_5) = 2$. Then we choose $med(T_j) = \min_{i=1,\ldots,n_C}\{med(T_i)\} = med(T_3)$ and we identify $\boldsymbol{x}_3$ as the median vector of C. In our example, the mean center and the median center coincide. In general, however, this is not the case.

The distances between $\boldsymbol{x} = [6, 4]^T$ and C when the mean point, the mean center, and the median center are used as representatives of C are 4.65, 5.39, and 5.39, respectively.

Hyperplane Representatives

Linear shaped clusters (or hyperplanar in the general case) are often encountered in computer vision applications. This type of cluster cannot be accurately represented by a single point. In such cases we use lines (hyperplanes) as representatives of the clusters (e.g., [Duda 73]).

The general equation of a hyperplane H is

$$\sum_{j=1}^{l} a_j x_j + a_0 = \boldsymbol{a}^T \boldsymbol{x} + a_0 = 0 \tag{11.47}$$

where $\boldsymbol{x} = [x_1, \ldots, x_l]^T$ and $\boldsymbol{a} = [a_1, \ldots, a_l]^T$ is the weight vector of H. The distance of a point $\boldsymbol{x}$ from H is defined as

$$d(\boldsymbol{x}, H) = \min_{z \in H} d(\boldsymbol{x}, z) \tag{11.48}$$

In the case of Euclidean distance between two points and using simple geometric arguments (see Figure 11.9a), we obtain

$$d(\boldsymbol{x}, H) = \frac{|\boldsymbol{a}^T \boldsymbol{x} + a_0|}{\|\boldsymbol{a}\|} \tag{11.49}$$

where $\|\boldsymbol{a}\| = \sqrt{\sum_{j=1}^{l} a_j^2}$.

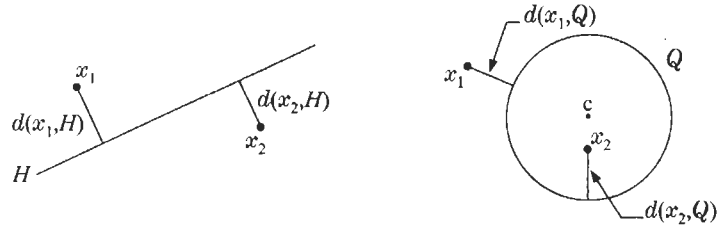


FIGURE 11.9: (a) Distance between a point and a hyperplane. (b) Distance between a point and hypersphere.

Hyperspherical Representatives

Clusters of another type are those that are circular (hyperspherical in higher dimensions). These are also frequently encountered in computer vision applications. For such clusters, the ideal representative is a circle (hypersphere).

The general equation of a hypersphere Q is

$$(\boldsymbol{x} - \boldsymbol{c})^T(\boldsymbol{x} - \boldsymbol{c}) = r^2 \tag{11.50}$$

where $\boldsymbol{c}$ is the center of the hypersphere and r its radius. The distance from a point $\boldsymbol{x}$ to Q is defined as

$$d(\boldsymbol{x}, Q) = \min_{z \in Q} d(\boldsymbol{x}, \boldsymbol{z}) \tag{11.51}$$

In most of the cases of interest, the Euclidean distance between two points is used in this definition. Figure 11.9b provides geometric insight into this definition. However, other nongeometric distances $d(\boldsymbol{x}, Q)$ have been used in the literature (e.g., [Dave 92, Kris 95, Frig 96]).

11.2.4 Proximity Functions between Two Sets

So far, we have been concerned with proximity measures between points in l-dimensional spaces and proximity functions between points and sets. Our major focus now is to define proximity functions between sets of points. As we will soon see, some of the clustering algorithms are built upon such information. Most of the proximity functions $\wp^{ss}$ used for the comparison of sets are based on proximity measures, $\wp$, between vectors (see [Duda 73]). If D_i, D_j are two sets of vectors,

the most common proximity functions are:

- The *max proximity function*:

$$\wp^{ss}_{max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \wp(x, y) \tag{11.52}$$

  It is easy to see that if $\wp$ is a dissimilarity measure, $\wp^{ss}_{max}$ is not a measure, since it does not satisfy the conditions in Section 11.2.1. $\wp^{ss}_{max}$ is fully determined by the pair (x, y) of the most dissimilar (distant) vectors, with $x \in D_i$ and $y \in D_j$. On the other hand, if $\wp$ is a similarity measure, $\wp^{ss}_{max}$ is a measure but it is not a metric (see Problem 11.12). In that case $\wp^{ss}_{max}$ is fully determined by the pair (x, y) of the most similar (closest) vectors, with $x \in D_i$ and $y \in D_j$.
- The *min proximity function*:

$$\wp^{ss}_{min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \wp(x, y) \tag{11.53}$$

  When $\wp$ is a similarity measure, $\wp^{ss}_{min}$ is not a measure. In this case $\wp^{ss}_{min}$ is fully determined by the pair (x, y) of the most dissimilar (distant) vectors, with $x \in D_i$ and $y \in D_j$. On the other hand, if $\wp$ is a dissimilarity measure, $\wp^{ss}_{min}$ is a measure but it is not a metric (see Problem 11.12). In this case $\wp^{ss}_{min}$ is fully determined by the pair (x, y) of the most similar (closest) vectors, with $x \in D_i$ and $y \in D_j$.
- The *average proximity function*:

$$\wp^{ss}_{avg}(D_i, D_j) = \frac{1}{n_{D_i} n_{D_j}} \sum_{x \in D_i} \sum_{y \in D_j} \wp(x, y) \tag{11.54}$$

  where n_{D_i} and n_{D_j} are the cardinalities of D_i and D_j, respectively. It is easily shown that $\wp^{ss}_{avg}$ is not a measure even though $\wp$ is a measure. In this case, all vectors of both D_i and D_j contribute to the computation of $\wp^{ss}_{avg}$.
- The *mean proximity function*:

$$\wp^{ss}_{mean}(D_i, D_j) = \wp(\boldsymbol{m}_{D_i}, \boldsymbol{m}_{D_j}) \tag{11.55}$$

  where $\boldsymbol{m}_{D_i}$ is the representative of D_i, $i = 1, 2$. For example, $\boldsymbol{m}_{D_i}$ may be the mean point, the mean center, or the median of D_i. Obviously, this is the proximity function between the representatives of D_i and D_j. It is clear that the mean proximity function is a measure provided that $\wp$ is a measure.

- Another proximity function that will be used later on is based on the mean proximity function and is defined as[4]

$$\wp_e^{ss}(D_i, D_j) = \sqrt{\frac{n_{D_i} n_{D_j}}{n_{D_i} + n_{D_j}}} \wp(\boldsymbol{m}_{D_i}, \boldsymbol{m}_{D_j}) \tag{11.56}$$

where $\boldsymbol{m}_{D_i}$ is defined as in the previous case.

In the last two alternatives we consider only the cases in which D_i's are represented by points. The need for a definition of a proximity function between two sets via their representatives, when the latter are not points, is of limited practical interest.

Example 11.11. (a) Consider the set $D_1 = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4\}$ and $D_2 = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_4\}$, with $\boldsymbol{x}_1 = [0, 0]^T$, $\boldsymbol{x}_2 = [0, 2]^T$, $\boldsymbol{x}_3 = [2, 0]^T$, $\boldsymbol{x}_4 = [2, 2]^T$, $\boldsymbol{y}_1 = [-3, 0]^T$, $\boldsymbol{y}_2 = [-5, 0]^T$, $\boldsymbol{y}_3 = [-3, -2]^T$, $\boldsymbol{y}_4 = [-5, -2]^T$. The Euclidean distance is employed as the distance between two vectors. The distances between D_1 and D_2 according to the proximity functions just defined are $d_{min}^{ss}(D_1, D_2) = 3$, $d_{max}^{ss}(D_1, D_2) = 8.06$, $d_{avg}^{ss}(D_1, D_2) = 5.57$, $d_{mean}^{ss}(D_1, D_2) = 5.39$, $d_e^{ss}(D_1, D_2) = 7.62$.

(b) Consider now the set $D_2' = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \boldsymbol{z}_4\}$, with $\boldsymbol{z}_1 = [1, 1.5]^T$, $\boldsymbol{z}_2 = [1, 0.5]^T$, $\boldsymbol{z}_3 = [0.5, 1]^T$, $\boldsymbol{z}_4 = [1.5, 1]^T$. Notice that the points of D_1 and D_2' lie in two concentric circles centered at $[1, 1]^T$. The radius corresponding to D_1 (D_2') is $\sqrt{2}$ (0.5). The distances between D_1 and D_2' according to the proximity functions are $d_{min}^{ss}(D_1, D_2') = 1.19$, $d_{max}^{ss}(D_1, D_2') = 1.80$, $d_{avg}^{ss}(D_1, D_2') = 1.46$, $d_{mean}^{ss}(D_1, D_2') = 0$, $d_e^{ss}(D_1, D_2') = 0$.

Notice that in the last case, in which one of the sets lies in the convex hull of the other, some proximity measures may not be appropriate. For example, the measure based on the distances between the two means of the clusters gives meaningless results. However, this distance is well suited for cases in which the two sets are compact and well separated, especially because of its low computational requirements.

Notice that the proximities between two sets are built upon proximities between two points. *Intuitively, one can understand that different choices of proximity functions between sets may lead to totally different clustering results.* Moreover, if we use different proximity measures between points, the same proximity function between sets will lead, in general, to different clustering results. *The only way to achieve proper clustering of the data is by trial and error and, of course, by taking into account the opinion of an expert in the field of application.*

Finally, proximity functions between a vector $\boldsymbol{x}$ and a set D_i may also be derived from the functions defined here, if we set $D_j = \{\boldsymbol{x}\}$.

[4]This definition is a generalization of that given in [Ward 63] (see Chapter 13).

Problems

11.1 Let s be a metric similarity measure on X with $s(\boldsymbol{x}, \boldsymbol{y}) > 0$, $\forall \boldsymbol{x}, \boldsymbol{y} \in X$ and $d(\boldsymbol{x}, \boldsymbol{y}) = a/s(\boldsymbol{x}, \boldsymbol{y})$, with $a > 0$. Prove that d is a metric dissimilarity measure.

11.2 Prove that the Euclidean distance satisfies the triangular inequality.
Hint: Use the Minkowski inequality, which states that for two vectors $\boldsymbol{x} = [x_1, \ldots, x_l]^T$ and $\boldsymbol{y} = [y, \ldots, y_l]^T$ it holds that

$$\left(\sum_{i=1}^{l} |x_i + y_i|^p\right)^{1/p} \leq \left(\sum_{i=1}^{l} |x_i|^p\right)^{1/p} + \left(\sum_{i=1}^{l} |y_i|^p\right)^{1/p}$$

11.3 Show that: (a) if s is a metric similarity measure on a set X with $s(\boldsymbol{x}, \boldsymbol{y}) \geq 0$, $\forall \boldsymbol{x}, \boldsymbol{y} \in X$, then $s(\boldsymbol{x}, \boldsymbol{y}) + a$ is also a metric similarity measure on X, $\forall a \geq 0$.
(b) If d is a metric dissimilarity measure on X then $d + a$ is also a metric dissimilarity measure on X, $\forall a \geq 0$.

11.4 Let $f : \mathcal{R}^+ \to \mathcal{R}^+$ be a continuous monotonically increasing function such that

$$f(x) + f(y) \geq f(x + y), \quad \forall x, y \in \mathcal{R}^+$$

and d a metric dissimilarity measure on a set X with $d_0 \geq 0$. Show that $f(d)$ is also a metric dissimilarity measure on X.

11.5 Let s be a metric similarity measure on a set X, with $s(\boldsymbol{x}, \boldsymbol{y}) > 0$, $\forall \boldsymbol{x}, \boldsymbol{y} \in X$ and $f : \mathcal{R}^+ \to \mathcal{R}^+$ be a continuous monotonically decreasing function such that

$$f(x) + f(y) \geq f\left(\frac{\mathrm{I}}{\frac{1}{x} + \frac{1}{y}}\right), \quad \forall x, y \in \mathcal{R}^+$$

Show that $f(s)$ is a metric dissimilarity measure on X.

11.6 Prove that

$$d_\infty(\boldsymbol{x}, \boldsymbol{y}) \leq d_2(\boldsymbol{x}, \boldsymbol{y}) \leq d_1(\boldsymbol{x}, \boldsymbol{y})$$

for any two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in X.

11.7 (a) Prove that the maximum and the minimum values of $s_F(\boldsymbol{x}, \boldsymbol{y})$ given in (11.37) are $l^{1/q}$ and $0.5l^{1/q}$, respectively.
(b) Prove that as $q \to +\infty$, Eq. (11.37) results in $s_F(\boldsymbol{x}, \boldsymbol{y}) = \max_{1 \leq i \leq l} s(x_i, y_i)$

11.8 Examine whether the similarity functions defined by Eqs. (11.29), (11.30) are metric SMs.

11.9 Let d be a dissimilarity measure on X and $s = d_{max} - d$ a corresponding similarity measure. Prove that

$$s_{avg}^{ps}(\boldsymbol{x}, C) = d_{max} - d_{avg}^{ps}(\boldsymbol{x}, C), \quad \forall \boldsymbol{x} \in X, \quad C \subset X$$

where s_{avg}^{ps} and d_{avg}^{ps} are defined in terms of s and d, respectively. The definition of $\wp_{avg}^{ps}$ may be obtained from (11.54), where the first set consists of a single vector.

11.10 Let $\boldsymbol{x}, \boldsymbol{y} \in \{0, 1\}^l$. Prove that $d_2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{d_{Hamming}(\boldsymbol{x}, \boldsymbol{y})}$.

11.11 Consider two points in an l-dimensional space, $\boldsymbol{x} = [x_1, \ldots, x_l]^T$ and $\boldsymbol{y} = [y_1, \ldots, y_l]^T$, and let $|x_i - y_i| = \max_{j=1,\ldots,l}\{|x_j - y_j|\}$. We define the distance $d_n(\boldsymbol{x}, \boldsymbol{y})$ as

$$d_n(\boldsymbol{x}, \boldsymbol{y}) = |x_i - y_i| + \frac{1}{l - [(l-2)/2]} \sum_{j=1, j\neq i}^{l} |x_j - y_j|$$

This distance has been proposed in [Chau 92] as an approximation of the d_2 (Euclidean) distance.

(a) Prove that d_n is a metric.

(b) Compare d_n with d_2 in terms of computational complexity.

11.12 Let d and s be a similarity and a dissimilarity measure, respectively. Let d^{ss}_{min} (s^{ss}_{min}), d^{ss}_{max} (s^{ss}_{max}), d^{ss}_{avg} (s^{ss}_{avg}), d^{ss}_{mean} (s^{ss}_{mean}) be defined in terms of $d(s)$.

(a) Prove that d^{ss}_{min}, d^{ss}_{mean} are measures and d^{ss}_{max}, d^{ss}_{avg} are not.

(b) Prove that s^{ss}_{max}, s^{ss}_{mean} are measures while s^{ss}_{min}, s^{ss}_{avg} are not.

11.13 Based on Eqs. (11.52), (11.53), (11.54), (11.55), derive the corresponding proximity functions between a point and a set. Are these proximity functions measures?

References

[Ande 73] Anderberg M.R. *Cluster Analysis for Applications*, Academic Press, 1973.

[Ball 71] Ball G.H. "Classification analysis," Stanford Research Institute, *SRI Project 5533*, 1971.

[Broc 81] Brockett P.L., Haaland P.D., Levine A. "Information theoretic analysis of questionnaire data," *IEEE Transactions on Information Theory*, Vol. 27, pp. 438–445, 1981.

[Chau 92] Chaudhuri D., Murthy C.A., Chaudhuri B.B. "A modified metric to compute distance," *Pattern Recognition*, Vol. 25(7), pp. 667–677, 1992.

[Dave 92] Dave R.N., Bhaswan K. "Adaptive fuzzy c-shells clustering and detection of ellipses," *IEEE Transactions on Neural Networks*, Vol. 3(5), pp. 643–662, 1992.

[Dixo 79] Dixon J.K. "Pattern recognition with partly missing data," *IEEE Transactions on Systems Man and Cybernetics*, Vol. SMC 9, 617–621, 1979.

[Duda 73] Duda R.O., Hart P. *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[Ever 81] Everitt B. *Cluster Analysis*, 2nd ed., Halsted Press, 1981.

[Frig 96] Frigui H., Krishnapuram R. "A comparison of fuzzy shell clustering methods for the detection of ellipses," *IEEE Transactions on Fuzzy Systems*, Vol. 4(2), May 1996.

[Fu 93] Fu L., Yang M., Braylan R., Benson N. "Real-time adaptive clustering of flow cytometric data," *Pattern Recognition*, Vol. 26(2), pp. 365–373, 1993.

[Gers 92] Gersho A., Gray R.M. *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[Good 66] Goodall D.W. "A new similarity index based on probability," *Biometrics*, Vol. 22, pp. 882–907, 1966.

[Gowe 67] Gower J.C. "A comparison of some methods of cluster analysis," *Biometrics*, Vol. 23, pp. 623–637, 1967.

[Gowe 71] Gower J.C. "A general coefficient of similarity and some of its properties," *Biometrics*, Vol. 27, pp. 857–872, 1971.

[Gowe 86] Gower J.C., Legendre P. "Metric and Euclidean properties of dissimilarity coefficients," *Journal of Classification*, Vol. 3, pp. 5–48, 1986.

[Hall 67] Hall A.V. "Methods for demonstrating resemblance in taxonomy and ecology," *Nature*, Vol. 214, pp. 830–831, 1967.

[Huba 82] Hubalek Z. "Coefficients of association and similarity based on binary (presence–absence) data—an evaluation," *Biological Review*, Vol. 57, pp. 669–689, 1982.

[Jain 88] Jain A.K., Dubes R.C. *Algorithms for Clustering Data*, Prentice Hall, 1988.

[John 67] Johnson S.C. "Hierarchical clustering schemes," *Psychometrika*, Vol. 32, pp. 241–254, 1967.

[Klir 95] Klir G., Yuan B., *Fuzzy sets and fuzzy logic*, Prentice Hall, 1995.

[Koho 89] Kohonen T. *Self-Organization and Associative Memory*, Springer-Verlag, 1989.

[Kris 95] Krishnapuram R., Frigui H., Nasraoui O. "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation—Part I," *IEEE Transactions on Fuzzy Systems*, Vol. 3(1), pp. 29–43, February 1995.

[Li 85] Li X., Dubes R.C. "The first stage in two-stage template matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 7, pp. 700–707, 1985.

[Lipp 87] Lippmann R.P. "An introduction to computing with neural nets," *IEEE ASSP Magazine*, Vol. 4(2), April 1987.

[Payk 72] Paykel E.S. "Depressive typologies and response to amitriptyline," *British Journal of Psychiatry*, Vol. 120, pp. 147–156, 1972.

[Snea 73] Sneath P.H.A., Sokal R.R. *Numerical Taxonomy*, W.H. Freeman & Co. San Francisco, 1973.

[Soka 63] Sokal R.R., Sneath P.H.A. *Principles of Numerical Taxonomy*, Freeman, 1963.

[Spat 80] Spath H. *Cluster Analysis Algorithms*, Ellis Horwood, 1980.

[Tani 58] Tanimoto T. "An elementary mathematical theory of classification and prediction," *Int. Rpt.*, IBM Corp., 1958.

[Wall 68] Wallace C.S., Boulton D.M. "An information measure for classification," *Computer Journal*, Vol. 11, pp. 185–194, 1968.

[Ward 63] Ward J.H., Jr. "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association.*, Vol. 58, pp. 236–244, 1963.

[Wind 82] Windham M.P. "Cluster validity for the fuzzy c-means clustering algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 4(4), pp. 357–363, 1982.

[Zade 65] Zadeh L.A. "Fuzzy sets," *Information and Control*, Vol. 8, pp. 338–353, 1965.

[Zade 73] Zadeh L.A. *IEEE Transactions on Systems Man and Cybernetics SMC-3*, Vol. 28, 1973.

CHAPTER 12

CLUSTERING ALGORITHMS I: SEQUENTIAL ALGORITHMS

12.1 INTRODUCTION

In the previous chapter, our major focus was on introducing a number of proximity measures. Each of them gives a different interpretation to the terms similar and dissimilar, associated with the types of clusters that our clustering procedure has to reveal. In the current and the following three chapters, the emphasis is on the various clustering algorithmic schemes and criteria that are available to the analyst. As has already been stated, different combinations of a proximity measure and a clustering scheme will lead to different results, which the expert has to interpret.

This chapter begins with a general overview of the various clustering algorithmic schemes and then focuses on one category, known as sequential algorithms.

12.1.1 Number of Possible Clusterings

Given the time and resources, the best way to assign the feature vectors $\boldsymbol{x}_i$, $i = 1, \ldots, N$, of a set X to clusters would be to identify all possible partitions and to select the most sensible one according to a preselected criterion. However, this is not possible even for moderate values of N. Indeed, let $S(N, m)$ denote the number of all possible clusterings of N vectors into m groups. Remember that, by definition, no cluster is empty. It is clear that the following conditions hold [Spat 80, Jain 88]:

- $S(N, 1) = 1$
- $S(N, N) = 1$
- $S(N, m) = 0, \quad \text{for } m > N$

Let L_{N-1}^{k} be the list containing all possible clusterings of the $N-1$ vectors into k clusters, for $k = m, m-1$. The Nth vector

- Either will be added to one of the clusters of any member of L_{N-1}^{m}
- Or will form a new cluster to each member of L_{N-1}^{m-1}

Thus, we may write

$$S(N, m) = mS(N-1, m) + S(N-1, m-1) \tag{12.1}$$

The solutions of (12.1) are the so-called Stirling numbers of the second kind (e.g., see [Liu 68])[1]:

$$S(N, m) = \frac{1}{m!} \sum_{i=0}^{m} (-1)^{m-i} \binom{m}{i} i^N \tag{12.2}$$

Example 12.1. Assume that $X = \{x_1, x_2, x_3\}$. We seek to find all possible clusterings of the elements of X in two clusters. It is easy to deduce that

$$L_2^1 = \{\{x_1, x_2\}\}$$

and

$$L_2^2 = \{\{x_1\}, \{x_2\}\}$$

Taking into account (12.1), we easily find that $S(3, 2) = 2 \times 1 + 1 = 3$. Indeed, the L_3^2 list is

$$L_3^2 = \{\{x_1, x_3\}, \{x_2\}\}, \{\{x_1\}, \{x_2, x_3\}\}, \{\{x_1, x_2\}, \{x_3\}\}$$

Especially for $m = 2$, (12.2) becomes

$$S(N, 2) = 2^{N-1} - 1 \tag{12.3}$$

(see Problem 12.1). Some numerical values of (12.2) are [Spat 80]

- $S(15, 3) = 2375101$
- $S(20, 4) = 45232115901$
- $S(25, 8) = 690223721118368580$
- $S(100, 5) \simeq 10^{68}$

It is clear that these calculations are valid for the case in which the number of clusters is fixed. If this is not the case, one has to enumerate all possible clusterings for all possible values of m. From the preceding analysis it is obvious that evaluating all of them to identify the most sensible one is impractical even for moderate values of N. Indeed, if, for example, one has to evaluate all possible clusterings of 100 objects into five clusters with a computer that evaluates each single clustering in 10^{-12} seconds, the most "sensible" clustering would be available after approximately 10^{48} years!

[1] Compare it with the number of dichotomies in Cover's theorem.

12.2 CATEGORIES OF CLUSTERING ALGORITHMS

Clustering algorithms may be viewed as schemes that provide us with sensible clusterings by considering *only a small fraction of the set containing all possible partitions of X. The result depends on the specific algorithm and the criteria used.* Thus a clustering algorithm is a learning procedure that tries to identify the specific characteristics of the clusters underlying the data set. Clustering algorithms may be divided into the following major categories.

- *Sequential algorithms.* These algorithms produce a single clustering. They are quite straightforward and fast methods. In most of them, all the feature vectors are presented to the algorithm once or a few times (typically no more than five or six times). The final result is, usually, dependent on the order in which the vectors are presented to the algorithm. These schemes tend to produce compact and hyperspherically or hyperellipsoidally shaped clusters, depending on the distance metric used. This category will be studied at the end of this chapter.
- *Hierarchical clustering algorithms.* These schemes are further divided into
  - —*Agglomerative algorithms.* These algorithms produce a sequence of clusterings of decreasing number of clusters, m, at each step. The clustering produced at each step results from the previous one by merging two clusters into one. The main representatives of the agglomerative algorithms are the *single and complete link* algorithms. The agglomerative algorithms may be further divided into the following subcategories:
    - * Algorithms that stem from the matrix theory
    - * Algorithms that stem from graph theory

    These algorithms are appropriate for the recovery of elongated clusters (as is the case with the single link algorithm) and compact clusters (as is the case with the complete link algorithm).
  - —*Divisive algorithms.* These algorithms act in the opposite direction; that is, they produce a sequence of clusterings of increasing m at each step. The clustering produced at each step results from the previous one by splitting a single cluster into two.
- *Clustering algorithms based on cost function optimization.* This category contains algorithms in which "sensible" is quantified by a cost function, J, in terms of which a clustering is evaluated. Usually, the number of clusters m is kept fixed. These algorithms use differential calculus concepts and produce successive clusterings while trying to optimize J. They terminate when a local optimum of J is determined. Algorithms of this category are also called *iterative function optimization schemes.* This category includes

the following subcategories:

- —*Hard or crisp clustering algorithms*, where a vector belongs exclusively to a specific cluster. The assignment of the vectors to individual clusters is carried out optimally, according to the adopted optimality criterion. The most famous algorithm of this category is the Isodata or Lloyd algorithm [Lloy 82, Duda 73].
- —*Probabilistic clustering algorithms* are a special type of hard clustering algorithms that follow Bayesian classification arguments and each vector $\boldsymbol{x}$ is assigned to the cluster C_i for which $P(C_i|\boldsymbol{x})$ (i.e., the a posteriori probability) is maximum. These probabilities are estimated via an appropriately defined optimization task.
- —*Fuzzy clustering algorithms*, where a vector belongs to a specific cluster up to a certain degree.
- —*Possibilistic clustering algorithms*. In this case we measure the possibility for a feature vector $\boldsymbol{x}$ to belong to a cluster C_i.
- —*Boundary detection algorithms*. Instead of determining the clusters by the feature vectors themselves, these algorithms adjust iteratively the boundaries of the regions where clusters lie. These algorithms, although they evolve around a cost function optimization philosophy, they are of different nature from the above algorithms. All the aforementioned schemes use cluster representatives and the goal is to locate them in space in an optimal way. In contrast, boundary detection algorithms seek ways of placing optimally boundaries between clusters. This has led us to the decision to treat these algorithms in a separate chapter, together with algorithms to be discussed next.

- *Other*: This last category contains some special clustering techniques that cannot be assigned to any of the previous categories. These include:
  - —*Branch and bound clustering algorithms*. These algorithms provide us with *the globally optimal clustering without having to consider all possible clusterings*, for fixed number m of clusters, and for a prespecified criterion. However, their computational burden is excessive.
  - —*Genetic clustering algorithms*. These algorithms use an initial population of possible clusterings and iteratively generate new populations, which, in general, contain better clusterings than those of the previous generations, according to a prespecified criterion.
  - —*Stochastic relaxation methods*. These are methods that guarantee, under certain conditions, convergence in probability to the globally optimum clustering, with respect to a prespecified criterion, at the expense of intensive computations.

—*Valley-seeking clustering algorithms.* These algorithms treat the feature vectors as instances of a (multidimensional) random variable $\boldsymbol{x}$. They are based on the commonly accepted assumption that regions of $\boldsymbol{x}$ where many vectors reside correspond to regions of increased values of the respective probability density function (pdf) of $\boldsymbol{x}$. Therefore, the estimation of the pdf may highlight the regions where clusters are formed.

—*Competitive learning algorithms.* These are iterative schemes that do not employ cost functions. They produce several clusterings and they converge to the most "sensible" one, according to a distance metric. Typical representatives of this category are the *basic competitive learning scheme* and the *leaky learning algorithm.*

—*Algorithms based on morphological transformation techniques.* These algorithms use morphological transformations in order to achieve better separation of the involved clusters.

12.3 SEQUENTIAL CLUSTERING ALGORITHMS

In this section we describe a basic sequential algorithmic scheme (BSAS) (which is a generalization of that discussed in [Hall 67]) and we also give some variants of it. First, we consider the case where all the vectors are presented to the algorithm *only once. The number of clusters is not known a priori in this case.* In fact, new clusters are created as the algorithm evolves.

Let $d(\boldsymbol{x}, C)$ denote the distance (or dissimilarity) between a feature vector $\boldsymbol{x}$ and a cluster C. This may be defined by taking into account either all vectors of C or a representative vector of it (see Chapter 11). *The user-defined parameters required by the algorithmic scheme are the threshold of dissimilarity* Θ *and the maximum allowable number of clusters,* q. The basic idea of the algorithm is the following: As each new vector is considered, it is either assigned to an existing cluster or assigned to a newly created cluster, depending on its distance from the already formed ones. Let m be the number of clusters that the algorithm has created up to now. Then the algorithmic scheme may be stated as:

Basic Sequential Algorithmic Scheme (BSAS)

- $m = 1$
- $C_m = \{\boldsymbol{x}_1\}$
- For $i = 2$ to N
  - —Find C_k: $d(\boldsymbol{x}_i, C_k) = \min_{1 \le j \le m} d(\boldsymbol{x}_i, C_j)$
  - —If $(d(\boldsymbol{x}_i, C_k) > \Theta)$ AND $(m < q)$ then
    - $m = m + 1$

* $C_m = \{\boldsymbol{x}_i\}$

—Else

* $C_k = C_k \cup \{\boldsymbol{x}_i\}$
* Where necessary, update representatives.[2]

—End {if}

- End {For}

Different choices of $d(\boldsymbol{x}, C)$ lead to different algorithms and any of the measures introduced in Chapter 11 can be employed. When C is represented by a single vector, $d(\boldsymbol{x}, C)$ becomes

$$d(\boldsymbol{x}, C) = d(\boldsymbol{x}, \boldsymbol{m}_C) \tag{12.4}$$

where $\boldsymbol{m}_C$ is the representative of C. In the case in which the mean vector is used as a representative, the updating may take place in an iterative fashion, that is,

$$\boldsymbol{m}_{C_k}^{new} = \frac{(n_{C_k^{new}} - 1)\boldsymbol{m}_{C_k}^{old} + \boldsymbol{x}}{n_{C_k^{new}}} \tag{12.5}$$

where $n_{C_k^{new}}$ is the cardinality of C_k after the assignment of $\boldsymbol{x}$ to it and $\boldsymbol{m}_{C_k}^{new}$ ($\boldsymbol{m}_{C_k}^{old}$) is the representative of C_k after (before) the assignment of $\boldsymbol{x}$ to it (Problem 12.2).

It is not difficult to realize that the order in which the vectors are presented to the BSAS plays an important role in the clustering results. Different presentation ordering may lead to totally different clustering results, in terms of the number of clusters as well as the clusters themselves (see Problem 12.3).

Another important factor affecting the result of the clustering algorithm is the choice of the threshold Θ. This value directly affects the number of clusters formed by BSAS. If Θ is too small, unnecessary clusters will be created. On the other hand, if Θ is too large a smaller than appropriate number of clusters will be created. In both cases, the number of clusters that best fits the data set is missed.

If the number q of the maximum allowable number of clusters is not constrained, we leave to the algorithm to "decide" about the appropriate number of clusters. Consider for example Figure 12.1, where three compact and well-separated clusters are formed by the points of X. If the maximum allowable number of clusters is set equal to two, the BSAS algorithm will be unable to discover three clusters. Probably, in this case the two rightmost groups of

[2]This statement is activated in the cases where each cluster is represented by a single vector. For example, if each cluster is represented by its mean vector, this must be updated each time a new vector becomes a member of the cluster.

FIGURE 12.1: Three clusters are formed by the feature vectors. When q is constrained to a value less than 3, the BSAS algorithm will not be able to reveal them.

points will form a single cluster. On the other hand, if q is unconstrained the BSAS algorithm will probably form three clusters (with an appropriate choice of Θ), at least for the case in which the mean vector is used as a representative. However, constraining q becomes necessary when dealing with implementations where the available computational resources are limited. In the next subsection a simple technique is given for determining the number of clusters.[3]

Remarks

- The BSAS scheme may be used with similarity instead of dissimilarity measures with appropriate modification; that is, the min operator is replaced by max.
- It turns out that BSAS, with point cluster representatives, favors compact clusters. Thus, it is not recommended if there is strong evidence that other types of clusters are present.
- The preceding algorithm is closely related to the algorithm implemented by the ART2 (adaptive resonance theory) neural architecture [Carp 87, Burk 91].

12.3.1 Estimation of the Number of Clusters

In this subsection, a simple method is described for determining the number of clusters (see also Chapter 16). The method is suitable for BSAS as well as other algorithms, for which the number of clusters is not required as an input parameter. In what follows, BSAS(Θ) denotes the BSAS algorithm with a specific threshold of dissimilarity Θ.

[3]This problem is also treated in Chapter 16.

- For $\Theta = a$ to b step c
  - —Run s times the algorithm BSAS(Θ), each time presenting the data in a different order.
  - —Estimate the number of clusters, m_Θ, as the most frequent number resulting from the s runs of BSAS(Θ).
- Next Θ

The values a and b are the minimum and maximum dissimilarity levels among all pairs of vectors in X, that is, $a = \min_{i,j=1,\ldots,N} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $b = \max_{i,j=1,\ldots,N} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The choice of c is directly influenced by the choice of $d(\boldsymbol{x}, C)$. As far as the value of s is concerned, the greater the s, the larger the statistical sample and, thus, the higher the accuracy of the results. In the sequel, we plot the number of clusters m_Θ versus Θ. This plot has a number of flat regions. We estimate the number of clusters as the number that corresponds to the widest flat region. It is expected that at least for the case in which the vectors form well-separated compact clusters, this is the desired number. Let us explain this argument intuitively. Suppose that the data form two compact and well-separated clusters C_1 and C_2. Let the maximum distance between two vectors in C_1 (C_2) be r_1 (r_2) and suppose that $r_1 < r_2$. Also let r ($> r_2$) be the minimum among all distances $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, with $\boldsymbol{x}_i \in C_1$ and $\boldsymbol{x}_j \in C_2$. It is clear that for $\Theta \in [r_2, r - r_2]$, the number of clusters created by BSAS is 2. In addition, if $r \gg r_2$, the interval has a wide range, and thus it corresponds to a wide flat region in the plot of m_Θ versus Θ. Example 12.2 illustrates the idea.

Example 12.2. Consider two 2-dimensional Gaussian distributions with means $[0, 0]^T$ and $[20, 20]^T$, respectively. The covariance matrices are $\Sigma = 0.5I$ for both distributions, where I is the 2×2 identity matrix. Generate 50 points from each distribution (Figure 12.2a). The number of underlying clusters is 2. The plot resulting from the application of the previously described procedure is shown in Figure 12.2b, with $a = \min_{\boldsymbol{x}_i, \boldsymbol{x}_j \in X} d_2(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $b = \max_{\boldsymbol{x}_i, \boldsymbol{x}_j \in X} d_2(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $c \simeq 0.3$. It can be seen that the widest flat region corresponds to the number 2, which is the number of underlying clusters.

In the foregoing procedure, we have implicitly assumed that the feature vectors do form clusters. If this is not the case, the method is useless. Methods that deal with the problem of discovering whether any clusters exist are discussed in Chapter 16. Moreover, if the vectors form compact clusters, which are not well separated, the procedure may give unreliable results, since it is unlikely for the plot of m_Θ versus Θ to contain wide flat regions.

In some cases, it may be advisable to consider all the numbers of clusters, m_Θ, that correspond to all flat regions of considerable size in the plot of m_Θ versus Θ. If, for example, we have three clusters and the first two of them lie close to each

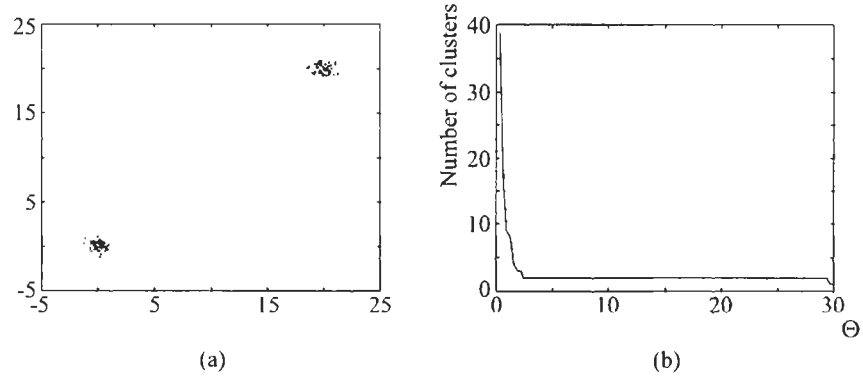


FIGURE 12.2: (a) The data set. (b) The plot of the number of clusters versus Θ. It can be seen that for a wide range of values of Θ, the number of clusters, m, is 2.

other and away from the third, the flattest region may occur for $m_\Theta = 2$ and the second flattest for $m_\Theta = 3$. If we discard the second flattest region, we will miss the three-cluster solution (Problem 12.6).

12.4 A MODIFICATION OF BSAS

As has already been stated, the basic idea behind BSAS is that each input vector $\boldsymbol{x}$ is assigned to an already created cluster or a new one is formed. Therefore, a decision for the vector $\boldsymbol{x}$ is reached *prior to the final cluster formation*, which is determined after all vectors have been presented. The following refinement of BSAS, which will be called modified BSAS (MBSAS), overcomes this drawback. The cost we pay for it is that the vectors of X have to be presented twice to the algorithm. The algorithmic scheme consists of two phases. The first phase involves the determination of the clusters, via the assignment of *some* of the vectors of X to them. During the second phase, the unassigned vectors are presented for a second time to the algorithm and are assigned to the appropriate cluster. The MBSAS may be written as follows:

Modified Basic Sequential Algorithmic Scheme (MBSAS)

Cluster Determination

- $m = 1$
- $C_m = \{\boldsymbol{x}_1\}$

- For $i = 2$ to N
  - —Find C_k: $d(\boldsymbol{x}_i, C_k) = \min_{1 \le j \le m} d(\boldsymbol{x}_i, C_j)$
  - —If $(d(\boldsymbol{x}_i, C_k) > \Theta)$ AND $(m < q)$ then
    - * $m = m + 1$
    - * $C_m = \{\boldsymbol{x}_i\}$
  - —End {if}
- End {For}

Pattern Classification

- For $i = 1$ to N
  - —If $\boldsymbol{x}_i$ has not been assigned to a cluster, then
    - * Find C_k: $d(\boldsymbol{x}_i, C_k) = \min_{1 \le j \le m} d(\boldsymbol{x}_i, C_j)$
    - * $C_k = C_k \cup \{\boldsymbol{x}_i\}$
    - * Where necessary, update representatives.
  - —End {if}
- End {For}

The number of clusters is determined in the first phase and then it is frozen. Thus, the decision taken during the second phase for each vector takes into account all clusters.

When the mean vector of a cluster is used as its representative, the appropriate cluster representative has to be adjusted using Eq. (12.5), after the assignment of each vector in a cluster.

Also, as it was the case with BSAS, MBSAS is sensitive to the order in which the vectors are presented.

Finally, it must be stated that, after minor modifications, MBSAS may be used when a similarity measure is employed (see Problem 12.7).

12.5 A TWO-THRESHOLD SEQUENTIAL SCHEME

As already has been pointed out, the results of BSAS and MBSAS are strongly dependent on the order in which the vectors are presented to the algorithm, as well as on the value of Θ. Improper choice of Θ may lead to meaningless clustering results. One way to overcome these difficulties is to define a "gray" region (see [Trah 89]). This is achieved by employing two thresholds, Θ_1 and $\Theta_2 (> \Theta_1)$. If the dissimilarity level $d(\boldsymbol{x}, C)$ of a vector $\boldsymbol{x}$ from its closest cluster C is less than Θ_1, $\boldsymbol{x}$ is assigned to C. If $d(\boldsymbol{x}, C) > \Theta_2$, a new cluster is formed and $\boldsymbol{x}$ is placed in it. Otherwise, if $\Theta_1 \le d(\boldsymbol{x}, C) \le \Theta_2$, there exists uncertainty and the assignment of $\boldsymbol{x}$ to a cluster will take place at a later stage. Let $clas(\boldsymbol{x})$ be a flag that indicates whether $\boldsymbol{x}$ has been classified (1) or not (0). Again, we denote by m the number of clusters that have been formed up to now. In the following, we assume no bounds to the number of clusters (i.e., $q = N$). The algorithmic

scheme is:

The Two-Threshold Sequential Algorithmic Scheme (TTSAS)

$m = 0$
$clas(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in X$
$prev_change = 0$
$cur_change = 0$
$exists_change = 0$

While (there exists at least one feature vector $\boldsymbol{x}$ with $clas(\boldsymbol{x}) = 0$) do

- For $i = 1$ to N
  - —if $clas(\boldsymbol{x}_i) = 0$ AND it is the first in the new while loop AND $exists_change = 0$ then
    - $m = m + 1$
    - $C_m = \{\boldsymbol{x}_i\}$
    - $clas(\boldsymbol{x}_i) = 1$
    - $cur_change = cur_change + 1$
  - —Else if $clas(\boldsymbol{x}_i) = 0$ then
    - Find $d(\boldsymbol{x}_i, C_k) = \min_{1 \le j \le m} d(\boldsymbol{x}_i, C_j)$
    - if $d(\boldsymbol{x}_i, C_k) < \Theta_1$ then
      - $C_k = C_k \cup \{\boldsymbol{x}_i\}$
      - $clas(\boldsymbol{x}_i) = 1$
      - $cur_change = cur_change + 1$
    - else if $d(\boldsymbol{x}_i, C_k) > \Theta_2$ then
      - $m = m + 1$
      - $C_m = \{\boldsymbol{x}_i\}$
      - $clas(\boldsymbol{x}_i) = 1$
      - $cur_change = cur_change + 1$
    - End {If}
  - —Else if $clas(\boldsymbol{x}_i) = 1$ then
    - $cur_change = cur_change + 1$
  - —End {If}
- End {For}
- $exists_change = |cur_change - prev_change|$
- $prev_change = cur_change$
- $cur_change = 0$

End {While}

The $exists_change$ checks whether there exists at least one vector that has been classified at the current pass on X (i.e., the current iteration of the while loop). This is achieved by comparing the number of vectors that have been classified up

to the current pass on X, cur_change, with the number of vectors that have been classified up to the previous pass on X, $prev_change$. If $exists_change = 0$, that is, no vector has been assigned to a cluster during the last pass on X, the first unclassified vector is used for the formation of a new cluster.

The first *if* condition in the *For* loop ensures that the algorithm terminates after N passes on X (N executions of the *while* loop) at the most. Indeed, this condition forces the first unassigned vector to a new cluster when no vector has been assigned during the last pass on X. This gives a way out to the case in which no vector has been assigned at a given circle.

However, in practice, the number of required passes is much less than N. It should be pointed out that this scheme is almost always at least as expensive as the previous two schemes, because in general it requires at least two passes on X. Moreover, since the assignment of a vector is postponed until enough information becomes available, it turns out that this algorithm is less sensitive to the order of data presentation.

As in the previous case, different choices of the dissimilarity between a vector and a cluster lead to different results. This algorithm also favors compact clusters, when used with point cluster representatives.

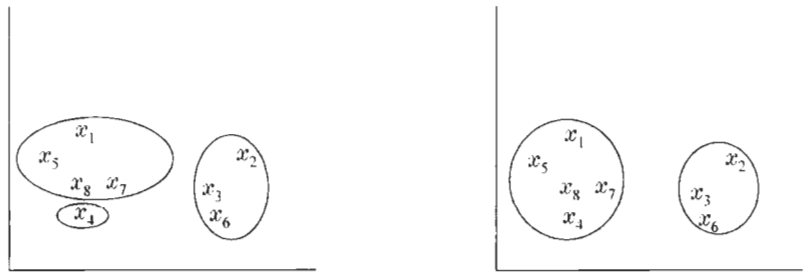
Remark

- Note that for all these algorithms no deadlock state occurs. That is, none of the algorithms enters into a state where there exist unassigned vectors that cannot be assigned either to existing clusters or to new ones, regardless of the number of passes of the data to the algorithm. The BSAS and MBSAS algorithms are guaranteed to terminate after a single and after two passes on X, respectively. In TTSAS the deadlock situation is avoided, as we arbitrarily assign the first unassigned vector at the current pass to a new cluster if no assignment of vectors occurred in the previous pass.

Example 12.3. Consider the vectors $\boldsymbol{x}_1 = [2, 5]^T$, $\boldsymbol{x}_2 = [6, 4]^T$, $\boldsymbol{x}_3 = [5, 3]^T$, $\boldsymbol{x}_4 = [2, 2]^T$, $\boldsymbol{x}_5 = [1, 4]^T$, $\boldsymbol{x}_6 = [5, 2]^T$, $\boldsymbol{x}_7 = [3, 3]^T$, and $\boldsymbol{x}_8 = [2, 3]^T$. The distance from a vector $\boldsymbol{x}$ to a cluster C is taken to be the Euclidean distance between $\boldsymbol{x}$ and the mean vector of C. If we present the vectors in the above order to the MBSAS algorithm and we set $\Theta = 2.5$, we obtain three clusters, $C_1 = \{\boldsymbol{x}_1, \boldsymbol{x}_5, \boldsymbol{x}_7, \boldsymbol{x}_8\}$, $C_2 = \{\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_6\}$, and $C_3 = \{\boldsymbol{x}_4\}$ (see Figure 12.3a).

On the other hand, if we present the vectors in the above order to the TTSAS algorithm, with $\Theta_1 = 2.2$ and $\Theta_2 = 4$, we obtain $C_1 = \{\boldsymbol{x}_1, \boldsymbol{x}_5, \boldsymbol{x}_7, \boldsymbol{x}_8, \boldsymbol{x}_4\}$ and $C_2 = \{\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_6\}$ (see Figure 12.3b). In this case, all vectors were assigned to clusters during the first pass on X, except $\boldsymbol{x}_4$. This was assigned to cluster C_1 during the second pass on X. At each pass on X, we had at least one vector assignment to a cluster. Thus, no vector is forced to a new cluster arbitrarily.

It is clear that the last algorithm leads to more reasonable results than MBSAS. However, it should be noted that MBSAS also leads to the same clustering if, for example, the vectors are presented with the following order: $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_5, \boldsymbol{x}_3, \boldsymbol{x}_8, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_4$.



(a) (b)

FIGURE 12.3: (a) The clustering produced by the MBSAS. (b) The clustering produced by the TTSAS.

12.6 REFINEMENT STAGES

In all the preceding algorithms, it may happen that two of the formed clusters are very closely located, and it may be desirable to merge them into a single one. Such cases cannot be handled by these algorithms. One way out of this problem is to run the following simple merging procedure, after the termination of the preceding schemes (see [Fu 93]).

Merging procedure

- (A) Find C_i, C_j $(i < j)$ such that $d(C_i, C_j) = \min_{k,r=1,\ldots,m,\ k \neq r} d(C_k, C_r)$
- If $d(C_i, C_j) \leq M_1$ then
  - —Merge C_i, C_j to C_i and eliminate C_j
  - —Update the cluster representative of C_i (if cluster representatives are used)
  - —Rename the clusters $C_{j+1}, \ldots, C_m$ to $C_j, \ldots, C_{m-1}$, respectively.
  - —$m = m - 1$
  - —Go to (A)
- Else
  - —Stop
- End {If}

M_1 is a user-defined parameter that quantifies the closeness of two clusters, C_i and C_j. The dissimilarity $d(C_i, C_j)$ between the clusters can be defined using the definitions given in Chapter 11.

The other drawback of the sequential algorithms is their sensitivity to the order of presentation of vectors. Suppose, for example, that in using BSAS, $\boldsymbol{x}_2$ is assigned to the first cluster, C_1, and after the termination of the algorithm four clusters are formed. Then it is possible for $\boldsymbol{x}_2$ to be closer to a cluster different from C_1.

However, there is no way for x_2 to move to its closest cluster once assigned to another one. A simple way to face this problem is to use the following reassignment procedure:

Reassignment procedure

- For $i = 1$ to N
  - —Find C_j such that $d(x_i, C_j) = \min_{k=1,\ldots,m} d(x_i, C_k)$
  - —Set $b(i) = j$
- End {For}
- For $j = 1$ to m
  - —Set $C_j = \{x_i \in X: b(i) = j\}$
  - —Update the representatives (if used)
- End {For}

In this procedure, $b(i)$ denotes the closest to x_i cluster. This procedure may be used after the termination of the algorithms or, if the merging procedure is also used, after the termination of the merging procedure.

A variant of the BSAS algorithm combining the two refinement procedures has been proposed in [MacQ 67]. Only the case in which point representatives are used is considered. According to this algorithm, instead of starting with a single cluster, we start with $m > 1$ clusters, each containing one of the first m of the vectors in X. We apply the merging procedure and then we present each of the remaining vectors to the algorithm. After assigning the current vector to a cluster and updating its representative, we run the merging procedure again. If the distance between a vector x_i and its closest cluster is greater than a prespecified threshold, we form a new cluster which contains only x_i. Finally, after all vectors have been presented to the algorithm, we run the reassignment procedure once. The merging procedure is applied $N - m + 1$ times. Finally, a variant of the algorithm is given in [Ande 73].

A different sequential clustering algorithm that requires a single pass on X is discussed in [Mant 85]. More specifically, it is assumed that the vectors are produced by a mixture of k Gaussian probability densities, $p(x|C_i)$, that is,

$$p(x) = \sum_{j=1}^{k} P(C_j) p(x|C_j; \mu_j, \Sigma_j) \tag{12.6}$$

where μ_j and Σ_j are the mean and the covariance matrix of the jth Gaussian distribution, respectively. Also, $P(C_j)$ is the a priori probability for C_j. For convenience, let us assume that all $P(C_j)$'s are equal to each other. The clusters formed by the algorithm are assumed to follow the Gaussian distribution. At the beginning, a single cluster is formed using the first vector. Then, for each newly

arrived vector, $\boldsymbol{x}_i$, the mean vector and covariance matrix of each of the m clusters, formed up to now, are appropriately updated and the conditional probabilities $P(C_j|\boldsymbol{x}_i)$ are estimated. If $P(C_q|\boldsymbol{x}_i) = \max_{j=1,\ldots,m} P(C_j|\boldsymbol{x}_i)$ is greater than a prespecifed threshold a, then $\boldsymbol{x}_i$ is assigned to C_q. Otherwise, a new cluster is formed where $\boldsymbol{x}_i$ is assigned.

12.7 NEURAL NETWORK IMPLEMENTATION

In this section, a neural network architecture is introduced and is then used to implement BSAS.

12.7.1 Description of the Architecture

The architecture is shown in Figure 12.4a. It consists of two modules, the matching score generator (MSG) and the MaxNet network (MN).[4]

The first module stores q parameter vectors[5] $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_q$ of dimension $l \times 1$ and implements a function $f(\boldsymbol{x}, \boldsymbol{w})$, which indicates the similarity between $\boldsymbol{x}$ and $\boldsymbol{w}$. The higher the value of $f(\boldsymbol{x}, \boldsymbol{w})$, the more similar $\boldsymbol{x}$ and $\boldsymbol{w}$ are.

When a vector $\boldsymbol{x}$ is presented to the network, the MSG module outputs a $q \times 1$ vector $\boldsymbol{v}$, with its ith coordinate being equal to $f(\boldsymbol{x}, \boldsymbol{w}_i)$, $i = 1, \ldots, q$.

The second module takes as input the vector $\boldsymbol{v}$ and identifies its maximum coordinate. Its output is a $q \times 1$ vector $\boldsymbol{s}$ with all its components equal to 0 except one that corresponds to the maximum coordinate of $\boldsymbol{v}$. This is set equal to 1. Most of the modules of this type require at least one coordinate of $\boldsymbol{v}$ to be positive.

Different implementations of the MSG can be used, depending on the proximity measure adopted. For example, if the function f is the inner product, the MSG module consists of q *linear* nodes with their threshold being equal to 0. Each of these nodes is associated with a parameter vector $\boldsymbol{w}_i$, and its output is the inner product of the input vector $\boldsymbol{x}$ with $\boldsymbol{w}_i$.

If the Euclidean distance is used, the MSG module consists also of q linear nodes. However, a different setup is required. The weight vector associated with the ith node is $\boldsymbol{w}_i$ and its threshold is set equal to $T_i = \frac{1}{2}(Q - \|\boldsymbol{w}_i\|^2)$, where Q is a positive constant that ensures that at least one of the first layer nodes will output a positive matching score, and $\|\boldsymbol{w}_i\|$ is the Euclidean norm of $\boldsymbol{w}_i$. Thus, the output of the node is

$$f(\boldsymbol{x}, \boldsymbol{w}_i) = \boldsymbol{x}^T \boldsymbol{w}_i + \frac{1}{2}(Q - \|\boldsymbol{w}_i\|^2) \tag{12.7}$$

[4]This is a generalization of the Hamming network proposed in [Lipp 87].

[5]These are also called exemplar patterns.

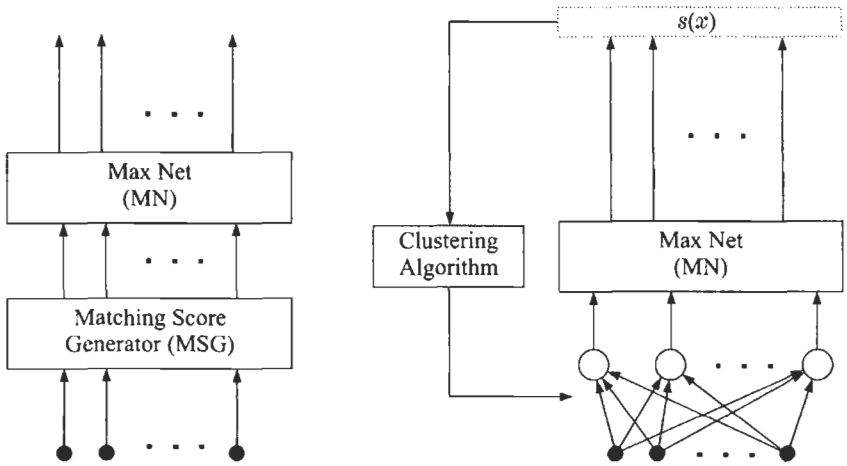


FIGURE 12.4: (a) The neural architecture. (b) Implementation of the BSAS algorithm when each cluster is represented by its mean vector and the Euclidean distance between two vectors is used.

It is easy to show that $d_2(\boldsymbol{x}, \boldsymbol{w}_i) < d_2(\boldsymbol{x}, \boldsymbol{w}_j)$ is equivalent to $f(\boldsymbol{x}, \boldsymbol{w}_i) > f(\boldsymbol{x}, \boldsymbol{w}_j)$ and thus the output of MSG corresponds to the $\boldsymbol{w}_i$ with the minimum Euclidean distance from $\boldsymbol{x}$ (see Problem 12.8).

The MN module can be implemented via a number of alternatives. One can use either neural network comparators such as the Hamming MaxNet and other feed-forward architectures [Lipp 87, Kout 95, Kout 98] or conventional comparators [Mano 79].

12.7.2 Implementation of the BSAS Algorithm

In this section, we demonstrate how the BSAS algorithm can be mapped to the neural network architecture when (a) each cluster is represented by its mean vector and (b) the Euclidean distance between two vectors is used (see Figure 12.4b). The structure of the Hamming network must also be slightly modified, so that each node in the first layer to have as an extra input the term $-\frac{1}{2}\|\boldsymbol{x}\|^2$. Let $\boldsymbol{w}_i$ and T_i be the weight vector and the threshold of the ith node in the MSG module, respectively. Also let $\boldsymbol{a}$ be a $q \times 1$ vector whose ith component indicates the number of vectors contained in the ith cluster. Also, let $\boldsymbol{s}(\boldsymbol{x})$ be the output of the MN module when the input to the network is $\boldsymbol{x}$. In addition, let t_i be the connection between the

ith node of the MSG and its corresponding node in the MN module. Finally, let $sgn(z)$ be the step function that returns 1 if $z > 0$ and 0 otherwise.

The first m of the q $\boldsymbol{w}_i$'s correspond to the representatives of the clusters defined so far by the algorithm. At each iteration step either one of the first m $\boldsymbol{w}_i$'s is updated or a new parameter vector $\boldsymbol{w}_{m+1}$ is employed, whenever a new cluster is created (if $m < q$). The algorithm may be stated as follows.

- Initialization
  - —$\boldsymbol{a} = \boldsymbol{0}$
  - —$\boldsymbol{w}_i = \boldsymbol{0}, i = 1, \ldots, q$
  - —$t_i = 0, i = 1, \ldots, q$
  - —$m = 1$
  - —For the first vector $\boldsymbol{x}_1$ set
    - * $\boldsymbol{w}_1 = \boldsymbol{x}_1$
    - * $a_1 = 1$
    - * $t_1 = 1$
- Main Phase
  - —Repeat
    - * Present the next vector $\boldsymbol{x}$ to the network.
    - * Compute the output vector $\boldsymbol{s}(\boldsymbol{x})$.
    - * $GATE(\boldsymbol{x}) = AND((1 - \sum_{j=1}^{q}(s_j(\boldsymbol{x}))), sgn(q - m))$.
    - * $m = m + GATE(\boldsymbol{x})$
    - * $a_m = a_m + GATE(\boldsymbol{x})$
    - * $\boldsymbol{w}_m = \boldsymbol{w}_m + GATE(\boldsymbol{x})\boldsymbol{x}$
    - * $T_m = \Theta - \frac{1}{2}\|\boldsymbol{w}_m\|^2$
    - * $t_m = 1$
    - * For $j = 1$ to m
      - $a_j = a_j + (1 - GATE(\boldsymbol{x}))s_j(\boldsymbol{x})$
      - $\boldsymbol{w}_j = \boldsymbol{w}_j - (1 - GATE(\boldsymbol{x}))s_j(\boldsymbol{x})(\frac{1}{a_j}(\boldsymbol{w}_j - \boldsymbol{x}))$
      - $T_j = \Theta - \frac{1}{2}\|\boldsymbol{w}_j\|^2$
    - * Next j
  - —Until all vectors have been presented once to the network.

Note that only the outputs of the m first nodes of the MSG module are taken into account, because only these correspond to clusters. The outputs of the remaining nodes are not taken into account, since $t_k = 0$, $k = m + 1, \ldots, q$. Assume that a new vector is presented to the network such that $\min_{1 \le j \le m} d(\boldsymbol{x}, \boldsymbol{w}_j) > \Theta$ and $m < q$. Then $GATE(\boldsymbol{x}) = 1$. Therefore, a new cluster is created and the next node is activated in order to represent it. Since $1 - GATE(\boldsymbol{x}) = 0$, the execution of the instructions in the *For* loop does not affect any of the parameters of the network.

Suppose next that $GATE(\boldsymbol{x}) = 0$. This is equivalent to the fact that either $\min_{1 \le j \le m} d(\boldsymbol{x}, \boldsymbol{w}_j) \le \Theta$ or there are no more nodes available to represent

additional clusters. Then the execution of the instructions in the For loop results in updating the weight vector and the threshold of the node, k, for which $\min_{1 \leq j \leq m} d(\boldsymbol{x}, \boldsymbol{w}_j)$. This happens because $s_k(\boldsymbol{x}) = 1$ and $s_j(\boldsymbol{x}) = 0$, $j = 1, \ldots, q$, $j \neq k$.

Problems

12.1 Prove Eq. (12.3) using induction.

12.2 Prove Eq. (12.5).

12.3 This problem aims at the investigation of the effects of the ordering of presentation of the vectors in the BSAS and MBSAS algorithms. Consider the following two-dimensional vectors: $\boldsymbol{x}_1 = [1, 1]^T$, $\boldsymbol{x}_2 = [1, 2]^T$, $\boldsymbol{x}_3 = [2, 2]^T$, $\boldsymbol{x}_4 = [2, 3]^T$, $\boldsymbol{x}_5 = [3, 3]^T$, $\boldsymbol{x}_6 = [3, 4]^T$, $\boldsymbol{x}_7 = [4, 4]^T$, $\boldsymbol{x}_8 = [4, 5]^T$, $\boldsymbol{x}_9 = [5, 5]^T$, $\boldsymbol{x}_{10} = [5, 6]^T$, $\boldsymbol{x}_{11} = [-4, 5]^T$, $\boldsymbol{x}_{12} = [-3, 5]^T$, $\boldsymbol{x}_{13} = [-4, 4]^T$, $\boldsymbol{x}_{14} = [-3, 4]^T$. Also consider the case that each cluster is represented by its mean vector.

(a) Run the BSAS and the MBSAS algorithms when the vectors are presented in the given order. Use the Euclidean distance between two vectors and take $\Theta = \sqrt{2}$.

(b) Change the order of presentation to $\boldsymbol{x}_1, \boldsymbol{x}_{10}, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_{11}, \boldsymbol{x}_{12}, \boldsymbol{x}_5, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_{13}, \boldsymbol{x}_8, \boldsymbol{x}_{14}, \boldsymbol{x}_9$ and rerun the algorithms.

(c) Run the algorithms for the following order of presentation: $\boldsymbol{x}_1, \boldsymbol{x}_{10}, \boldsymbol{x}_5, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_{11}, \boldsymbol{x}_{12}, \boldsymbol{x}_4, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_{13}, \boldsymbol{x}_{14}, \boldsymbol{x}_8, \boldsymbol{x}_9$.

(d) Plot the given vectors and discuss the results of these runs.

(e) Perform a visual clustering of the data. How many clusters do you claim are formed by the given vectors?

12.4 Consider the setup of Example 12.2. Run BSAS and MBSAS algorithms, with $\Theta = 5$, using the mean vector as representative for each cluster. Discuss the results.

12.5 Consider Figure 12.5. The inner square has side $S_1 = 0.3$, and the sides of the inner and outer square of the outer frame are $S_2 = 1$ and $S_3 = 1.3$, respectively. The inner square contains 50 points that stem from a uniform distribution in the square. Similarly, the outer frame contains 50 points that stem from a uniform distribution in the frame.

(a) Perform a visual clustering of the data. How many clusters do you claim are formed by the given points?

(b) Consider the case in which each cluster is represented by its mean vector and the Euclidean distance between two vectors is employed. Run BSAS and MBSAS algorithms, with

$$\Theta = \min_{i,j=1,\ldots,100} d(\boldsymbol{x}_i, \boldsymbol{x}_j), \quad \text{to} \quad \max_{i,j=1,\ldots,100} d(\boldsymbol{x}_i, \boldsymbol{x}_j) \text{ with step } 0.2$$

and with random ordering of the data. Give a quantitative explanation for the results. Compare them with the results obtained from the previous problem.

(c) Repeat (b) for the case in which d_{min}^{ps} is chosen as the dissimilarity between a vector and a cluster (see Chapter 11).

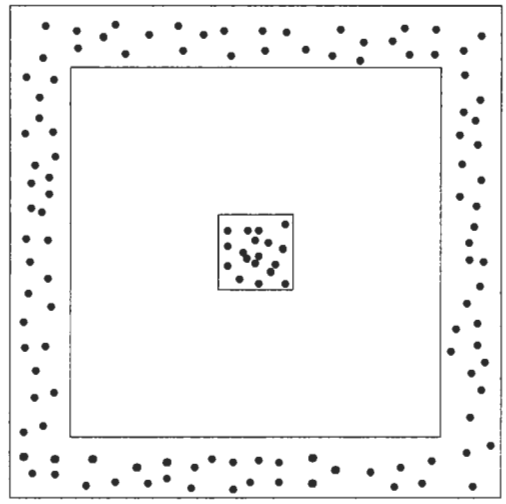


FIGURE 12.5: The setup of Problem 12.5.

12.6 Consider three two-dimensional Gaussian distributions with means $[0, 0]^T$, $[6, 0]^T$ and $[12, 6]^T$, respectively. The covariance matrices for all distributions are equal to the identity matrix I. Generate 30 points from each distribution and let X be the resulting data set. Employ the Euclidean distance and apply the procedure discussed in section 12.3.1 for the estimation of the number of clusters underlying in X, with $a = \min_{i,j=1,\ldots,100} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $b = \max_{i,j=1,\ldots,100} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $c = 0.3$. Plot m versus Θ and draw your conclusions.

12.7 Let s be a similarity measure between a vector and a cluster. Express the BSAS, MBSAS, and TTSAS algorithms in terms of s.

12.8 Show that when the Euclidean distance between two vectors is in use and the output function of the MSG module is given by Eq. (12.7), the relations $d_2(\boldsymbol{x}, \boldsymbol{w}_1) < d_2(\boldsymbol{x}, \boldsymbol{w}_2)$ and $f(\boldsymbol{x}, \boldsymbol{w}_1) > f(\boldsymbol{x}, \boldsymbol{w}_2)$ are equivalent.

12.9 Describe a neural network implementation similar to the one given in Section 12.7 for the BSAS algorithm when each cluster is represented by the first vector assigned to it.

12.10 The neural network architecture that implements the MBSAS algorithm, if the mean vector is in use, is similar to the one given in Figure 12.4b for the Euclidean distance case. Write the algorithm in a form similar to the one given in Section 12.7 for the MBSAS when the mean vector is in use, and highlight the differences between the two implementations.

References

[Ande 73] Anderberg M.R. *Cluster Analysis for Applications*, Academic Press, 1973.

[Ball 65] Ball G.H. "Data analysis in social sciences," *Proceedings FJCC*, Las Vegas, 1965.

[Burk 91] Burke L.I. "Clustering characterization of adaptive reasonance," *Neural Networks*, Vol. 4, pp. 485–491, 1991.

[Carp 87] Carpenter G.A., Grossberg S. "ART2: Self-organization of stable category recognition codes for analog input patterns," *Applied Optics*, Vol. 26, pp. 4919–4930, 1987.

[Duda 73] Duda R.O., Hart P. *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[Flor 91] Floreen P. "The convergence of the Hamming memory networks," *IEEE Transactions on Neural Networks*, Vol. 2(4), July 1991.

[Fu 93] Fu L., Yang M., Braylan R., Benson N. "Real-time adaptive clustering of flow cytometric data," *Pattern Recognition*, Vol. 26(2), pp. 365–373, 1993.

[Hall 67] Hall A.V. "Methods for demonstrating resemblance in taxonomy and ecology," *Nature*, Vol. 214, pp. 830–831, 1967.

[Jain 88] Jain A.K., Dubes R.C. *Algorithms for Clustering Data*, Prentice Hall, 1988.

[Kout 94] Koutroumbas K., Kalouptsidis N. "Qualitative analysis of the parallel and asynchronous modes of the Hamming network," *IEEE Transactions on Neural Networks*, Vol. 5(3), pp. 380–391, May 1994.

[Kout 95] Koutroumbas K. "Hamming neural networks, architecture design and applications," Ph.D. thesis, Department of Informatics, University of Athens, 1995.

[Kout 98] Koutroumbas K., Kalouptsidis N. "Neural network architectures for selecting the maximum input," *International Journal of Computer Mathematics*, Vol. 68(1–2), 1998.

[Lipp 87] Lippmann R.P. "An introduction to computing with neural nets," *IEEE ASSP Magazine*, Vol. 4(2), April 1987.

[Liu 68] Liu C.L. *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.

[Lloy 82] Lloyd S.P. "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, Vol. 28(2), March 1982.

[MacQ 67] MacQuenn J.B. "Some methods for classification and analysis of multivariate observations," *Proceedings of the Symposium on Mathematical Statistics and Probability*, 5th Berkeley, Vol. 1, pp. 281–297, AD 669871, University of California Press, 1967.

[Mano 79] Mano M. *Digital Logic and Computer Design*, Prentice Hall, 1979.

[Mant 85] Mantaras R.L., Aguilar-Martin J. "Self-learning pattern classification using a sequential clustering technique," *Pattern Recognition*, Vol. 18(3/4), pp. 271–277, 1985.

[Sebe 62] Sebestyen G.S. "Pattern recognition by an adaptive process of sample set construction," *IRE Transactions on Information Theory*, Vol. 8(5), pp. S82–S91, 1962.

[Snea 73] Sneath P.H.A., Sokal R.R. *Numerical Taxonomy*, W.H. Freeman, San Francisco, 1973.

[Spat 80] Spath H. *Cluster Analysis Algorithms*, Ellis Horwood, 1980.

[Trah 89] Trahanias P., Scordalakis E. "An efficient sequential clustering method," *Pattern Recognition*, Vol. 22(4), pp. 449–453, 1989.

CHAPTER 13

CLUSTERING ALGORITHMS II: HIERARCHICAL ALGORITHMS

13.1 INTRODUCTION

Hierarchical clustering algorithms are of a different philosophy from the algorithms described in the previous chapter. Specifically, instead of producing a single clustering they produce a hierarchy of clusterings. This kind of algorithms is usually met in the social sciences and biological taxonomy (e.g., [El-G 68, Prit 71, Shea 65, McQu 62]). In addition, they have been used in many other fields, including modern biology, medicine, and archaeology (e.g., [Stri 67, Bobe 93, Solo 71, Hods 71]). Also, applications of the hierarchical algorithms may be found in computer science and engineering (e.g., [Murt 95, Kank 96]).

Before we describe their basic idea, let us recall that

$$X = \{\boldsymbol{x}_i,\ i = 1, \ldots, N\}$$

is a set of l-dimensional vectors that are to be clustered. Also, recall from Chapter 11 the definition of a clustering

$$\mathcal{R} = \{C_j,\ j = 1, \ldots, m\}$$

where $C_j \subseteq X$.

A clustering $\mathcal{R}_1$ containing k clusters is said to be *nested* in the clustering $\mathcal{R}_2$, which contains $r(<k)$ clusters, if *each* cluster in $\mathcal{R}_1$ is a subset of a set in $\mathcal{R}_2$. Note that at least one cluster of $\mathcal{R}_1$ is a proper subset of $\mathcal{R}_2$. In this case we write $\mathcal{R}_1 \sqsubset \mathcal{R}_2$. For example, the clustering $\mathcal{R}_1 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_3\}, \{\boldsymbol{x}_4\}, \{\boldsymbol{x}_2, \boldsymbol{x}_5\}\}$ is nested in $\mathcal{R}_2 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_3, \boldsymbol{x}_4\}, \{\boldsymbol{x}_2, \boldsymbol{x}_5\}\}$. On the other hand, $\mathcal{R}_1$ is nested neither in $\mathcal{R}_3 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_4\}, \{\boldsymbol{x}_3\}, \{\boldsymbol{x}_2, \boldsymbol{x}_5\}\}$ nor in $\mathcal{R}_4 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_4\}, \{\boldsymbol{x}_3, \boldsymbol{x}_5\}\}$. It is clear that a clustering is not nested to itself.

Hierarchical clustering algorithms produce a *hierarchy of nested clusterings*. More specifically, these algorithms involve N steps, as many as the number of

data vectors. At each step t, a new clustering is obtained based on the clustering produced at the previous step $t-1$. There are two main categories of these algorithms, the *agglomerative* and the *divisive hierarchical algorithms*.

The initial clustering $\mathcal{R}_0$ for the agglomerative algorithms consists of N clusters each containing a single element of X. At the first step, the clustering $\mathcal{R}_1$ is produced. It contains $N-1$ sets, such that $\mathcal{R}_0 \sqsubset \mathcal{R}_1$. This procedure continues until the final clustering, $\mathcal{R}_{N-1}$, is obtained, which contains a single set, that is, the set of data, X. Notice that for the hierarchy of the resulting clusterings we have

$$\mathcal{R}_0 \sqsubset \mathcal{R}_1 \sqsubset \cdots \sqsubset \mathcal{R}_{N-1}$$

The divisive algorithms follow the inverse path. In this case, the initial clustering $\mathcal{R}_0$ consists of a single set, X. At the first step the clustering $\mathcal{R}_1$ is produced. It consists of two sets, such that $\mathcal{R}_1 \sqsubset \mathcal{R}_0$. This procedure continues until the final clustering $\mathcal{R}_{N-1}$ is obtained, which contains N sets, each consisting of a single element of X. In this case we have

$$\mathcal{R}_{N-1} \sqsubset \mathcal{R}_{N-2} \sqsubset \ldots, \sqsubset \mathcal{R}_0$$

The next section is devoted to the agglomerative algorithms. The divisive algorithms are briefly discussed in Section 13.4.

13.2 AGGLOMERATIVE ALGORITHMS

Let $g(C_i, C_j)$ be a function defined for all possible pairs of clusters of X. This function measures the proximity between C_i and C_j. Let t denote the current level of hierarchy. Then, the general agglomerative scheme may be stated as follows:

Generalized Agglomerative Scheme (GAS)

- 1. Initialization:

  —1.1. Choose $\mathcal{R}_0 = \{C_i = \{\boldsymbol{x}_i\},\ i = 1, \ldots, N\}$ as the initial clustering.
  —1.2. $t = 0$.

- 2. Repeat:

  —2.1. $t = t + 1$
  —2.2. Among all possible pairs of clusters (C_r, C_s) in $\mathcal{R}_{t-1}$ find the one, say (C_i, C_j), such that

$$g(C_i, C_j) = \begin{cases} \min_{r,s} g(C_r, C_s), & \text{if } g \text{ is a dissimilarity function} \\ \max_{r,s} g(C_r, C_s), & \text{if } g \text{ is a similarity function} \end{cases} \tag{13.1}$$

— 2.3. Define $C_q = C_i \cup C_j$ and produce the new clustering $\mathfrak{R}_t = (\mathfrak{R}_{t-1} - \{C_i, C_j\}) \cup \{C_q\}$.

- Until all vectors lie in a single cluster.

It is clear that this scheme creates a hierarchy of N clusterings, so that each one is nested in all successive clusterings, that is, $\mathfrak{R}_{t_1} \sqsubset \mathfrak{R}_{t_2}$, for $t_1 < t_2, t_2 = 1, \ldots, N-1$. Alternatively, we can say that *if two vectors come together into a single cluster at level t of the hierarchy, they will remain in the same cluster for all subsequent clusterings*. This is another way of viewing the nesting property.

A disadvantage of the nesting property is that there is no way to recover from a "poor" clustering that may have occurred in an earlier level of the hierarchy (see [Gowe 67]).[1]

At each level t, there are $N - t$ clusters. Thus, in order to determine the pair of clusters that is going to be merged at the $t + 1$ level, $\binom{N-t}{2} \equiv \frac{(N-t)(N-t-1)}{2}$ pairs of clusters have to be considered. Thus, the total number of pairs that have to be examined throughout the whole clustering process is

$$\sum_{t=0}^{N-1} \binom{N-t}{2} = \sum_{k=1}^{N} \binom{k}{2} = \frac{(N-1)N(N+1)}{6}$$

that is, the total number of operations required by an agglomerative scheme is proportional to N^3. Moreover, the exact complexity of the algorithm depends on the definition of g.

13.2.1 Definition of Some Useful Quantities

There are two main categories of agglomerative algorithms. Algorithms of the first category are based on matrix theory concepts, while algorithms of the second one are based on graph theory concepts. Before we enter into their discussion, some definitions are required. The *pattern matrix* $D(X)$ is the $N \times l$ matrix, whose ith row is the (transposed) ith vector of X. The *similarity (dissimilarity) matrix*, $P(X)$, is an $N \times N$ matrix whose (i, j) element equals the similarity $s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ (dissimilarity $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$) between vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. It is also referred to as the *proximity matrix* to include both cases. P is a symmetric matrix.[2] Moreover, if P is a similarity matrix, its diagonal elements are equal to the maximum value of s. On the other hand, if P is a dissimilarity matrix, its diagonal elements are equal to the minimum value of d. Notice that for a single pattern matrix there

[1] Recently, a method that produces hierarchies which do not, necessarily, possess the nesting property has been proposed ([Frig 97]).

[2] Note, however, that in [Ozaw 83] a hierarchical clustering algorithm, called RANCOR, is discussed, which is based on asymmetric proximity matrices.

exists more than one proximity matrix depending on the choice of the proximity measure $\wp(\boldsymbol{x}_i, \boldsymbol{x}_j)$. However, fixing $\wp(\boldsymbol{x}_i, \boldsymbol{x}_j)$, one can easily observe that for a given pattern matrix there exists an associated single proximity matrix. On the other hand, a proximity matrix may correspond to more than one pattern matrices (see Problem 13.1).

Example 13.1. Let $X = \{\boldsymbol{x}_i, i = 1, \ldots, 5\}$, with $\boldsymbol{x}_1 = [1, 1]^T$, $\boldsymbol{x}_2 = [2, 1]^T$, $\boldsymbol{x}_3 = [5, 4]^T$, $\boldsymbol{x}_4 = [6, 5]^T$, and $\boldsymbol{x}_5 = [6.5, 6]^T$. The pattern matrix of X is

$$D(X) = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 5 & 4 \\ 6 & 5 \\ 6.5 & 6 \end{bmatrix}$$

and its corresponding dissimilarity matrix, when the Euclidean distance is in use, is

$$P(X) = \begin{bmatrix} 0 & 1 & 5 & 6.4 & 7.4 \\ 1 & 0 & 4.2 & 5.7 & 6.7 \\ 5 & 4.2 & 0 & 1.4 & 2.5 \\ 6.4 & 5.7 & 1.4 & 0 & 1.1 \\ 7.4 & 6.7 & 2.5 & 1.1 & 0 \end{bmatrix}$$

When the Tanimoto measure is used, the similarity matrix of X becomes

$$P'(X) = \begin{bmatrix} 1 & 0.75 & 0.26 & 0.21 & 0.18 \\ 0.75 & 1 & 0.44 & 0.35 & 0.20 \\ 0.26 & 0.44 & 1 & 0.96 & 0.90 \\ 0.21 & 0.35 & 0.96 & 1 & 0.98 \\ 0.18 & 0.20 & 0.90 & 0.98 & 1 \end{bmatrix}$$

Note that in $P(X)$ all diagonal elements are 0, since $d_2(\boldsymbol{x}, \boldsymbol{x}) = 0$, while in $P'(X)$ all diagonal elements are equal to 1, since $s_T(\boldsymbol{x}, \boldsymbol{x}) = 1$.

A *threshold dendrogram*, or simply a *dendrogram*, is an effective means of representing the sequence of clusterings produced by an agglomerative algorithm. To clarify this idea, let us consider again the data set given in Example 13.1. Let us define $g(C_i, C_j)$ as $g(C_i, C_j) = d_{min}^{ss}(C_i, C_j)$ (see Section 11.2). One may easily see that, in this case, the clustering sequence for X produced by the generalized agglomerative scheme, when the Euclidean distance between two vectors is used, is the one shown in Figure 13.1. At the first step $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ form a new cluster. At the second step $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$ stick together, forming a single cluster. At the third step $\boldsymbol{x}_3$ joins the cluster $\{\boldsymbol{x}_4, \boldsymbol{x}_5\}$ and, finally, at the fourth step the clusters $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ and $\{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5\}$ are merged into a single set, X. The right-hand side of Figure 13.1 shows the corresponding dendrogram. Each step of GAS corresponds

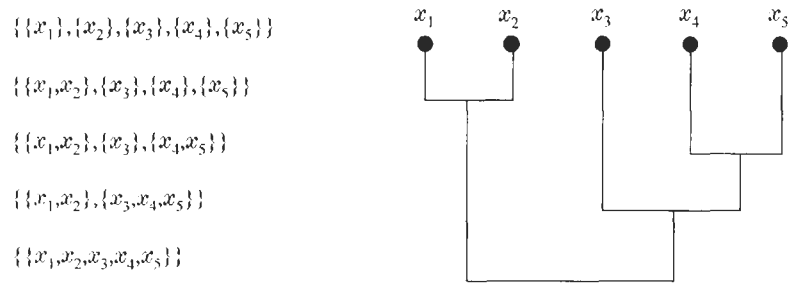


FIGURE 13.1: The clustering hierarchy for X of Example 13.1 and its corresponding dendrogram.

to a level of the dendrogram. *Cutting the dendrogram at a specific level results in a clustering.*

A *proximity dendrogram* is a dendrogram that takes into account the level of proximity where two clusters are merged *for the first time.* When a dissimilarity (similarity) measure is in use, the proximity dendrogram is called a *dissimilarity (similarity) dendrogram.* This tool may be used as an indicator of the natural or forced formation of clusters at any level. That is, it may provide a clue about the clustering that best fits the data, as will be explained in Section 13.5. Figure 13.2 shows the similarity and dissimilarity dendrograms for X of Example 13.1 when $P'(X)$ and $P(X)$ are in use, respectively.

Before we proceed to a more detailed discussion of the hierarchical algorithms, an important note is in order. As explained earlier, this kind of algorithm determines a whole hierarchy of clusterings, rather than a single clustering. The determination of the whole dendrogram may be very useful in some applications, such as biological taxonomy (e.g., see [Prit 71]). However, in other applications we are interested only in the specific clustering that best fits the data. If one is willing to use hierarchical algorithms for applications of the latter type, he or she has to decide which clustering of the produced hierarchy is most suitable for the data. Equivalently, one must determine the appropriate level to cut the dendrogram that corresponds to the resulting hierarchy. Similar comments also hold for the divisive algorithms to be discussed later. Methods for determining the cutting level are discussed in the last section of the chapter.

In the sequel, unless otherwise stated, we consider only dissimilarity matrices. Similar arguments hold for similarity matrices.

13.2.2 Agglomerative Algorithms Based on Matrix Theory

These algorithms may be viewed as special cases of GAS. The input in these schemes is the $N \times N$ dissimilarity matrix, $P_0 = P(X)$, derived from X. At each

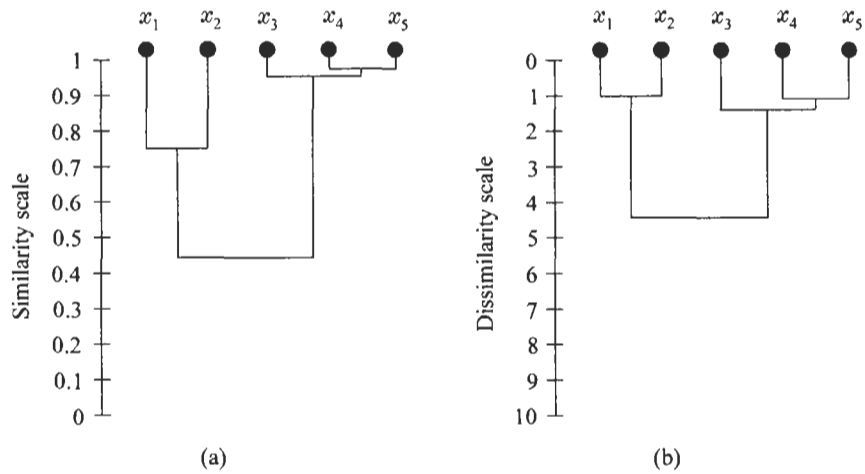


FIGURE 13.2: (a) The proximity (similarity) dendrogram for X using $P'(X)$ from Example 13.1. (b) The proximity (dissimilarity) dendrogram for X using $P(X)$ from Example 13.1.

level, t, when two clusters are merged into one, the size of the dissimilarity matrix P_t becomes $(N-t) \times (N-t)$. P_t follows from P_{t-1} by (a) deleting the two rows and columns that correspond to the merged clusters and (b) adding a new row and a new column that contain the distances between the newly formed cluster and the old (unaffected at this level) clusters. The distance between the newly formed cluster C_q (the result of merging C_i and C_j) and an old cluster, C_s, is a function of the form

$$d(C_q, C_s) = f(d(C_i, C_s), d(C_j, C_s), d(C_i, C_j)) \tag{13.2}$$

The procedure justifies the name *matrix updating algorithms*, often used in the literature. In the sequel, we give an algorithmic scheme, the *matrix updating algorithmic scheme (MUAS)*, that includes most of the algorithms of this kind. Again, t denotes the current level of the hierarchy.

Matrix Updating Algorithmic Scheme (MUAS)

- 1. Initialization:

  —1.1. $\Re_0 = \{\{\boldsymbol{x}_i\}, i = 1, \ldots, N\}$.
  —1.2. $P_0 = P(X)$.
  —1.3. $t = 0$

- 2. Repeat:
  - —2.1. $t = t + 1$
  - —2.2. Find C_i, C_j such that $d(C_i, C_j) = \min_{r,s=1,\ldots,N,\ r\neq s} d(C_r, C_s)$.
  - —2.3. Merge C_i, C_j into a single cluster C_q and form $\mathfrak{R}_t = (\mathfrak{R}_{t-1} - \{C_i, C_j\}) \cup \{C_q\}$.
  - —2.4. Define the proximity matrix P_t from P_{t-1} as explained in the text.
- Until $\mathfrak{R}_{N-1}$ clustering is formed, that is, all vectors lie in the same cluster.

Notice that this scheme is in the spirit of the GAS. In [Lanc 67] it is pointed out that a number of distance functions comply with the following update equation:

$$d(C_q, C_s) = a_i d(C_i, C_s) + a_j d(C_j, C_s) + b d(C_i, C_j) + c|d(C_i, C_s) - d(C_j, C_s)| \tag{13.3}$$

Different values of a_i, a_j, b, and c correspond to different choices of the dissimilarity measure $d(C_i, C_j)$. Equation (13.3) is also a recursive definition of a distance between two clusters, initialized from the distance between the initial point clusters. Another formula, not involving the last term and allowing a_i, a_j, and b to be functions of C_i, C_j, and C_s, is discussed in [Bobe 93]. In the sequel we present algorithms stemming from MUAS and following from Eq. (13.3) for different values of the parameters a_i, a_j, b, c.

The simpler algorithms included in this scheme are:

- The *single link algorithm*. This is obtained from Eq. (13.3) if we set $a_i = 1/2$, $a_j = 1/2, b = 0, c = -1/2$. In this case,

$$d(C_q, C_s) = \min\{d(C_i, C_s), d(C_j, C_s)\} \tag{13.4}$$

  The d_{min}^{ss} measure, defined in Section 11.2, falls under this umbrella.
- The *complete link algorithm*. This follows from Eq. (13.3) if we set $a_i = \frac{1}{2}$, $a_j = \frac{1}{2}, b = 0$ and $c = \frac{1}{2}$. Then we may write[3]

$$d(C_q, C_s) = \max\{d(C_i, C_s), d(C_j, C_s)\}. \tag{13.5}$$

Note that the distance between the merged clusters C_i and C_j does not enter into the above formulae. In the case where a similarity, instead of a dissimilarity, measure is used then (a) for the single link algorithm the operator min should be replaced by max in Eq. (13.4) and (b) for the complete link algorithm the operator max should be replaced by the operator min in Eq. (13.5). To gain a further insight into the behavior of the above algorithms, let us consider the following example.

[3]Equations (13.4) and (13.5) suggest that merging clusters is a min/max problem for the complete link and a min/min problem for the single link.

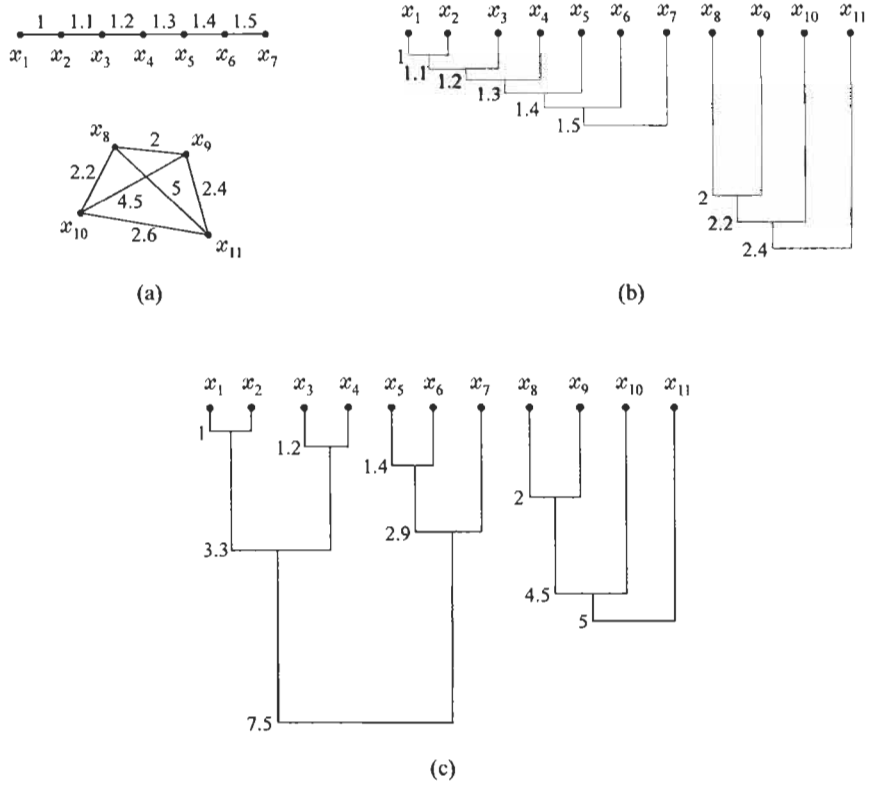


FIGURE 13.3: (a) The data set X. (b) The dissimilarity dendrogram produced by the single link algorithm. (c) The dissimilarity dendrogram produced by the complete link algorithm (the level of the final clustering is not shown).

Example 13.2. Consider the data set shown in Figure 13.3a. The first seven points form an elongated cluster while the remaining four form a rather compact cluster. The numbers on top of the edges connecting the points correspond to the respective (Euclidean) distances between vectors. These distances are also taken to measure the distance between two initial point clusters. Distances that are not shown are assumed to have very large values. Figure 13.3b shows the dendrogram produced by the application of the single link algorithm to this data set. As one can easily observe, the algorithm first recovers the elongated cluster, and the second cluster is recovered at a higher dissimilarity level.

Figure 13.3c shows the dendrogram produced by the complete link algorithm. It is easily noticed that this algorithm proceeds by recovering compact clusters.

Remark

- The preceding algorithms are the two extremes of the family described by Eq. (13.3). Indeed, the clusters produced by the single link algorithm are formed at low dissimilarities in the dissimilarity dendrogram. On the other hand, the clusters produced by the complete link algorithm are formed at high dissimilarities in the dissimilarity dendrogram. This happens because in the single link (complete link) algorithm the minimum (maximum) of the distances $d(C_i, C_s)$ and $d(C_j, C_s)$ is used as the distance between $d(C_q, C_s)$. This implies that the single link algorithm has a tendency to favour elongated clusters. This characteristic is also known as *chaining effect*. On the other hand, the complete link algorithm proceeds by recovering small compact clusters, and it should be preferred if there is evidence that compact clusters underlie X.

The rest of the algorithms, to be discussed next, are compromises between these two extremes.[4]

- The *weighted pair group method average (WPGMA)* algorithm is obtained from Eq. (13.3) if we set $a_i = a_j = \frac{1}{2}$, $b = 0$, and $c = 0$, that is,

$$d(C_q, C_s) = \frac{1}{2}(d(C_i, C_s) + d(C_j, C_s)) \tag{13.6}$$

  Thus, in this case the distance between the newly formed cluster C_q and an old one C_s is defined as the average of distances between C_i, C_s and C_j, C_s.
- The *unweighted pair group method average (UPGMA)* algorithm is defined if we choose $a_i = \frac{n_i}{n_i+n_j}$, $a_j = \frac{n_j}{n_i+n_j}$, $b = 0$, $c = 0$, where n_i and n_j are the cardinalities of C_i and C_j, respectively. In this case the distance between C_q and C_s is defined as

$$d(C_q, C_s) = \frac{n_i}{n_i + n_j} d(C_i, C_s) + \frac{n_j}{n_i + n_j} d(C_j, C_s) \tag{13.7}$$

- The *unweighted pair group method centroid (UPGMC)* algorithm results on setting $a_i = \frac{n_i}{n_i+n_j}$, $a_j = \frac{n_j}{n_i+n_j}$, $b = -\frac{n_i n_j}{(n_i+n_j)^2}$, $c = 0$, that is,

$$d_{qs} = \frac{n_i}{n_i + n_j} d_{is} + \frac{n_j}{n_i + n_j} d_{js} - \frac{n_i n_j}{(n_i + n_j)^2} d_{ij} \tag{13.8}$$

[4]The terminology used here follows that given in [Jain 88].

This algorithm has an interesting interpretation. Let the representatives of the clusters be chosen as the respective means (centroids), that is,

$$\boldsymbol{m}_q = \frac{1}{n_q} \sum_{\boldsymbol{x} \in C_q} \boldsymbol{x} \tag{13.9}$$

and the dissimilarity to be the squared Euclidean distance between cluster representatives. Then it turns out that this recursive definition of d_{qs} is nothing but the square Euclidean distance between the respective representatives (see Problem 13.2), that is,

$$d_{qs} = \|\boldsymbol{m}_q - \boldsymbol{m}_s\|^2 \tag{13.10}$$

- The *weighted pair group method centroid (WPGMC)* algorithm is obtained if we choose $a_i = a_j = \frac{1}{2}$, $b = -\frac{1}{4}$, and $c = 0$. That is,

$$d_{qs} = \frac{1}{2} d_{is} + \frac{1}{2} d_{js} - \frac{1}{4} d_{ij} \tag{13.11}$$

Note that Eq. (13.11) results from (13.8) if the merging clusters have the same number of vectors. Of course, this is not true in general, and the algorithm basically computes the distance between weighted versions of the respective centroids. A notable feature of the WPGMC algorithm is that there are cases where $d_{qs} \leq \min(d_{is}, d_{js})$ (Problem 13.3).

- The *Ward or minimum variance algorithm*. Here, the distance between two clusters C_i and C_j, d'_{ij}, is defined as a weighted version of the squared Euclidean distance of their mean vectors, that is,

$$d'_{ij} = \frac{n_i n_j}{n_i + n_j} d_{ij} \tag{13.12}$$

where $d_{ij} = \|\boldsymbol{m}_i - \boldsymbol{m}_j\|^2$. Thus, in step 2.2 of MUAS we seek the pair of clusters C_i, C_j so that the quantity d'_{ij} is minimum. Furthermore, it can be shown (Problem 13.4) that this distance belongs to the family of Eq. (13.3) and we can write

$$d'_{qs} = \frac{n_i + n_s}{n_i + n_j + n_s} d'_{is} + \frac{n_j + n_s}{n_i + n_j + n_s} d'_{js} - \frac{n_s}{n_i + n_j + n_s} d'_{ij} \tag{13.13}$$

The preceding distance can also be viewed from a different perspective. Let us define

$$e_r^2 = \sum_{\boldsymbol{x} \in C_r} \|\boldsymbol{x} - \boldsymbol{m}_r\|^2$$

as the variance of the rth cluster around its mean and

$$E_t = \sum_{r=1}^{N-t} e_r^2 \tag{13.14}$$

as the total variance of the clusters at the tth level (where $N - t$ clusters are present). We will now show that *Ward's algorithm forms $\Re_{t+1}$ by merging the two clusters that lead to the smallest possible increase of the total variance.* Suppose that clusters C_i and C_j are chosen to be merged into one, say C_q. Let E_{t+1}^{ij} be the total variance after the clusters C_i and C_j are merged in C_q at the $t+1$ level. Then, since all other clusters remain unaffected, the difference $\Delta E_{t+1}^{ij} = E_{t+1}^{ij} - E_t$ is equal to

$$\Delta E_{t+1}^{ij} = e_q^2 - e_i^2 - e_j^2 \tag{13.15}$$

Taking into account that

$$\sum_{\boldsymbol{x} \in C_r} \|\boldsymbol{x} - \boldsymbol{m}_r\|^2 = \sum_{\boldsymbol{x} \in C_r} \|\boldsymbol{x}\|^2 - n_r \|\boldsymbol{m}_r\|^2 \tag{13.16}$$

Eq. (13.15) is written as

$$\Delta E_{t+1}^{ij} = n_i \|\boldsymbol{m}_i\|^2 + n_j \|\boldsymbol{m}_j\|^2 - n_q \|\boldsymbol{m}_q\|^2 \tag{13.17}$$

Using the fact that

$$n_i \boldsymbol{m}_i + n_j \boldsymbol{m}_j = n_q \boldsymbol{m}_q \tag{13.18}$$

Eq. (13.17) becomes

$$\Delta E_{t+1}^{ij} = \frac{n_i n_j}{n_i + n_j} \|\boldsymbol{m}_i - \boldsymbol{m}_j\|^2 = d'_{ij} \tag{13.19}$$

which is the distance minimized by Ward's algorithm. This justifies the name minimum variance.

Example 13.3. Consider the following dissimilarity matrix:

$$P_0 = \begin{bmatrix} 0 & 1 & 2 & 26 & 37 \\ 1 & 0 & 3 & 25 & 36 \\ 2 & 3 & 0 & 16 & 25 \\ 26 & 25 & 16 & 0 & 1.5 \\ 37 & 36 & 25 & 1.5 & 0 \end{bmatrix}$$

where the corresponding squared Euclidean distance is adopted. As one can easily observe, the first three vectors, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_3$, are very close to each other and far away from

the others. Likewise, $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$ lie very close to each other and far away from the first three vectors. For this problem all seven algorithms discussed before result in the same dendrogram. The only difference is that each clustering is formed at a different dissimilarity level.

Let us first consider the single link algorithm. Since P_0 is symmetric, we consider only the upper diagonal elements. The smallest of these elements equals 1 and occurs at position $(1, 2)$ of P_0. Thus, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ come into the same cluster and $\mathcal{R}_1 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \{\boldsymbol{x}_3\}, \{\boldsymbol{x}_4\}, \{\boldsymbol{x}_5\}\}$ is produced. In the sequel, the dissimilarities among the newly formed cluster and the remaining ones have to be computed. This can be achieved via Eq. (13.4). The resulting proximity matrix, P_1, is

$$P_1 = \begin{bmatrix} 0 & 2 & 25 & 36 \\ 2 & 0 & 16 & 25 \\ 25 & 16 & 0 & 1.5 \\ 36 & 25 & 1.5 & 0 \end{bmatrix}$$

Its first row and column correspond to the cluster $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$. The smallest of the upper diagonal elements of P_1 equals 1.5. This means that at the next stage, the clusters $\{\boldsymbol{x}_4\}$ and $\{\boldsymbol{x}_5\}$ will get together into a single cluster, producing $\mathcal{R}_2 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \{\boldsymbol{x}_3\}, \{\boldsymbol{x}_4, \boldsymbol{x}_5\}\}$. Employing Eq. (13.4), we obtain

$$P_2 = \begin{bmatrix} 0 & 2 & 25 \\ 2 & 0 & 16 \\ 25 & 16 & 0 \end{bmatrix}$$

where the first row (column) corresponds to $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$, and the second and third rows (columns) correspond to $\{\boldsymbol{x}_3\}$ and $\{\boldsymbol{x}_4, \boldsymbol{x}_5\}$, respectively. Proceeding as before, at the next stage $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ and $\{\boldsymbol{x}_3\}$ will get together in a single cluster and $\mathcal{R}_3 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}, \{\boldsymbol{x}_4, \boldsymbol{x}_5\}\}$ is produced. The new proximity matrix, P_3, becomes

$$P_3 = \begin{bmatrix} 0 & 16 \\ 16 & 0 \end{bmatrix}$$

where the first and the second row (column) correspond to $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$ and $\{\boldsymbol{x}_4, \boldsymbol{x}_5\}$ clusters, respectively. Finally, $\mathcal{R}_4 = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5\}\}$ will be formed at dissimilarity level equal to 16.

Working in a similar fashion, we can apply the remaining six algorithms to P_0. Note that in the case of Ward's algorithm, the initial dissimilarity matrix should be $\frac{1}{2}P_0$, due to the definition in Eq. (13.12). However, care must be taken when we apply UPGMA, UPGMC, and Ward's method. In these cases, when a merging takes place the parameters a_i, a_j, b, and c must be properly adjusted. The proximity levels at which each clustering is formed for each algorithm are shown in Table 13.1.

It is worth noting that the considered task is a nice problem with two well-defined compact clusters lying away from each other. The preceding example demonstrates that in such "easy" cases all algorithms work satisfactorily (as happens with most of the clustering algorithms proposed in the literature). The particular characteristics of each algorithm are revealed when more demanding situations are faced. Thus, in Example 13.2, we saw the

Table 13.1: The results obtained with the seven algorithms discussed when they are applied to the proximity matrix of Example 13.3

	SL	CL	WPGMA	UPGMA	WPGMC	UPGMC	Ward's Algorithm
$\mathcal{R}_0$	0	0	0	0	0	0	0
$\mathcal{R}_1$	1	1	1	1	1	1	0.5
$\mathcal{R}_2$	1.5	1.5	1.5	1.5	1.5	1.5	0.75
$\mathcal{R}_3$	2	3	2.5	2.5	2.25	2.25	1.5
$\mathcal{R}_4$	16	37	25.75	27.5	24.69	26.46	31.75

different behaviors of the single link and complete link algorithms. Characteristics of other algorithms, such as the WPGMC and the UPGMC, are discussed next.

13.2.3 Monotonicity and Crossover

Let us consider the following dissimilarity matrix:

$$P = \begin{bmatrix} 0 & 1.8 & 2.4 & 2.3 \\ 1.8 & 0 & 2.5 & 2.7 \\ 2.4 & 2.5 & 0 & 1.2 \\ 2.3 & 2.7 & 1.2 & 0 \end{bmatrix}$$

Application of the single and complete link algorithms to P gives rise to the dissimilarity dendrograms depicted in Figure 13.4a and 13.4b, respectively. Application of the UPGMC and WPGMC algorithms to P results in the same dissimilarity dendrogram, which is shown in Figure 13.4c. In this dendrogram something interesting occurs. The cluster $\{x_1, x_2, x_3, x_4\}$ is formed at a lower dissimilarity level than cluster $\{x_1, x_2\}$. This phenomenon is called *crossover*. More specifically, crossover occurs when a cluster is formed at a lower dissimilarity level than any of its components. The opposite of the crossover is *monotonicity*. Satisfaction of the latter condition implies that each cluster is formed at a higher dissimilarity level than any one of its components. More formally, the monotonicity condition may be stated as follows:

"If clusters C_i and C_j are selected to be merged in cluster C_q, at the tth level of the hierarchy, then the following condition must hold:

$$d(C_q, C_k) \geq d(C_i, C_j)$$

for all C_k, $k \neq i, j, q$."

Monotonicity is a property that is exclusively related to the clustering algorithm and not to the (initial) proximity matrix. Recall Eq. (13.3) defined in terms of the parameters a_i, a_j, b, and c.

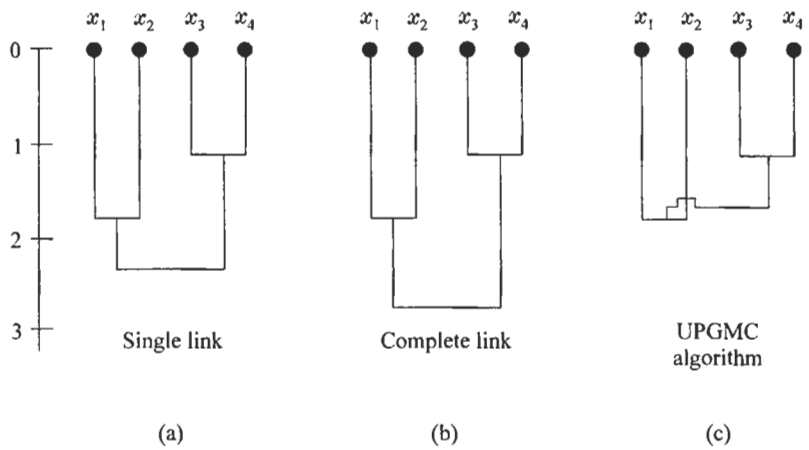


FIGURE 13.4: Dissimilarity dendrograms derived from (a) single link, (b) complete link, and (c) UPGMC and WPGMC when they apply to P. The third dendrogram exhibits the crossover phenomenon.

In the sequel, a proposition is stated and proved that allows us to decide whether an algorithm satisfies the monotonicity condition.

Proposition 1. If a_i and a_j are nonnegative, $a_i + a_j + b \geq 1$, and either (a) $c \geq 0$ or (b) $\max\{-a_i, -a_j\} \leq c \leq 0$, then the corresponding clustering method satisfies the monotonicity condition.

Proof. (a) From the hypothesis we have that

$$b \geq 1 - a_i - a_j$$

Substituting this result in Eq. (13.3) and after some rearrangements, we obtain

$$\begin{aligned} d(C_q, C_s) \geq\ & d(C_i, C_j) + a_i(d(C_i, C_s) - d(C_i, C_j)) \\ & + a_j(d(C_j, C_s) - d(C_i, C_j)) + c|d(C_i, C_s) - d(C_j, C_s)| \end{aligned}$$

Since, from step 2.2 of the MUAS in Section 13.2.2,

$$d(C_i, C_j) = \min_{r,u} d(C_r, C_u)$$

the second and third terms of the last inequality are nonnegative. Moreover, the fourth term of the same inequality is also nonnegative. Therefore, we obtain

$$d(C_q, C_s) \geq d(C_i, C_j)$$

Thus, the monotonicity condition is satisfied.

(b) Let $d(C_i, C_s) \geq d(C_j, C_s)$ (the case where $d(C_i, C_s) < d(C_j, C_s)$ may be treated similarly). As in the previous case,

$$b \geq 1 - a_i - a_j$$

Taking into account this inequality, Eq. (13.3) gives

$$\begin{aligned} d(C_q, C_s) \geq\ & d(C_i, C_j) + a_i(d(C_i, C_s) - d(C_i, C_j)) \\ & + a_j(d(C_j, C_s) - d(C_i, C_j)) + c(d(C_i, C_s) - d(C_j, C_s)) \end{aligned}$$

By adding and subtracting on the right-hand side of this inequality the term $cd(C_i, C_j)$ and after some manipulations, we obtain

$$\begin{aligned} d(C_q, C_s) \geq\ & (a_j - c)(d(C_j, C_s) - d(C_i, C_j)) + d(C_i, C_j) \\ & + (a_i + c)(d(C_i, C_s) - d(C_i, C_j)) \end{aligned}$$

Since, from the hypothesis, $a_j - c \geq 0$ and

$$d(C_i, C_j) = \min_{r,u} d(C_r, C_u)$$

from step 2.2 of the MUAS we obtain that

$$d(C_q, C_s) \geq d(C_i, C_j) \quad \blacksquare$$

Note that the conditions of Proposition 1 are sufficient but not necessary. This means that algorithms that do not satisfy the premises of this proposition may still satisfy the monotonicity condition. It is easy to note that the single link, the complete link, the UPGMA, the WPGMA, and Ward's algorithm satisfy the premises of Proposition 1. Thus, all these algorithms satisfy the monotonicity condition. The other two algorithms, the UPGMC and the WPGMC, do not satisfy the monotonicity condition. Moreover, we can construct examples that demonstrate that these two algorithms violate the monotonicity property, as follows from Figure 13.4c. However, it is important to note that this does not mean that they always lead to dendrograms with crossovers.

Finally, we note that there have been several criticisms concerning the usefulness of algorithms that do not satisfy the monotonicity condition (e.g., [Will 77, Snea 73]). However, these algorithms may give satisfactory results in the frame of certain applications. Moreover, there is no theoretical guideline suggesting that the algorithms satisfying the monotonicity condition always lead to acceptable results. After all, this ought to be the ultimate criterion for the usefulness of an algorithm (unfortunately, such a criterion does not exist in general).

13.2.4 Implementational Issues

As stated earlier, the computational time of GAS is $O(N^3)$. However, many efficient implementations of these schemes have been proposed in the literature, that reduce the computational time by an order of N. For example, in [Kuri 91] an implementation is discussed, for which the required computational time is reduced to $O(N^2 \log N)$. Also, in [Murt 83, Murt 84, Murt 85], implementations for widely used agglomerative algorithms are discussed that require $O(N^2)$ computational time and either $O(N^2)$ or $O(N)$ storage. Finally, parallel implementations on Single Instruction Multiple Data (SIMD) machines are discussed in [Will 89] and [Li 90].

13.2.5 Agglomerative Algorithms Based on Graph Theory

Before we describe the algorithms of this family, let us first recall some basic definitions from graph theory.

Basic Definitions from Graph Theory

A *graph* G is defined as an ordered pair $G = (V, E)$, where $V = \{v_i,\ i = 1, \ldots, N\}$ is a set of *vertices* and E is a set of *edges* connecting some pairs of vertices. An edge, connecting the vertices v_i and v_j, will be denoted either by e_{ij} or by (v_i, v_j). When the ordering of v_i and v_j is of no importance, then we deal with *undirected graphs*. Otherwise, we deal with *directed graphs*. In addition, if no cost is associated with the edges of the graph, we deal with *unweighted graphs*. Otherwise, we deal with *weighted graphs*. In the sequel, we consider graphs where a pair of vertices may be connected by at most one edge. In the framework of clustering, we deal with undirected graphs where each vertex corresponds to a feature vector (or, equivalently, to the pattern represented by the feature vector).

A *path* in G, between vertices v_{i_1} and v_{i_n}, is a sequence of vertices and edges of the form $v_{i_1} e_{i_1 i_2} v_{i_2} \ldots v_{i_{n-1}} e_{i_{n-1} i_n} v_{i_n}$ (Figure 13.5a). Of course, it is not guaranteed that there always exists a path from v_{i_1} to v_{i_n}. If in this path, v_{i_1} coincides with v_{i_n}, the path is called a *loop* or *circle* (Figure 13.5b). In the special case in which an edge connects a vertex to itself, we have a *self-loop*.

A *subgraph* $G' = (V', E')$ of G is a graph with $V' \subseteq V$ and $E' \subseteq E_1$, where E_1 is a subset of E, whose edges connect vertices that lie in V'. Clearly, G is a subgraph of itself.

A *subgraph* $G' = (V', E')$ is *connected* if there exists at least one path connecting any pair of vertices in V' (Figure 13.5c). For example, in Figure 13.5c the subgraph with vertices v_1, v_2, v_4, and v_5 is not connected. The subgraph G' is *complete* if every vertex $v_i \in V'$ is connected with every vertex in $V' - \{v_i\}$ (Figure 13.5d).

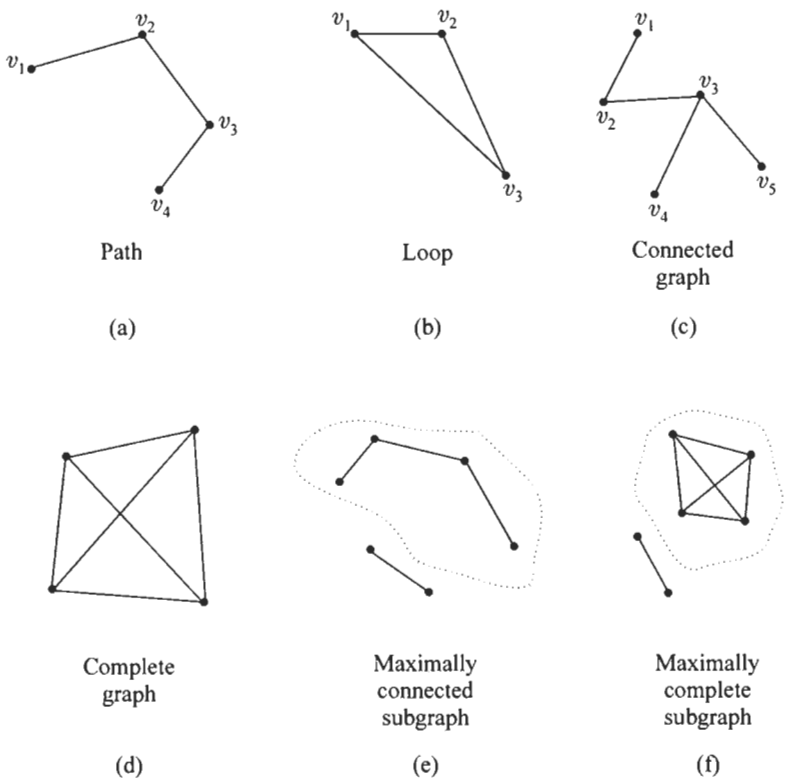


FIGURE 13.5: (a) A path connecting the vertices v_1 and v_4. (b) A loop. (c) A connected graph. (d) A complete graph. (e) Maximally connected subgraph. (f) Maximally complete subgraph.

A *maximally connected subgraph of* G is a connected subgraph G' of G that contains as many vertices of G as possible (Figure 13.5e). A *maximally complete subgraph* is a complete subgraph G' of G that contains as many vertices of G as possible (Figure 13.5f).

A concept that is closely related to the algorithms based on graph theory is that of the *threshold graph*. A threshold graph is an undirected, unweighted graph with N nodes, each corresponding to a vector of the data set X. In this graph there are no self-loops or multiple edges between any two nodes. Let a be a dissimilarity level. A threshold graph $G(a)$ with N nodes contains an edge between two nodes i and j *if the dissimilarity between the corresponding vectors* $\boldsymbol{x}_i$ *and* $\boldsymbol{x}_j$ *is less than or equal to* a, $i, j = 1, \ldots, N$. Alternatively, we may write

$$(v_i, v_j) \in G(a), \quad \text{if } d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq a, \quad i, j = 1, \ldots, N \tag{13.20}$$

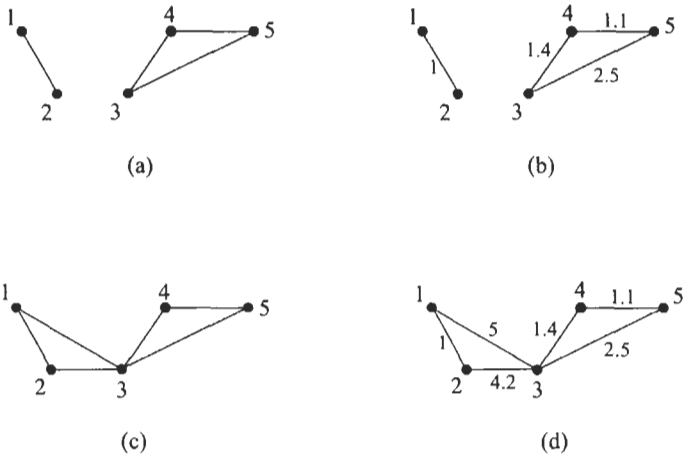


FIGURE 13.6: (a) The threshold graph $G(3)$, (b) the proximity (dissimilarity) graph $G_p(3)$, (c) the threshold graph $G(5)$, (d) the proximity (dissimilarity) graph $G_p(5)$, obtained from the dissimilarity matrix $P(X)$ of Example 13.1.

If similarity measures are used, this definition is altered to

$$(v_i, v_j) \in G(a), \quad \text{if } s(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq a, \quad i, j = 1, \ldots, N$$

A *proximity graph* $G_p(a)$ is a threshold graph $G(a)$, all of whose edges (v_i, v_j) are weighted with the proximity measure between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. If a dissimilarity (similarity) measure is used as proximity between two vectors, then the proximity graph is called a *dissimilarity (similarity) graph*. Figure 13.6 shows the threshold and proximity graphs $G(3)$ and $G_p(3)$, $G(5)$ and $G_p(5)$ obtained from the dissimilarity matrix $P(X)$ given in Example 13.1.

The Algorithms

In this section, we discuss agglomerative algorithms based on graph theory concepts. More specifically, we consider graphs, G, of N nodes with each node corresponding to a vector of X. Clusters are formed by connecting nodes together, and this leads to connected subgraphs. Usually, an additional graph property, $h(k)$, must be satisfied by the subgraphs in order to define valid clusters. In this context, the function g involved in GAS is replaced by $g_{h(k)}$, where $h(k)$ is the graph property. Some typical properties that can be adopted are [Jain 88, Ling 72]

- *Node connectivity.* The node connectivity of a connected subgraph is the largest integer k such that all pairs of nodes are joined by at least k paths having no nodes in common.

- *Edge connectivity.* The edge connectivity of a connected subgraph is the largest integer k such that all pairs of nodes are joined by at least k paths having no edges in common.
- *Node degree.* The degree of a connected subgraph is the largest integer k such that each node has at least k incident edges.

The general agglomerative scheme in the context of graph theory is known as the *graph theory–based algorithmic scheme (GTAS).* The algorithm follows exactly the same iteration steps as the generalized agglomerative scheme (GAS), with the exception of step 2.2. This is expressed as

$$g_{h(k)}(C_i, C_j) = \begin{cases} \min_{r,s} g_{h(k)}(C_r, C_s), & \text{for dissimilarity functions} \\ \max_{r,s} g_{h(k)}(C_r, C_s), & \text{for similarity functions} \end{cases} \tag{13.21}$$

The proximity function $g_{h(k)}(C_r, C_s)$ between two clusters is defined in terms of (a) a proximity measure between vectors (i.e., nodes in the graph) *and* (b) certain constraints imposed by the property $h(k)$ on the subgraphs that are formed. In a more formal way, $g_{h(k)}$ is defined as

$$\begin{aligned} &g_{h(k)}(C_r, C_s) \\ &\quad = \min_{x_u \in C_r, x_v \in C_s} \{d(x_u, x_v) \equiv a : \text{ the } G(a) \text{ subgraph} \\ &\qquad\qquad \text{defined by } C_r \cup C_s \text{ is (a) connected and either} \\ &\qquad\qquad \text{(b1) has the property } h(k) \text{ or (b2) is complete}\}^5 \end{aligned} \tag{13.22}$$

In words, clusters (i.e., connected subgraphs) are merged (a) based on the proximity measure between their nodes and (b) provided that their merging leads to a connected subgraph that either has property $h(k)$ or is complete. Let us now consider a few examples.

Single link algorithm: Here connectedness is the only prerequisite. That is, no property $h(k)$ is imposed and no completeness is required. Thus (b1) and (b2) in (13.22) are ignored and (13.22) is simplified to

$$\begin{aligned} g_{h(k)}(C_r, C_s) = \min_{x_u \in C_r, x_v \in C_s} &\{d(x_u, x_v) \equiv a: \text{ the } G(a) \\ &\text{subgraph defined by } C_r \cup C_s \text{ is connected}\} \end{aligned} \tag{13.23}$$

Let us demonstrate the algorithm via an example.

[5] This means that *all* nodes of $C_r \cup C_s$ participate in the required properties.

Example 13.4. Consider the following dissimilarity matrix:

$$P = \begin{bmatrix} 0 & 1.2 & 3 & 3.7 & 4.2 \\ 1.2 & 0 & 2.5 & 3.2 & 3.9 \\ 3 & 2.5 & 0 & 1.8 & 2.0 \\ 3.7 & 3.2 & 1.8 & 0 & 1.5 \\ 4.2 & 3.9 & 2.0 & 1.5 & 0 \end{bmatrix}$$

The first clustering, $\mathfrak{R}_0$, is the one where each vector of X forms a single cluster (see Figure 13.7). In order to determine the next clustering, $\mathfrak{R}_1$, via the single link algorithm, we need

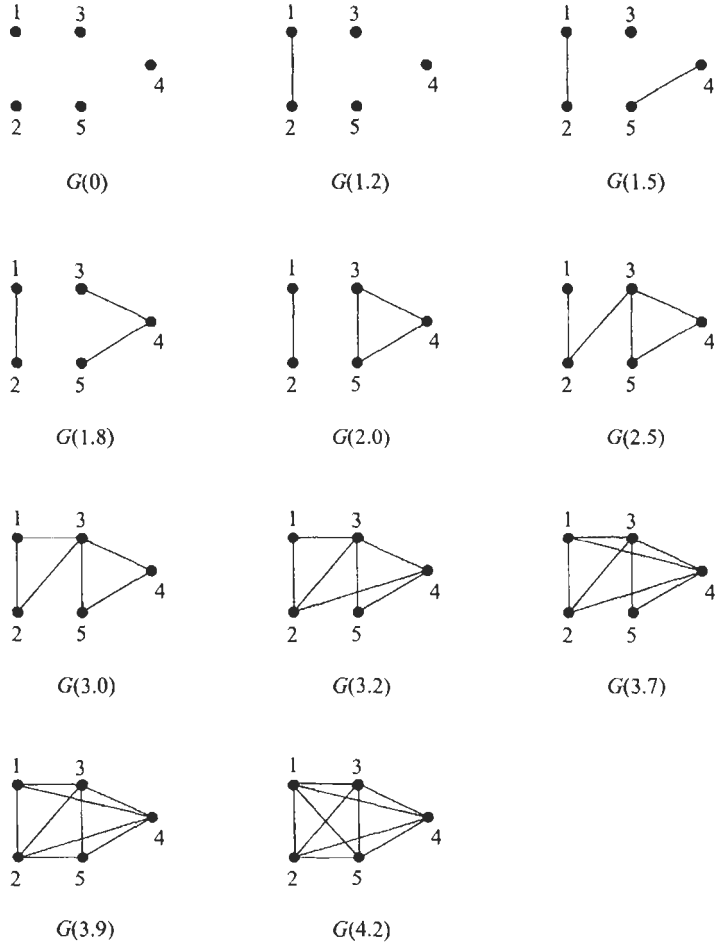


FIGURE 13.7: Threshold graphs derived by the dissimilarity matrix P given in Example 13.4.

to compute $g_{h(k)}(C_r, C_s)$ for all pairs of the existing clusters. For $\{x_1\}$ and $\{x_2\}$, the value of $g_{h(k)}$ is equal to 1.2, since $\{x_1\} \cup \{x_2\}$ become connected for a first time in $G(1.2)$. Likewise, $g_{h(k)}(\{x_1\}, \{x_3\}) = 3$. The rest of the $g_{h(k)}$ values are computed in a similar fashion. Then, using Eq. (13.21), we find that $g_{h(k)}(\{x_1\}, \{x_2\}) = 1.2$ is the minimum value of $g_{h(k)}$ and thus $\{x_1\}$ and $\{x_2\}$ are merged in order to produce

$$\mathfrak{R}_1 = \{\{x_1, x_2\}, \{x_3\}, \{x_4\}, \{x_5\}\}$$

Following the same procedure, we find that the minimum $g_{h(k)}$ among all pairs of clusters is $g_{h(k)}(\{x_4\}, \{x_5\}) = 1.5$. Thus, $\mathfrak{R}_2$ is given by

$$\mathfrak{R}_2 = \{\{x_1, x_2\}, \{x_3\}, \{x_4, x_5\}\}$$

For the formation of $\mathfrak{R}_3$ we first consider the clusters $\{x_3\}$ and $\{x_4, x_5\}$. In this case $g_{h(k)}(\{x_3\}, \{x_4, x_5\}) = 1.8$, since $\{x_3\} \cup \{x_4, x_5\}$ becomes connected at $G(1.8)$ for a first time. Similarly, we find that $g_{h(k)}(\{x_1, x_2\}, \{x_3\}) = 2.5$, and $g_{h(k)}(\{x_1, x_2\}, \{x_4, x_5\}) = 3.2$. Thus,

$$\mathfrak{R}_3 = \{\{x_1, x_2\}, \{x_3, x_4, x_5\}\}$$

Finally, we find that $g_{h(k)}(\{x_1, x_2\}, \{x_3, x_4, x_5\}) = 2.5$ and $\mathfrak{R}_4$ is formed at this level. Observe that at $G(2.0)$ no clustering is formed.

Remark

- In the single link algorithm no property $h(k)$ is required, and Eq. (13.23) is basically the same as

$$g_{h(k)}(C_r, C_s) = \min_{x \in C_r, y \in C_s} d(x, y) \tag{13.24}$$

  Hence the algorithm is equivalent to its single link counterpart based on matrix theory and both produce the same results (see Problem 13.7).

Complete link algorithm. The only prerequisite here is that of completeness; that is, the graph property $h(k)$ is omitted. Since, connectedness is a weaker condition than completeness, subgraphs form valid clusters only if they are complete. Let us demonstrate the algorithms via the case of Example 13.4.

Clusterings $\mathfrak{R}_0$, $\mathfrak{R}_1$, and $\mathfrak{R}_2$ are the same as those produced by the single link algorithm and are formed by $G(0)$, $G(1.2)$, and $G(1.5)$, respectively. Let us derive the $\mathfrak{R}_3$ clustering. It is $g_{h(k)}(\{x_3\}, \{x_4, x_5\}) = 2$, because at $G(2.0)$, $\{x_3\} \cup \{x_4, x_5\}$ becomes complete for the first time. Similarly, $g_{h(k)}(\{x_1, x_2\}, \{x_3\}) = 3$ and $g_{h(k)}(\{x_1, x_2\}, \{x_4, x_5\}) = 4.2$. Thus, the resulting $\mathfrak{R}_3$ clustering is the same as the one obtained by the single link algorithm. The only difference is that it is formed at graph $G(2.0)$ instead of $G(1.8)$, which was the case with the single link algorithm. Finally, the last clustering, $\mathfrak{R}_4$ is defined at $G(4.2)$.

Remark

- A little thought suffices to see that Eq. (13.22) for the complete link algorithm is equivalent to

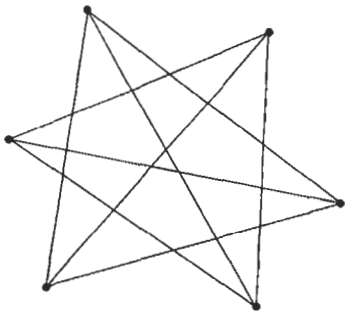
$$g_{h(k)}(C_r, C_s) = \max_{x \in C_r, y \in C_s} d(x, y) \tag{13.25}$$

  and, thus, this algorithm is equivalent to its matrix-based counterpart (see Problem 13.8)

The single and the complete link algorithms may be seen as the extreme cases of the GTAS scheme. This is because the criteria adopted for the formation of a new cluster are the weakest possible for the single link and the strongest possible for the complete link algorithm. A variety of algorithms between these two extremes may be obtained if we make different choices for $g_{h(k)}$. From Eq. (13.22) it is clear that this may be achieved by changing the property $h(k)$, where k is a parameter whose meaning depends on the adopted property $h(k)$. For example, in Figure 13.8, the value of k for the properties of node connectivity, edge connectivity, and node degree is 3.

Example 13.5. In this example we demonstrate the operation of the property $h(k)$. Let us consider the following dissimilarity matrix:

$$P(X) = \begin{bmatrix} 0 & 1 & 9 & 18 & 19 & 20 & 21 \\ 1 & 0 & 8 & 13 & 14 & 15 & 16 \\ 9 & 8 & 0 & 17 & 10 & 11 & 12 \\ 18 & 13 & 17 & 0 & 5 & 6 & 7 \\ 19 & 14 & 10 & 5 & 0 & 3 & 4 \\ 20 & 15 & 11 & 6 & 3 & 0 & 2 \\ 21 & 16 & 12 & 7 & 4 & 2 & 0 \end{bmatrix}$$



Node connectivity : 3

Edge connectivity : 3

Node degree : 3

FIGURE 13.8: A graph with node connectivity, edge connectivity, and node degree equal to 3.

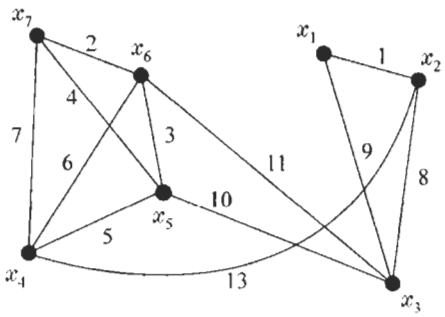


FIGURE 13.9: The proximity graph $G(13)$ derived by the dissimilarity matrix P given in Example 13.5.

Figure 13.9 shows the $G(13)$ proximity graph produced by this dissimilarity matrix. Let $h(k)$ be the node degree property with $k = 2$; that is, it is required that each node has at least two incident edges. Then the obtained threshold dendrogram is shown in Figure 13.10a. At dissimilarity level 1, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ form a single cluster. This happens because $\{\boldsymbol{x}_1\} \cup \{\boldsymbol{x}_2\}$ is complete at $G(1)$, despite the fact that property $h(2)$ is not satisfied (remember the disjunction between conditions (b1) and (b2) in Eq. (13.22)). Similarly, $\{\boldsymbol{x}_6\} \cup \{\boldsymbol{x}_7\}$ forms a cluster at dissimilarity level 2. The next clustering is formed at level 4, since $\{\boldsymbol{x}_5\} \cup \{\boldsymbol{x}_6, \boldsymbol{x}_7\}$ becomes complete in $G(4)$. At level 6, $\boldsymbol{x}_4$, $\boldsymbol{x}_5$, $\boldsymbol{x}_6$, and $\boldsymbol{x}_7$ lie for the first time in the same cluster. Although this subgraph is not complete, it does satisfy $h(2)$. Finally, at level 9, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_3$ come into the same cluster. Note that, although all nodes in the graph have node degree equal to 2, the final clustering will be formed at level 10 because at level 9 the graph is not connected.

Assume now that $h(k)$ is the node connectivity property, with $k = 2$; that is, all pairs of nodes in a connected subgraph are joined by at least two paths having no nodes in common. The dissimilarity dendrogram produced in this case is shown in Figure 13.10b.

Finally, the dissimilarity dendrogram produced when the edge connectivity property with $k = 2$ is employed is shown in Figure 13.10c.

It is not difficult to see that all these properties for $k = 1$ result in the single link algorithm. On the other hand, as k increases, the resulting subgraphs approach completeness.

Example 13.6. Consider again the dissimilarity matrix of the previous example. Assume now that $h(k)$ is the node degree property with $k = 3$. The corresponding dendrogram is shown in Figure 13.11. Comparing the dendrograms of Figures 13.10a and 13.11, we observe that the same clusters in the second case are formed in larger dissimilarity levels.

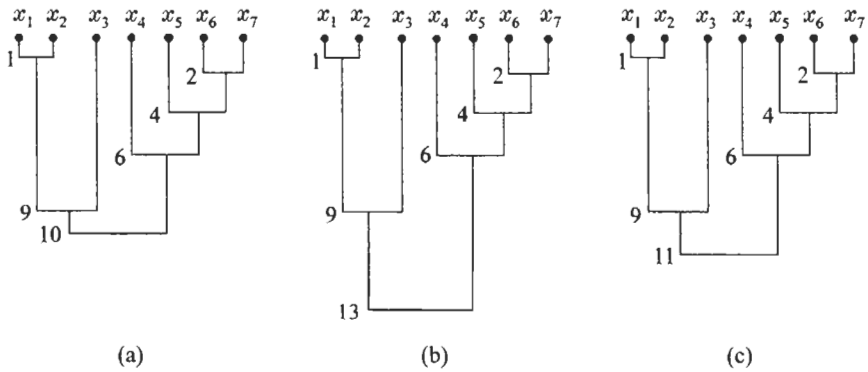


FIGURE 13.10: Dissimilarity dendrograms related to Example 13.5. (a) Dissimilarity dendrogram produced when $h(k)$ is the node degree property, with $k = 2$. (b) Dissimilarity dendrogram produced when $h(k)$ is the node connectivity property, with $k = 2$. (c) Dissimilarity dendrogram produced when $h(k)$ is the edge connectivity property, with $k = 2$.

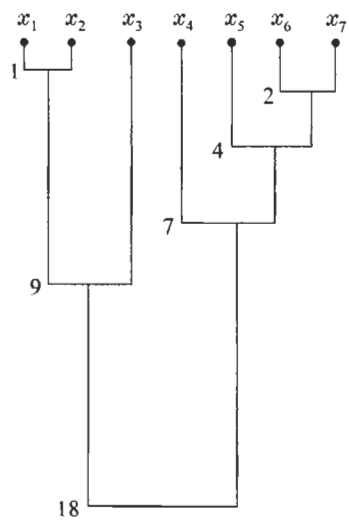


FIGURE 13.11: Threshold dendrogram related to Example 13.6 for the node degree property and $k = 3$.

Clustering Algorithms Based on the Minimum Spanning Tree

A *spanning tree* is a connected graph (containing all the vertices of the graph) and having no loops (i.e., there exists only one path connecting any two pairs of nodes in the graph.). If the edges of the graph are weighted, we define as the *weight of the spanning tree* the sum of the weights of its edges. A *minimum spanning tree (MST)* is a spanning tree with the smallest weight among all spanning trees connecting the nodes of the graph. An MST of a graph may be derived with Prim's algorithm or Kruskal's algorithm (e.g., see [Horo 78]). Note that there may be more than one minimum spanning trees for a given graph. However, when the weights of the edges of G are different from each other, then the MST is unique. In our case, the weights of a graph are derived from the proximity matrix $P(X)$.

Searching for the MST can also be seen as a special case of the GTAS scheme, if we adopt in place of $g_{h(k)}(C_r, C_s)$ the following proximity function:

$$g(C_r, C_s) = \min_{ij}\{w_{ij} : \boldsymbol{x}_i \in C_r,\ \boldsymbol{x}_j \in C_s\} \tag{13.26}$$

where $w_{ij} = d(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

In words, this measure identifies the minimum weight of the MST that connects the subgraphs corresponding to C_r and C_s.

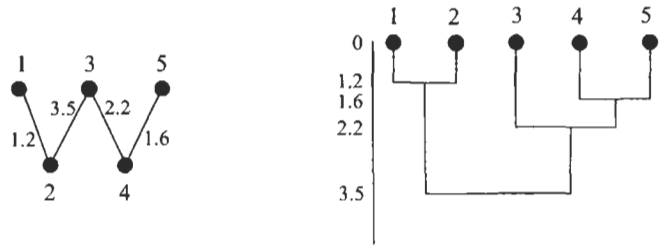
Once the MST has been determined (using any suitable algorithm), we may identify a hierarchy of clusterings as follows: the clusters of the clustering at level t are identified as the *connected* components of G if only the edges of its MST with the smallest t weights are considered. It takes a little thought to see that this hierarchy is identical to the one defined by the single link algorithm, at least for the case in which all the distances between any two vectors of X are different from each other. Thus, this scheme may also be viewed as an alternative implementation of the single link algorithm. The following example demonstrates the operation of this scheme.

Example 13.7. Let us consider the following proximity matrix.

$$P = \begin{bmatrix} 0 & 1.2 & 4.0 & 4.6 & 5.1 \\ 1.2 & 0 & 3.5 & 4.2 & 4.7 \\ 4.0 & 3.5 & 0 & 2.2 & 2.8 \\ 4.6 & 4.2 & 2.2 & 0 & 1.6 \\ 5.1 & 4.7 & 2.8 & 1.6 & 0 \end{bmatrix}$$

The MST derived from this proximity matrix is given in Figure 13.12a. The corresponding dendrogram is given in Figure 13.12(b).

It is easy to observe that a minimum spanning tree uniquely specifies the dendrogram of the single link algorithm. Thus, MST can be used as an alternative to the single link algorithm and can lead to computational savings.



(a) (b)

FIGURE 13.12: (a) The minimum spanning tree derived with the dissimilarity matrix given in Example 13.7. (b) The dissimilarity dendrogram obtained with the algorithm based on the MST.

13.2.6 Ties in the Proximity Matrix

In cases in which the vectors consist of interval-scaled or ratio-scaled features, the probability of a vector of the data set X being equidistant from two other vectors of X is very small for most practical problems. However, if we deal with ordinal data, this probability is not negligible. The fact that a vector is equidistant from two other vectors implies that a proximity matrix P will have at least two equal entries in the triangle above its main diagonal (see Example 13.8). It is interesting to see how the hierarchical algorithms behave with such proximity matrices. Let us consider first the family of algorithms based on the graph theory via the following example.

Example 13.8. Consider the following dissimilarity matrix:

$$P = \begin{bmatrix} 0 & 4 & 9 & 6 & 5 \\ 4 & 0 & 3 & 8 & 7 \\ 9 & 3 & 0 & 3 & 2 \\ 6 & 8 & 3 & 0 & 1 \\ 5 & 7 & 2 & 1 & 0 \end{bmatrix}$$

Note that $P(2, 3) = P(3, 4)$. The corresponding dissimilarity graph $G(9)$ is shown in Figure 13.13a. Figure 13.13b shows the corresponding dissimilarity dendrogram obtained by the single link algorithm. No matter which of the two edges is considered first, the resulting dendrogram remains the same. Figure 13.13c (13.13d) depicts the dendrogram obtained by the complete link algorithm when the $(3, 4)$ $((2, 3))$ edge is considered first. Note that the dendrograms of Figure 13.13c and 13.13d are different.

Let us interchange the $P(1, 2)$ and $P(1, 3)$ entries of P[5] and let P_1 be the new dissimilarity matrix. Figure 13.14a shows the dendrogram obtained by the single link algorithm

[5]Since a dissimilarity matrix is symmetric, $P(2, 1)$ and $P(3, 1)$ entries are also interchanged.

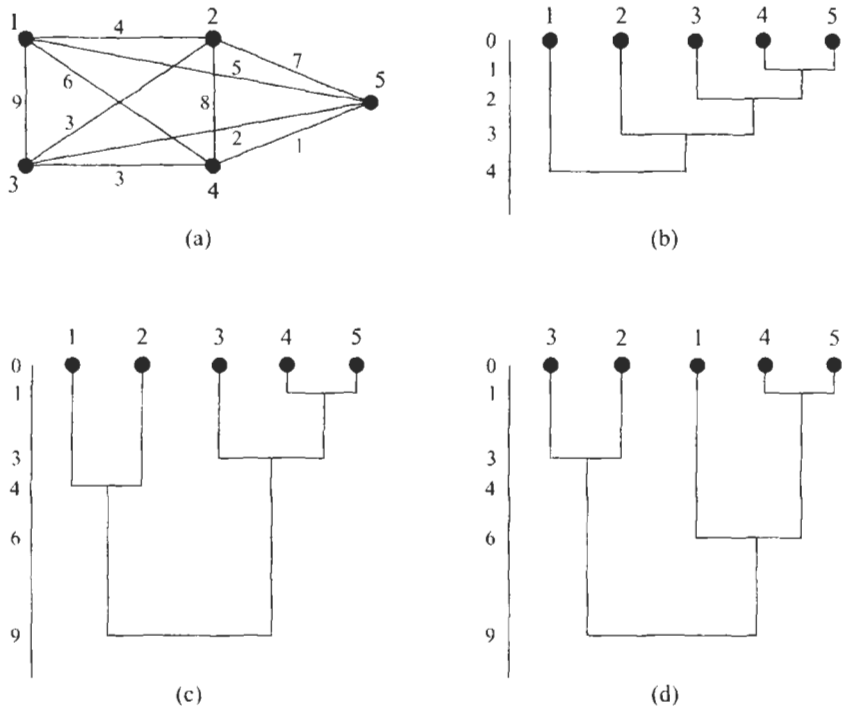


FIGURE 13.13: (a) The dissimilarity graph ($G(9)$) for the dissimilarity matrix given in Example 13.8. (b) The dissimilarity dendrogram obtained by the single link algorithm. (c) The dissimilarity dendrogram obtained by the complete link algorithm when edge (3, 4) is considered first. (d) The dissimilarity dendrogram obtained by the complete link algorithm when edge (2, 3) is considered first.

and Figure 13.14b depicts the dendrogram obtained by the complete link algorithm. In this case, the complete link algorithm produces the same dendrogram regardless the order in which edges (1, 2) and (3, 4) are considered.

This example indicates that the single link algorithm leads to the same dendrogram, regardless of how the ties are considered. On the other hand, the complete link algorithm *may* lead to different dendrograms if it follows different ways of considering the ties. The other graph theory–based algorithms, which fall between the single and the complete algorithm, exhibit behavior similar to that of the complete link algorithm (see Problem 13.11).

The same trend is true for the matrix-based algorithms. *Note, however, that in matrix-based schemes ties may appear at a later stage in the proximity matrix*

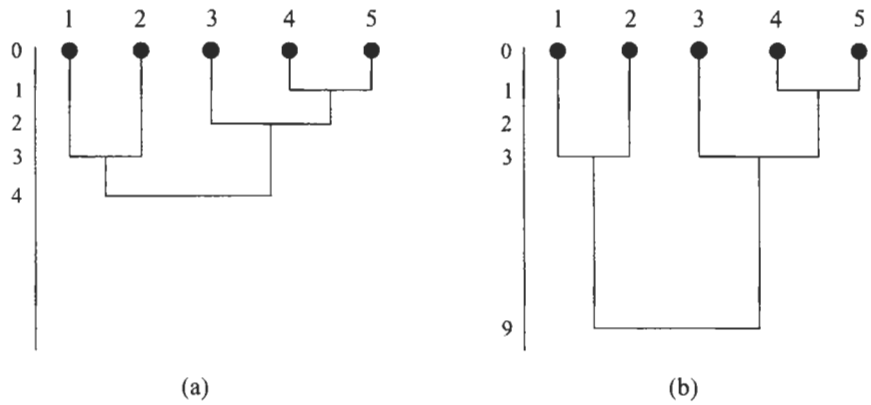


FIGURE 13.14: (a) The dissimilarity dendrogram obtained by the single link algorithm for P_1 given in Example 13.8. (b) The dissimilarity dendrogram obtained by the complete link algorithm for P_1 given in Example 13.8.

(see Problem 13.12). Thus, as is showed in [Jard 71], the single link algorithm treats the ties in the proximity matrix in the most reliable way; it always leads to the same proximity dendrogram. It seems that every requirement additional to the connectivity property (for graph theory based algorithms) or to Eq. (13.4) (for matrix theory based algorithms) produces ambiguity and the results become sensitive to the order in which ties are processed. From this point of view, the single link algorithm seems to outperform its competitors. This does not mean that all the other algorithms are inferior. The algorithm that gives the best results is problem dependent. However, if one decides to use any other algorithm, different from the single link, he or she must treat the possible ties in the proximity matrix very carefully.

13.3 THE COPHENETIC MATRIX

Another quantity associated with the hierarchical algorithms is the *cophenetic matrix*. This will be used as a tool for the validation of clustering hierarchies in Chapter 16.

Let $\mathfrak{R}_{t_{ij}}$ be the clustering at which $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are merged in the same cluster *for the first time* (of course, they will remain in the same cluster for all subsequent clusterings). Also let $L(t_{ij})$ be the proximity level at which clustering $\mathfrak{R}_{t_{ij}}$ is defined. We define the *cophenetic distance* between two vectors as

$$d_C(\boldsymbol{x}_i, \boldsymbol{x}_j) = L(t_{ij})$$

In words, the cophenetic distance between two vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is defined as the proximity level at which the two vectors are found in the same cluster for the first time.

The cophenetic distance is a metric, *under the assumption of monotonicity*. To prove this, we need to show that the five conditions stated in Chapter 11 are satisfied. Indeed, the first condition holds; that is, $0 \leq d_C(\boldsymbol{x}, \boldsymbol{y}) < +\infty$. Note that the minimum value of d_C is zero, since $L(0) = 0$. Also, $d_C(\boldsymbol{x}, \boldsymbol{x}) = 0$, since a vector $\boldsymbol{x}$ lies in the same cluster with itself at the zero level clustering, where dissimilarity is 0. Furthermore, it is obvious that $d_C(\boldsymbol{x}, \boldsymbol{y}) = d_C(\boldsymbol{y}, \boldsymbol{x})$. Also, it is easy to see that $d_C(\boldsymbol{x}, \boldsymbol{y}) = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{y}$, since at $L(0)$ each cluster consists of a single vector. Finally, the triangular inequality is also valid. Indeed, for a triple $(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_r)$, let $t_1 = \max\{t_{ij}, t_{jr}\}$ and $L_1 = \max\{L(t_{ij}), L(t_{jr})\}$. It is clear, from the property of hierarchy, that in the clustering formed at the t_1 level, $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, and $\boldsymbol{x}_r$ fall into the same cluster. Furthermore, *assuming monotonicity*, $d_C(\boldsymbol{x}_i, \boldsymbol{x}_r) \leq L_1$, or

$$d_C(\boldsymbol{x}_i, \boldsymbol{x}_r) \leq \max\{d_C(\boldsymbol{x}_i, \boldsymbol{x}_j), d_C(\boldsymbol{x}_j, \boldsymbol{x}_r)\} \tag{13.27}$$

Note that two of these three distances are always equal, depending on which pair of vectors came first under a single cluster. This condition is stronger than the triangular inequality and is called *ultrametric inequality*. Notice the close relation of ultrametricity and monotonicity. Monotonicity ensures ultrametricity and, as a consequence, the triangular inequality.

The *cophenetic matrix* is defined as

$$D_C(X) = [d_C(\boldsymbol{x}_i, \boldsymbol{x}_j)] = [L(t_{ij})], \quad i, j = 1, \ldots, N$$

It is clear that the cophenetic matrix is symmetric. Moreover, apart from its diagonal elements, it has only $N - 1$ distinct entries, i.e., it has many ties (duplicate entries). $D_C(X)$ is a special case of dissimilarity matrix, since $d_C(\boldsymbol{x}_i, \boldsymbol{x}_j)$ satisfy the ultrametric inequality.

A hierarchical algorithm can, thus, be viewed as a mapping of the data proximity matrix into a cophenetic matrix.

Example 13.9. Let us consider the dissimilarity dendrogram of Figure 13.2b. The corresponding cophenetic matrix is

$$D_C(X) = \begin{bmatrix} 0 & 1 & 4.2 & 4.2 & 4.2 \\ 1 & 0 & 4.2 & 4.2 & 4.2 \\ 4.2 & 4.2 & 0 & 1.4 & 1.4 \\ 4.2 & 4.2 & 1.4 & 0 & 1.1 \\ 4.2 & 4.2 & 1.4 & 1.1 & 0 \end{bmatrix}$$

13.4 DIVISIVE ALGORITHMS

The divisive algorithms follow the reverse strategy from that of the agglomerative schemes. The first clustering contains a single set, X. At the first step, we search for the best possible partition of X into two clusters. The straightforward method is to consider all possible $2^{N-1} - 1$ partitions of X into two sets and to select the optimum, according to a prespecified criterion. This procedure is then applied iteratively to each of the two sets produced in the previous stage. The final clustering consists of N clusters, each containing a single vector of X.

Let us state the general divisive scheme more formally. Here, the tth clustering contains $t + 1$ clusters. In the sequel, C_{tj} will denote the jth cluster of the tth clustering $\mathfrak{R}_t$, $t = 0, \ldots, N-1$, $j = 1, \ldots, t+1$. Let $g(C_i, C_j)$ be a dissimilarity function defined for all possible pairs of clusters. The initial clustering $\mathfrak{R}_0$ contains only the set X, that is, $C_{01} = X$. To determine the next clustering, we consider all possible pairs of clusters that form a partition of X. Among them we choose the pair, denoted by (C_{11}, C_{12}), that maximizes g.[6] These clusters form the next clustering $\mathfrak{R}_1$, that is, $\mathfrak{R}_1 = \{C_{11}, C_{12}\}$. At the next time step we consider all possible pairs of clusters produced by C_{11} and we choose the one that maximizes g. The same procedure is repeated for C_{12}. Assume now that from the two resulting pairs of clusters, the one originating from C_{11} gives the larger value of g. Let this pair be denoted by (C_{11}^1, C_{11}^2). Then the new clustering, $\mathfrak{R}_2$, consists of C_{11}^1, C_{11}^2, and C_{12}. Relabelling these clusters as C_{21}, C_{22}, C_{23}, respectively, we have $\mathfrak{R}_2 = \{C_{21}, C_{22}, C_{23}\}$. Carrying on in the same way, we form all subsequent clusterings. The general divisive scheme may be stated as follows:

Generalized Divisive Scheme (GDS)

- 1. Initialization
  - — 1.1. Choose $\mathfrak{R}_0 = \{X\}$ as the initial clustering.
  - — 1.2. $t = 0$
- 2. Repeat
  - — 2.1. $t = t + 1$
  - — 2.2. For $i = 1$ to t
    - * 2.2.1 Among all possible pairs of clusters (C_r, C_s) that form a partition of $C_{t-1,i}$, find the pair $(C_{t-1,i}^1, C_{t-1,i}^2)$ that gives the maximum value for g.
  - — Next i

[6] We can also use a similarity function. In that case we should choose the pair of clusters that minimizes g.

— 2.3. From the t pairs defined in the previous step choose the one that maximizes g. Suppose that this is $(C_{t-1,j}^1, C_{t-1,j}^2)$.

— 2.4. The new clustering is

$$\mathcal{R}_t = (\mathcal{R}_{t-1} - \{C_{t-1,j}\}) \cup \left\{C_{t-1,j}^1, C_{t-1,j}^2\right\}$$

— 2.5. Relabel the clusters of $\mathcal{R}_t$.

- Until each vector lies in a single distinct cluster.

Different choices of g give rise to different algorithms. One can easily observe that this divisive scheme is computationally very demanding, even for moderate values of N. This is its main drawback, compared with the agglomerative scheme. Thus, if these schemes are to be of any use in practice, some further computational simplifications are required. One possibility is to make compromises and not search for all possible partitions of a cluster. This can be done by ruling out many partitions as not "reasonable," under a prespecified criterion. Examples of such algorithms are discussed in [Gowd 95] and [MacN 64]. The latter scheme is discussed next.

Let C_i be an already formed cluster. Our goal is to split it further, so that the two resulting clusters, C_i^1 and C_i^2, are as "dissimilar" as possible. Initially, we have $C_i^1 = \emptyset$ and $C_i^2 = C_i$. Then, we identify the vector in C_i^2 whose average dissimilarity from the remaining vectors is maximum and we move it to C_i^1. In the sequel, for each of the remaining $\boldsymbol{x} \in C_i^2$, we compute its average dissimilarity with the vectors of C_i^1, $g(\boldsymbol{x}, C_i^1)$, as well as its average dissimilarity with the rest of the vectors in C_i^2, $g(\boldsymbol{x}, C_{i2} - \{\boldsymbol{x}\})$. If for *every* $\boldsymbol{x} \in C_i^2$, $g(\boldsymbol{x}, C_i^2 - \{\boldsymbol{x}\}) < g(\boldsymbol{x}, C_i^1)$, then we stop. Otherwise, we select the vector $\boldsymbol{x} \in C_i^2$ for which the difference $D(\boldsymbol{x}) = g(\boldsymbol{x}, C_i^2 - \{\boldsymbol{x}\}) - g(\boldsymbol{x}, C_i^1)$ is maximum (among the vectors of C_i^2, $\boldsymbol{x}$ exhibits the maximum dissimilarity with $C_i^2 - \{\boldsymbol{x}\}$ and the maximum similarity to C_i^1) and we move it to C_i^1. The procedure is repeated until the termination criterion is met. This iterative procedure accounts for step 2.2.1 of the GDS.

In the preceding algorithm the splitting of the clusters is based on all the features (coordinates) of the feature vectors. Algorithms of this kind are also called *polythetic algorithms*. In fact, all the algorithms that have been or will be considered in this book are polythetic. In contrast, there are divisive algorithms that achieve the division of a cluster based on a single feature at each step. These are the so-called *monothetic algorithms*. Such algorithms are discussed in [Ever 81]. For more details see [Lamb 62, Lamb 66, MacN 65].

A large research effort has been devoted to comparing the performance of a number of the various agglomerative algorithms in the context of different applications. The interested reader may consult, for example, [Bake 74, Hube 74, Kuip 75, Dube 76, Mill 80, Mill 83].

13.5 CHOICE OF THE BEST NUMBER OF CLUSTERS

So far, we have focused on various hierarchical algorithms. In the sequel we turn our attention to the important task of determining the best clustering within a given hierarchy. Clearly, this is equivalent to the identification of the number of clusters that best fits the data. An intuitive approach is to search in the proximity dendrogram for clusters that have a large *lifetime*. The lifetime of a cluster is defined as the absolute value of the difference between the proximity level at which it is created and the proximity level at which it is absorbed into a larger cluster. For example, the dendrogram of Figure 13.15a suggests that two major clusters are present and that of Figure 13.15b suggests only one. In [Ever 81], experiments are conducted to assess the behavior of various agglomerative algorithms when (a) a single compact cluster and (b) two compact clusters are formed by the vectors of X.

However, this method involves human subjectivity. Many formal methods that may be used in cooperation with both hierarchical and nonhierarchical algorithms for identifying the best number of clusters for the data at hand have been proposed (e.g., [Cali 74, Duda 73, Hube 76]).

A comparison of many such methods is given in [Mill 85]. In the sequel, we discuss two methods, proposed in [Bobe 93] for the identification of the clustering that best fits the data that are appropriate for agglomerative algorithms. The clustering algorithm does not necessarily produce the whole hierarchy of N clusterings, but it terminates when the clustering that best fits the data has been achieved, according to a criterion.

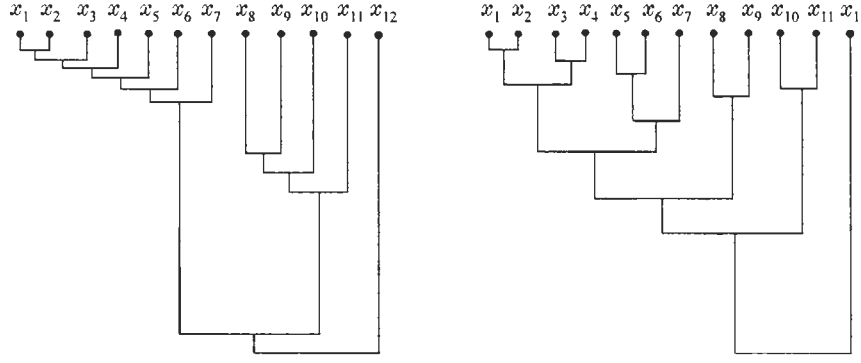
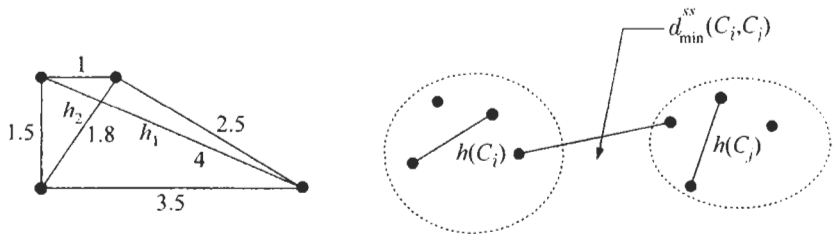


FIGURE 13.15: (a) A dendrogram that suggests that there are two major clusters in the data set. (b) A dendrogram indicating that there is a single major cluster in the data set.



(a) (b)

FIGURE 13.16: (a) Examples of "self-similarity" measures. (b) Illustration of the termination condition for method II.

Method I. This is an extrinsic method, in the sense that it requires the determination of the value of a specific parameter by the user. It involves the definition of a function $h(C)$ that measures the dissimilarity between the vectors of the same cluster C. That is, we can view it as a "self-similarity" measure. For example, $h(C)$ may be defined as

$$h_1(C) = \max\{d(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{x}, \boldsymbol{y} \in C\} \tag{13.28}$$

or

$$h_2(C) = \text{med}\{d(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{x}, \boldsymbol{y} \in C\} \tag{13.29}$$

(see Figure 13.16a).

When d is a metric distance, $h(C)$ may be defined as

$$h_3(C) = \frac{1}{2n_C} \sum_{\boldsymbol{x} \in C} \sum_{\boldsymbol{y} \in C} d(\boldsymbol{x}, \boldsymbol{y}) \tag{13.30}$$

where n_C is the cardinality of C. Other definitions of $h(C)$ are also possible for the last case.

Let θ be an appropriate threshold for the adopted $h(C)$. Then, the algorithm terminates at the $\mathcal{R}_t$ clustering if

$$\exists C_j \in \mathcal{R}_{t+1} : h(C_j) > \theta \tag{13.31}$$

In words, $\mathcal{R}_t$ is the final clustering if there exists a cluster C in $\mathcal{R}_{t+1}$ with dissimilarity between its vectors ($h(C)$) greater than θ.

Sometimes the threshold θ is defined as

$$\theta = \mu + \lambda\sigma \tag{13.32}$$

where μ is the average distance between any two vectors in X and σ is its variance. The parameter λ is a user-defined parameter. Thus, the need for specifying an appropriate value for θ is transferred to the choice of λ. However, λ may be estimated more reasonably than θ.

Method II. This is an intrinsic method; that is, in this case only the structure of the data set X is taken into account. According to this method, the final clustering $\mathcal{R}_t$ *must* satisfy the following relation:

$$d_{min}^{ss}(C_i, C_j) > \max\{h(C_i), h(C_j)\}, \quad \forall C_i, C_j \in \mathcal{R}_t \tag{13.33}$$

where d_{min}^{ss} is defined in Chapter 11. In words, in the final clustering, the dissimilarity between every pair of clusters is larger than the "self-similarity" of each of them (see Figure 13.16b). Note that this is only a necessary condition.

Finally, it must be stated that all these methods are based on heuristic arguments and they are indicative only of the best clustering.

Problems

13.1 Consider the Euclidean distance as the proximity measure between two vectors. Consider only ratio-scaled vectors. Prove that:

(a) A pattern matrix uniquely identifies the corresponding proximity matrix.

(b) A proximity matrix does not identify a pattern matrix uniquely. Furthermore, there are proximity matrices which do not correspond to any pattern matrix.

Hint: (b) Consider, for example, the translations of the points of the data set X.

13.2 Derive Eq. (13.10) from Eq. (13.8).

Hint: Make use of the following identities:

$$n_3\boldsymbol{m}_3 = n_1\boldsymbol{m}_1 + n_2\boldsymbol{m}_2 \tag{13.34}$$

and

$$n_1\|\boldsymbol{m}_1 - \boldsymbol{m}_3\|^2 + n_2\|\boldsymbol{m}_2 - \boldsymbol{m}_3\|^2 = \frac{n_1 n_2}{n_1 + n_2}\|\boldsymbol{m}_1 - \boldsymbol{m}_2\|^2 \tag{13.35}$$

where C_1 and C_2 are any two clusters and $C_3 = C_1 \cup C_2$.

13.3 Show that for the WPGMC algorithm there are cases where $d_{qs} \leq \min(d_{is}, d_{js})$.

13.4 Prove

$$d'_{qs} = \frac{n_q n_s}{n_q + n_s} d_{qs} \tag{13.36}$$

Hint: Multiply both sides of

$$\|\boldsymbol{m}_q - \boldsymbol{m}_s\|^2 = \frac{n_i}{n_i + n_j} d_{is} + \frac{n_j}{n_i + n_j} d_{js} - \frac{n_i n_j}{(n_i + n_j)^2} d_{ij}$$

by $(n_i + n_j)n_s/(n_i + n_j + n_s)$ (This equation holds from Problem 13.2).

13.5 (a) Prove Eq. (13.16).

(b) Complete the proof of Eq. (13.19).

Hint: Take the squares of both sides of Eq. (13.18).

13.6 Consider the proximity matrix given in Example 13.5. Find the proximity dendrograms derived by the GTAS algorithm when $h(k)$ is (a) the node connectivity property and (b) the edge connectivity property, with $k = 3$.

13.7 Prove that the distance between two clusters C_r and C_s, $d(C_r, C_s)$, which are at the same level of the hierarchy produced by the single link algorithm, given by Eq. (13.4), may be written as

$$d(C_r, C_s) = \min_{\boldsymbol{x} \in C_r, \boldsymbol{y} \in C_s} d(\boldsymbol{x}, \boldsymbol{y}). \tag{13.37}$$

That is, the single link algorithms based on matrix and graph theory are equivalent.

Hint: Proceed by induction on the level of hierarchy t. Take into account that the clusterings $\mathcal{R}_t$ and $\mathcal{R}_{t+1}$ have $N - t - 2$ common clusters.

13.8 Prove that the distance between two clusters C_r and C_s, $d(C_r, C_s)$, which are at the same level of the hierarchy produced by the complete link algorithm, given by Eq. (13.5), may be written as

$$d(C_r, C_s) = \max_{\boldsymbol{x} \in C_r, \boldsymbol{y} \in C_s} d(\boldsymbol{x}, \boldsymbol{y}). \tag{13.38}$$

That is, the complete link algorithms based on matrix and graph theory are equivalent.

Hint: Take into account the hints of the previous problem.

13.9 State and prove the propositions of the previous two problems when similarity measures between two vectors are used.

13.10 Consider the following proximity matrix:

$$P = \begin{bmatrix} 0 & 4 & 9 & 6 & 5 \\ 4 & 0 & 1 & 8 & 7 \\ 9 & 1 & 0 & 3 & 2 \\ 6 & 8 & 3 & 0 & 1 \\ 5 & 7 & 2 & 1 & 0 \end{bmatrix}$$

Apply the single and complete link algorithms to P and comment on the resulting dendrograms.

13.11 Consider the dissimilarity matrix P given in Example 13.5. Let us change $P(3, 4)$ to 6 ($P(4, 6)$ is also equal to 6). Also let $h(k)$ be the node degree property with $k = 2$. Run the corresponding graph theory algorithm when (a) the edge (3, 4) is considered first and (b) the edge (4, 6) is considered first. Comment on the resulting dendrograms.

13.12 Consider the following dissimilarity matrix:

$$P = \begin{bmatrix} 0 & 4 & 9 & 6 & 5 \\ 4 & 0 & 3 & 8 & 7 \\ 9 & 3 & 0 & 3 & 2 \\ 6 & 8 & 3 & 0 & 1 \\ 5 & 7 & 2 & 1 & 0 \end{bmatrix}$$

(a) Determine all possible dendrograms resulting from the application of the single and the complete link algorithms to P and comment on the results.

(b) Set $P(3, 4) = 4$, $P(1, 2) = 10$, and let P_1 be the new proximity matrix. Note that P_1 contains no ties. Determine all possible dendrograms resulting from the application of the UPGMA algorithm to P_1.

13.13 Consider the general divisive scheme. Assume that at step 2.2.1 of the algorithm the cluster $C_{t-1,i}^{1}$ consists of a single vector, for $i = 1, \ldots, t$, $t = 1, \ldots, N$. Compute the number of pairs of clusters that have to be examined during the whole clustering process. Discuss the merits and the disadvantages of this scheme.

13.14 Does the alternative divisive algorithm discussed in Section 13.4 guarantee the determination of the optimum possible clustering at each level? Is it reasonable to extend the answer to this question to other such divisive algorithms?

13.15 Explain the physical meaning of Eq. (13.32).

References

[Ande 73] Anderberg M.R. *Cluster Analysis for Applications*, Academic Press, 1973.

[Bake 74] Baker F.B. "Stability of two hierarchical grouping techniques. Case 1. Sensitivity to data errors," *J. Am. Statist. Assoc.*, Vol. 69, pp. 440–445, 1974.

[Bobe 93] Boberg J., Salakoski T. "General formulation and evaluation of agglomerative clustering methods with metric and non-metric distances," *Pattern Recognition*, Vol. 26(9), pp. 1395–1406, 1993.

[Cali 74] Calinski R.B., Harabasz J. "A dendrite method for cluster analysis," *Communications in Statistics*, Vol. 3, pp. 1–27, 1974.

[Day 84] Day W.H.E., Edelsbrunner H. "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, Vol. 1(1), pp. 7–24, 1984.

[Dube 76] Dubes R., Jain A.K. "Clustering techniques: The user's dilemma," *Pattern Recognition*, Vol. 8, pp. 247–260, 1976.

[Dube 87] Dubes R. "How many clusters are best?—An experiment," *Pattern Recognition*, Vol. 20(6), pp. 645–663, 1987.

[Duda 73] Duda R., Hart P. *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[El-G 68] El-Gazzar A., Watson L., Williams W.T., Lance G. "The taxonomy of Salvia: A test of two radically different numerical methods," *J. Linn. Soc. (Bot.)*, Vol. 60, pp. 237–250, 1968.

[Ever 81] Everitt B. *Cluster Analysis*, Halsted Press, 1981.

[Frig 97] Frigui H., Krishnapuram R. "Clustering by competitive agglomeration", *Pattern Recognition*, Vol. 30(7), pp. 1109–1119, 1997.

[Gowd 95] Gowda Chidananda K., Ravi T.V. "Divisive clustering of symbolic objects using the concepts of both similarity and dissimilarity," *Pattern Recognition*, Vol. 28(8), pp. 1277–1282, 1995.

[Gowe 67] Gower J.C. "A comparison of some methods of cluster analysis," *Biometrics*, Vol. 23, pp. 623–628, 1967.

[Hods 71] Hodson F.R. "Numerical typology and prehistoric archaeology," in *Mathematics in Archaeological and Historical Sciences* (Hodson F.R., Kendell D.G., Tautu P.A., eds.), University Press, Edinburgh, 1971.

[Horo 78] Horowitz E., Sahni S. *Fundamentals of Computer Algorithms*, Computer Science Press, 1978.

[Hube 74] Hubert L.J. "Approximate evaluation techniques for the single link and complete link hierarchical clustering procedures," *J. Am. Statist. Assoc.*, Vol. 69, pp. 698–704, 1974.

[Hube 76] Hubert L.J., Levin J.R. "A general statistical framework for assessing categorical clustering in free recall," *Psychological Bulletin*, Vol. 83, pp. 1072–1080, 1976.

[Jain 88] Jain A.K., Dubes R.C. *Algorithms for Clustering Data*, Prentice Hall, 1988.

[Jard 71] Jardine N., Sibson R. *Numerical taxonomy*, John Wiley and Sons, 1971.

[Kank 96] Kankanhalli M.S., Mehtre B.M., Wu J.K. "Cluster-based color matching for image retrieval," *Pattern Recognition*, Vol. 29(4), pp. 701–708, 1996.

[Kuip 75] Kuiper F.K., Fisher L. "A Monte Carlo comparison of six clustering procedures," *Biometrics*, Vol. 31, pp. 777–783, 1975.

[Kuri 91] Kurita T. "An efficient agglomerative clustering algorithm using a heap," *Pattern Recognition*, Vol. 24, pp. 205–209, 1991.

[Lamb 62] Lambert J.M., Williams W.T. "Multivariate methods in plant technology, IV. Nodal analysis," *J. Ecol.*, Vol. 50, pp. 775–802, 1962.

[Lamb 66] Lambert J.M., Williams W.T. "Multivariate methods in plant technology, IV. Comparison of information analysis and association analysis," *J. Ecol.*, Vol. 54, pp. 635–664, 1966.

[Lanc 67] Lance G.N., Williams W.T. "A general theory of classificatory sorting strategies: II. Clustering systems," *Computer Journal*, Vol. 10, pp. 271–277, 1967.

[Li 90] Li X. "Parallel algorithms for hierarchical clustering and cluster validity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12(11), pp. 1088–1092, 1990.

[Ling 72] Ling R.F. "On the theory and construction of k-clusters," *Computer Journal*, Vol. 15, pp. 326–332, 1972.

[Liu 68] Liu C.L. *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.

[MacN 64] MacNaughton-Smith P., Williams W.T., Dale M.B., Mockett L.G. "Dissimilarity analysis," *Nature*, Vol. 202, pp. 1034–1035, 1964.

[MacN 65] MacNaughton-Smith P. "Some statistical and other numerical techniques for classifying individuals," *Home Office Research Unit Report No.* 6, London: H.M.S.O., 1965.

[Marr 71] Marriot F.H.C. "Practical problems in a method of cluster analysis," *Biometrics*, Vol. 27, pp. 501–514, 1971.

[McQu 62] McQuitty L.L. "Multiple hierarchical classification of institutions and persons with reference to union-management relations and psychological well-being," *Educ. Psychol. Measur.*, Vol. 23, pp. 55–61, 1962.

[Mill 80] Milligan G.W. "An examination of the effect of six types of error perturbation on fifteen clustering algorithms," *Psychometrika*, Vol. 45, pp. 325–342, 1980.

[Mill 83] Milligan G.W., Soon S.C., Sokol L.M. "The effect of cluster size, dimensionality and the number of clusters on recovery of true cluster structure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 5(1), January 1983.

[Mill 85] Milligan G.W., Cooper M.C. "An examination of procedures for determining the number of clusters in a data set," *Psychometrika*, Vol. 50(2), pp. 159–179, 1985.

[Murt 83] Murtagh F. "A survey of recent advances in hierarchical clustering algorithms," *Journal of Computation*, Vol. 26, pp. 354–359, 1983.

[Murt 84] Murtagh F. "Complexities of hierarchic clustering algorithms: State of the art," *Computational Statistics Quarterly*, Vol. 1(2), pp. 101–113, 1984.

[Murt 85] Murtagh F. *Multidimensional clustering algorithms*, Vienna: Physica-Verlag, COMPSTAT Lectures, Vol. 4, 1985.

[Murt 95] Murthy M.N., Jain A.K. "Knowledge-based clustering scheme for collection, management and retrieval of library books," *Pattern Recognition*, Vol. 28(7), pp. 949–963, 1995.

[Ozaw 83] Ozawa K. "Classic: A hierarchical clustering algorithm based on asymmetric similarities," *Pattern Recognition*, Vol. 16(2), pp 201–211, 1983.

[Prit 71] Pritchard N.M., Anderson A.J.B. "Observations on the use of cluster analysis in botany with an ecological example," *J. Ecol.*, Vol. 59, pp. 727–747, 1971.

[Rolp 73] Rolph F.J. "Algorithm 76: Hierarchical clustering using the minimum spanning tree," *Journal of Computation*, Vol. 16, pp. 93–95, 1973.

[Rolp 78] Rolph F.J. "A probabilistic minimum spanning tree algorithm," *Information Processing Letters*, Vol. 7, pp. 44–48, 1978.

[Shea 65] Sheals J.G. "An application of computer techniques to Acrine taxonomy: A preliminary examination with species of the Hypoaspio-Androlaelaps complex Acarina," *Proc. Linn. Soc. Lond.*, Vol. 176, pp. 11–21, 1965.

[Snea 73] Sneath P. H. A. , Sokal R. R. *Numerical Taxonomy*, W. H. Freeman & Co., San Francisco, 1973.

[Solo 71] Solomon H. "Numerical taxonomy," in *Mathematics in the Archaeological and Historical Sciences* (Hobson F.R., Kendall D.G., Tautu P.A., eds.), University Press, 1971.

[Stri 67] Stringer P. "Cluster analysis of non-verbal judgement of facial expressions," *Br. J. Math. Statist. Psychol.*, Vol. 20, pp. 71–79, 1967.

[Will 89] Willett P. "Efficiency of hierarchic agglomerative clustering using the ICL distributed array processor," *Journal of Documentation*, Vol. 45, pp. 1–45, 1989.

[Will 77] Williams W.T., Lance G.N. "Hierarchical classificatory methods," in *Statistical Methods for Digital Computers* (Enslein K., Ralston A., Wilf H.S., eds.), John Wiley & Sons, 1977.

CHAPTER 14

CLUSTERING ALGORITHMS III: SCHEMES BASED ON FUNCTION OPTIMIZATION

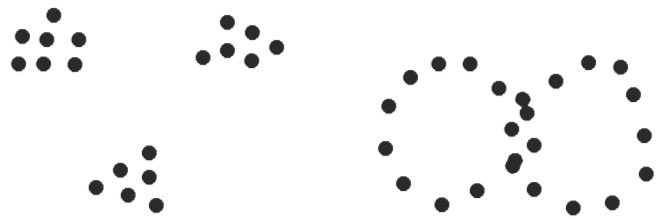
14.1 INTRODUCTION

One of the most commonly used families of clustering schemes relies on the optimization of a cost function J using differential calculus techniques (e.g., see [Duda 73], [Bezd 80], [Bobr 91], [Kris 95a], [Kris 95b]). The cost J is a function of the vectors of the data set X and it is parameterized in terms of an unknown parameter vector, $\boldsymbol{\theta}$. For most of the schemes of the family, the number of clusters, m, is assumed to be known.

Our goal is the estimation of $\boldsymbol{\theta}$ that characterizes best the clusters underlying X. The parameter vector $\boldsymbol{\theta}$ is strongly dependent on the shape of the clusters. For example, for compact clusters (see Figure 14.1a), it is reasonable to adopt as parameters a set of m points, $\boldsymbol{m}_i$, in the l-dimensional space, each corresponding to a cluster—thus, $\boldsymbol{\theta} = [\boldsymbol{m}_1{}^T, \boldsymbol{m}_2{}^T, \ldots, \boldsymbol{m}_m{}^T]^T$. On the other hand, if ring-shaped clusters are expected (see Figure 14.1b), it is reasonable to use m hyperspheres $C(\boldsymbol{c}_i, r_i)$, $i = 1, \ldots, m$, as representatives, where $\boldsymbol{c}_i$ and r_i are the center and the radius of the ith hypersphere, respectively. In this case, $\boldsymbol{\theta} = [\boldsymbol{c}_1{}^T, r_1, \boldsymbol{c}_2{}^T, r_2, \ldots, \boldsymbol{c}_m{}^T, r_m]^T$.

Spherical or, in general, shell-shaped clusters[1] are encountered in many robot vision applications. The basic problem here is the identification of objects (patterns) lying in a *scene* (which is a region in the three-dimensional space), and the estimation of their relative positions, using a single or several *images* (two-dimensional projections of the scene). An important task of this problem is the identification of the boundaries of the objects in the image. Given an image (see, e.g., Figure 14.2a), we may identify the pixels that constitute the boundary of the objects using appropriate operators (see, e.g., [Horn 86], [Kare 94]) (see Figure 14.2b). Then, the boundaries of the objects may be considered as shell-shaped or linear-shaped clusters and clustering algorithms may be mobilized

[1] These may be hyperellipsoids, hyperparabolas, etc.

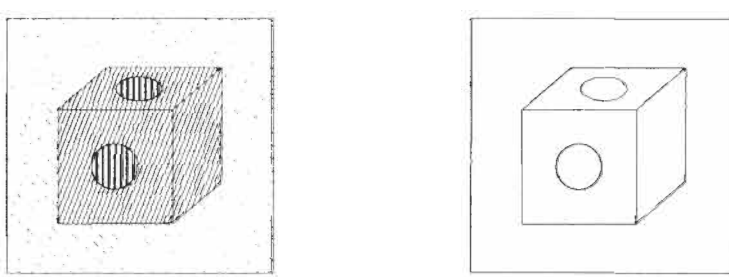


(a) (b)

FIGURE 14.1: (a) Compact clusters. (b) Spherical clusters.

in order to recover their exact shape and location in the image. In fact, clustering techniques have exhibited satisfactory results at least for the case in which the boundaries are known to be shell shaped (e.g., [Kris 95a]).

A distinct characteristic of most of the algorithms of this chapter, compared with the algorithms of the previous chapter, is that the cluster representatives are computed using *all* the available vectors of X and not only the vectors which have been assigned to the respective cluster. We will focus on three major categories of algorithms: the mixture decomposition, the fuzzy, and the possibilistic approaches. In the first, the cost function is constructed on the basis of random vectors, and assignment to clusters follows probabilistic arguments, in the spirit of the Bayesian classification. The conditional probabilities here result from the optimization process. In the fuzzy approach a proximity function between a vector and a cluster is defined and the "grade of membership" of a vector in a cluster is provided by the set of membership functions. As is always the case with fuzzy approaches, the values of the membership functions of a vector in the various



(a) (b)

FIGURE 14.2: (a) The original image of a scene. (b) The image after the application of appropriate operators.

clusters are interrelated. This constraint is removed in the case of the possibilistic approach. Finally, hard clustering may be viewed as a special case of the fuzzy clustering approach, where each vector belongs exclusively to a cluster.

14.2 MIXTURE DECOMPOSITION SCHEMES

The basic reasoning behind this algorithmic family springs from our familiar Bayesian philosophy. We assume that there are m clusters, C_j, $j = 1, \ldots, m$, underlying the data set.[2] Each vector $\boldsymbol{x}_i$, $i = 1, \ldots, N$, belongs to a cluster C_j with probability $P(C_j|\boldsymbol{x}_i)$. A vector $\boldsymbol{x}_i$ is appointed to the cluster C_j if

$$P(C_j|\boldsymbol{x}_i) > P(C_k|\boldsymbol{x}_i), \quad k = 1, \ldots, m, k \neq j$$

The differences from the classification task of Chapter 2 are that (a) no training data with known cluster labelling are available and (b) the a priori probabilities $P(C_j) \equiv P_j$ are not known either. Thus, although the goal is the same, the tools have to be different. Basically, this is a typical task with an incomplete training data set. We are missing the corresponding cluster labelling information for each data point $\boldsymbol{x}_i$. Thus, the task fits nicely in the framework introduced in Section 2.5.5.

From equation (2.81) and adopting the notation for the needs of the current chapter we have

$$Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}(t)) = \sum_{i=1}^{N} \sum_{j=1}^{m} P(C_j|\boldsymbol{x}_i; \boldsymbol{\Theta}(t)) \ln(p(\boldsymbol{x}_i|C_j; \boldsymbol{\theta}) P_j), \tag{14.1}$$

where $\boldsymbol{\theta} = [\boldsymbol{\theta}_1^T, \ldots, \boldsymbol{\theta}_m^T]^T$, with $\boldsymbol{\theta}_j$ being the parameter vector corresponding to the j-th cluster, $\boldsymbol{P} = [P_1, \ldots, P_m]^T$, with P_j being the a priori probability for the j-th cluster and $\boldsymbol{\Theta} = [\boldsymbol{\theta}^T, \boldsymbol{P}^T]^T$. The above equation results from the application of the E-step of the EM algorithm. The M-step of the algorithm is

$$\boldsymbol{\Theta}(t+1) = arg \max_{\boldsymbol{\Theta}} Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}(t)). \tag{14.2}$$

Assuming that all pairs of $\boldsymbol{\theta}_k, \boldsymbol{\theta}_j$'s are functionally independent, that is, no $\boldsymbol{\theta}_k$ gives any information about $\boldsymbol{\theta}_j (j \neq i)$, we estimate $\boldsymbol{\theta}_j$ from eq. (14.2) as follows:

$$\sum_{i=1}^{N} \sum_{j=1}^{m} P(C_j|\boldsymbol{x}_i; \boldsymbol{\Theta}(t)) \frac{\partial}{\partial \boldsymbol{\theta}_j} \ln p(\boldsymbol{x}_i|C_j; \boldsymbol{\theta}_j) = \boldsymbol{0} \tag{14.3}$$

[2] Recall that the number m is assumed to be known.

Maximization with respect to $\boldsymbol{P}$ is a constraint optimization problem since

$$P_k \geq 0, \quad k = 1, \ldots, m, \quad \text{and} \quad \sum_{k=1}^{m} P_k = 1. \tag{14.4}$$

The corresponding Lagrangian function is

$$\mathcal{Q}(\boldsymbol{P}, \lambda) = Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}(t)) - \lambda \left(\sum_{k=1}^{m} P_k - 1 \right) \tag{14.5}$$

Taking the partial derivative of $\mathcal{Q}(\boldsymbol{P}, \lambda)$ with respect to P_j and setting it equal to 0, and after some algebra we obtain

$$P_j = \frac{1}{\lambda} \sum_{i=1}^{N} P(C_j | \boldsymbol{x}_i; \boldsymbol{\Theta}(t)), \quad j = 1, \ldots, m \tag{14.6}$$

Substituting the above equations into Eq. (14.4) we obtain

$$\lambda = \sum_{i=1}^{N} \sum_{j=1}^{m} P(C_j | \boldsymbol{x}_i; \boldsymbol{\Theta}(t)) = N \tag{14.7}$$

Thus, Eq. (14.6) gives

$$P_j = \frac{1}{N} \sum_{i=1}^{N} P(C_j | \boldsymbol{x}_i; \boldsymbol{\Theta}(t)) \quad j = 1, \ldots, m \tag{14.8}$$

Taking into account Eqs. (14.3), (14.8) and (2.87), the EM algorithm for this case may be stated as

Generalized Mixture Decomposition Algorithmic Scheme (GMDAS)

- Choose initial estimates, $\boldsymbol{\theta} = \boldsymbol{\theta}(0)$ and $\boldsymbol{P} = \boldsymbol{P}(0)$.[3]
- $t = 0$
- Repeat

  —Compute

$$P(C_j | \boldsymbol{x}_i; \boldsymbol{\Theta}(t)) = \frac{p(\boldsymbol{x}_i | C_j; \boldsymbol{\theta}_j(t)) P_j(t)}{\sum_{k=1}^{m} p(\boldsymbol{x}_i | C_k, \boldsymbol{\theta}_k(t)) P_k(t)} \tag{14.9}$$

  $i = 1, \ldots, N, \quad j = 1, \ldots, m.$

[3] Initial conditions must satisfy the constraints.

—Set $\boldsymbol{\theta}_j(t+1)$ equal to the solution of the equation

$$\sum_{i=1}^{N}\sum_{j=1}^{m} P(C_j|\boldsymbol{x}_i;\boldsymbol{\Theta}(t))\frac{\partial}{\partial \boldsymbol{\theta}_j}\ln p(\boldsymbol{x}_i|C_j;\boldsymbol{\theta}_j)=\mathbf{0} \tag{14.10}$$

with respect to $\boldsymbol{\theta}_j$, for $j=1,\ldots,m$.

—Set

$$P_j(t+1)=\frac{1}{N}\sum_{i=1}^{N} P(C_j|\boldsymbol{x}_i;\boldsymbol{\Theta}(t)), \quad j=1,\ldots,m \tag{14.11}$$

—$t=t+1$

- Until convergence, with respect to $\boldsymbol{\Theta}$, is achieved.

An appropriate termination criterion for the algorithm is the following:

$$\|\boldsymbol{\Theta}(t+1)-\boldsymbol{\Theta}(t)\|<\varepsilon$$

where $\|\cdot\|$ is an appropriate vector norm and ε is a "small" user-defined constant. This scheme is guaranteed to converge to a global or a local maximum of the loglikelihood function. However, even if a local maximum solution is reached, it may still capture satisfactorily the underlying clustering structure of X.

Once the algorithm has converged, vectors are assigned to clusters according to the final estimates $P(C_j|\boldsymbol{x}_i)$ of Eq. (14.9). Hence, the task now becomes a typical Bayesian classification problem, if we treat each cluster as a separate class.

14.2.1 Compact and Hyperellipsoidal Clusters

In this section, we focus our attention on the case in which the vectors of X form compact clusters. A distribution that is suitable for clusters of this scheme is the normal distribution, that is,

$$p(\boldsymbol{x}|C_j;\boldsymbol{\theta}_j)=\frac{1}{(2\pi)^{l/2}|\Sigma_j|^{1/2}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T\Sigma_j^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right), \quad j=1,\ldots,m \tag{14.12}$$

or

$$\ln p(\boldsymbol{x}|C_j;\boldsymbol{\theta}_j)=\ln\frac{|\Sigma_j|^{-1/2}}{(2\pi)^{l/2}}-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T\Sigma_j^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j), \quad j=1,\ldots,m \tag{14.13}$$

In this case, each vector $\boldsymbol{\theta}_j$ consists of the l parameters of the mean $\boldsymbol{\mu}_j$ and the $l(l+1)/2$ independent parameters of Σ_j. A parameter reduction may be obtained

by assuming that the covariance matrices are diagonal. If this assumption is too strict, another commonly used assumption is that all covariance matrices are equal. In the former case θ consists of $2ml$ parameters, while in the latter it consists of $ml + l(l+1)/2$ parameters.

Combining Eq. (14.12) and Eq. (14.9) results in

$$P(C_j|\boldsymbol{x}; \boldsymbol{\Theta}(t)) = \frac{|\Sigma_j(t)|^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_j(t))^T \Sigma_j^{-1}(t)(\boldsymbol{x} - \boldsymbol{\mu}_j(t))\right) P_j(t)}{\sum_{k=1}^{m} |\Sigma_k(t)|^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k(t))^T \Sigma_k^{-1}(t)(\boldsymbol{x} - \boldsymbol{\mu}_k(t))\right) P_k(t)} \tag{14.14}$$

In the sequel, we consider the problem in its most general form; that is, we assume that all the means $\boldsymbol{\mu}_j$ and the covariance matrices Σ_j are unknown. We also assume that, in general, all Σ_j's are different from each other. Following an approach similar to the one described in Chapter 2, the updating equations for $\boldsymbol{\mu}_j$'s and Σ_j's from the M-step are

$$\boldsymbol{\mu}_j(t+1) = \frac{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))\boldsymbol{x}_k}{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))} \tag{14.15}$$

and

$$\Sigma_j(t+1) = \frac{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))(\boldsymbol{x}_k - \boldsymbol{\mu}_j(t))(\boldsymbol{x}_k - \boldsymbol{\mu}_j(t))^T}{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))} \tag{14.16}$$

$j = 1, \ldots, m$.

Thus, in the Gaussian case these two iterations replace Eq. (14.10), and Eq. (14.14) replaces Eq. (14.9) in the corresponding steps of the GMDAS algorithm.

Remark

- Notice that this scheme is computationally very demanding, because at each iteration step the inverses of m covariance matrices are required for the computation of $P(C_j|\boldsymbol{x}_i; \boldsymbol{\Theta}(t))$. As stated earlier, one way to relax this demand is to assume that all covariance matrices are diagonal or that all are equal to each other. In the latter case, only one inversion is required at each iteration step.

Example 14.1. (a) Consider 3 two-dimensional normal distributions with means $\boldsymbol{\mu}_1 = [1, 1]^T$, $\boldsymbol{\mu}_2 = [3.5, 3.5]^T$, $\boldsymbol{\mu}_3 = [6, 1]^T$ and covariance matrices

$$\Sigma_1 = \begin{bmatrix} 1 & -0.3 \\ -0.3 & 1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}$$

respectively.

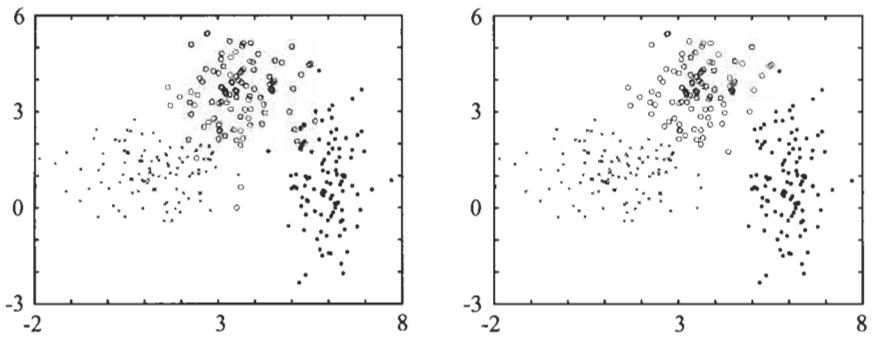


FIGURE 14.3: (a) A data set that consists of three groups of points. (b) The results from the application of GMDAS when normal mixtures are used.

A group of 100 vectors is generated from each distribution. These groups constitute the data set X. Figure 14.3a is a plot of the data generated.

We initialize $P_j = 1/3$, $j = 1, 2, 3$. Also, we set $\boldsymbol{\mu}_i(0) = \boldsymbol{\mu}_i + \boldsymbol{y}_i$, where $\boldsymbol{y}_i$ is an 2×1 vector with random coordinates, uniformly distributed in the interval $[-1, 1]^T$. Similarly, we define $\Sigma_i(0)$, $i = 1, 2, 3$. We set $\varepsilon = 0.01$. Using these initial conditions, the GMDAS for Gaussian pdf's terminates after 38 iterations. The final parameter estimates obtained are $\boldsymbol{P}' = [0.35, 0.31, 0.34]^T$, $\boldsymbol{\mu}_1' = [1.28, 1.16]^T$, $\boldsymbol{\mu}_2' = [3.49, 3.68]^T$, $\boldsymbol{\mu}_3' = [5.96, 0.84]^T$, and

$$\Sigma_1' = \begin{bmatrix} 1.45 & 0.01 \\ 0.01 & 0.57 \end{bmatrix}, \quad \Sigma_2' = \begin{bmatrix} 0.62 & 0.09 \\ 0.09 & 0.74 \end{bmatrix}, \quad \Sigma_3' = \begin{bmatrix} 0.30 & 0.0024 \\ 0.0024 & 1.94 \end{bmatrix}$$

For comparison, the sample mean values are $\hat{\boldsymbol{\mu}}_1 = [1.16, 1.13]^T$, $\hat{\boldsymbol{\mu}}_2 = [3.54, 3.56]^T$, $\hat{\boldsymbol{\mu}}_3 = [5.97, 0.76]^T$, respectively. Also, the sample covariance matrices are

$$\hat{\Sigma}_1 = \begin{bmatrix} 1.27 & -0.03 \\ -0.03 & 0.52 \end{bmatrix}, \quad \hat{\Sigma}_2 = \begin{bmatrix} 0.70 & 0.07 \\ 0.07 & 0.98 \end{bmatrix}, \quad \hat{\Sigma}_3 = \begin{bmatrix} 0.32 & 0.05 \\ 0.05 & 1.81 \end{bmatrix}$$

respectively.

As we can see, the final estimates of the algorithm are close enough to the means and the covariance matrices of the three groups of vectors.

Once the unknown parameters of the model have been estimated, the data vectors are assigned to clusters according to the estimated values of $P(C_j|\boldsymbol{x}_i)$. Figure 14.3b shows the assignment of points to the three clusters, which is in close agreement with the original structure. A way to assess the performance of the resulting model estimates is via the

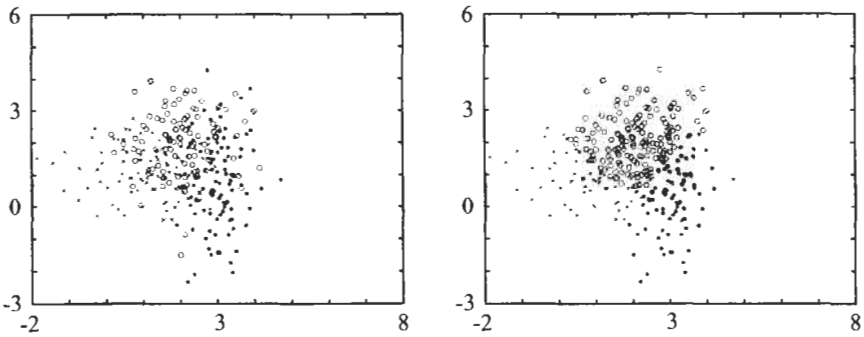


FIGURE 14.4: (a) The data set, which consists of three overlapping groups of points. (b) The results of the GMDAS when Gaussian mixtures are used.

so-called *confusion matrix*. This is a matrix A whose (i, j) element is the number of vectors that originate from the ith distribution and are assigned to the jth cluster.[4] For our example this is

$$A_1 = \begin{bmatrix} 99 & 0 & 1 \\ 0 & 100 & 0 \\ 3 & 4 & 93 \end{bmatrix}$$

This example indicates that 99% of the data from the first distribution are assigned to the same cluster (the first one). Similarly, all the data from the second distribution are assigned to the same cluster (the second one) and, finally, 93% of the data from the third distribution are assigned to the same cluster (the third one).

(b) Let us now consider the case in which the three normal distributions are located closer, for example, $\boldsymbol{\mu}_1 = [1, 1]^T$, $\boldsymbol{\mu}_2 = [2, 2]^T$, $\boldsymbol{\mu}_3 = [3, 1]^T$, and with the same covariance matrices as before (Figure 14.4). We initialize $\boldsymbol{\mu}_i$ and Σ_i, $i = 1, 2, 3$, as in the previous case and run the GMDAS for Gaussian pdf's. The confusion matrix for this case is

$$A_2 = \begin{bmatrix} 85 & 4 & 11 \\ 35 & 56 & 9 \\ 26 & 0 & 74 \end{bmatrix}$$

As expected, each one of the obtained clusters contains a significant percentage of points from more than one distribution.

[4] It should be noted here that in real clustering applications the confusion matrix cannot be defined, since we do not know a priori the cluster where each feature vector belongs. However, we may use it in artificial experiments such as this one, in order to evaluate the performance of the clustering algorithms.

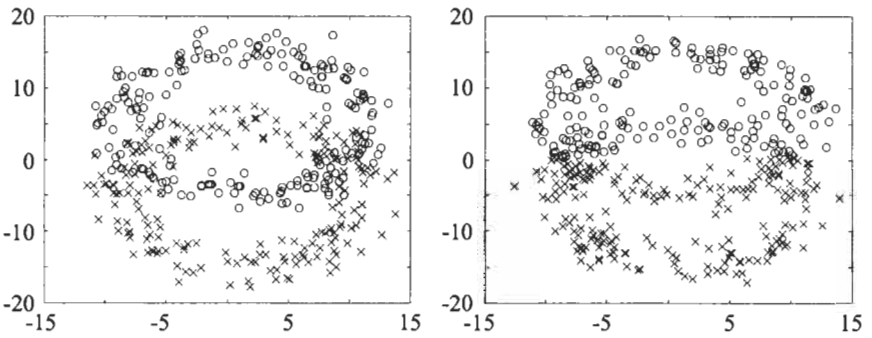


FIGURE 14.5: (a) A data set that consists of ring-shaped intersecting clusters. (b) The results from the application of GMDAS when Gaussian mixtures are used.

Example 14.2. The data set X, which is depicted in Figure 14.5a, consists of two intersecting ring-shaped clusters. Each cluster consists of 500 points. We run the GMDAS with Gaussians and $m = 2$ and $\varepsilon = 0.01$. The algorithm terminates after 72 iterations and the results are shown in Figure 14.5b. *As expected, the algorithm fails to recover the underlying clustering structure of X, because it seeks compact clusters. Generally speaking, GMDAS using Gaussians reveals clusters that are as compact as possible, even though the clusters underlying X may have a different shape.* Even worse, it will identify clusters in X even though there is no clustering structure in it.[5]

In [Zhua 96], the case of Gaussian pdf's, contaminated by unknown outlier distributions, $h(\boldsymbol{x}_i|C_j)$, is considered. In this case, we can write $p(\boldsymbol{x}|C_j) = (1-\varepsilon_j)G(\boldsymbol{x}|C_j)+\varepsilon_j h(\boldsymbol{x}|C_j)$, where ε_j is the level of contamination and $G(\boldsymbol{x}|C_j)$ is the jth Gaussian distribution. Under the assumption that all $h(\boldsymbol{x}_i|C_j)$ are constant, that is, $h(\boldsymbol{x}_i|C_j) = c_j$, $i = 1, \ldots, N$, $p(\boldsymbol{x}|C_j)$ may be written as $p(\boldsymbol{x}|C_j) = (1 - \varepsilon_j)G(\boldsymbol{x}|C_j) + \varepsilon_j c_j$. Then we may use the preceding methodology in order to identify the mean and the covariance matrices of the normal distributions $G(\boldsymbol{x}|C_j)$ as well as the values of ε_j and c_j.

14.2.2 A Geometrical Interpretation

As mentioned earlier, the conditional probability, $P(C_j|\boldsymbol{x}_i)$, indicates how likely it is that $\boldsymbol{x}_i \in X$ belongs to C_j, $i = 1, \ldots, N$, subject, of course, to

[5] In general, before we apply any clustering algorithm to identify clusters contained in X, we should first check whether *there exists* any clustering structure in X. This procedure refers to *clustering tendency* and is considered in Chapter 16.

the constraint

$$\sum_{j=1}^{m} P(C_j|\boldsymbol{x}_i) = 1 \tag{14.17}$$

This may be viewed as the equation of an $(m-1)$-dimensional hyperplane. For notational purposes, let $P(C_j|\boldsymbol{x}_i) \equiv y_j$, $j = 1, \ldots, m$. Then Eq. (14.17) may be written as

$$\boldsymbol{a}^T \boldsymbol{y} = 1 \tag{14.18}$$

where $\boldsymbol{y}^T = [y_1, \ldots, y_m]$ and $\boldsymbol{a}^T = [1, 1, \ldots, 1]$. That is, $\boldsymbol{y}$ is allowed to move on the hyperplane defined by the previous equation. In addition, since $0 \leq y_j \leq 1$, $j = 1, \ldots, m$, $\boldsymbol{y}$ lies also inside the unit hypercube (see Figure 14.6).

This interpretation allows us to derive some useful conclusions for the so-called *noisy feature vectors* or *outliers*. Let $\boldsymbol{x}_i$ be such a vector. Since Eq. (14.17) holds for $\boldsymbol{x}_i$, at least one of the y_j's, $j = 1, \ldots, m$, is significant (it lies in the interval $[1/m, 1]$). Thus, $\boldsymbol{x}_i$ will affect, at least, the estimates for the corresponding cluster C_j, through Eqs. (14.9), (14.10), and (14.11), and this makes GMDAS sensitive to outliers. The following example clarifies this idea further.

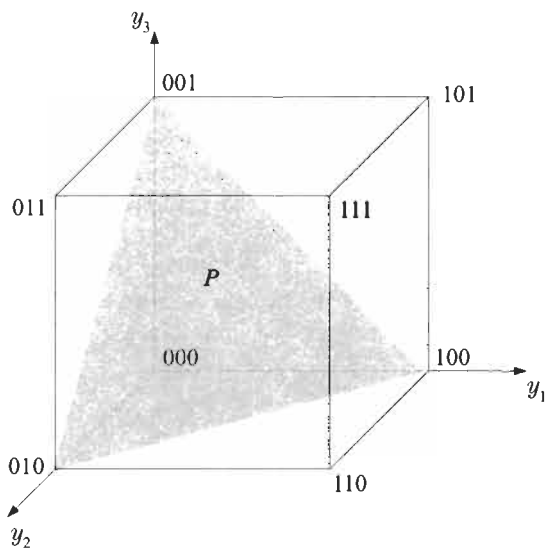


FIGURE 14.6: The hypercube for $m = 3$. The point $\boldsymbol{y}$ is allowed to move only on the shaded region of P.

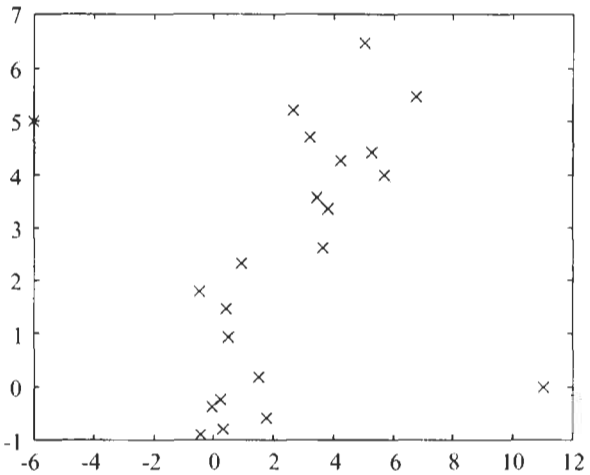


FIGURE 14.7: The data set for Example 14.3.

Example 14.3. Consider the data set X shown in Figure 14.7. It consists of 22 vectors. The first (next) 10 vectors are drawn from the normal distribution with mean $\boldsymbol{\mu}_1 = [0, 0]^T$ ($\boldsymbol{\mu}_2 = [4.5, 4.5]^T$) and covariance matrix $\Sigma_1 = I$ ($\Sigma_2 = I$), where I is the 2×2 identity matrix. The last two points are $\boldsymbol{x}_{21} = [-6, 5]^T$ and $\boldsymbol{x}_{22} = [11, 0]^T$, respectively. We run the GMDAS for Gaussian pdf's on X. The estimates of $\boldsymbol{P}$, $\boldsymbol{\mu}_j$, and Σ_j, $j = 1, 2$, obtained after five iterations, are

$$\boldsymbol{P}' = [0.5, 0.5]^T, \quad \boldsymbol{\mu}_1' = [-0.58, 0.35]^T, \quad \boldsymbol{\mu}_2' = [4.98, 4.00]^T$$

$$\Sigma_1' = \begin{bmatrix} 4.96 & -2.01 \\ -2.01 & 2.63 \end{bmatrix}, \qquad \Sigma_2' = \begin{bmatrix} 3.40 & -2.53 \\ -2.53 & 3.27 \end{bmatrix}$$

The resulting values of $P(C_j|\boldsymbol{x}_i)$, $j = 1, 2$, $i = 1, \ldots, 22$, are shown in Table 14.1. Although $\boldsymbol{x}_{21}$ and $\boldsymbol{x}_{22}$ may be considered as outliers, since they lie away from the two clusters, we get that $P(C_1|\boldsymbol{x}_{21}) = 1$ and $P(C_2|\boldsymbol{x}_{22}) = 1$, due to the constraint $\sum_{j=1}^{2} P(C_j|\boldsymbol{x}_i) = 1$. This implies that these points have a nonnegligible impact on $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, Σ_1, and Σ_2. Indeed, if we run GMDAS for Gaussian pdf's on $X_1 = \{\boldsymbol{x}_i : i = 1, \ldots, 20\}$ (i.e., we exclude the last two points), using the same initial conditions, we obtain after five iterations:

$$\boldsymbol{P}'' = [0.5, 0.5]^T, \quad \boldsymbol{\mu}_1'' = [-0.03, -0.12]^T, \quad \boldsymbol{\mu}_2'' = [4.37, 4.40]^T$$

$$\Sigma_1'' = \begin{bmatrix} 0.50 & -0.01 \\ -0.01 & 1.22 \end{bmatrix}, \qquad \Sigma_2'' = \begin{bmatrix} 1.47 & 0.44 \\ 0.44 & 1.13 \end{bmatrix}$$

Comparing the results of the two experiments, it is easily observed that the last setup gives more accurate estimates of the unknown parameters.

Table 14.1: The Resulting A Posteriori Probabilities for the Data Set of Example 14.3

feat. vec.	$P(C_1\|\boldsymbol{x})$	$P(C_2\|\boldsymbol{x})$	feat. vec.	$P(C_1\|\boldsymbol{x})$	$P(C_2\|\boldsymbol{x})$
$\boldsymbol{x}_1$	0	1	$\boldsymbol{x}_{12}$	1	0
$\boldsymbol{x}_2$	0	1	$\boldsymbol{x}_{13}$	1	0
$\boldsymbol{x}_3$	0	1	$\boldsymbol{x}_{14}$	1	0
$\boldsymbol{x}_4$	0	1	$\boldsymbol{x}_{15}$	1	0
$\boldsymbol{x}_5$	0	1	$\boldsymbol{x}_{16}$	1	0
$\boldsymbol{x}_6$	0	1	$\boldsymbol{x}_{17}$	1	0
$\boldsymbol{x}_7$	0	1	$\boldsymbol{x}_{18}$	1	0
$\boldsymbol{x}_8$	0	1	$\boldsymbol{x}_{19}$	0.99	0.01
$\boldsymbol{x}_9$	0	1	$\boldsymbol{x}_{20}$	1	0
$\boldsymbol{x}_{10}$	0	1	$\boldsymbol{x}_{21}$	0	1
$\boldsymbol{x}_{11}$	1	0	$\boldsymbol{x}_{22}$	1	0

Another interesting observation can be derived by examining the following situation. Let $l = 1$. Consider two clusters described by normal distributions $p(x|C_j)$, $j = 1, 2$, with the same variance and means μ_1 and μ_2, respectively ($\mu_1 < \mu_2$). Also let $P_1 = P_2$. It is not difficult to prove that for $x < (>)\frac{\mu_1+\mu_2}{2}$, $P(C_1|x) > (<) P(C_2|x)$. Now consider the points $x_1 = \frac{3\mu_1+\mu_2}{4}$ and $x_2 = \frac{5\mu_1-\mu_2}{4}$. Although these points have the same distance from μ_1 (i.e., they are symmetric with respect to μ_1), it is not hard to show that $P(C_1|x_1) > P(C_1|x_2)$. This happens because $P(C_1|x)$ and $P(C_2|x)$ are *related* through Eq. (14.17). Thus, the probability of having x in one cluster is affected by the probability of belonging to the other. We will soon see that we can free ourselves from such an interrelation.

14.3 FUZZY CLUSTERING ALGORITHMS

One of the difficulties associated with the previously discussed probabilistic algorithms is the involvement of the pdf's, for which a suitable model has to be assumed. In addition, it is not easy to handle cases where the clusters are not compact but are shell shaped. A family of clustering algorithms that emancipates itself from such constraints is that of fuzzy clustering algorithms. These schemes have been the subject of intensive research during the past two decades. The major point that differentiates the two approaches is that in the fuzzy schemes a vector *belongs simultaneously* to more than one cluster, whereas in the probabilistic schemes, each vector belongs *exclusively* to a single cluster.

As already discussed in Chapter 11, a *fuzzy m-clustering of X* is defined by a set of functions $u_j : X \to A$, $j = 1, \ldots, m$, where $A = [0, 1]$.

In the case where $A = \{0, 1\}$, a *hard m-clustering of X* is defined. In this case, each vector belongs exclusively to a single cluster.

As in the previous section, it is assumed that the number of clusters as well as their shape is known a priori. The shape of the clusters is characterized by the adopted set of parameters. For example, if we deal with compact clusters, a point representative is used to represent each cluster; that is, each cluster is represented by l parameters. On the other hand, if we deal with noncompact but, say, hyperspherical clusters, a hypersphere is used as a representative of each cluster. In this case, each cluster is represented by $l + 1$ parameters (l for the center of the hypersphere and 1 for its radius).

In the sequel we use the following notation: $\boldsymbol{\theta}_j$ is the parameterized representative of the jth cluster, $\boldsymbol{\theta} \equiv [\boldsymbol{\theta}_1^T, \ldots, \boldsymbol{\theta}_m^T]^T$, U is an $N \times m$ matrix whose (i, j) element equals $u_j(\boldsymbol{x}_i)$, $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ is the dissimilarity between $\boldsymbol{x}_i$ and $\boldsymbol{\theta}_j$, and $q(> 1)$ is a parameter called a *fuzzifier*. The role of the latter will be clarified shortly. Most of the well-known fuzzy clustering algorithms are those derived by minimizing a cost function of the form

$$J_q(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) \tag{14.19}$$

with respect to $\boldsymbol{\theta}$ and U, subject to the constraints

$$\sum_{j=1}^{m} u_{ij} = 1, \quad i = 1, \ldots, N \tag{14.20}$$

where

$$\begin{aligned} u_{ij} \in [0, 1], \quad i = 1, \ldots, N, \quad j = 1, \ldots, m, \\ 0 < \sum_{i=1}^{N} u_{ij} < N, \quad j = 1, 2, \ldots, m \end{aligned} \tag{14.21}$$

In other words, the grade of membership of $\boldsymbol{x}_i$ in the j-th cluster is related to the grade of membership of $\boldsymbol{x}_i$ to the rest $m - 1$ clusters through Eq. (14.20). Different values of q in Eq. (14.19) bias $J_q(\boldsymbol{\theta}, U)$ toward either the fuzzy or the hard clusterings. More specifically, for fixed $\boldsymbol{\theta}$, *if* $q = 1$, *no fuzzy clustering is better than the best hard clustering in terms of* $J_q(\boldsymbol{\theta}, U)$. *However, if* $q > 1$, *there are cases in which fuzzy clusterings lead to lower values of* $J_q(\boldsymbol{\theta}, U)$ *than the best hard clustering.* Let us clarify these ideas further using the following example.

Example 14.4. Let $X = \{x_1, x_2, x_3, x_4\}$, where $x_1 = [0, 0]^T$, $x_2 = [2, 0]^T$, $x_3 = [0, 3]^T$, $x_4 = [2, 3]^T$. Let $\theta_1 = [1, 0]^T$, $\theta_2 = [1, 3]^T$ be the cluster representatives. Suppose also that the Euclidean distance between a vector and a representative is in use. The hard two-cluster clustering that minimizes $J_q(\theta, U)$, for the above choice of θ_1, θ_2, can be represented by

$$U_{hard} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

The value of $J_q(\theta, U)$ in this case (Eq. 14.19) is $J_q^{hard}(\theta, U) = 4$. Obviously, hard clusterings do not depend on q.

Assume now that $q = 1$ and u_{ij}'s are between 0 and 1. Then the value of the cost function becomes

$$J_1^{fuzzy}(\theta, U) = \sum_{i=1}^{2}(u_{i1} + u_{i2}\sqrt{10}) + \sum_{i=3}^{4}(u_{i1}\sqrt{10} + u_{i2})$$

Since for each x_i both u_{i1} and u_{i2} are positive and $u_{i1} + u_{i2} = 1$, it easily follows that $J_1^{fuzzy}(\theta, U) > 4$. Thus, the hard clustering always results in better values of $J_q^{fuzzy}(\theta, U)$, compared with their fuzzy counterparts, when $q = 1$.

Assume now that $q = 2$. The reader should easily verify that when $u_{i2} \in [0, 0.48]$ for $i = 1, 2$ and $u_{i1} \in [0, 0.48]$ for $i = 3, 4$, and, of course, $u_{i1} = 1 - u_{i2}$, for each x_i, then the value of $J_2^{fuzzy}(\theta, U)$ is less than 4 (see Problem 14.7). Thus, in this case fuzzy clusterings are favored over hard ones.

Finally, let $q = 3$. In this case, it is easily verified that when $u_{i2} \in [0, 0.67]$ for $i = 1, 2$ and $u_{i1} \in [0, 0.67]$ for $i = 3, 4$ and $u_{i1} = 1 - u_{i2}$, for each x_i, then the value of $J_3^{fuzzy}(\theta, U)$ is also less than 4.

Minimization of $J_q(\theta, U)$

We first assume that no x_i coincides with any of the representatives. More formally, for an x_i let Z_i be the set that contains the indices of the representatives θ_j for which $d(x_i, \theta_j) = 0$. According to our assumption, $Z_i = \emptyset$, for all i. In the sequel, for ease of notation, we drop the index q from $J_q(\theta, U)$. Let us consider first U. Minimization of $J_q(\theta, U)$ with respect to U, subject to the constraint (14.20), leads to the following Lagrangian function:

$$\mathcal{J}(\theta, U) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^q d(x_i, \theta_j) - \sum_{i=1}^{N}\lambda_i\left(\sum_{j=1}^{m} u_{ij} - 1\right) \tag{14.22}$$

The partial derivative of $\mathcal{J}(\boldsymbol{\theta}, U)$ with respect to u_{rs} is

$$\frac{\partial \mathcal{J}(\boldsymbol{\theta}, U)}{\partial u_{rs}} = q u_{rs}^{q-1} d(\boldsymbol{x}_r, \boldsymbol{\theta}_s) - \lambda_r \tag{14.23}$$

Setting $\partial \mathcal{J}(\boldsymbol{\theta}, U)/\partial u_{rs}$ equal to 0 and solving with respect to u_{rs}, we obtain

$$u_{rs} = \left(\frac{\lambda_r}{q d(\boldsymbol{x}_r, \boldsymbol{\theta}_s)} \right)^{\frac{1}{q-1}}, \quad s = 1, \ldots, m \tag{14.24}$$

Substituting u_{rs} from the previous equation in the constraint equation $\sum_{j=1}^{m} u_{rj} = 1$, we obtain

$$\sum_{j=1}^{m} \left(\frac{\lambda_r}{q d(\boldsymbol{x}_r, \boldsymbol{\theta}_j)} \right)^{\frac{1}{q-1}} = 1$$

or

$$\lambda_r = \frac{q}{\left(\sum_{j=1}^{m} \left(\frac{1}{d(\boldsymbol{x}_r, \boldsymbol{\theta}_j)} \right)^{\frac{1}{q-1}} \right)^{q-1}} \tag{14.25}$$

Combining Eq. (14.25) with (14.24) and using a bit of algebra, we obtain

$$u_{rs} = \frac{1}{\sum_{j=1}^{m} \left(\frac{d(\boldsymbol{x}_r, \boldsymbol{\theta}_s)}{d(\boldsymbol{x}_r, \boldsymbol{\theta}_j)} \right)^{\frac{1}{q-1}}} \tag{14.26}$$

$r = 1, \ldots, N$, $s = 1, \ldots, m$.

Now consider the parameter vector $\boldsymbol{\theta}_j$. Taking the gradient of $J(\boldsymbol{\theta}, U)$ with respect to $\boldsymbol{\theta}_j$ and setting it equal to zero, we obtain

$$\frac{\partial J(\boldsymbol{\theta}, U)}{\partial \boldsymbol{\theta}_j} = \sum_{i=1}^{N} u_{ij}^{q} \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j} = \mathbf{0}, \quad j = 1, \ldots, m \tag{14.27}$$

Equations (14.26) and (14.27) are coupled and, in general, cannot give closed form solutions. One way to proceed is to employ the following iterative algorithmic scheme, in order to obtain estimates for U and $\boldsymbol{\theta}$.

Generalized Fuzzy Algorithmic Scheme (GFAS)

- Choose $\boldsymbol{\theta}_j(0)$ as initial estimates for $\boldsymbol{\theta}_j$, $j = 1, \ldots, m$.
- $t = 0$
- Repeat

  —For $i = 1$ to N

  * For $j = 1$ to m

    - $$u_{ij}(t) = \frac{1}{\sum_{k=1}^{m} \left(\frac{d(\boldsymbol{x}_i, \boldsymbol{\theta}_j(t))}{d(\boldsymbol{x}_i, \boldsymbol{\theta}_k(t))} \right)^{\frac{1}{q-1}}}$$

  * End {For-j}

  —End {For-i}

  —$t = t + 1$

  —For $j = 1$ to m

  * *Parameter updating:* Solve

$$\sum_{i=1}^{N} u_{ij}^{q}(t-1) \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j} = \mathbf{0} \tag{14.28}$$

  with respect to $\boldsymbol{\theta}_j$ and set $\boldsymbol{\theta}_j(t)$ equal to this solution.

  —End {For-j}

- Until a termination criterion is met.

As the termination criterion we may employ $\|\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-1)\| < \varepsilon$, where $\|\cdot\|$ is any vector norm and ε is a "small" user-defined constant.

Remarks

- If, for a given $\boldsymbol{x}_i$, $Z_i \neq \emptyset$, we arbitrarily choose u_{ij}'s, with $j \in Z_i$, such that $\sum_{j \in Z_i} u_{ij} = 1$ and $u_{ij} = 0$, for $j \notin Z_i$. That is, $\boldsymbol{x}_i$ is shared arbitrarily among the clusters whose representatives coincide with $\boldsymbol{x}_i$, subject to the constraint (14.20). In the case in which $\boldsymbol{x}_i$ coincides with a single representative, say $\boldsymbol{\theta}_j$, the condition becomes $u_{ij} = 1$, and $u_{ik} = 0$, $k \neq j$.
- The algorithmic scheme may also be initialized from $U(0)$ instead of $\boldsymbol{\theta}_j(0)$, $j = 1, \ldots, m$, and start iterations with computing $\boldsymbol{\theta}_j$ first.
- The above iterative algorithmic scheme is also known as the *alternating optimization (AO) scheme*, since at each iteration step U is updated for fixed $\boldsymbol{\theta}$, and then $\boldsymbol{\theta}$ is updated for fixed U ([Bezd 95], [Hopp 99]).

In the sequel the algorithm is specialized to three commonly encountered cases.

14.3.1 Point Representatives

In the case of compact clusters, a point representative is used for each cluster; that is, $\boldsymbol{\theta}_j$ consists of l parameters. In this case, the dissimilarity $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ may be any distance measure between two points. Two common choices for $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ are (see also Chapter 11)

$$d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = (\boldsymbol{x}_i - \boldsymbol{\theta}_j)^T A(\boldsymbol{x}_i - \boldsymbol{\theta}_j) \tag{14.29}$$

where A is a symmetric, positive definite matrix, and the *Minkowski distance*,

$$d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = \left(\sum_{k=1}^{l} |x_{ik} - \theta_{jk}|^p \right)^{\frac{1}{p}} \tag{14.30}$$

where p is a positive integer and x_{ik}, θ_{jk} are the kth coordinates of $\boldsymbol{x}_i$ and $\boldsymbol{\theta}_j$, respectively. Let us now see the specific form of GFAS under these choices.

- When the distance given in Eq. (14.29) is in use, we have

$$\frac{\partial d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j} = 2A(\boldsymbol{\theta}_j - \boldsymbol{x}_i) \tag{14.31}$$

  Substituting Eq. (14.31) into Eq. (14.28), we obtain

$$\sum_{i=1}^{N} u_{ij}^q (t-1) 2A(\boldsymbol{\theta}_j - \boldsymbol{x}_i) = \mathbf{0}$$

  Since A is positive definite, it is invertible. Premultiplying both sides of this equation with A^{-1} and after some simple algebra, we obtain

$$\boldsymbol{\theta}_j(t) = \frac{\sum_{i=1}^{N} u_{ij}^q (t-1) \boldsymbol{x}_i}{\sum_{i=1}^{N} u_{ij}^q (t-1)} \tag{14.32}$$

  The resulting algorithm is also known as *Fuzzy c-Means (FCM)* or *Fuzzy k-means algorithm* and has been discussed extensively in the literature (e.g. [Bezd 80], [Cann 86], [Hath 86], [Hath 89], [Isma 86]).
- Let us now examine the case in which Minkowski distances are in use. In the sequel, we consider only the case where p is even and $p < +\infty$. In this

case, we can guarantee the differentiability of $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ with respect to $\boldsymbol{\theta}_j$. Equation (14.30) then gives

$$\frac{\partial d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \theta_{jr}} = \frac{(\theta_{jr} - x_{ir})^{p-1}}{\left(\sum_{k=1}^{l} |x_{ik} - \theta_{jk}|^p\right)^{1-\frac{1}{p}}}, \quad r = 1, \ldots, l \tag{14.33}$$

Substituting Eq. (14.33) into Eq. (14.28), we obtain

$$\sum_{i=1}^{N} u_{ij}^q (t-1) \frac{(\theta_{jr} - x_{ir})^{p-1}}{\left(\sum_{k=1}^{l} |x_{ik} - \theta_{jk}|^p\right)^{1-\frac{1}{p}}} = 0, \quad r = 1, \ldots, l \tag{14.34}$$

Hence, we end up with a system of l nonlinear equations and l unknowns, that is, the coordinates of $\boldsymbol{\theta}_j$. This can be solved by an iterative technique, such as the Gauss–Newton or the Levenberg–Marquardt (L-M) method (e.g., [Luen 84]).

The resulting algorithms are also known as *pFCM*, where p indicates the employed Minkowski distance ([Bobr 91]).

In the iterative technique, the initial estimates at step t can be the estimates obtained from the previous iteration step $t - 1$.

Example 14.5. (a) Consider the setup of Example 14.1(a). We run the GFAS first for the distance defined in Eq. (14.29), when (i) A is the identity 2×2 matrix, and (ii) $A = \begin{bmatrix} 2 & 1.5 \\ 1.5 & 2 \end{bmatrix}$, and (iii) the Minkowski distance with $p = 4$ is used. The algorithm is initialized as in the example 14.1, with $\boldsymbol{\theta}_j$ in the place of $\boldsymbol{\mu}_j$. The fuzzifier q was set equal to 2.

The estimates for $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, and $\boldsymbol{\theta}_3$ are $\boldsymbol{\theta}_1 = [1.37, 0.71]^T$, $\boldsymbol{\theta}_2 = [3.14, 3.12]^T$, and $\boldsymbol{\theta}_3 = [5.08, 1.21]^T$ for case (i), $\boldsymbol{\theta}_1 = [1.47, 0.56]^T$, $\boldsymbol{\theta}_2 = [3.54, 1.97]^T$, and $\boldsymbol{\theta}_3 = [5.21, 2.97]^T$ for case (ii), and $\boldsymbol{\theta}_1 = [1.13, 0.74]^T$, $\boldsymbol{\theta}_2 = [2.99, 3.16]^T$, and $\boldsymbol{\theta}_3 = [5.21, 3.16]^T$ for case (iii). The corresponding confusion matrices (see Example 14.1) are

$$A_i = \begin{bmatrix} 98 & 2 & 0 \\ 14 & 84 & 2 \\ 11 & 0 & 89 \end{bmatrix}, \quad A_{ii} = \begin{bmatrix} 63 & 11 & 26 \\ 5 & 95 & 0 \\ 39 & 23 & 38 \end{bmatrix}, \quad A_{iii} = \begin{bmatrix} 96 & 0 & 4 \\ 11 & 89 & 0 \\ 13 & 2 & 85 \end{bmatrix}$$

Notice that in the cases of A_i and A_{iii}, almost all vectors from the same distribution are assigned to the same cluster. Note that for the construction of the confusion matrices we took the liberty to assign each point $\boldsymbol{x}_i$ to the cluster, for which the respective u_{ij} has the maximum value.

(b) Let us now consider the setup of Example 14.1(b). We run the GFAS algorithm for the three cases described in (a). The estimates for $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, and $\boldsymbol{\theta}_3$ are $\boldsymbol{\theta}_1 = [1.60, 0.12]^T$, $\boldsymbol{\theta}_2 = [1.15, 1.67]^T$, and $\boldsymbol{\theta}_3 = [3.37, 2.10]^T$ for case (i), $\boldsymbol{\theta}_1 = [1.01, 0.38]^T$, $\boldsymbol{\theta}_2 = [2.25, 1.49]^T$,

$\boldsymbol{\theta}_3 = [3.75, 2.68]^T$ for case (ii), and $\boldsymbol{\theta}_1 = [1.50, -0.13]^T$, $\boldsymbol{\theta}_2 = [1.25, 1.77]^T$, $\boldsymbol{\theta}_3 = [3.54, 1.74]^T$ for case (iii). The corresponding confusion matrices are

$$A_i = \begin{bmatrix} 51 & 46 & 3 \\ 14 & 47 & 39 \\ 43 & 0 & 57 \end{bmatrix}, \quad A_{ii} = \begin{bmatrix} 79 & 21 & 0 \\ 19 & 58 & 23 \\ 28 & 41 & 31 \end{bmatrix}, \quad A_{iii} = \begin{bmatrix} 51 & 3 & 46 \\ 37 & 62 & 1 \\ 11 & 36 & 53 \end{bmatrix}$$

Let us now comment on these results. First, as expected, the closer the clusters are, the worse the performance of all the algorithms. Also, when the distance given in Eq. (14.29) is employed, the choice of the matrix A is critical. For the case of our example, when $A = I$, the GFAS identifies almost perfectly the clusters underlying X, when they are not too close to each other. The same holds true for the Minkowski distance with $p = 4$.

14.3.2 Quadric Surfaces as Representatives

In this section we consider the case of clusters of quadric shape, such as hyperellipsoids and hyperparaboloids. In the sequel, we present four algorithms of this type, out of a large number that have appeared in the literature.

Our first concern is to define the distance between a point and a quadric surface, as Eq. (14.19) demands. The next section is devoted to the definition and the physical explanation of some commonly used distances of this kind.

Distances between a Point and a Quadric Surface

In this section we introduce definitions in addition to those discussed in Chapter 11 concerning the distance between a point and a quadric surface.

We recall that the general equation of a quadric surface, Q, is

$$\boldsymbol{x}^T A \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c = 0 \tag{14.35}$$

where A is an $l \times l$ symmetric matrix, $\boldsymbol{b}$ is an $l \times 1$ vector, c is a scalar, and $\boldsymbol{x} = [x_1, \ldots, x_l]^T$. The A, $\boldsymbol{b}$ and c quantities are the parameters defining Q. For various choices of these quantities we obtain hyperellipses, hyperparabolas, and so on. An alternative to the Eq. (14.35) form is easily verified to be (see Problem 14.8)

$$\boldsymbol{q}^T \boldsymbol{p} = 0 \tag{14.36}$$

where

$$\boldsymbol{q} = \Big[\overbrace{x_1^2, x_2^2, \ldots, x_l^2}^{l}, \overbrace{x_1 x_2, \ldots, x_{l-1} x_l}^{l(l-1)/2}, \overbrace{x_1, x_2, \ldots, x_l, 1}^{l+1} \Big]^T \tag{14.37}$$

and

$$\boldsymbol{p} = [p_1, p_2, \ldots, p_l, p_{l+1}, \ldots, p_r, p_{r+1}, \ldots, p_s]^T \tag{14.38}$$

with $r = l(l+1)/2$ and $s = r + l + 1$. Vector $\boldsymbol{p}$ is easily derived from A, $\boldsymbol{b}$, and c so that Eq. (14.36) is satisfied.

Algebraic Distance

The squared *algebraic distance* between a point $\boldsymbol{x}$ and a quadric surface Q is defined as

$$d_a^2(\boldsymbol{x}, Q) = (\boldsymbol{x}^T A \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c)^2 \tag{14.39}$$

Using the alternative formulation in Eq. (14.36), $d_a^2(\boldsymbol{x}, Q)$ can be written as

$$d_a^2(\boldsymbol{x}, Q) = \boldsymbol{p}^T M \boldsymbol{p} \tag{14.40}$$

where $M = \boldsymbol{q}\boldsymbol{q}^T$. The algebraic distance could be seen as a generalization of the distance of a point from a hyperplane (see Chapter 11). Its physical meaning will become clear later on. For the derivation of the GFAS algorithm, based on the squared algebraic distance, it is more convenient to use the last formulation in (14.40).

Perpendicular Distance

Another distance between a point $\boldsymbol{x}$ and a quadric surface Q is the squared *perpendicular distance* defined as

$$d_p^2(\boldsymbol{x}, Q) = \min_{\boldsymbol{z}} \|\boldsymbol{x} - \boldsymbol{z}\|^2 \tag{14.41}$$

subject to the constraint that

$$\boldsymbol{z}^T A \boldsymbol{z} + \boldsymbol{b}^T \boldsymbol{z} + c = 0 \tag{14.42}$$

In words, this definition states that the distance between $\boldsymbol{x}$ and Q is defined as the squared Euclidean distance between $\boldsymbol{x}$ and the point $\boldsymbol{z}$ of Q closest to $\boldsymbol{x}$. $d_p(\boldsymbol{x}, Q)$ is the length of the perpendicular line segment from $\boldsymbol{x}$ to Q. Although this definition seems to be the most reasonable one from an intuitive point of view, its computation is not straightforward. More precisely, it involves

Lagrangian formalization. Specifically, we define

$$\mathcal{D}(\boldsymbol{x}, Q) = \|\boldsymbol{x} - z\|^2 - \lambda(z^T A z + \boldsymbol{b}^T z + c) \tag{14.43}$$

Taking into account that A is symmetric, the gradient of $\mathcal{D}(\boldsymbol{x}, Q)$ with respect to z is

$$\frac{\partial \mathcal{D}(\boldsymbol{x}, Q)}{\partial z} = 2(\boldsymbol{x} - z) - 2\lambda A z - \lambda \boldsymbol{b}$$

Setting $\partial \mathcal{D}(\boldsymbol{x}, Q)/\partial z$ equal to $\mathbf{0}$ and after some algebra, we obtain

$$z = \frac{1}{2}(I + \lambda A)^{-1}(2\boldsymbol{x} - \lambda \boldsymbol{b}) \tag{14.44}$$

To compute λ, we substitute z in Eq. (14.42) and we obtain a polynomial of λ of degree $2l$. For each of the real roots, λ_k, of this polynomial, we determine the corresponding z_k. Then, $d_p(\boldsymbol{x}, Q)$ is defined as

$$d_p^2(\boldsymbol{x}, Q) = \min_{z_k} \|\boldsymbol{x} - z_k\|^2$$

Radial Distance

This distance is suitable when Q is a hyperellipsoidal. Then Eq. (14.35) can be brought into the form

$$(\boldsymbol{x} - \boldsymbol{c})^T A(\boldsymbol{x} - \boldsymbol{c}) = 1 \tag{14.45}$$

where $\boldsymbol{c}$ is the center of the ellipse and A is a symmetric positive definite matrix,[6] which determines the major and the minor axis of the ellipse as well as its orientation.

The squared *radial distance* [Frig 96] between a point $\boldsymbol{x}$ and Q is defined as

$$d_r^2(\boldsymbol{x}, Q) = \|\boldsymbol{x} - z\|^2 \tag{14.46}$$

subject to the constraints

$$(z - \boldsymbol{c})^T A(z - \boldsymbol{c}) = 1 \tag{14.47}$$

and

$$(z - \boldsymbol{c}) = a(\boldsymbol{x} - \boldsymbol{c}) \tag{14.48}$$

In words, we first determine the intersection point, z, between the line segment $\boldsymbol{x} - \boldsymbol{c}$ and Q and then we compute $d_r(\boldsymbol{x}, Q)$ as the squared Euclidean distance between $\boldsymbol{x}$ and z (see Figure 14.8).

[6] Obviously, this matrix is in general different (yet related) from the A matrix used in Eq. (14.35). We use the same symbol for notational convenience.

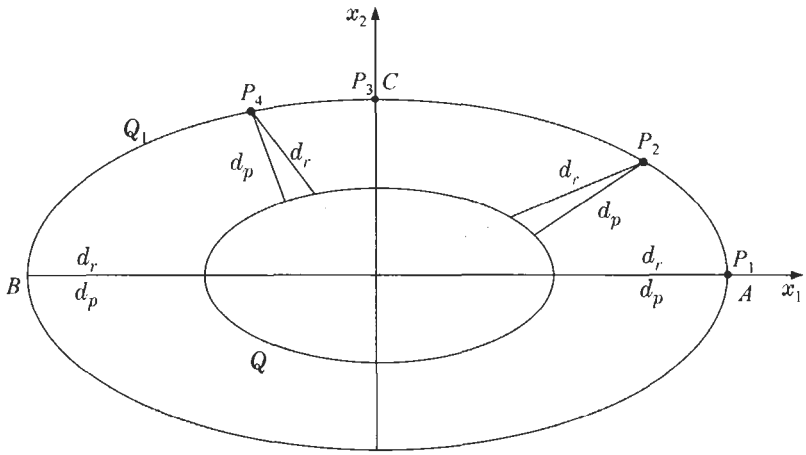


FIGURE 14.8: Graphical representation of the perpendicular and radial distances.

Normalized Radial Distance

The squared *normalized radial distance* between $\boldsymbol{x}$ and Q is also appropriate for hyperellipsoids and is defined as

$$d_{nr}^2(\boldsymbol{x}, Q) = \left(\left((\boldsymbol{x} - \boldsymbol{c})^T A (\boldsymbol{x} - \boldsymbol{c}) \right)^{1/2} - 1 \right)^2 \tag{14.49}$$

It can be shown (Problem 14.10) that

$$d_r^2(\boldsymbol{x}, Q) = d_{nr}^2(\boldsymbol{x}, Q) \|z - \boldsymbol{c}\|^2 \tag{14.50}$$

where z is the intersection of the line segment $\boldsymbol{x} - \boldsymbol{c}$ with Q. This justifies the term "normalized."

The following examples give some insight into the distances that have been defined.

Example 14.6. Consider an ellipse Q centered at $\boldsymbol{c} = [0, 0]^T$, with

$$A = \begin{bmatrix} 0.25 & 0 \\ 0 & 1 \end{bmatrix}$$

and an ellipse Q_1 centered at $\boldsymbol{c} = [0, 0]^T$, with

$$A_1 = \begin{bmatrix} 1/16 & 0 \\ 0 & 1/4 \end{bmatrix}$$

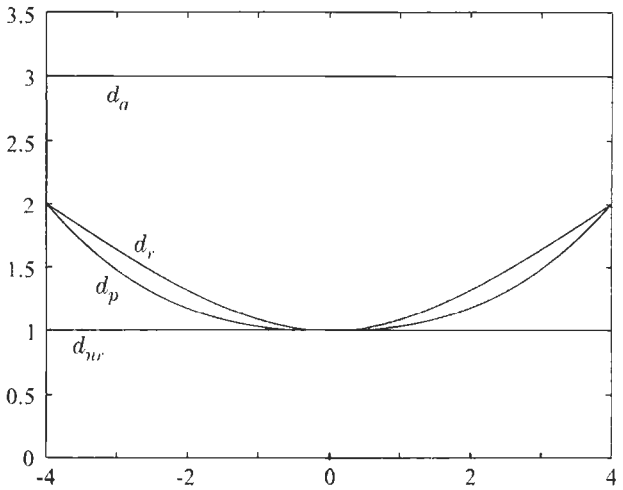


FIGURE 14.9: Variation of the distances d_p, d_a, d_{nr}, and d_r as P moves from $A(4, 0)$ to $B(-4, 0)$, with its x_2 coordinate being positive. The horizontal axis corresponds to the x_1 coordinate of the various points considered.

Let $P(x_1, x_2)$ be a point in Q_1 moving from $A(4, 0)$ to $B(-4, 0)$ and always having its x_2 coordinate positive (Figure 14.8). Figure 14.9 illustrates how the four distances vary as P moves from A to B. One can easily observe that d_a and d_{nr} do not vary as P moves. This means that all points lying on an ellipse sharing the same center as Q and, having the same orientation as it, have the same d_a and d_{nr} distances from Q. However, this is not the case with the other two distances. Figure 14.8 shows graphically the d_p and d_r distances for various instances of P. As expected, the closer P is to the point $C(2, 0)$, the smaller the d_p and d_r distances. Also, Figure 14.8 indicates that d_r can be used as an approximation of d_p, since, as we saw earlier, it is hard to compute d_p. However, it should be recalled that d_p is applicable when general quadric surfaces are considered, whereas d_r is used only when hyperellipsoids are considered.

Example 14.7. Consider the two ellipses Q_1 and Q_2 shown in Figure 14.10, with equations

$$(\boldsymbol{x} - \boldsymbol{c}_j)^T A_j (\boldsymbol{x} - \boldsymbol{c}_j) = 1, \quad j = 1, 2$$

where $\boldsymbol{c}_1 = [0, 0]^T$, $\boldsymbol{c}_2 = [8, 0]^T$ and

$$A_1 = \begin{bmatrix} 1/16 & 0 \\ 0 & 1/4 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} 1/4 & 0 \\ 0 & 1 \end{bmatrix}$$

Also consider the points $A(5, 0)$, $B(3, 0)$, $C(0, 2)$, and $D(5.25, 1.45)$. The distances d_a, d_p, d_{nr}, d_r between each of these points and Q_1 and Q_2 are shown in Table 14.2. From this

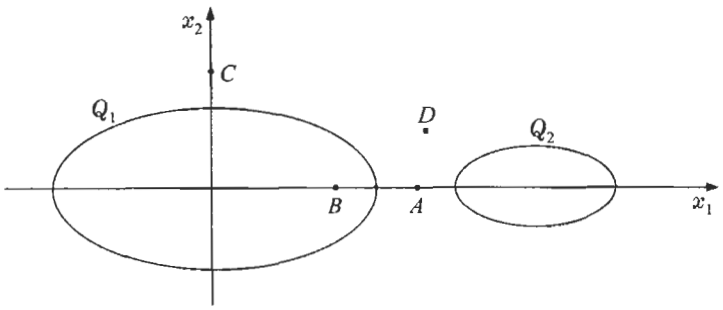


FIGURE 14.10: The setup of Example 14.7.

Table 14.2: Comparison of the Various Distances between Points and Hyperellipsoids

	d_a		d_p		d_{nr}		d_r	
	Q_1	Q_2	Q_1	Q_2	Q_1	Q_2	Q_1	Q_2
A	0.32	1.56	1	1	0.06	0.25	1	1
B	0.19	27.56	1	9	0.06	2.25	1	9
C	1.56	576	1	44.32	0.25	16	1	46.72
D	1.56	9.00	2.78	1.93	0.25	1	3.30	2.42

table we observe that the d_p distances from A, B, and C to Q_1 are equal. Moreover, as expected, the d_r distance is always greater than or equal to d_p (when the equality holds?). Also, d_p is unbiased toward the size of the ellipses ($d_p(A, Q_1) = d_p(A, Q_2)$). Finally, d_a and d_{nr} are biased toward larger ellipses ($d_a(A, Q_1) < d_a(A, Q_2)$ and $d_{nr}(A, Q_1) < d_{nr}(A, Q_2)$).

In the sequel, we derive some well-known algorithms suitable for shell-shaped clusters. These algorithms are usually called *fuzzy shell clustering algorithms*, and the representatives of the clusters are (in most cases) hyperquadrics.

Fuzzy Shell Clustering Algorithms

The first two algorithms that are examined are suitable for hyperellipsoid-shaped clusters. The first of them [Dave 92a], [Dave 92b] is called the *adaptive fuzzy C-shells (AFCS) clustering algorithm* and the second one is known as the *fuzzy C ellipsoidal shells (FCES) algorithm* [Kris 95a].

The Adaptive Fuzzy C-Shells (AFCS) algorithm

The AFCS uses the squared distance d_{nr} between a point and a hyperellipsoidal (Eq. 14.49). Thus, Eq. (14.19) becomes

$$J_{nr}(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^{q} d_{nr}^{2}(\boldsymbol{x}_i, Q_j) \tag{14.51}$$

It is clear that in this case the parameters used to identify a representative (an ellipse) are its center, $\boldsymbol{c}_j$, and the symmetric, positive definite matrix, A_j. Thus, the parameter vector $\boldsymbol{\theta}_j$ of the j-th cluster contains the l parameters of $\boldsymbol{c}_j$ plus the $l(l+1)/2$ independent parameters of A_j, $j = 1, \ldots, m$. In the sequel, we write $d_{nr}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ instead of $d_{nr}(\boldsymbol{x}_i, Q_j)$, in order to show explicitly the dependence on the parameter vector.

As in the case of point representatives, our first step is the computation of the gradient of $d_{nr}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ with respect to $\boldsymbol{c}_j$ and A_j. The gradient $\partial d_{nr}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)/\partial \boldsymbol{c}_j$ after some algebra becomes

$$\frac{\partial d_{nr}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{c}_j} = -2 \frac{d_{nr}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\phi(\boldsymbol{x}_i, \boldsymbol{\theta}_j)} A_j(\boldsymbol{x}_i - \boldsymbol{c}_j) \tag{14.52}$$

where

$$\phi^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = (\boldsymbol{x}_i - \boldsymbol{c}_j)^T A_j (\boldsymbol{x}_i - \boldsymbol{c}_j) \tag{14.53}$$

Let a_{rs}^j be the (r, s) element of A_j and x_{ir}, c_{jr} the rth coordinates of $\boldsymbol{x}_i$ and $\boldsymbol{c}_j$, respectively. Then, the partial derivative of $d_{nr}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ with respect to a_{rs}^j, after some elementary manipulations, becomes

$$\frac{\partial d_{nr}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial a_{rs}^j} = \frac{d_{nr}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\phi(\boldsymbol{x}_i, \boldsymbol{\theta}_j)} (x_{ir} - c_{jr})(x_{is} - c_{js})$$

Thus,

$$\frac{\partial d_{nr}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial A_j} = \frac{d_{nr}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\phi(\boldsymbol{x}_i, \boldsymbol{\theta}_j)} (\boldsymbol{x}_i - \boldsymbol{c}_j)(\boldsymbol{x}_i - \boldsymbol{c}_j)^T \tag{14.54}$$

Substituting Eqs. (14.52) and (14.54) in (14.28), and after some minor manipulations, the parameter updating part of GFAS becomes

- Parameter updating:

  —Solve with respect to $\boldsymbol{c}_j$ and A_j the following equations.

$$\sum_{i=1}^{N} u_{ij}^{q}(t-1)\frac{d_{nr}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\phi(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}(\boldsymbol{x}_i - \boldsymbol{c}_j) = \boldsymbol{0}$$

  and

$$\sum_{i=1}^{N} u_{ij}^{q}(t-1)\frac{d_{nr}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\phi(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}(\boldsymbol{x}_i - \boldsymbol{c}_j)(\boldsymbol{x}_i - \boldsymbol{c}_j)^T = O$$

  where

$$\phi^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = (\boldsymbol{x}_i - \boldsymbol{c}_j)^T A_j (\boldsymbol{x}_i - \boldsymbol{c}_j)$$

  and

$$d_{nr}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = (\phi(\boldsymbol{x}_i, \boldsymbol{\theta}_j) - 1)^2$$

  —Set $\boldsymbol{c}_j(t)$ and $A_j(t)$, $j = 1, \ldots, m$, equal to the resulting solutions.

- End parameter updating.

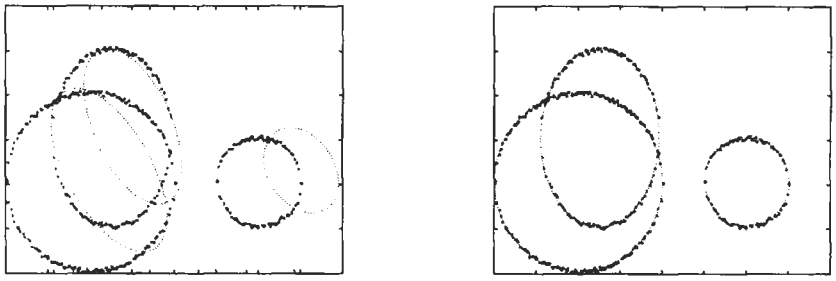
Once more, the above system of equations can be solved by employing iterative techniques.

Variants of the algorithm, imposing certain constraints on A_j, $j = 1, \ldots, m$, have also been proposed in [Dave 92a] and [Dave 92b].

Example 14.8. Consider the three ellipses in Figure 14.11a, with centers $\boldsymbol{c}_1 = [0, 0]^T$, $\boldsymbol{c}_2 = [8, 0]^T$, and $\boldsymbol{c}_3 = [1, 1]^T$, respectively. The corresponding matrices that specify their major and minor axes, as well as their orientation, are

$$A_1 = \begin{bmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}, \qquad A_2 = \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & 1 \end{bmatrix}, \qquad A_3 = \begin{bmatrix} \frac{1}{8} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}$$

respectively. We generate 100 points, $\boldsymbol{x}_i$, from each ellipse and we add to each of these points a random vector whose coordinates stem from the uniform distribution in $[-0.5, 0.5]$. The initial values for the $\boldsymbol{c}_i$'s and the A_i's, $i = 1, 2, 3$, are $\boldsymbol{c}_i(0) = \boldsymbol{c}_i + \boldsymbol{z}$, $i = 1, 2, 3$, with $\boldsymbol{z}$ taken to be $\boldsymbol{z} = [0.3, 0.3]^T$ and $A_i(0) = A_i + Z$, $i = 1, 2, 3$, where all the elements of Z are equal to 0.2. The fuzzifier q was also set equal to 2. Application of the AFCS algorithm to this data set gives, after four iterations, the results shown in Figure 14.11b. Thus, the algorithm has identified the ellipses to a good approximation.



(a) (b)

FIGURE 14.11: The setup of Example 14.8. Thick dots represent the points of the data set. Thin dots represent (a) the initial estimates and (b) the final estimates of the ellipses.

The Fuzzy C Ellipsoidal Shells (FCES) Algorithm

This algorithm uses the squared radial distance between a point and a hyperellipsoidal. Equation (14.19) now becomes

$$J_r(\boldsymbol{\theta}, U) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^q d_r^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j) \tag{14.55}$$

Defining the $\boldsymbol{\theta}_j$'s as in the previous case and carrying out the steps followed for the derivation of the AFCS, we end up with the following equations for $\boldsymbol{c}_j$ and A_j (see Problem 14.11):

$$\sum_{i=1}^{N} u_{ij}^q(t-1)\left[\frac{\|\boldsymbol{x}_i-\boldsymbol{c}_j\|^2(1-\phi(\boldsymbol{x}_i,\boldsymbol{\theta}_j))}{\phi^4(\boldsymbol{x}_i,\boldsymbol{\theta}_j)}A_j-\left(1-\frac{1}{\phi(\boldsymbol{x}_i,\boldsymbol{\theta}_j)}\right)^2 I\right](\boldsymbol{x}_i-\boldsymbol{c}_j)=\mathbf{0} \tag{14.56}$$

and

$$\sum_{i=1}^{N} u_{ij}^q(t-1)\frac{\phi(\boldsymbol{x}_i,\boldsymbol{\theta}_j)-1}{\phi^4(\boldsymbol{x}_i,\boldsymbol{\theta}_j)}\|\boldsymbol{x}_i-\boldsymbol{c}_j\|^2(\boldsymbol{x}_i-\boldsymbol{c}_j)(\boldsymbol{x}_i-\boldsymbol{c}_j)^T = O \tag{14.57}$$

where $\phi(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ is defined as in the case of the AFCS algorithm.

The following two algorithms are proposed in [Kris 95a] and [Frig 96]. In contrast to the previous algorithms, they may fit quadrics of any shape to the data set. They are called the *fuzzy C quadric shells (FCQS) algorithm* and *modified fuzzy C quadric shells (MFCQS) algorithm*, respectively.

Fuzzy C Quadric Shells (FCQS) Algorithm

The FCQS algorithm is suitable for recovering general hyperquadric shapes. It uses the squared algebraic distance. Equation (14.19) now becomes

$$J_a(\boldsymbol{\theta}, U) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^q d_a^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^q \boldsymbol{p}_j{}^T M_i \boldsymbol{p}_j \tag{14.58}$$

where $\boldsymbol{p}_j$ is defined in Eq. (14.38) and $M_i = \boldsymbol{q}_i \boldsymbol{q}_i{}^T$, with $\boldsymbol{q}_i$ defined in Eq. (14.37).

We recall that $\boldsymbol{p}_j$ incorporates all the parameters of the jth quadric surface (see Eq. 14.40), that is, $\boldsymbol{\theta}_j = \boldsymbol{p}_j$. Direct minimization of $J_a(\boldsymbol{\theta}, U)$ with respect to $\boldsymbol{p}_j$ would lead to the trivial zero solution for $\boldsymbol{p}_j$. Thus, constraints on $\boldsymbol{p}_j$ must be imposed, and a number of those have been proposed in the literature. Different constraints lead to different algorithms. Examples of such constraints are [Kris 95a] (i) $\|\boldsymbol{p}_j\|^2 = 1$, (ii) $\sum_{k=1}^{r+l} p_{jk}^2 = 1$, (iii) $p_{j1} = 1$, (iv) $p_{js}^2 = 1$, and (v) $\| \sum_{k=1}^{l} p_{jk}^2 + 0.5 \sum_{k=l+1}^{r} p_{jk}^2 \|^2 = 1$. Each of these constraints has its advantages and disadvantages. For example, constraints (i) and (ii) [Gnan 77],[Pato 70] do not preserve the invariance under translation and rotation of d_a. However, they are able to identify planar clusters. Also, constraint (iii) [Chen 89] precludes linear clusters and can lead to poor results if the points in X are approximately coplanar.

In the Appendix at the end of the chapter, the algorithm under constraint (v) is derived.

Modified Fuzzy C Quadric Shells (MFCQS) Algorithm

A different C-shells quadric algorithm is obtained if we employ the squared perpendicular distance d_p between a point and a quadric surface. However, because of the difficulty of its estimation, the resulting problem is much more difficult than those examined before. In this case, due to the complex nature of d_p, minimization of the $J_p(\boldsymbol{\theta}, U)$ with respect to the parameter vector $\boldsymbol{\theta}_j$ becomes very complex [Kris 95a].

One way to simplify things is to use the following alternative scheme. For the computation of u_{ij}'s the perpendicular distance d_p is used, and for the estimation of the parameters $\boldsymbol{\theta}_j$, $j = 1, \ldots, m$, the updating scheme of FCQS is employed (recall that in FCQS, $\boldsymbol{\theta}_i = \boldsymbol{p}_i$). In other words, the grade of membership of a vector $\boldsymbol{x}_i$ in a cluster is determined using the perpendicular distance, and the updating of the parameters of the representatives is carried out using the parameter updating part of the FCQS algorithm. However, this simplification implies that the algebraic and the perpendicular distances should be close to each other (see also Problem 14.13). This modification leads to the so called modified FCQS (MFCQS) algorithm.

Another algorithm, discussed in [Kris 95a] and [Frig 96], is the fuzzy C plano-quadric shells (FCPQS) algorithm. This algorithm uses a first-order approximation of the algebraic distance and it is derived, as are all the others, by taking derivatives of the resulting cost function with respect to the parameter vector, $\boldsymbol{\theta}_j$, and setting them equal to zero.

Finally, fuzzy clustering algorithms that are able to detect spherical clusters are discussed in [Dave 92a], [Kris 92a], [Kris 92b],[Man 94]. However, most of these may be viewed as special cases of the algorithms developed to fit ellipses.

14.3.3 Hyperplane Representatives

In this section, we discuss algorithms that are suitable for the recovery of hyperplanar clusters [Kris 92a]. Algorithms of this kind can be applied to the surface-fitting problem, which is one of the most important tasks in computer vision. In this problem, the surfaces of an object depicted in the image are approximated by planar surfaces. Successful identification of the surfaces is a prerequisite for the identification of the objects depicted in an image.

Some of these algorithms, such as the fuzzy c-varieties (FCV) algorithm [Ande 85], are based on the minimization of the distances of the vectors in X from hyperplanes (see Chapter 11). However, FCV tends to recover very long clusters and, thus, collinear distinct clusters may be merged into a single cluster.

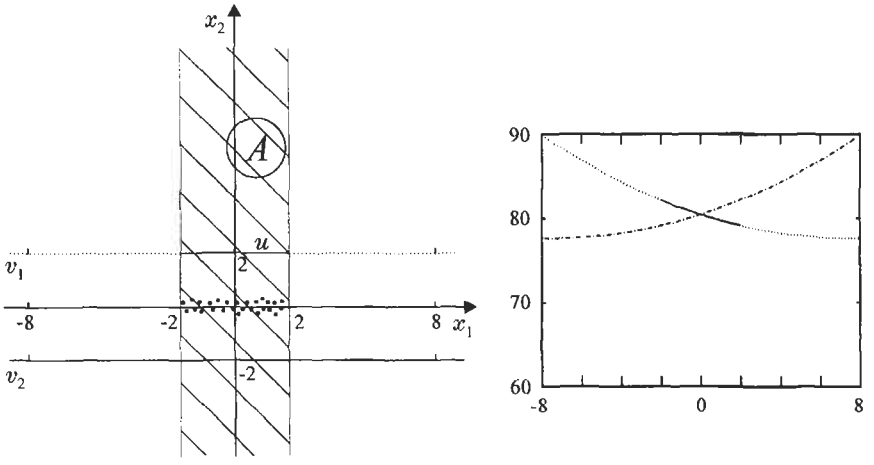
In this section we describe an algorithm, known as the Gustafson–Kessel (G-K) algorithm (see, e.g., [Kris 92a]). According to this algorithm, planar clusters are represented by centers $\boldsymbol{c}_j$ and covariance matrices Σ_j. Defining $\boldsymbol{\theta}_j$ as in previous cases, we define the squared distance between a vector $\boldsymbol{x}$ and the jth cluster as the scaled Mahalanobis distance

$$d_{GK}^2(\boldsymbol{x}, \boldsymbol{\theta}_j) = |\Sigma_j|^{1/l}(\boldsymbol{x} - \boldsymbol{c}_j)^T \Sigma_j^{-1}(\boldsymbol{x} - \boldsymbol{c}_j) \tag{14.59}$$

Let us now gain some insight into the behavior of this distance. A well-known property that characterizes the distance d_H of a point from a hyperplane, as defined in Chapter 11, is that all points lying on a hyperplane H_1 parallel to a given hyperplane H, have the same d_H distance from H. This will be our starting point for the investigation of d_{GK}^2.

Example 14.9. Consider the setup of Figure 14.12a, where a single cluster C is present, and let $\boldsymbol{\theta}$ be its parameter vector. The points of C are of the form $[x_{i1}, x_{i2}]^T$ where $x_{i1} = -2 + 0.1i$, $i = 0, 1, 2, \ldots, 40$, and the corresponding x_{i2}'s are random numbers following the uniform distribution in $[-0.1, 0.1]$.

Consider also the points of the line segment u connecting the points $(-2, 2)$ and $(2, 2)$. Figure 14.12b shows the distances $d_{GK}(\boldsymbol{x}, \boldsymbol{\theta})$ of the points $\boldsymbol{x} \in u$ from C. As we can see, all these distances are almost the same. Indeed, the relative difference $(d_{max} - d_{min})/d_{max}$ between the maximum d_{max} and the minimum d_{min} values is approximately equal to 0.02.



(a) (b)

FIGURE 14.12: (a) The setup of the Example 14.9. (b) The solid line corresponds to the distances of the points of u from C. The dashed line corresponds to the distances of the points of the line segment v_1 from C (the solid line is part of the dashed line). Also, the dash-dotted line corresponds to the distances of the points of the line segment v_2 from C.

Now consider the larger line segment $v_1(v_2)$ that connects $(-8, 2)((-8, -2))$ and $(8, 2)((8, -2))$. The distances $d_{GK}(\boldsymbol{x}, \boldsymbol{\theta})$ of the points $\boldsymbol{x} \in v_1(v_2)$ from C are shown in Figure 14.12b. Note that although we have larger variations compared to the previous case, they still remain relatively small (the relative difference $(d_{max} - d_{min})/d_{max}$ is approximately 0.12).

The G-K algorithm can be derived via the minimization of

$$J_{GK}(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^{q} d_{GK}^{2}(\boldsymbol{x}_i, \boldsymbol{\theta}_j) \tag{14.60}$$

Taking the gradient of $J_{GK}(\boldsymbol{\theta}, U)$ with respect to $\boldsymbol{c}_j$, we obtain

$$\frac{\partial J_{GK}(\boldsymbol{\theta}, U)}{\partial \boldsymbol{c}_j} = \sum_{i=1}^{N} u_{ij}^{q} \frac{\partial d_{GK}^{2}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{c}_j} \tag{14.61}$$

The gradient of the distance, after a bit of algebra, becomes

$$\frac{\partial d_{GK}^{2}(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{c}_j} = -2|\Sigma_j|^{1/l} \Sigma_j^{-1}(\boldsymbol{x}_i - \boldsymbol{c}_j) \tag{14.62}$$

Substituting $\partial d^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)/\partial \boldsymbol{c}_j$ from Eq. (14.62) into (14.61) and setting $\partial J_{GK}(\boldsymbol{\theta}, U)/\partial \boldsymbol{c}_j$ equal to zero, we obtain[7]

$$\boldsymbol{c}_j = \frac{\sum_{i=1}^{N} u_{ij}^q \boldsymbol{x}_i}{\sum_{i=1}^{N} u_{ij}^q} \tag{14.63}$$

Now taking the derivative of $J_{GK}(\boldsymbol{\theta}, U)$ with respect to the elements of the covariance matrix, Σ_j results (Appendix of this chapter) in

$$\Sigma_j = \frac{\sum_{i=1}^{N} u_{ij}^q (\boldsymbol{x}_i - \boldsymbol{c}_j)(\boldsymbol{x}_i - \boldsymbol{c}_j)^T}{\sum_{i=1}^{N} u_{ij}^q} \tag{14.64}$$

Having derived Eqs. (14.63) and (14.64), the parameter updating part of GFAS for the G-K algorithm becomes

- $\boldsymbol{c}_j(t) = \dfrac{\sum_{i=1}^{N} u_{ij}^q(t-1)\boldsymbol{x}_i}{\sum_{i=1}^{N} u_{ij}^q(t-1)}$
- $\Sigma_j(t) = \dfrac{\sum_{i=1}^{N} u_{ij}^q(t-1)\big(\boldsymbol{x}_i - \boldsymbol{c}_j(t)\big)\big(\boldsymbol{x}_i - \boldsymbol{c}_j(t)\big)^T}{\sum_{i=1}^{N} u_{ij}^q(t-1)}$

Example 14.10. (a) Consider Figure 14.13a. It consists of three linear clusters. Each cluster contains 41 points. The points of the first cluster lie around the line $x_2 = x_1 + 1$, while the points of the second and the third clusters lie around the lines $x_2 = 0$ and $x_2 = -x_1 + 1$, respectively. The $\boldsymbol{c}_j$'s, $j = 1, 2, 3$, are randomly initialized and the threshold of the termination criterion, ε, is set to 0.01. The G-K algorithm converges after 32 iterations. As shown in Figure 14.13b, the G-K identifies correctly the clusters underlying X.

(b) Now consider Figure 14.14a. We also have three clusters, each consisting of 41 points. The first and the third clusters are the same as in Figure 14.13a, while the points of the second cluster lie around the line $x_2 = 0.5$. Note that in this case the three intersection points between any pair of lines lie very close to each other. The G-K algorithm terminates after 42 iterations. The results obtained are shown in Figure 14.14b. In this case, the G-K algorithm fails to identify the clusters correctly.

14.3.4 Combining Quadric and Hyperplane Representatives

In this section, we assume that $l = 2$. Consider the case in which X contains quadric-shaped clusters as well as linear clusters. How can we accurately identify

[7] We also make the mild assumption that the covariance matrix is invertible.

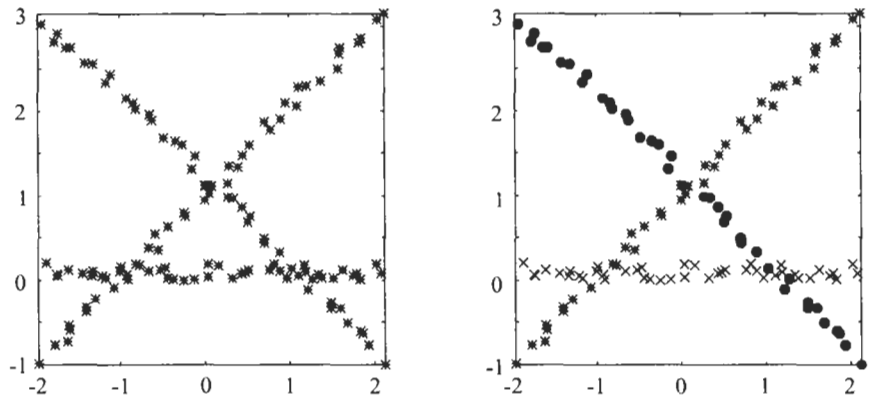


FIGURE 14.13: (a) The data set X for Example 14.10(a). (b) The results of the G-K algorithm.

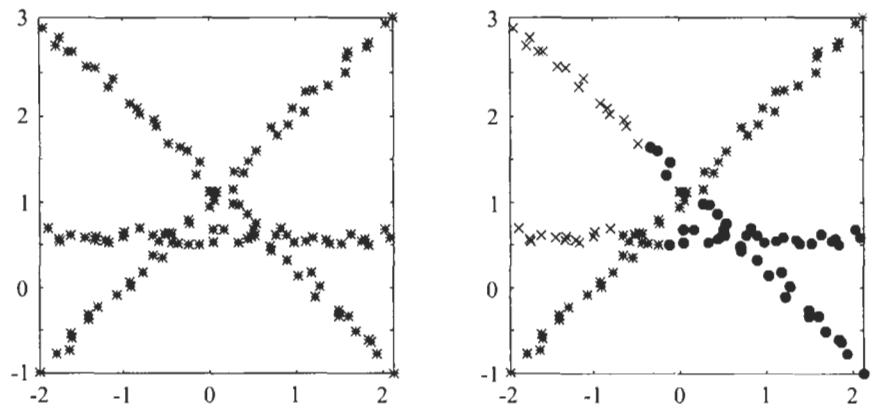


FIGURE 14.14: (a) The data set X for Example 14.10(b). (b) The results of the G-K algorithm.

both kinds of clusters? If we run an algorithm that fits quadric curves to the clusters, the linear clusters will not be properly represented. On the other hand, if we run an algorithm that fits lines to the clusters, the ellipsoidally and hyperbolically shaped clusters will be poorly represented. A way out of this problem is discussed in [Kris 95a]. The idea is to run the FCQS algorithm first on the whole data set X.

This algorithm can be used to detect linear clusters, even though the adopted constraints forces all representatives to be of second degree. This happens since, in practice, FCQS fits a pair of coincident lines for a single line, a hyperbola for two intersecting lines and a very "flat" hyperbola or a very elongated ellipse or a pair of lines for two parallel lines [Kris 95a]. The identification of "extreme" quadric curves (i.e., extremely elongated ellipses, "flat" hyperbolas, a set of lines) after the termination of the algorithm is a strong indication that X contains linear clusters. In order to represent these extreme clusters more accurately, we can run the G-K algorithm on the set X', which contains only the vectors that belong to them (with a high grade of membership). However, different actions have to be carried out depending on the shape of each extreme quadric curve. Let Q_j be the representative curve of the jth cluster, $j = 1, \ldots, m$, identified by FCQS and Q'_j's the representative curves of the linear clusters that are identified by the G-K algorithm. Specifically, we have

- If Q_j is a pair of coincident lines, then initialize Q'_j using one of the two lines.
- If Q_j is a nonflat hyperbola *or* a pair of intersecting lines OR a pair of parallel lines, then initialize two representatives Q'_{j1} and Q'_{j2} using the asymptotes of the hyperbola (for the first case) or using each of the lines (for the last two cases).
- If Q_j is an ellipse with a very large ratio of major to minor axis, then initialize two representatives Q'_{j1} and Q'_{j2} using the tangents of the ellipse at the two ends of the minor axis.
- If Q_j is a hyperbola with a very large ratio of conjugate axis to transverse axis, then initialize two representatives Q'_{j1} and Q'_{j2} using the two tangents of the hyperbola at its two vertices

Since the initialization of the Q'_j representatives is very good, it is expected that the G-K algorithm will converge in a few iterations to a satisfactory solution.

14.3.5 A Geometrical Interpretation

Arguments similar to those given in Section 14.2.2 can also be repeated here. Now u_{ij} takes the place of $P(C_j|\boldsymbol{x}_i)$. The constraint equation in this case is

$$\sum_{j=1}^{m} u_{ij} = 1, \quad i = 1, \ldots, N. \tag{14.65}$$

The vector $\boldsymbol{y}$ associated with vector $\boldsymbol{x}_i$ becomes $\boldsymbol{y} = [u_{i1}, u_{i2}, \ldots, u_{im}]$ and it is also restricted on the hyperplane defined by the constraint (14.65) in the H_m hypercube. If we carry out the experiments discussed in Section 14.2.2.

we will draw similar conclusions with respect to the effect of the outliers on the performance of the fuzzy algorithms.

In [Mena 00] an algorithm called *fuzzy c+2 means* is introduced. This is an extension of GFAS for point representatives, where the outliers as well as the points that lie near the cluster boundaries are treated so as to control their effect on the estimates of the cluster representatives.

14.3.6 Convergence Aspects of the Fuzzy Clustering Algorithms

Although fuzzy clustering algorithms are obtained by minimizing a cost function of the form of Eq. (14.19), little is known about their convergence behavior. More specifically, it has been proved [Bezd 80], [Hath 86], using the global convergence theorem of Zangwill [Luen 84], that when a Mahalanobis distance is used (or other distances satisfying certain conditions discussed in [Bezd 80]), the iteration sequence produced by the fuzzy c-means (FCM) algorithm either converges to a stationary point of the cost function in a finite number of iteration steps or it has at least one subsequence that converges to a stationary point of the cost function. This point may be a local (or global) optimum or a saddle point. Tests for the identification of the nature of the convergence point are discussed in [Isma 86], [Hath 86], [Kim 88]. Issues concerning numerical convergence aspects of the FCM algorithms are discussed in [Bezd 92].

14.3.7 Alternating Cluster Estimation

It is not difficult to notice that the membership functions $u_j(\boldsymbol{x}_i)$, associated with the u_{ij}'s used in GFAS (Eq. (14.26)), are neither convex nor monotonous (see, for example, Figure 14.15a). However, in fuzzy rule based systems convexity of the membership functions is an important requirement. For example, linguistic characterizations such as "low," "medium," or "high" require convex membership functions of the form shown in Figure 14.15b. In such cases it may be preferable to adopt a specific membership function and use the alternating updating philosophy used in GFAS to estimate u_{ij}'s and $\boldsymbol{\theta}_j$. The resulting algorithmic scheme is known as *Alternating Cluster Estimation (ACE)* ([Runk 99], [Hopp 99]) and GFAS may be viewed as a special case of it. Obviously, in this case, the obtained solution is not necessarily related to an optimizing criterion.

14.4 POSSIBILISTIC CLUSTERING

The algorithms of this section are relaxed from constraints such as in (14.17) and (14.65) [Kris 93], [Kris 96]. Speaking in the terms of Section 14.3.5, this means that the vector $\boldsymbol{y}$, with coordinates the u_{ij}'s, will be allowed to move anywhere in

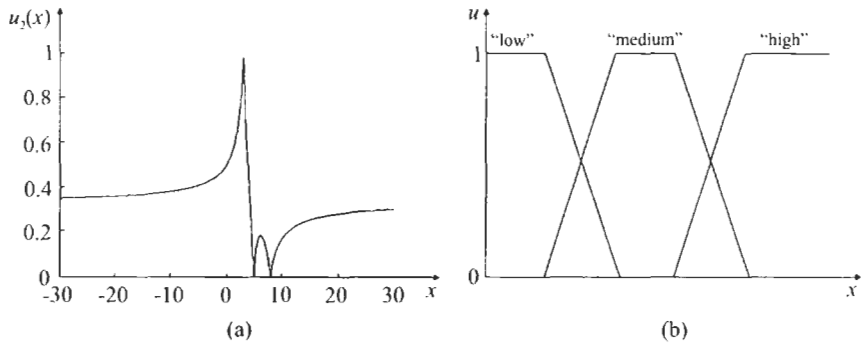


FIGURE 14.15: (a) The membership function $u_2(x)$ given by Eq. (14.26), for the one-dimensional case, with $\theta_1 = 5, \theta_2 = 3, \theta_3 = 8, q = 2$ and $d(x, \theta_i) = |x - \theta_i|$. (b) Examples of membership functions characterizing "low,", "medium," and "high" for a specific quantity.

the H_m hypercube, that is,

$$u_{ij} \in [0, 1]$$
$$\max_{j=1,\ldots,m} u_{ij} > 0, \quad i = 1, \ldots, N$$

and

$$0 < \sum_{i=1}^{N} u_{ij} \leq N, \quad i = 1, \ldots, N \tag{14.66}$$

This change in the constraints has an important impact on the interpretation of the u_{ij}'s. In the fuzzy framework, u_{ij} denotes the grade of membership of $\boldsymbol{x}_i$ in the jth cluster. Here, u_{ij} may be interpreted as the degree of compatibility of $\boldsymbol{x}_i$ with the jth cluster representative, or, following [Zade 78], the possibility that $\boldsymbol{x}_i$ belongs to the jth cluster. Note that *the possibility that $\boldsymbol{x}_i$ belongs to the jth cluster depends exclusively on $\boldsymbol{x}_i$ and the cluster representative of the jth cluster; that is, it is independent of the possibilities that $\boldsymbol{x}_i$ belongs to any other cluster.*

For convenience, let us recall here that the cost function to be minimized is

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^{q} d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) \tag{14.67}$$

Obviously, direct minimization with respect to U will lead to the trivial zero solution. In order to avoid this situation, we must insert an additional term in $J(\boldsymbol{\theta}, U)$.

This term, $f(U)$, will be a function of u_{ij}'s only. Motivated by the discussion in Section 14.2.2, it will be chosen in such a way so that to minimize the effects of outliers. As will become apparent soon, one such choice of $f(U)$ is

$$f(U) = \sum_{j=1}^{m} \eta_j \sum_{i=1}^{N} (1 - u_{ij})^q \tag{14.68}$$

Then, the cost function becomes

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) + \sum_{j=1}^{m} \eta_j \sum_{i=1}^{N} (1 - u_{ij})^q \tag{14.69}$$

where η_j are suitably chosen positive constants.

The minimum of $J(\boldsymbol{\theta}, U)$, with respect to u_{ij}, is obtained by

$$\frac{\partial J(\boldsymbol{\theta}, U)}{\partial u_{ij}} = q u_{ij}^{q-1} d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) - q \eta_j (1 - u_{ij})^{q-1} = 0$$

or

$$u_{ij} = \frac{1}{1 + \left(\frac{d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\eta_j} \right)^{\frac{1}{q-1}}} \tag{14.70}$$

In words, u_{ij} is inversely proportional to the dissimilarity between $\boldsymbol{x}_i$, and the representative of the jth cluster. Loosely speaking, u_{ij} denotes the degree to which the representative of the jth cluster *should be "stretched"* in order to match $\boldsymbol{x}_i$. Large (small) values of u_{ij} indicate little (large) stretch for the jth representative. *It is clear that for a specific vector $\boldsymbol{x}_i$, this "stretching" action can be carried out independently for each cluster.*

The meaning of the second term in Eq. (14.69) is more clear now. Its effect is to minimize the influence of outliers in the estimation of the $\boldsymbol{\theta}_j$'s. Indeed, large dissimilarity levels correspond to small u_{ij}'s and they have little effect on the first term in the cost function, which controls the estimation of $\boldsymbol{\theta}_j$'s.

Since the second term does not involve the representatives of the clusters, one may easily conclude that in possibilistic clustering schemes, the updating of the parameters of each cluster is carried out exactly the same way as in the case of their fuzzy counterparts.

Generalized Possibilistic Algorithmic Scheme (GPAS)

- Fix η_j, $j = 1, \ldots, m$.
- Choose $\boldsymbol{\theta}_j(0)$ as the initial estimates of $\boldsymbol{\theta}_j$, $j = 1, \ldots, m$.
- $t = 0$
- Repeat
  - —For $i = 1$ to N
    - * For $j = 1$ to m

$$u_{ij}(t) = \frac{1}{1 + \left(\frac{d(\boldsymbol{x}_i, \boldsymbol{\theta}_j(t))}{\eta_j}\right)^{\frac{1}{q-1}}}.$$

    - * End {For-j}
  - —End {For-i}
  - —$t = t + 1$
  - —For $j = 1$ to m
    - * *Parameter updating*: Solve

$$\sum_{i=1}^{N} u_{ij}^q(t-1) \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j} = \mathbf{0} \tag{14.71}$$

      with respect to $\boldsymbol{\theta}_j$ and set $\boldsymbol{\theta}_j(t)$ equal to the computed solution.
  - —End {For-j}.
- Until a termination criterion is met.

As usual, we may employ $\|\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-1)\| < \varepsilon$ as a termination criterion. Based on the preceding generalized scheme, for each of the fuzzy clustering algorithms, defined in the previous section, we can derive a corresponding possibilistic one.

An interesting observation is that, since for each vector $\boldsymbol{x}_i$, u_{ij}'s, $j = 1, \ldots, m$, are independent of each other, we can write $J(\boldsymbol{\theta}, U)$ as

$$J(\boldsymbol{\theta}, U) = \sum_{j=1}^{m} J_j$$

where

$$J_j = \sum_{i=1}^{N} u_{ij}^q d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) + \eta_j \sum_{i=1}^{N} (1 - u_{ij})^q \tag{14.72}$$

Each J_j corresponds to a different cluster and the minimization of $J(\boldsymbol{\theta}, U)$ with respect to the u_{ij}'s can be carried out separately for each J_j.

The value of η_j determines the relative significance of the two terms in (14.72) and it is related to the size and the "shape" of the jth cluster, $j = 1, \ldots, m$. More specifically, as can be seen from Figure 14.16, η_j determines the dissimilarity level between a vector $\boldsymbol{x}_i$ and the representative $\boldsymbol{\theta}_j$ at which u_{ij} becomes equal to 0.5. Thus, η_j determines the influence of a specific point on the estimation of the jth cluster representative.

In general, the size of η_j is assumed constant during the execution of the algorithm. One way to estimate its value, under the assumption that X does not contain many outliers, is to run the generalized fuzzy algorithmic scheme (GFAS) and after its convergence to estimate η_j as [Kris 96]

$$\eta_j = \frac{\sum_{i=1}^{N} u_{ij}^q d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\sum_{i=1}^{N} u_{ij}^q} \tag{14.73}$$

or

$$\eta_j = \frac{\sum_{u_{ij} > a} d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\sum_{u_{ij} > a} 1} \tag{14.74}$$

where a is an appropriate threshold. In words, η_j is defined as a weighted average of the dissimilarities between the vectors $\boldsymbol{x}_i$ and $\boldsymbol{\theta}_j$. Once η_j's have been fixed, the GPAS algorithm can be applied.

In Figure 14.16, u_{ij} versus $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)/\eta_j$ is plotted for various choices of q (see Eq. (14.70)). From this diagram, it can be seen that q determines the rate of decrease of u_{ij} with respect to $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$. For $q = 1$, all points $\boldsymbol{x}_i$ with $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) > \eta_j$ have $u_{ij} = 0$. On the other hand, as $q \to +\infty$, u_{ij} tends to a constant and all the vectors of X contribute equally to the estimation of the representative of the jth cluster.

It is worth noting here that q has different meanings in the possibilistic and the fuzzy framework. In the first case, high values of q imply almost equal contributions of *all* feature vectors to *all* clusters, whereas in the second case, high values of q imply increased *sharing* of the vectors among all clusters [Kris 96]. This implies that, in general, different values of q are required to provide satisfactory results for the two cases.

14.4.1 The Mode-Seeking Property

The generalized mixture decomposition algorithmic scheme (GMDAS) and the generalized fuzzy algorithmic scheme (GFAS) are *partition algorithmic schemes*, that is, schemes that always end up with the predetermined number of clusters m, no matter how many "naturally formed" clusters underlie X. If, for example, the

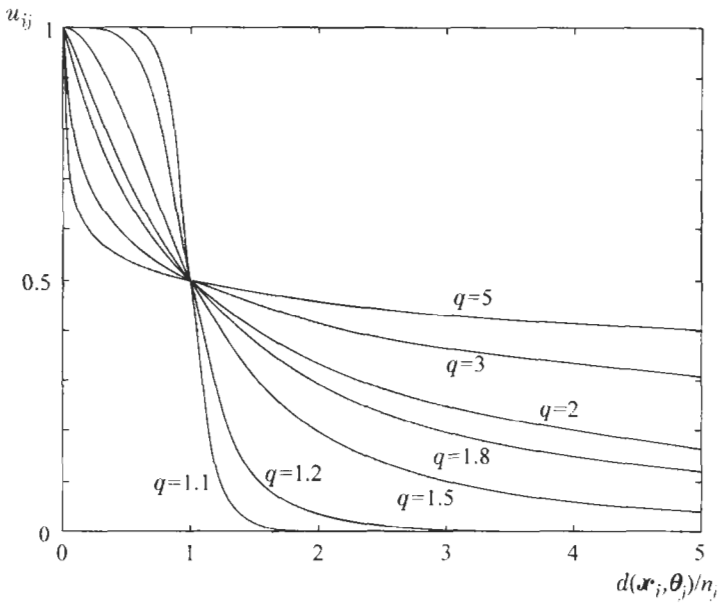


FIGURE 14.16: Plots of the membership function for various values of q.

data set X contains two clusters and we run GMDAS or GFAS with $m = 3$, these algorithms will split at least one natural cluster and will end up with three clusters.

However, this is not the case with the generalized possibilistic algorithmic scheme (GPAS). Algorithms of this kind are known as *mode-seeking algorithms*, that is, algorithms searching for dense regions of vectors in X.[8] In order to see this, let us consider again the individual functions J_j. Solving Eq. (14.70) with respect to $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$, we obtain

$$d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = \eta_j \left(\frac{1 - u_{ij}}{u_{ij}} \right)^{q-1}$$

Substituting $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ from this equation into Eq. (14.72) results in

$$J_j = \eta_j \sum_{i=1}^{N} (1 - u_{ij})^{q-1} \tag{14.75}$$

[8] Such algorithms are also considered in Chapter 15.

For fixed η_j, minimization of J_j requires maximization of u_{ij}'s, which, in turn, requires minimization of $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$. The last requirement implies that $\boldsymbol{\theta}_j$ should be placed in a region dense in vectors of X.

The mode-seeking property of the GPAS implies that the number of clusters in X need not be known a priori. Indeed, if we run a possibilistic algorithm for m clusters while X contains k natural clusters, with $m>k$, then, after proper initialization, *some of the m clusters will coincide with others* [Kris 96]. It is hoped that the number of the noncoincident clusters will be equal to k. If, on the other hand, $m<k$, proper initialization of the possibilistic algorithm will potentially lead to m different clusters. Of course, these are not all the natural clusters formed in X, but at least they are some of them [Kris 96].

Example 14.11. This example demonstrates the mode seeking property. Consider three two-dimensional Gaussian distributions with means $\boldsymbol{\mu}_1 = [1, 1]^T$, $\boldsymbol{\mu}_2 = [6, 1]^T$, $\boldsymbol{\mu}_3 = [6, 6]^T$ and covariance matrices $\Sigma_j = I$, $j = 1, 2, 3$. One hundred vectors are generated from each distribution. These constitute the data set X. We set $q = 1.5$ and, finally, we employ the squared Euclidean distance. It is not difficult to realize that under the above choice, Eq. (14.71) gives

$$\boldsymbol{\theta}_j(t) = \frac{\sum_{i=1}^{N} u_{ij}^q(t-1)\boldsymbol{x}_i}{\sum_{i=1}^{N} u_{ij}^q(t-1)}. \tag{14.76}$$

(a) Let $m = 3$. The initial estimates of $\boldsymbol{\theta}_j$'s (which, in this case, are vectors in the 2-dimensional space) in GPAS are $\boldsymbol{\theta}_j(0) = \boldsymbol{\mu}_j + \boldsymbol{z}_j$, $j = 1, 2, 3$, where the $\boldsymbol{z}_j$'s are two-dimensional vectors whose components are drawn from the uniform distribution in $[-2, 2]$. Also, we set $\eta_j = 1.5$, $j = 1, 2, 3$. Application of the GPAS causes the movement of each one of the $\boldsymbol{\theta}_j$'s towards the mean of each distribution (i.e., towards dense regions). Indeed, the final estimates for $\boldsymbol{\theta}_j$'s obtained after 12 iterations, are $\boldsymbol{\theta}_1 = [0.93, 0.60]^T$, $\boldsymbol{\theta}_2 = [5.88, 1.12]^T$ and $\boldsymbol{\theta}_3 = [6.25, 5.86]^T$, which compare very favorably to $\boldsymbol{\mu}_j$'s.

(b) Let $m = 4$. In this case, $\boldsymbol{\theta}_j$'s, $j = 1, 2, 3$ are initialized as in the previous example, while $\boldsymbol{\theta}_4$ is initialized as $\boldsymbol{\mu}_1 + \boldsymbol{z}_4$. Application of GPAS in this case causes the movement of $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_4$ towards the dense region that corresponds to the first distribution. Also, $\boldsymbol{\theta}_2$ and $\boldsymbol{\theta}_3$ move towards the dense regions that correspond to the second and the third distribution, respectively. The resulting values for $\boldsymbol{\theta}_j$'s, obtained after 12 iterations, are $\boldsymbol{\theta}_1 = [0.93, 0.60]^T$, $\boldsymbol{\theta}_2 = [5.88, 1.12]^T$, $\boldsymbol{\theta}_3 = [6.25, 5.86]^T$ and $\boldsymbol{\theta}_4 = [0.94, 0.60]^T$.

(c) Let $m = 2$. We initialize $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ as in (a). Application of the GPAS algorithm causes the movement of $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ towards the dense regions corresponding to first and the second distribution, respectively. The resulting values for $\boldsymbol{\theta}_j$'s, obtained after 11 iterations, are $\boldsymbol{\theta}_1 = [0.93, 0.60]^T$ and $\boldsymbol{\theta}_2 = [5.88, 1.12]^T$.

14.4.2 An Alternative Possibilistic Scheme

An alternative possibilistic algorithm may be derived from the function [Kris 96]

$$J_1(\boldsymbol{\theta}, U) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij} d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) + \sum_{j=1}^{m} \eta_j \sum_{i=1}^{N} (u_{ij} \ln u_{ij} - u_{ij}) \tag{14.77}$$

Note that q is not involved in the definition of $J_1(\boldsymbol{\theta}, U)$. Also, in this case the second term is negative. Setting the partial derivative of $J_1(\boldsymbol{\theta}, U)$ with respect to u_{ij} equal to 0 and solving for u_{ij}, we obtain the following necessary condition for each u_{ij} to be a minimum of $J_1(\boldsymbol{\theta}, U)$:

$$u_{ij} = \exp\left(- \frac{d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\eta_j} \right) \tag{14.78}$$

Hence, u_{ij} decreases more rapidly with $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ than in the previous case (Eq. 14.70). Let us consider a point $\boldsymbol{x}_i$ and a cluster representative $\boldsymbol{\theta}_j$. For the same distance d, (14.78) leads to smaller values of u_{ij} than those derived from (14.70). This means that increased "stretching" is demanded for the former case. *This is an indication that this algorithmic scheme may be used when the clusters are expected to lie close to each other.*

14.5 HARD CLUSTERING ALGORITHMS

In this section we return to the world where each vector belongs *exclusively* to a single cluster. This is why such schemes are called *hard* or *crisp* clustering algorithms. *It turns out that some of the most well known and widely used clustering algorithms fall into this category.* Our starting point is the assumption that the membership coefficients u_{ij} are either 1 or 0. Moreover, they are 1 for one cluster, C_j, and zero for all the others, C_k, $k \neq j$, that is,

$$u_{ij} \in \{0, 1\}, \qquad j = 1, \ldots, m \tag{14.79}$$

and

$$\sum_{j=1}^{m} u_{ij} = 1 \tag{14.80}$$

This situation may be seen as a special case of the fuzzy algorithmic schemes. However, the cost function

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij} d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) \tag{14.81}$$

is *no longer differentiable with respect to* $\boldsymbol{\theta}_j$. Despite that, the general framework of the generalized fuzzy algorithmic schemes, can be adopted for the special case of hard clustering. Such schemes have been used extensively in practice (e.g., [Duda 73]).

Let us fix $\boldsymbol{\theta}_j$, $j = 1, \ldots, m$. Since for each vector $\boldsymbol{x}_i$ only one u_{ij} is 1 and all the others are 0, it is straightforward to see that $J(\boldsymbol{\theta}, U)$ in Eq. (14.81) is minimized if we assign each $\boldsymbol{x}_i$ to its closest cluster, that is,

$$u_{ij} = \begin{cases} 1, & \text{If } d(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = \min_{k=1,\ldots,m} d(\boldsymbol{x}_i, \boldsymbol{\theta}_k) \\ 0, & \text{otherwise} \end{cases} \qquad i = 1, \ldots, N \tag{14.82}$$

Let us now fix u_{ij}s. Working as in the fuzzy algorithms case, the updating equations of the parameter vectors, $\boldsymbol{\theta}_j$, of the clusters are

$$\sum_{i=1}^{N} u_{ij} \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j} = \mathbf{0}, \qquad j = 1, \ldots, m \tag{14.83}$$

Having derived Eqs. (14.82) and (14.83), we are now in a position to write down the generalized hard clustering algorithmic scheme

Generalized Hard Algorithmic Scheme (GHAS)

- Choose $\boldsymbol{\theta}_j(0)$ as initial estimates for $\boldsymbol{\theta}_j$, $j = 1, \ldots, m$.
- $t = 0$
- Repeat
  - —For $i = 1$ to N
    - For $j = 1$ to m
      - *Determination of the partition:* [9]

$$u_{ij}(t) = \begin{cases} 1, & \text{if } d(\boldsymbol{x}_i, \boldsymbol{\theta}_j(t)) = \min_{k=1,\ldots,m} d(\boldsymbol{x}_i, \boldsymbol{\theta}_k(t)) \\ 0, & \text{otherwise,} \end{cases}$$

    - End {For-j}
  - —End {For-i}
  - —$t = t + 1$
  - —For $j = 1$ to m
    - *Parameter updating*: Solve

$$\sum_{i=1}^{N} u_{ij}(t-1) \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j} = \mathbf{0} \tag{14.84}$$

with respect to $\boldsymbol{\theta}_j$ and set $\boldsymbol{\theta}_j(t)$ equal to the computed solution.

[9] In the case in which two or more minima occur, an arbitrary choice is made.

—End {For $-j$}.

- Until a termination criterion is met.

Note that in the update of each $\boldsymbol{\theta}_j$, only the vectors $\boldsymbol{x}_i$ closest to it (i.e., those $\boldsymbol{x}_i$'s for which $u_{ij}(t-1) = 1$) are used. As usual, the termination criterion $\|\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-1)\| < \varepsilon$ can be used. Alternatively, GHAS may terminate if U remains unchanged for two successive iterations.

Each hard clustering algorithm has its corresponding fuzzy clustering algorithm. As with the fuzzy clustering algorithms, we may obtain hard clustering algorithms when $\boldsymbol{\theta}_j$'s represent points, quadric surfaces, or hyperplanes. The updating equations for the parameter vectors $\boldsymbol{\theta}_j$ in the hard clustering algorithms are obtained from their fuzzy counterparts if we set $q = 1$.

Remarks

- Hard clustering algorithms are not as robust as the fuzzy clustering algorithms when other than point representatives are employed. If, for example, hyperplane representatives are used and the G-K algorithm is adopted, we must have an adequate number of vectors N from all underlying clusters in order to avoid degenerate cases where Σ_j is not invertible [Kris 92a].
- The determination of the partition part of the algorithms optimizes $J(\boldsymbol{\theta}, U)$ with respect to U given a set of representatives $\boldsymbol{\theta}_j$. On the other hand, the parameter updating phase optimizes $J(\boldsymbol{\theta}, U)$ with respect to $\boldsymbol{\theta}$ given a specific partition. Note that this procedure does not necessarily lead to a (local) optimum of $J(\boldsymbol{\theta}, U)$.

14.5.1 The Isodata or k-Means or c-Means Algorithm

This is one of the most popular and well-known clustering algorithms [Duda 73], [Ball 67], [Lloy 82]. It can be viewed as a special case of the generalized hard clustering algorithmic scheme when point representatives are used and the *squared* Euclidean distance is adopted to measure the dissimilarity between vectors $\boldsymbol{x}_i$ and cluster representatives $\boldsymbol{\theta}_j$. Before we state the algorithm explicitly, some further comments may be of interest. For this case Eq. (14.81) becomes

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij} \|\boldsymbol{x}_i - \boldsymbol{\theta}_j\|^2 \tag{14.85}$$

This is nothing but the trace of the within scatter matrix S_w, defined in Chapter 5. That is,

$$J(\boldsymbol{\theta}, U) = \text{trace}\{S_w\} \tag{14.86}$$

For the above choice of distance, Eq. (14.82) gives that $\boldsymbol{\theta}_j$ is the mean vector of the j-th cluster. *Applying the generalized hard algorithmic scheme for this specific choice, it turns out that the algorithm converges to a minimum of the cost function.* In words, the isodata algorithm recovers clusters that are as compact as possible. It must be emphasized however, that this convergence result is not valid for other distances, including the Euclidean distance. For example, when Minkowski distances are used, the algorithm converges but not necessarily to a minimum of the corresponding cost function [Seli 84a].

The Isodata or k-Means or c-Means Algorithm

- Choose arbitrary initial estimates $\boldsymbol{\theta}_j(0)$ for the $\boldsymbol{\theta}_j$'s, $j = 1, \ldots, m$.
- Repeat
  - —For $i = 1$ to N
    - * Determine the closest representative, say $\boldsymbol{\theta}_j$, for $\boldsymbol{x}_i$.
    - * Set $b(i) = j$.
  - —End {For}
  - —For $j = 1$ to m
    - * Parameter updating: Determine $\boldsymbol{\theta}_j$ as the mean of the vectors $\boldsymbol{x}_i \in X$ with $b(i) = j$.
  - —End {For}.
- Until no change in $\boldsymbol{\theta}_j$'s occurs between two successive iterations.

An advantage of this algorithm is its computational simplicity. Also, as with all the algorithms that use point representatives, isodata is suitable for recovering compact clusters. A version of the k-means algorithm, where in each cluster C_i the number of vectors is constrained a priori to be n_i, is proposed in [Ng 00].

Example 14.12. (a) Consider the setup of example 14.1(a). In this case $\boldsymbol{\theta}_j$'s correspond to the $\boldsymbol{\mu}_j$'s. We set $m = 3$ and we initialize randomly $\boldsymbol{\theta}_j$'s. After convergence, the isodata algorithm identifies successfully the underlying clusters in X, as indicated by the corresponding confusion matrix,

$$A = \begin{bmatrix} 94 & 3 & 3 \\ 0 & 100 & 0 \\ 9 & 0 & 91 \end{bmatrix}.$$

The resulting values of $\boldsymbol{\theta}_j$'s are $\boldsymbol{\theta}_1 = [1.19, 1.16]^T, \boldsymbol{\theta}_2 = [3.76, 3.63]^T$ and $\boldsymbol{\theta}_3 = [5.93, 0.55]^T$.

(b) Let us now consider two 2-dimensional Gaussian distributions with means $\boldsymbol{\mu}_1 = [1, 1]^T$ and $\boldsymbol{\mu}_2 = [8, 1]^T$ and covariance matrices $\Sigma_1 = 1.5I$ and $\Sigma_2 = I$, respectively. We generate 300 points from the first distribution and 10 points from the second distribution

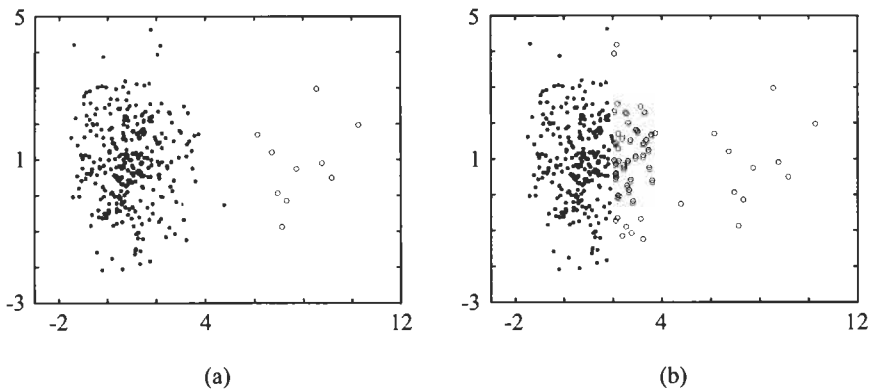


FIGURE 14.17: (a) The data set. (b) The results of the isodata algorithm.

in order to form the data set X (Figure 14.17a). Also, we set $m = 2$ and we initialize randomly $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. After convergence, we observe that the large group has been split into two parts and, in addition, the right part joins the vectors of the second distribution in the same cluster (Figure 14.17b). Specifically, the results of the algorithm are $\boldsymbol{\theta}_1 = [0.54, 0.94]^T$ and $\boldsymbol{\theta}_2 = [3.53, 0.99]^T$ and 61 vectors from the first distribution are assigned to the same cluster with the ten vectors of the second distribution. The above situation reveals a weakness of the algorithm to deal accurately with clusters having significantly different sizes.

In practice, a number of variants of the isodata algorithm have been suggested, employing various splitting, merging and discarding operations among the resulting clusters [Ande 73]. No doubt, such ad-hoc techniques are no more the result of an optimization process.

14.6 VECTOR QUANTIZATION

An area that has close affinity with clustering is that of vector quantization (VQ), and it has been the focus of intensive research effort over the past years (e.g. [Gray 84], [Gers 92]). Vector quantization techniques are used mainly for data compression, which is a prerequisite in order to achieve better computer storage utilization and better bandwidth utilization (in communications). Let T be the set of all possible vectors for the problem at hand. The task of VQ may be stated in the general case in which T is a continuous subset of $\mathcal{R}^l$. The idea is rather simple. We separate T into m distinct regions R_j that exhaust T, and we represent each of them with an l-dimensional vector, the so-called *code vector* or *reproduction vector*, $\boldsymbol{\theta}_j$, $j = 1, \ldots, m$. In the sequel, given an $\boldsymbol{x} \in T$, we determine the region where it belongs, say R_j, and we adopt the corresponding representative $\boldsymbol{\theta}_j$,

instead of $\boldsymbol{x}$, for further use, that is, storage or transmission. This is obviously associated with some information loss, which is known as *distortion*. The major goal in VQ is to define the regions R_j and their representatives $\boldsymbol{\theta}_j$ so that distortion is minimized.

After stating the general idea, let us proceed now to some more formal definitions. A *vector quantizer* Q of dimension l and size m is a mapping of T to a finite set C, which is called the *reproduction set* and contains m output reproduction points, the code vectors or *code words*. Thus

$$Q : T \rightarrow C$$

where $C = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_m\}$ with $\boldsymbol{\theta}_i \in T$. Each code vector $\boldsymbol{\theta}_i$ represents a specific region R_i of the vector space.

The next question that naturally arises is how one can select the code vectors $\boldsymbol{\theta}_j$ in such a way as to achieve the least possible distortion. A usual approach is to optimize an appropriate criterion function, which in this framework is also known as a *distortion function*, with respect to $\boldsymbol{\theta}_j$'s. Let $\boldsymbol{x}$ be a random vector that models T and $p(\boldsymbol{x})$ its corresponding pdf.

A commonly used distortion criterion is the *average expected quantization error*, which is defined as

$$D(Q) = \sum_{j=1}^{m} D_j(Q) \tag{14.87}$$

where

$$D_j(Q) = \int_{R_j} d(\boldsymbol{x}, \boldsymbol{\theta}_j) p(\boldsymbol{x})\, d\boldsymbol{x} \tag{14.88}$$

$D_j(Q)$ is known as the average quantization error for region R_j. The quantity d is a distance measure, e.g., Euclidean, and it is also referred to as a *distortion measure*.

When a finite number of samples, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, of $\boldsymbol{x}$ is available, the distortion criterion becomes

$$D(Q) = \sum_{i=1}^{N} d(\boldsymbol{x}_i, Q(\boldsymbol{x}_i)) P(\boldsymbol{x}_i) \tag{14.89}$$

where $Q(\boldsymbol{x}_i) \in C$ is the code vector that represents $\boldsymbol{x}_i$ and $P(\boldsymbol{x}_i)(>0) i = 1, \ldots, N$, the respective probabilities.

In [Gers 92] it is shown that the following conditions are *necessary* for a given quantizer to be optimal. The first refers to the *encoder* part of the vector quantizer, that is, the optimal way in which T is partitioned in the regions R_j, $j = 1, \ldots, m$, given the code vectors $\boldsymbol{\theta}_j$. It is known as the *nearest neighbor condition* and it

states that

- *For fixed C,*

$$Q(\boldsymbol{x}) = \boldsymbol{\theta}_j \quad \text{only if } d(\boldsymbol{x}, \boldsymbol{\theta}_j) \leq d(\boldsymbol{x}, \boldsymbol{\theta}_k), \quad k = 1, \ldots, m, k \neq j$$

The second condition refers to the *decoder* part of the VQ, that is, the optimal way the code words $\boldsymbol{\theta}_j$ are chosen, given the partition regions R_j, $j = 1, \ldots, m$. It is known as the *centroid condition* and it is stated as

- *For a fixed partition $R_1, R_2, \ldots, R_m$, each $\boldsymbol{\theta}_j$ is chosen such that*

$$\int_{R_j} d(\boldsymbol{x}, \boldsymbol{\theta}_j) p(\boldsymbol{x})\, d\boldsymbol{x} = \min_{\boldsymbol{y}} \int_{R_j} d(\boldsymbol{x}, \boldsymbol{y}) p(\boldsymbol{x})\, d\boldsymbol{x}$$ [10]

In the case that T is finite, the integrals are replaced with summations. One way to compute the code vectors of the set C is to start with an arbitrary initial estimate of the code vectors and to iteratively apply the nearest neighbor condition and the centroid condition, interchangeably, until a termination criterion is satisfied.[11] This is the well-known Lloyd's algorithm [Lloy 82].[12] Note that if $P(\boldsymbol{x}_i) = 1/N$, $\forall \boldsymbol{x}_i \in T$, Lloyd's algorithm coincides with the generalized hard clustering algorithmic scheme. This is not surprising. Both algorithms try to place optimally point representatives in space. Note, however, that despite algorithmic similarities, the two tasks have different goals. The goal of VQ is to place points in space in a way representative of the data distribution. On the other hand, clustering focuses on revealing the underlying clusters in X, if they exist.

Finally, it is worth pointing out that many other models for vector quantization have been proposed in the literature. For example, hierarchical and fuzzy vector quantizers are discussed in [Lutt 89] and [Kara 96], respectively.

APPENDIX

Derivation of $\boldsymbol{\mu}_j$ and Σ_j for the EM Algorithm (Section 14.2)

Equations (14.4) and (14.13) for $\boldsymbol{\mu}_j$ lead to

$$\boldsymbol{\mu}_j = \frac{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))\boldsymbol{x}_k}{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))} \tag{14.90}$$

for $j = 1, \ldots, m$.

[10] An additional optimality condition, given in [Gers 92], is that no vector in T is equidistant from two (or more) code vectors.

[11] One such criterion is to have the same values for all $\boldsymbol{\theta}_j$'s, $j = 1, \ldots, m$, for two successive iterations.

[12] For the special case in which the squared Euclidean distance is considered, the centroid condition becomes $\boldsymbol{\theta}_j = (1/n_j) \sum_{\boldsymbol{x} \in R_j} \boldsymbol{x}$, where n_j is the number of vectors that lie in R_j. The corresponding algorithm is the isodata algorithm, which in this framework is also called the LBG algorithm [Lind 80].

Let us now turn our attention to Σ_j. Let σ_{rs} be the (r, s) element of Σ_j^{-1}. Then Eq. (14.13) gives

$$\frac{\partial}{\partial \sigma_{rs}} \ln p(\boldsymbol{x}|C_j; \boldsymbol{\theta}_j) = \frac{1}{2}|\Sigma_j| \frac{\partial}{\partial \sigma_{rs}} |\Sigma_j^{-1}| - \frac{1}{2}(x_r - \mu_{jr})(x_s - \mu_{js})$$

or

$$\frac{\partial}{\partial \sigma_{rs}} \ln p(\boldsymbol{x}|C_j; \boldsymbol{\theta}_j) = \frac{1}{2}|\Sigma_j|\sigma_{rs} - \frac{1}{2}(x_r - \mu_{jr})(x_s - \mu_{js})$$

where σ_{rs} is the cofactor of σ_{rs}[13] and x_r, μ_{jr} (x_s, μ_{js}) are the rth (sth) coordinates of $\boldsymbol{x}$ and $\boldsymbol{\mu}_j$, respectively. Thus,

$$\frac{\partial \ln p(\boldsymbol{x}|C_j; \boldsymbol{\theta}_j)}{\partial \Sigma_j^{-1}} = \frac{1}{2}|\Sigma_j|\boldsymbol{\sigma} - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_j)(\boldsymbol{x} - \boldsymbol{\mu}_j)^T \tag{14.91}$$

where $\boldsymbol{\sigma}$ is the matrix of the cofactors of Σ_j^{-1}. Since, in general, $|\Sigma^{-1}| \neq 0$, the following identity holds from linear algebra:

$$\Sigma_j^{-1}\boldsymbol{\sigma}^T = |\Sigma_j^{-1}|I$$

Premultiplying both sides of this equation by Σ_j and noting that $\boldsymbol{\sigma}$ is a symmetric matrix, we obtain

$$\boldsymbol{\sigma} = |\Sigma_j^{-1}|\Sigma_j$$

Substituting the last result into Eq. (14.3), we obtain

$$\frac{\partial \ln p(\boldsymbol{x}|C_j; \boldsymbol{\theta}_j)}{\partial \Sigma_j^{-1}} = \frac{1}{2}\Sigma_j - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_j)(\boldsymbol{x} - \boldsymbol{\mu}_j)^T \tag{14.92}$$

Substituting this result into Eq. (14.3) and after some manipulations, we finally obtain

$$\Sigma_j = \frac{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))(\boldsymbol{x}_k - \boldsymbol{\mu}_j)(\boldsymbol{x}_k - \boldsymbol{\mu}_j)^T}{\sum_{k=1}^{N} P(C_j|\boldsymbol{x}_k; \boldsymbol{\Theta}(t))} \tag{14.93}$$

for $j = 1, \ldots, m$.

[13] Recall that the cofactor of the element a_{ij} of a matrix A is the determinant of the matrix that results from A if we delete its ith row and its jth column.

Derivation of the FCQS Algorithm

In the sequel we adopt constraint (v) (see Section 14.3.2). Since not all of the parameters of $\boldsymbol{p}_j$ are involved in (v), it is convenient to split $\boldsymbol{p}_j$ into two smaller vectors $\boldsymbol{a}_j$ and $\boldsymbol{b}_j$, that is, $\boldsymbol{p}_j = [\boldsymbol{a}_j{}^T, \boldsymbol{b}_j{}^T]^T$. The first vector contains the parameters of $\boldsymbol{p}_j$ that are involved in the constraint and the second contains the remaining parameters. More specifically, let us define

$$a_{jt} = \begin{cases} p_{jt}, & t = 1, \ldots, l \\ p_{jt}/\sqrt{2}, & t = l+1, \ldots, r \end{cases} \tag{14.94}$$

and

$$b_{jt} = p_{j(r+t)}, \qquad t = 1, \ldots, l+1 \tag{14.95}$$

Under these two definitions, constraint (v) becomes

$$\|\boldsymbol{a}_j\|^2 = 1 \tag{14.96}$$

Since $\boldsymbol{a}_j$ and, as a consequence, $\boldsymbol{p}_j$ are subject to constraints, we use once again the Lagrange multipliers method. Specifically, we define the function

$$\mathcal{J}_a(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q \boldsymbol{p}_j{}^T M_i \boldsymbol{p}_j - \sum_{j=1}^{m} \lambda_j (\|\boldsymbol{a}_j\|^2 - 1) \tag{14.97}$$

Let

$$\boldsymbol{r}_i{}^T = \left[x_{i1}^2, \ldots, x_{il}^2, \sqrt{2}x_{i1}x_{i2}, \ldots, \sqrt{2}x_{i,l-1}x_{il}\right]$$

and

$$\boldsymbol{t}_i{}^T = [x_{i1}, x_{i2}, \ldots, x_{il}, 1]$$

Using these definitions, we may write Eq. (14.97) as follows:

$$\mathcal{J}_a(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q [\boldsymbol{a}_j{}^T \boldsymbol{b}_j{}^T] \begin{bmatrix} \boldsymbol{r}_i \boldsymbol{r}_i{}^T & \boldsymbol{r}_i \boldsymbol{t}_i{}^T \\ \boldsymbol{t}_i \boldsymbol{r}_i{}^T & \boldsymbol{t}_i \boldsymbol{t}_i{}^T \end{bmatrix} [\boldsymbol{a}_j \boldsymbol{b}_j] - \sum_{j=1}^{m} \lambda_j (\|\boldsymbol{a}_j\|^2 - 1)$$

or, after some manipulations,

$$\mathcal{J}_a(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q \left(\boldsymbol{a}_j{}^T R_i \boldsymbol{a}_j + \boldsymbol{b}_j{}^T T_i \boldsymbol{b}_j + 2\boldsymbol{a}_j{}^T S_i^T \boldsymbol{b}_j\right) - \sum_{j=1}^{m} \lambda_j (\|\boldsymbol{a}_j\|^2 - 1) \tag{14.98}$$

where $R_i = \boldsymbol{r}_i \boldsymbol{r}_i^T$, $S_i = \boldsymbol{t}_i \boldsymbol{r}_i^T$, and $T_i = \boldsymbol{t}_i \boldsymbol{t}_i^T$. The gradient of $\mathcal{J}_a$ with respect to $\boldsymbol{b}_j$, after some manipulations, becomes

$$\frac{\partial \mathcal{J}_a}{\partial \boldsymbol{b}_j} = \sum_{i=1}^{N} u_{ij}^q \left(2T_i \boldsymbol{b}_j + 2S_i \boldsymbol{a}_j\right)$$

If we define $H_j = \sum_{i=1}^{N} u_{ij}^q T_i$ and $G_j = \sum_{i=1}^{N} u_{ij}^q S_i$ and set $\partial \mathcal{J}_a / \partial \boldsymbol{b}_j$ equal to $\mathbf{0}$, we finally obtain that

$$\boldsymbol{b}_j = -H_j^{-1} G_j \boldsymbol{a}_j \tag{14.99}$$

The gradient of $\mathcal{J}_a$ with respect to $\boldsymbol{a}_j$, after some manipulations, becomes

$$\frac{\partial \mathcal{J}_a}{\partial \boldsymbol{a}_j} = \sum_{i=1}^{N} u_{ij}^q \left(2R_i \boldsymbol{a}_j + 2S_i^T \boldsymbol{b}_j\right) - 2\lambda_j \boldsymbol{a}_j$$

Defining $F_j = \sum_{i=1}^{N} u_{ij}^q R_i$ and setting $\partial \mathcal{J}_a / \partial \boldsymbol{a}_j$ equal to $\mathbf{0}$, we obtain

$$\left(F_j - \lambda_j I\right) \boldsymbol{a}_j = -G_j^T \boldsymbol{b}_j$$

Substituting $\boldsymbol{b}_j$ from Eq. (14.99) into this equation, we finally obtain

$$\left(F_j - G_j^T H_j^{-1} G_j\right) \boldsymbol{a}_j = \lambda_j \boldsymbol{a}_j \tag{14.100}$$

Thus λ_j is an eigenvalue of $(F_j - G_j^T H_j^{-1} G_j)$ and $\boldsymbol{a}_j$ is its corresponding eigenvector with length equal to 1 due to the constraint given by Eq. (14.95). In fact, we choose the λ_j to be the smallest eigenvalue of $(F_j - G_j^T H_j^{-1} G_j)$ (Problem 14.12).

It should be noted here that the inverse of H_j exists if there are at least $l + 1$ noncoplanar vectors in the data set.

As soon as we have computed $\boldsymbol{a}_j$ from Eq. (14.100), we can compute $\boldsymbol{b}_j$ using Eq. (14.99). Then, the algorithm may be written as

- *Parameter updating*
  - —$F_j(t) = \sum_{i=1}^{N} u_{ij}^q (t-1) R_i$.
  - —$H_j(t) = \sum_{i=1}^{N} u_{ij}^q (t-1) T_i$.
  - —$G_j(t) = \sum_{i=1}^{N} u_{ij}^q (t-1) S_i$.
  - —Determine $\boldsymbol{a}_j(t)$ as the eigenvector of unit length that corresponds to the smallest eigenvalue of $(F_j(t) - G_j^T(t) H_j^{-1}(t) G_j(t))$.
  - —$\boldsymbol{b}_j(t) = -H_j^{-1}(t) G_j(t) \boldsymbol{a}_j(t)$.
  - —Determine $\boldsymbol{p}_j(t)$ from Eqs. (14.94) and (14.95).
- *End* parameter updating □

Computation of Σ_j in the G-K Algorithm

Taking into account that $|\Sigma_j| = |\Sigma_j^{-1}|^{-1}$, $d_{GK}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ becomes

$$d_{GK}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j) = |\Sigma_j^{-1}|^{-1/l}(\boldsymbol{x}_i - \boldsymbol{c}_j)^T \Sigma_j^{-1}(\boldsymbol{x}_i - \boldsymbol{c}_j)$$

Let f_{rs} be the (r, s) element of Σ_j^{-1}. Also let x_{ir} and c_{jr} be the rth coordinates of $\boldsymbol{x}_i$ and $\boldsymbol{c}_j$, respectively. The partial derivative of $d_{GK}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)$ with respect to f_{rs}, after some manipulations, becomes

$$\frac{\partial d_{GK}^2(\boldsymbol{x}, \boldsymbol{\theta}_j)}{\partial f_{rs}} = -\frac{1}{l}|\Sigma_j^{-1}|^{-\frac{1}{l}-1}\sigma_{rs}(x_{ir} - c_{jr})(x_{is} - c_{js}) + |\Sigma_j^{-1}|^{-\frac{1}{l}}(x_{ir} - c_{jr})(x_{is} - c_{js})$$

where σ_{rs} is the cofactor of f_{rs}. Taking into account the preceding equation and the fact that the cofactor matrix $\boldsymbol{\sigma}$ of Σ_j^{-1} can be written as $\boldsymbol{\sigma} = |\Sigma_j^{-1}|\Sigma_j$, we can write

$$\frac{\partial d_{GK}^2(\boldsymbol{x}_i, \boldsymbol{\theta}_j)}{\partial \Sigma_j} = |\Sigma_j^{-1}|^{-1/l}\left[-\frac{1}{l}(\boldsymbol{x}_i - \boldsymbol{c}_j)^T \Sigma_j^{-1}(\boldsymbol{x}_i - \boldsymbol{c}_j)\Sigma_j + (\boldsymbol{x}_i - \boldsymbol{c}_j)(\boldsymbol{x}_i - \boldsymbol{c}_j)^T\right]$$

Taking into account the last equation, and setting the gradient of $J_{GK}(\boldsymbol{\theta}, U)$ with respect to Σ_j^{-1} equal to zero, we obtain

$$\frac{1}{l}\sum_{i=1}^{N} u_{ij}^q (\boldsymbol{x}_i - \boldsymbol{c}_j)^T \Sigma_j^{-1}(\boldsymbol{x}_i - \boldsymbol{c}_j)\Sigma_j = \sum_{i=1}^{N} u_{ij}^q (\boldsymbol{x}_i - \boldsymbol{c}_j)(\boldsymbol{x}_i - \boldsymbol{c}_j)^T \quad (14.101)$$

The last equation is satisfied if we choose (Problem 14.19)

$$\Sigma_j = \frac{\sum_{i=1}^{N} u_{ij}^q (\boldsymbol{x}_i - \boldsymbol{c}_j)(\boldsymbol{x}_i - \boldsymbol{c}_j)^T}{\sum_{i=1}^{N} u_{ij}^q} \quad (14.102)$$

Problems

14.1 Consider the case in which there exist m clusters in X, which are characterized by normal distributions of unknown means and known covariance matrices; that is, the parameter vector $\boldsymbol{\theta}$ consists only of the parameters of $\boldsymbol{\mu}_j$, $j = 1, \ldots, m$. State the corresponding generalized mixture decomposition algorithmic scheme (GMDAS).

14.2 Consider the case that there exist m clusters in X which are described by normal distributions of unknown means and unknown diagonal covariance matrices. Derive the corresponding GMDAS.

14.3 Consider the case that there exist m clusters in X which are described by normal distributions. Derive the Maximum Likelihood estimates of the means $\boldsymbol{\mu}_j$ and covariance matrices Σ_j when:
(a) the means and the covariance matrices are unknown and
(b) the means and the covariance matrices are unknown but $\Sigma_j = \Sigma$, $j = 1, \ldots, m$.
Compare the results of (a) with those of section 14.2.1.

14.4 Consider the data set $X = \{\boldsymbol{x}_i \in \mathcal{R}^2, i = 1, \ldots, 16\}$, where $\boldsymbol{x}_1 = [2, 0]^T$, $\boldsymbol{x}_2 = [\sqrt{2}, \sqrt{2}]^T$, $\boldsymbol{x}_3 = [0, 2]^T$, $\boldsymbol{x}_4 = [-\sqrt{2}, \sqrt{2}]^T$, $\boldsymbol{x}_5 = [-2, 0]^T$, $\boldsymbol{x}_6 = [-\sqrt{2}, -\sqrt{2}]^T$, $\boldsymbol{x}_7 = [0, -2]^T$, $\boldsymbol{x}_8 = [\sqrt{2}, -\sqrt{2}]^T$. The remaining points $\boldsymbol{x}_i$, $i = 9, \ldots, 16$, are obtained from the first eight points as follows. The first coordinate of $\boldsymbol{x}_i$, $i = 9, \ldots, 16$, equals the first coordinate of $\boldsymbol{x}_{i-8}$ plus 6, while the second coordinate of $\boldsymbol{x}_i$, $i = 9, \ldots, 16$, equals the second coordinate of $\boldsymbol{x}_{i-8}$.
(a) Run the GMDAS, with Gaussian pdf's, to obtain estimates of $\boldsymbol{\mu}_j$, and Σ_j, $j = 1, 2$.
(b) Does the algorithm determine the clusters that underlie X correctly? Justify your answer.

14.5 Consider the setup of Problem 14.4, with the difference that the points $\boldsymbol{x}_i$, $i = 9, \ldots, 16$, are derived as follows. The first coordinate of $\boldsymbol{x}_i$, $i = 9, \ldots, 16$, equals the first coordinate of $\boldsymbol{x}_{i-8}$ plus 2, while the second coordinate of $\boldsymbol{x}_i$, $i = 9, \ldots, 16$, equals to the second coordinate of $\boldsymbol{x}_{i-8}$.
(a) Run the GMDAS, with Gaussian pdf's, to obtain estimates for $\boldsymbol{\mu}_j$, and Σ_j, $j = 1, 2$.
(b) Does the algorithm determine the clusters that underlie X accurately? Justify your answer.
(c) Compare the results obtained in this and the previous problem.

14.6 Consider four two-dimensional distributions with means $\boldsymbol{\mu}_1 = [0, 0]^T$, $\boldsymbol{\mu}_2 = [2, 2]^T$, $\boldsymbol{\mu}_3 = [4, 0]^T$, $\boldsymbol{\mu}_4 = [7, 0]^T$, respectively, and covariance matrices

$$\Sigma_1 = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}, \qquad \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\Sigma_3 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}, \qquad \Sigma_4 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

respectively. Draw 80 points from each distribution and let X be the set that contains these 320 points. Initialize $\boldsymbol{\mu}_i$ and Σ_i, $i = 1, \ldots, 4$, as in Example 14.1. Set $m = 4$, $\varepsilon = 0.01$ and run the GMDAS, with Gaussian pdf's.
(a) What are the estimates of $\boldsymbol{\mu}_j$, $j = 1, \ldots, 4$, and Σ_j, $j = 1, \ldots, 4$?
(b) Assign each feature vector $\boldsymbol{x} \in X$ to a cluster C_j according to the Bayes decision rule.
(c) Derive the respective confusion matrix.
(d) Run the algorithm for $m = 3$ and $m = 2$ and repeat steps (a) and (b). Discuss the results.

14.7 In the framework of Example 14.4, prove that for $m = 2, q \in \{2, 3\}$ and for fixed θ, there are cases where the fuzzy clusterings are favored against the hard ones.

14.8 Find the relation between $\boldsymbol{p}$ and A, $\boldsymbol{b}$, and c so that Eqs. (14.35) and (14.36) are equivalent.
Hint: Consider each coordinate of $\boldsymbol{p}$ separately.

14.9 Let $l = 2$. Prove that the substitution of $\mathbf{z}$, as given by Eq. (14.44), into Eq. (14.42) leads to a fourth-degree polynomial, with respect to λ.

14.10 Prove Eq. (14.50).

14.11 (a) Derive Eqs. (14.56) and (14.57) for the fuzzy C ellipsoidal shells (FCES) algorithm. (b) Write the parameter determination part of the fuzzy C ellipsoidal shells (FCES) algorithm.

14.12 Fill in the gaps in the derivation of the fuzzy C quadric shells (FCQS) algorithm. (see Appendix)

14.13 (a) State explicitly the modified fuzzy C quadric shells (MFCQS) algorithm.
(b) Under what general conditions are the algebraic and the perpendicular distances close to each other?

14.14 What is the relation between the perpendicular and the radial distance between a point $\boldsymbol{x}$ and a hyperspherical cluster?

14.15 (a) Derive the AFCS algorithm [Dave 92b] for the case that spherical clusters are to be recovered. The distance between a point $\boldsymbol{x}$ and a hypersphere Q with center $\boldsymbol{c}$ and radius r is

$$d^2(\boldsymbol{x}, Q) = (\|\boldsymbol{x} - \boldsymbol{c}\| - r)^2$$

(b) Derive the fuzzy C spherical shells (FCSS) algorithm [Kris 92b] for the case that spherical clusters are to be identified.

14.16 Derive the possibilistic algorithm obtained via minimization of the function J_1 given in Eq. (14.77).

14.17 Plot u_{ij} versus $d(\boldsymbol{x}_i, \boldsymbol{\theta}_j)/\eta_j$, using Eq. (14.78). Compare this plot with the one shown in Figure 14.15.

14.18 Compare the isodata algorithm with the variant of the BSAS proposed in [MacQ 67] and outlined in section 12.6.

14.19 Prove that Eq. (14.100) is satisfied if Σ_j is equal to

$$\Sigma_j = \frac{\sum_{i=1}^{N} u_{ij}^q (\boldsymbol{x}_i - \boldsymbol{c}_j)(\boldsymbol{x}_i - \boldsymbol{c}_j)^T}{\sum_{i=1}^{N} u_{ij}^q}$$

References

[Ande 73] Anderberg M. R. *Cluster analysis for applications*, Academic Press, 1973.

[Ande 85] Anderson I., Bezdek J.C. "An application of the c-varieties clustering algorithms to polygonal curve fitting," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 15, pp. 637–639, 1985.

[Ball 67] Ball G.H., Hall D.J. "A clustering technique for summarizing multivariate data," *Behavioral Science*, Vol. 12, pp. 153–155, March 1967.

[Barn 96] Barni M., Cappellini V., Mecocci A. "Comments on 'A possibilistic approach to clustering'," *IEEE Transactions on Fuzzy Systems*, Vol. 4(3), pp. 393–396, August 1996.

[Bezd 80] Bezdek J.C. "A convergence theorem for the fuzzy Isodata clustering algorithms," *IEEE Transactions on PAMI*, Vol. 2(1), pp. 1–8, 1980.

[Bezd 92] Bezdek J.C., Hathaway R.J. "Numerical convergence and interpretation of the fuzzy c-shells clustering algorithm," *IEEE Transactions on Neural Networks*, Vol. 3(5), pp. 787–793, September 1992.

[Bezd 95] Bezdek J. C., Hathaway R. J., Pal N. R., "Norm-induced shell prototypes (NISP) clustering," *Neural, Parallel and Scientific Computations*, Vol. 3, pp. 431–450, 1995.

[Bobr 91] Bobrowski L., Bejdek.J.C. "c-Means clustering with l_1 and l_∞ norms," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 21(3), pp. 545–554, May/June 1991.

[Cann 86] Cannon R.L., Dave J.V., Bezdek J.C. "Efficient implementation of the fuzzy c-means clustering algorithms," *IEEE Transactions on PAMI*, Vol. 8(2), pp. 248–255, March 1986.

[Chen 89] Chen D.S. "A data-driven intermediate level feature extraction algorithm," *IEEE Transactions on PAMI*, Vol. 11(7), pp. 749–758, July 1989.

[Dave 92a] Dave R.N., Bhaswan K. "Adaptive fuzzy c-shells clustering and detection of ellipses," *IEEE Transactions on Neural Networks*, Vol. 3(5), pp. 643–662, 1992.

[Dave 92b] Dave R.N. "Generalized fuzzy c-shells clustering and detection of circular and elliptical boundaries," *Pattern Recognition*, Vol. 25(7), pp. 713–721, 1992.

[Duda 73] Duda R.O., Hart P.E. *Pattern Classification and Scene Analysis*, John Wiley Sons, 1973.

[Frig 96] Frigui H., Krishnapuram R. "A comparison of fuzzy shell clustering methods for the detection of ellipses," *IEEE Transactions on Fuzzy Systems*, Vol. 4(2), pp. 193–199, May 1996.

[Gers 79] Gersho A. "Asymptotically optimal block quantization," *IEEE Transactions on Information Theory*, Vol. 25(4), pp. 373–380, 1979.

[Gers 92] Gersho A., Gray R.M. *Vector Quantization and Signal Compression*, Kluwer Publishers, 1992.

[Gnan 77] Gnanadesikan R. *Methods for Statistical Data Analysis of Multivariate Observations*, John Wiley & Sons, 1977.

[Gray 84] Gray R.M. "Vector quantization," *IEEE ASSP Magazine*, Vol. 1, pp. 4–29, 1984.

[Hath 86] Hathaway R.J., Bezdek J.C. "Local convergence of the fuzzy c-means algorithms," *Pattern Recognition* Vol. 19(6), pp. 477–480, 1986.

[Hath 89] Hathaway R.J., Davenport J.W., Bezdek J.C. "Relational duals of the c-means clustering algorithms," *Pattern Recognition*, Vol. 22(2), pp. 205–212, 1989.

[Hath 93] Hathaway R.J., Bezdek J.C. "Switching regression models and fuzzy clustering," *IEEE Transactions on Fuzzy Systems*, Vol. 1(3), pp. 195–204, August 1993.

[Hath 95] Hathaway R.J., Bezdek J.C. "Optimization of clustering criteria by reformulation," *IEEE Transactions on Fuzzy Systems*, Vol. 3(2), pp. 241–245, 1995.

[Hopp 99] Hoppner F., Klawonn F., Kruse R., Runkler T., *Fuzzy cluster analysis*, J. Wiley and Sons, 1999

[Horn 86] Horn B.K.P. *Robot Vision*, MIT Press, 1986.

[Isma 86] Ismail M.A., Selim S.Z. "Fuzzy c-means: Optimality of solutions and effective termination of the algorithm," *Pattern Recognition*, Vol. 19(6), pp. 481–485, 1986.

[Kara 96] Karayiannis N.B., Pai P.-I. "Fuzzy algorithms for learning vector quantization," *IEEE Transactions on Neural Networks*, Vol. 7(5), pp. 1196–1211, September 1996.

[Kare 94] Karen D., Cooper D., Subrahmonia J. "Describing complicated objects by implicit polynomials," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16(1), pp. 38–53, 1994.

[Kim 88] Kim T., Bezdek J.C., Hathaway R.J. "Optimality tests for fixed points of the fuzzy c-means algorithm," *Pattern Recognition*, Vol. 21(6), pp. 651–663, 1988.

[Kris 92a] Krishnapuram R., Freg C.-P. "Fitting an unknown number of lines and planes to image data through compatible cluster merging," *Pattern Recognition*, Vol. 25(4), pp. 385–400, 1992.

[Kris 92b] Krishnapuram R., Nasraoui O., Frigui H. "The fuzzy c spherical shells algorithm: A new approach," *IEEE Transactions on Neural Networks*, Vol. 3(5), pp. 663–671, 1992.

[Kris 93] Krishnapuram R., Keller J.M. "A possibilistic approach to clustering," *IEEE Transactions on Fuzzy Systems*, Vol. 1(2), pp. 98–110, May 1993.

[Kris 95a] Krishnapuram R., Frigui H., Nasraoui O. "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation—Part I," *IEEE Transactions on Fuzzy Systems*, Vol. 3(1), pp. 29–43, February 1995.

[Kris 95b] Krishnapuram R., Frigui H., Nasraoui O. "Fuzzy and possibilistic shell clus- tering algorithms and their application to boundary detection and surface approximation—Part II," *IEEE Transactions on Fuzzy Systems*, Vol. 3(1), pp. 44–60, February 1995.

[Kris 96] Krishnapuram R., Keller J.M. "The possibilistic c-means algorithm: Insights and recommendations," *IEEE Transaction on Fuzzy Systems*, Vol. 4(3), pp. 385–393, August 1996.

[Lind 80] Linde Y., Buzo A., Gray R.M. "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, Vol. 28, pp. 84–95, 1980.

[Lloy 82] Lloyd S.P. "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, Vol. 28(2), pp. 129–137 March 1982.

[Luen 84] Luenberger D.G. *Linear and Nonlinear Programming*, Addison Wesley, 1984.

[Lutt 89] Luttrell S.P. "Hierarchical vector quantization," *IEE Proceedings (London)*, Vol. 136 (Part I), pp. 405–413, 1989.

[MacQ 67] MacQueen J.B., "Some methods for classification and analysis of multivariate observations", *Proceedings of the Symposium on Mathematical Statistics and Probability, 5th Berkeley*, Vol. 1, 281–297, AD 669871, University of California Press, 1967.

[Man 94] Man Y., Gath I. "Detection and separation of ring-shaped clusters using fuzzy clustering," *IEEE Transactions on PAMI*, Vol. 16(8), pp. 855–861, August 1994.

[Mena 00] Menard M., Demko C., Loonis P., "The fuzzy c+2 means: solving the ambiguity rejection in clustering," *Pattern Recognition*, Vol. 33, pp. 1219–1237, 2000.

[Ng 00] Ng M. K., "A note on constrained k-means algorithms," *Pattern Recognition*, Vol. 33, pp. 515–519, 2000.

[Pato 70] Paton K. "Conic sections in chromosome analysis," *Pattern Recognition*, Vol. 2(1), pp. 39–51, January 1970.

[Runk 99] Runkler T. A., Bezdek J. C., "Alternating cluster estimation: A new tool for clustering and function approximation," *IEEE Trans. on Fuzzy Systems*, Vol. 7, No. 4, pp. 377–393, Aug. 1999.

[Sabi 87] Sabin M.J. "Convergence and consistency of fuzzy c-means/Isodata algorithms," *IEEE Transactions on PAMI*, Vol. 9(5), pp. 661–668, September 1987.

[Seli 84a] Selim S.Z., Ismail M.A. "K-means type algorithms: A generalized convergence theorem and characterization of local optimality, "*IEEE Transactions on PAMI*, Vol. 6(1), pp. 81–87, 1984.

[Seli 84b] Selim S.Z., Ismail M.A. "Soft clustering of multidimensional data: A semifuzzy approach," *Pattern Recognition*, Vol. 17(5), pp. 559–568, 1984.

[Seli 86] Selim, S.Z., Ismail, M.A. "On the local optimality of the fuzzy Isodata clustering algorithm," *IEEE Transactions on PAMI*, Vol. 8(2), pp. 284–288, March 1986.

[Spra 66] Spragins J. "Learning without a teacher," *IEEE Transactions on Information Theory*, Vol. IT-12, pp. 223–230, April 1966.

[Wei 94] Wei W., Mendel J.M. "Optimality tests for the fuzzy c-means algorithm," *Pattern Recognition*, Vol. 27(11), pp. 1567–1573, 1994.

[Yama 80] Yamada Y., Tazaki S., Gray R.M. "Asymptotic performance of block quantizers with difference distortion measures," *IEEE Transactions on Information Theory*, Vol. 26(1), pp. 6–14, 1980.

[Zade 78] Zadeh L.A. "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems*, Vol. 1, pp. 3–28, 1978.

[Zhua 96] Zhuang X., Huang Y., Palaniappan K., Zhao Y. "Gaussian mixture density modelling, decomposition and applications," *IEEE Transactions on Image Processing*, Vol. 5, pp. 1293–1302, September 1996.

CHAPTER 15

CLUSTERING ALGORITHMS IV

15.1 INTRODUCTION

The clustering algorithms presented in the previous two chapters evolved along two distinct major philosophies. The current chapter presents categories of algorithms that cannot be included in either of the previous two families, and they stem from various ideas. The first one includes clustering algorithms based on graph theory concepts, such as the minimum spanning tree and the directed tree. The second category includes competitive learning algorithms. The third category includes branch and bound algorithms. These schemes guarantee to provide globally optimal clustering, in terms of a prespecified optimality criterion, at the cost of increased computational requirements. The fourth category contains algorithms that are based on morphological transformations. These have been inspired by the corresponding methods used in signal and image processing. The fifth category contains algorithms that are not based on cluster representatives but, instead, seek to place boundaries between clusters. Algorithms of the sixth category treat clusters as dense in data regions of the feature space separated by regions sparse in data. Alternatively, clusters may be viewed as peaks of the pdf, underlying the data in X, separated by valleys. The seventh category includes additional algorithms that are based on function optimization, such as simulated annealing and deterministic annealing. The difference from the algorithms presented in Chapter 14 is that the optimizing methods used in this chapter do not involve differential calculus concepts. Finally, the eighth category includes genetic algorithms modified suitably for clustering tasks.

15.2 CLUSTERING ALGORITHMS BASED ON GRAPH THEORY

The algorithms of this family are capable of detecting clusters of various shapes, at least for the case in which they are well separated. Detection of clusters of various shapes is a feature that is shared by only a few other clustering algorithms.

15.2.1 Minimum Spanning Tree Algorithms

The first algorithm is based on the idea of the minimum spanning tree (MST) (Chapter 13) and is motivated by the way human perception works [Zahn 71]. More precisely, humans organize information with the most economical encoding [Hoch 64]. For example, the more likely way for a human to organize the points in Figure 15.1 is in four groups (clusters).

Let us consider the complete graph G, each node of which corresponds to a point of X. The weight of an edge $e = (\boldsymbol{x}_i, \boldsymbol{x}_j)$, w_e, connecting two nodes $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, is set equal to the distance $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ of the corresponding points in the feature space. Also, we say that two edges e_1 and e_2 are k *steps away* from each other if the minimum path that connects a vertex of e_1 and a vertex of e_2 has length equal to $k - 1$, that is, contains $k - 1$ edges.

The idea of the algorithm is the following: *determine the minimum spanning tree of G and then remove the edges that are "unusually" large compared with their neighboring edges.* These edges are called *inconsistent*, and it is expected that they connect points from different clusters. Next, we discuss a way to determine inconsistent edges. For each edge e, we consider all the edges, e_i, that lie k steps away from it, at the most, and we compute the mean, m_e, and the standard deviation, σ_e, of their weights. If w_e lies more than q (typically $q = 2$) standard deviations (σ_e) away from m_e, then we consider e as inconsistent. From this, it is clear that the characterization of an edge as inconsistent is somewhat subjective and depends on k and q, which are preselected.

Example 15.1. Consider Figure 15.2. Let $k = 2$ and $q = 3$. The edges lying two steps at the most from e_0 are e_i, $i = 1, \ldots, 10$. The mean m_{e_0} and the standard deviation σ_{e_0}, corresponding to e_0 are 2.3 and 0.95, respectively. Thus w_{e_0} lies 15.5 standard deviations (σ_{e_0}) away from m_{e_0}. Therefore, e_0 is an inconsistent edge.

Let us now consider the edge e_{11}. Working as before, we find that $m_{e_{11}}$ and $\sigma_{e_{11}}$ are 2.5 and 2.12, respectively. Thus $w_{e_{11}}$ is 0.24 standard deviations ($\sigma_{e_{11}}$) away from $m_{e_{11}}$. Therefore, e_{11} is not an inconsistent edge.

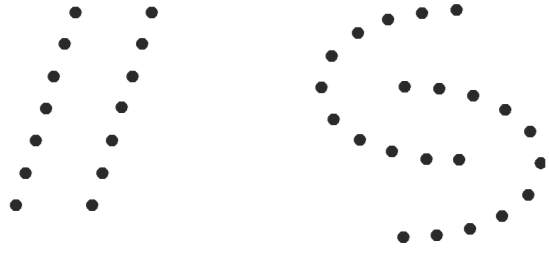


FIGURE 15.1: An arrangement of clusters.

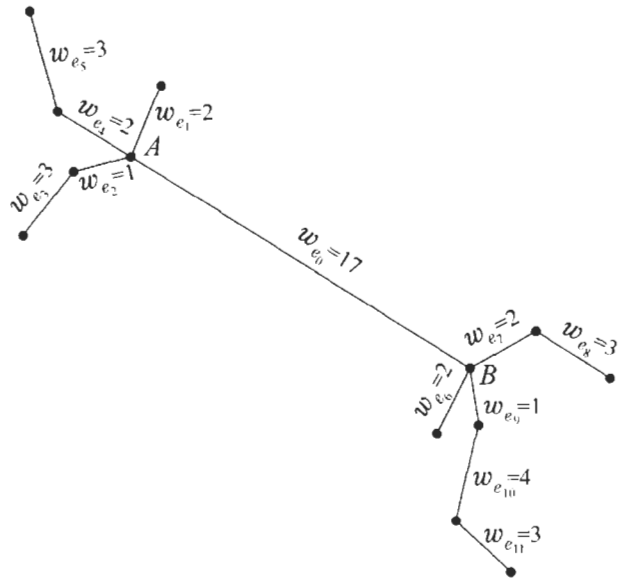


FIGURE 15.2: The minimum spanning tree of a graph. The edge e_0 is inconsistent, while e_1 is consistent.

After these definitions, the MST clustering algorithm may be stated as follows.

The MST Clustering Algorithm

- Construct a complete graph G such that:
  - —its vertices correspond to the vectors of X.
  - —$w_{(\boldsymbol{x}_i, \boldsymbol{x}_j)} = d(\boldsymbol{x}_i, \boldsymbol{x}_j), i, j = 1, \ldots, N, i \neq j$.
- Determine the MST of G.
- Identify the inconsistent edges of the MST.
- The clusters are the connected components of the MST after the removal of the inconsistent edges.

This algorithm works satisfactorily for many cases where the clusters are well separated. However, this is not a panacea. Let us consider for example Figure 15.3. The edge AB has a very large neighboring edge (BC), which increases w_{AB} and σ_{AB}. Thus, AB may not be characterized as inconsistent and, as a consequence, the vectors from regions R_1 and R_2 are considered as members of the same cluster [Jarv 78].

Some suggestions for the use of the MST algorithm for the cases where we have touching clusters (Figure 15.4a), as well as for the case where the clusters

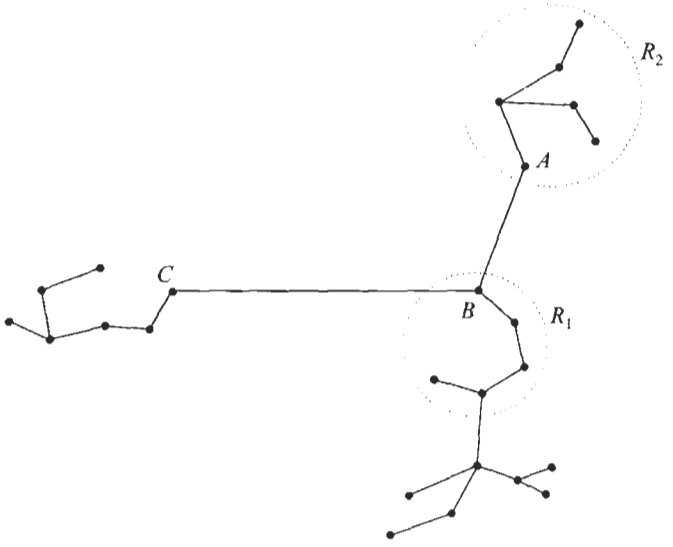


FIGURE 15.3: The MST clustering algorithm will assign the vectors of the regions R_1 and R_2 to the same cluster.

(a) (b)

FIGURE 15.4: (a) Touching clusters. (b) Clusters with different densities.

have different densities (Figure 15.4b), are discussed in [Zahn 71]. However, they implicitly require knowledge of the shape of the clusters.

Remark

- Note that this algorithm does not depend on the order in which data are considered by the algorithm and, also, no initial conditions are required, as is the case with the algorithms of Chapter 14.

15.2.2 Algorithms Based on Regions of Influence

An extension of the MST involves regions of influence for each pair of vectors of X. This idea was used by many researchers (e.g., [Tous 80], [Gabr 69],[Urqu 82]) in order to overcome the problems associated with the MST algorithms.

Let us consider two distinct vectors, $\boldsymbol{x}_i, \boldsymbol{x}_j \in X$. Their region of influence is defined as

$$R(\boldsymbol{x}_i, \boldsymbol{x}_j) = \{\boldsymbol{x}: cond(d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j), d(\boldsymbol{x}_i, \boldsymbol{x}_j)), \boldsymbol{x}_i \neq \boldsymbol{x}_j\} \tag{15.1}$$

where $cond(d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j), d(\boldsymbol{x}_i, \boldsymbol{x}_j))$ is a condition among the distances $d(\boldsymbol{x}, \boldsymbol{x}_i)$, $d(\boldsymbol{x}, \boldsymbol{x}_j)$, and $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Different choices of *cond* give rise to different shapes of regions of influence. Typical choices of *cond*, proposed in [Tous 80] and [Gabr 69], are

$$\max\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{15.2}$$

and

$$d^2(\boldsymbol{x}, \boldsymbol{x}_i) + d^2(\boldsymbol{x}, \boldsymbol{x}_j) < d^2(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{15.3}$$

respectively. Also, in [Urqu 82], the following two alternatives are proposed:

$$\begin{aligned} &(d^2(\boldsymbol{x}, \boldsymbol{x}_i) + d^2(\boldsymbol{x}, \boldsymbol{x}_j) < d^2(\boldsymbol{x}_i, \boldsymbol{x}_j)) \, OR \\ &(\sigma \min\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)) \end{aligned} \tag{15.4}$$

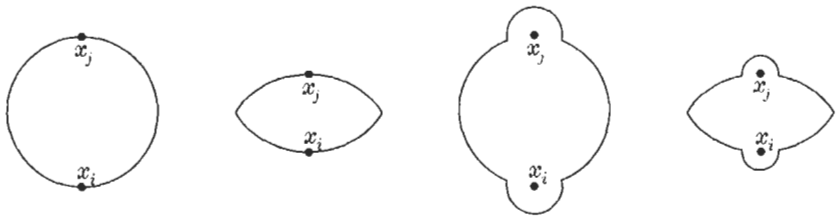
and

$$\begin{aligned} &(\max\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)) \, OR \\ &(\sigma \min\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)) \end{aligned} \tag{15.5}$$

where σ is a factor called *relative edge consistency*. This factor affects the size of the region of influence defined by $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The shapes of these regions are shown in Figure 15.5. Other choices of *cond* are also possible. An algorithm based on the idea of the regions of influence is described next.

Algorithm Based on Regions of Influence

- For $i = 1$ to N
  - —For $j = i + 1$ to N
    - * Determine the region of influence $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
    - * If $R(\boldsymbol{x}_i, \boldsymbol{x}_j) \cap (X - \{\boldsymbol{x}_i, \boldsymbol{x}_j\}) = \emptyset$ then
      - Add the edge connecting $\boldsymbol{x}_i, \boldsymbol{x}_j$
    - * End {If }
  - —End {For}



(a) (b) (c) (d)

FIGURE 15.5: The shapes of the regions defined by (a) condition (15.2), (b) condition (15.3), (c) condition (15.4), and (d) condition (15.5).

- End {For}
- Determine the connected components of the resulted graph and identify them as clusters.

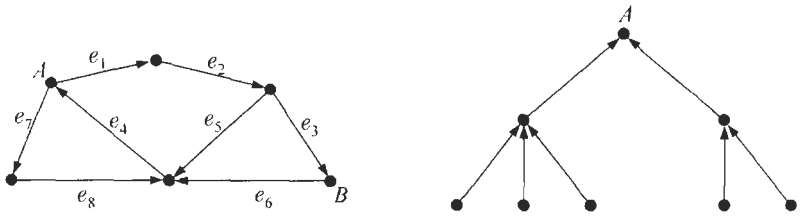
In words, the edge between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is added if no other vector of X lies in $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This is because when $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are closely located, it is expected that no other points of X will be in $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The opposite is obviously true for points located further away.

The algorithm is insensitive to the order in which the pairs of vectors are considered. Also, for the last two choices of *cond*, the value of σ must be chosen a priori. The graphs produced by these algorithms when (15.2) and (15.3) are used are also called *relative neighborhood graphs (RNGs)* and *Gabriel graphs (GGs)*, respectively.

These techniques avoid situations such as the one shown in Figure 15.3. Moreover, several results are given in [Urqu 82] showing the superior performance for the last two choices of *cond* compared with the first two. Also, in [Urqu 82] it is shown how the idea of the regions of influence may be used to give rise to hierarchical algorithms. Finally, in [Ozbo 95], empirically defined regions of influence are used, that exhibit satisfactory behavior.

15.2.3 Algorithms Based on Directed Trees

An alternative clustering scheme, based on the idea of directed trees, is proposed in [Koon 76]. Before we proceed, let us give some definitions. We recall that a *directed graph* is a graph whose edges are directed (see Figure 15.6a). We say that a set of edges $e_{i_1}, \ldots, e_{i_q}$ constitute a *directed path* from a vertex A to a vertex B, if A is the initial vertex of e_{i_1}, B is the final vertex of e_{i_q}, and the destination vertex of the edge e_{i_j}, $j = 1, \ldots, q-1$, is the departure vertex of the edge $e_{i_{j+1}}$.



(a) (b)

FIGURE 15.6: (a) A directed graph. (b) A directed tree.

For example, in Figure 15.6a, the sequence e_1, e_2, e_3 constitutes a directed path connecting the vertices A and B. Finally, a *directed tree* is a directed graph with a specific node A, known as *root*, such that (a) every node $B \neq A$ of the tree is the initial node of exactly one edge, (b) no edge departs from A, and (c) no circles are encountered, that is, there is no directed path from a node to itself (see Figure 15.6b).

The idea of the algorithm is the identification of directed trees in a graph, corresponding to the points of X, so that each of them corresponds to a cluster. The vectors of X are processed sequentially. For each point $\boldsymbol{x}_i$, we define its neighborhood as

$$\rho_i(\theta) = \{\boldsymbol{x}_j \in X : d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \theta, \boldsymbol{x}_j \neq \boldsymbol{x}_i\} \tag{15.6}$$

where θ determines the size of the neighborhood and $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the distance between the corresponding vectors of X. Let $n_i = |\rho_i(\theta)|$ be the number of points of X lying in $\rho_i(\theta)$. Finally, let $g_{ij} = (n_j - n_i)/d(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This quantity will be used to determine the position of the point $\boldsymbol{x}_i$ in a directed tree. After these definitions, the clustering algorithm may be stated as follows.

Clustering Algorithm Based on Directed Trees

Set θ to a specific value.
Determine n_i, $i = 1, \ldots, N$.
Compute g_{ij}, $i, j = 1, \ldots, N$, $i \neq j$.
For $i = 1$ to N

- if $n_i = 0$ then
  —$\boldsymbol{x}_i$ is the root of a new directed tree.
- else
  —Determine $\boldsymbol{x}_r$ such that $g_{ir} = \max_{\boldsymbol{x}_j \in \rho_i(\theta)} g_{ij}$.

—If $g_{ir} < 0$ then
  * $\boldsymbol{x}_i$ is the root of a new directed tree.
—Else if $g_{ir} > 0$ then
  * $\boldsymbol{x}_r$ is the parent of $\boldsymbol{x}_i$.[1]
—Else if $g_{ir} = 0$ then
  * Define $T_i = \{\boldsymbol{x}_j : \boldsymbol{x}_j \in \rho_i(\theta), g_{ij} = 0\}$.
  * Eliminate all the elements $\boldsymbol{x}_j \in T_i$, for which there exists a directed path from $\boldsymbol{x}_j$ to $\boldsymbol{x}_i$.
  * If the resulting T_i is empty then
    - $\boldsymbol{x}_i$ is the root of a new directed tree.
  * Else
    - The parent of $\boldsymbol{x}_i$ is $\boldsymbol{x}_q$ such that $d(\boldsymbol{x}_i, \boldsymbol{x}_q) = \min_{\boldsymbol{x}_s \in T_i} d(\boldsymbol{x}_i, \boldsymbol{x}_s)$.
  * End {if}
—End {if}

- End {if}

End {for}

The directed trees formed by these steps identify the clusters.

It is clear from the preceding algorithm that the root, say $\boldsymbol{x}_i$, of a directed tree has the largest n_i among the points lying in $\rho_i(\theta)$. That is, among the points lying in $\rho_i(\theta)$, $\boldsymbol{x}_i$ is the point with the most dense neighborhood. It should be pointed out that the branch that handles the case in which $g_{ir} = 0$ ensures that no circles will occur. Also, this algorithm is sensitive to the order in which the vectors are processed. Finally, it can be shown that for proper values of θ and large N this scheme behaves as a mode-seeking algorithm [Koon 76].

Example 15.2. Consider Figure 15.7. The size of the edge of the grid is 1 and the diagonal of a small rectangle equals $\sqrt{2}$. Also, let $X = \{\boldsymbol{x}_i, i = 1, \ldots, 11\}$. It is clear that the vectors of X form two well-separated clusters. Let $\theta = 1.1$. Applying the preceding algorithm on X, we determine the two directed trees shown in Figure 15.7. However, if we present $\boldsymbol{x}_5$ before $\boldsymbol{x}_4$, the left-directed tree will have a different root. Nevertheless, the final results of the algorithm remain the same in this (rather easy) case.

15.3 COMPETITIVE LEARNING ALGORITHMS

These algorithms employ a set of representatives $\boldsymbol{w}_j$, $j = 1, \ldots, m$.[2] *Their goal is to move each of them to regions of the vector space that are "dense"*

[1] We say that $\boldsymbol{x}_r$ is the parent of $\boldsymbol{x}_i$ if there exists a directed edge from $\boldsymbol{x}_i$ to $\boldsymbol{x}_r$.

[2] We use $\boldsymbol{w}_j$ here instead of $\boldsymbol{\theta}_j$ to comply with the notation usually adopted for this type of schemes.

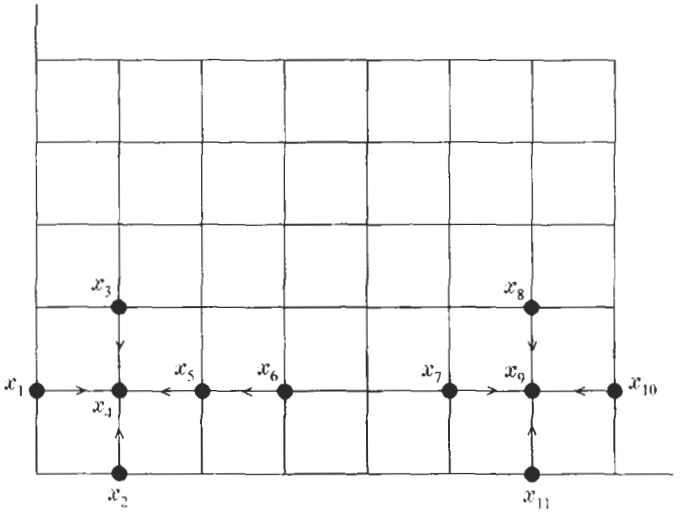


FIGURE 15.7: The setup of Example 15.2.

in vectors of X. The representatives are updated each time a new vector $\boldsymbol{x} \in X$ is presented to the algorithm. Algorithms of this type are called *pattern mode algorithms*. This is a point of differentiation from the hard clustering algorithms discussed in Chapter 14. There the updating of the representatives takes place after the presentation of *all* the vectors of X to the algorithm. Algorithms of the latter kind are also called *batch mode algorithms*. It must be emphasized that competitive learning algorithms do not necessarily stem from the optimization of a cost function.

The general idea is very simple. When a vector $\boldsymbol{x}$ is presented to the algorithm, all representatives *compete* with each other. The winner of this competition is the representative that lies closer (according to some distance measure) to $\boldsymbol{x}$. Then, the winner is updated so as to move toward $\boldsymbol{x}$, while the losers either remain unchanged or are updated toward $\boldsymbol{x}$ but at a much slower rate.

Although, in most of the cases, $\boldsymbol{w}_j$'s are points in the l-dimensional space, other choices are also possible. For example, the representatives may be hyperplanes [Likh 97]. In the sequel, we consider only the case in which $\boldsymbol{w}_j$'s are points in the l-dimensional space.

Let t be the current iteration and t_{max} the maximum allowable number of iterations. Also, let m be the current number, m_{init} the initial number, and m_{max} the maximum allowable number of clusters (representatives). Then, a generalized competitive learning scheme (GCLS) may be stated as follows.

Generalized Competitive Learning Scheme (GCLS)

- $t = 0$
- $m = m_{init}$
- (A) Initialize any other necessary parameters (depending on the specific scheme).
- Repeat
  - —$t = t + 1$
  - —Present a new randomly selected $\boldsymbol{x} \in X$ to the algorithm.
  - —(B) Determine the winning representative $\boldsymbol{w}_j$.
  - —(C) If (($\boldsymbol{x}$ is not "similar" to $\boldsymbol{w}_j$) OR (other condition)) AND ($m < m_{max}$) then
    - $m = m + 1$
    - $\boldsymbol{w}_m = \boldsymbol{x}$
  - —Else
    - (D) *Parameter updating*

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta h(\boldsymbol{x}, \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is the winner} \\ \boldsymbol{w}_j(t-1) + \eta' h(\boldsymbol{x}, \boldsymbol{w}_j(t-1)), & \text{otherwise} \end{cases} \tag{15.7}$$

  - —End
- (E) Until (convergence has occurred) OR ($t > t_{max}$)
- Identify the clusters represented by $\boldsymbol{w}_j$'s, by assigning each vector, $\boldsymbol{x} \in X$, to the cluster that corresponds to the representative closest to $\boldsymbol{x}$.

The function $h(\boldsymbol{x}, \boldsymbol{w}_i)$ is an appropriately defined function. Also, η and η' are the *learning rates* controlling the update of the winner and the losers, respectively. The parameter η' may be different for different losers. The similarity between a vector $\boldsymbol{x}$ and a representative $\boldsymbol{w}_j$ may be characterized by a threshold of similarity Θ; that is, if $d(\boldsymbol{x}, \boldsymbol{w}_j) > \Theta$, for some distance measure, $\boldsymbol{x}$ and $\boldsymbol{w}_j$ are considered as dissimilar and $\boldsymbol{w}_j$ cannot be used to represent $\boldsymbol{x}$ accurately. It is clear that improper choice of the value of Θ may lead to misleading results.

Termination of the algorithm is achieved via our familiar criterion $\|\boldsymbol{W}(t) - \boldsymbol{W}(t-1)\| < \varepsilon$, where $\boldsymbol{W} = [\boldsymbol{w}_1^T, \ldots \boldsymbol{w}_m^T]^T$.

With appropriate choices of the parts (A), (B), (C), and (D), most of the competitive learning algorithms may be viewed as special cases of the GCLS. In the sequel, unless otherwise stated, we use the Euclidean distance, although other distances may also be used.

15.3.1 Basic Competitive Learning Algorithm

In this algorithm $m = m_{init} = m_{max}$; that is, the number of representatives is constant. Thus, condition (C) is never satisfied. Also, no other parameters are

necessary, so part (A) is omitted. The determination of the winning representative (part (B)) is carried out using the following rule.

- The representative $\boldsymbol{w}_j$ is the winner on $\boldsymbol{x}$ if

$$d(\boldsymbol{x}, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d(\boldsymbol{x}, \boldsymbol{w}_k)$$

Besides the Euclidean distance, other distances may also be used, depending on the application at hand. For example, in [Ahal 90], the Itakura–Saito distortion is proposed when dealing with speech coding applications in the clustering framework. Moreover, similarity measures may also be used (see, e.g., [Fu 93]). In this case, the min operator in the preceding relation is replaced by the max operator. Finally, the updating of the representatives (part (D)) is carried out as follows:

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is the winner} \\ \boldsymbol{w}_j(t-1), & \text{otherwise} \end{cases} \tag{15.8}$$

where η is the learning rate and takes values in $[0, 1]$. According to this algorithm, the losers remain unchanged. On the other hand, the winner $\boldsymbol{w}_j(t)$ moves toward $\boldsymbol{x}$. The size of the movement depends on η. In the extreme case where $\eta = 0$, no updating takes place. On the other hand, if $\eta = 1$, the winning representative is placed on $\boldsymbol{x}$. For all other choices of η, the new value of the winner lies in the line segment formed by $\boldsymbol{w}_j(t-1)$ and $\boldsymbol{x}$.

It is clear that this algorithm requires an accurate determination of the number of representatives; that is, knowledge of the number of clusters is required. Another related problem that may arise is associated with the initialization of $\boldsymbol{w}_j$'s. If a representative is initialized far away from its closest vector in X,[3] it will never win. Thus, the vectors of X will be represented by the remaining representatives. An easy way to avoid this situation is to initialize all representatives using vectors of X.

In the special case in which the vectors are always presented in the same order, that is, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N, \ldots$, and under the assumption that after an iteration t_0 each representative wins on the same vectors, which is reasonable at least for the case where well-separated clusters are formed by the vectors of X, it can be shown that each representative converges to a weighted mean of the vectors it represents [Kout 95].

This algorithm has also been studied for a variable learning rate (e.g., [Likh 97]). Typical constraints for $\eta(t)$ in this case are:

- $\eta(t)$ is a positive decreasing sequence and $\eta(t) \to 0$.

[3] More specifically, if it lies far away from the convex hull defined by the vectors of X.

- $\sum_{t=0}^{\infty} \eta(t) = \infty$.
- $\sum_{t=0}^{\infty} \eta^r(t) < +\infty$, for $r > 1$.

Note that these constraints are very similar to those required by the Robbins–Monro schemes, discussed in Section 3.4.2. This is not a coincidence. Let us consider for example the trivial case of a single representative ($m = 1$). If $\eta = \eta(t)$, the updating equation may be seen as the Robbins–Monro iteration for solving the problem

$$E[h(\boldsymbol{x}, \boldsymbol{w})] = 0$$

where $h(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x} - \boldsymbol{w}$.

Finally, competitive learning algorithms for binary-valued vectors are discussed in [Rume 86], [Mals 73].

15.3.2 Leaky Learning Algorithm

This algorithm is the same as the basic competitive learning algorithm except for the updating equation of the representatives, which is

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta_w(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is the winner} \\ \boldsymbol{w}_j(t-1) + \eta_l(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is a loser} \end{cases} \tag{15.9}$$

where η_w and η_l are the learning rates in $[0, 1]$ and $\eta_w \gg \eta_l$. The basic competitive learning scheme may be viewed as a special case of the leaky learning scheme, for $\eta_l = 0$. In the general case where η_w and η_l are both positive, all representatives move toward $\boldsymbol{x}$. However, the losers move toward $\boldsymbol{x}$ at a much slower rate than the winner.

This algorithm does not suffer from the problem of poor initialization of the representatives. This is because the second branch of (15.9) ensures that even if some representatives are initialized away from their closest vectors of X, they will eventually come closer to the region where the vectors of X are located.

15.3.3 Conscientious Competitive Learning Algorithms

Another way to utilize the representative power of $\boldsymbol{w}_j$'s is to discourage a representative from winning if it has won many times in the past. This is achieved by assigning a "conscience" to each representative. Several models of conscience have been proposed in the literature (e.g., [Gros 76a], [Gros 76b], [Gros 87], [Hech 88], [Chen 94], [Uchi 94]).

Perhaps the simplest way to implement this idea is to equip each representative, $\boldsymbol{w}_j$, $j = 1, \ldots, m$, with a counter f_j, that counts the times that $\boldsymbol{w}_j$ wins. One way

to proceed is the following [Ahal 90]. At the initialization stage (part (A)) we set $f_j = 1$, $j = 1, \ldots, m$. We define

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = d(\boldsymbol{x}, \boldsymbol{w}_j) f_j$$

and part (B) becomes the following:

- The representative $\boldsymbol{w}_j$ is the winner on $\boldsymbol{x}$ if

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d^*(\boldsymbol{x}, \boldsymbol{w}_k)$$

- $f_j(t) = f_j(t-1) + 1$.

This setup ensures that the distance is penalized to discourage representatives that have won many times. The parts (C) and (D) are the same as their corresponding parts of the basic competitive learning algorithm, and also $m = m_{init} = m_{max}$.

An alternative way is to utilize f_j via the equation [Chou 97]

$$f_j = f_j + d(\boldsymbol{x}, \boldsymbol{w}_j)$$

Other schemes of this kind may be found in [Ueda 94], [Zhu 94], [Butl 96], [Chou 97].

A different approach is followed in [Chen 94]. Here, in part (A), all f_j's are initialized to $1/m$. We define $d^*(\boldsymbol{x}, \boldsymbol{w}_j)$ as

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = d(\boldsymbol{x}, \boldsymbol{w}_j) + \gamma(f_j - 1/m)$$

where γ is a conscience parameter. Letting $z_j(\boldsymbol{x})$ be 1 if $\boldsymbol{w}_j$ wins on $\boldsymbol{x}$ and 0 otherwise, part (B) of the algorithm becomes

- The representative $\boldsymbol{w}_j$ is the winner on $\boldsymbol{x}$ if

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d^*(\boldsymbol{x}, \boldsymbol{w}_k)$$

- $f_j(t) = f_j(t-1) + \varepsilon(z_j(\boldsymbol{x}) - f_j(t-1))$

where $0 < \varepsilon \ll 1$. As we can easily observe, f_j increases for the winner and, in contrast to the previous case, decreases for the losers. Guidelines for the choice of the appropriate values of ε and γ, as well as a version of the algorithm where the value of γ is adaptively adjusted, are discussed in [Chen 94].

15.3.4 Competitive Learning–Like Algorithms Associated with Cost Functions

The basic philosophy behind the competitive learning schemes is to move representatives toward their closest points. If we want to express this in terms of a cost function, then a possible way is the following. Let us consider the cost function

$$J(\boldsymbol{W}) = \frac{1}{2m} \sum_{i=1}^{N} \sum_{j=1}^{m} z_j(\boldsymbol{x}_i) \|\boldsymbol{x}_i - \boldsymbol{w}_j\|^2 \tag{15.10}$$

where $\boldsymbol{W} = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_m^T]^T$ and $z_j(\boldsymbol{x}) = 1$, if $\boldsymbol{w}_j$ lies closer to $\boldsymbol{x}$, and 0 otherwise. This is basically the cost function associated with the isodata algorithm (Chapter 14), and it is not differentiable, due to the presence of $z_j(\boldsymbol{x})$. One way to overcome the problem of differentiability of $J(\boldsymbol{W})$ is to smooth $z_j(\boldsymbol{x})$. This implies that the concept of the competition is abandoned. Instead, each representative is updated in proportion to its distance from $\boldsymbol{x}$.

One way to smooth $z_j(\boldsymbol{x})$ is to redefine it as

$$z_j(\boldsymbol{x}) = \frac{\|\boldsymbol{x} - \boldsymbol{w}_j\|^{-2}}{\sum_{r=1}^{m} \|\boldsymbol{x} - \boldsymbol{w}_r\|^{-2}}, \quad j = 1, \ldots, m \tag{15.11}$$

where $\|\cdot\|$ is the Euclidean distance between two vectors. Clearly, $z_j(\boldsymbol{x})$ is no longer strictly equal to 0 or 1 but lies in [0, 1]. More specifically, the closer the $\boldsymbol{w}_j$ to $\boldsymbol{x}$, the larger the $z_j(\boldsymbol{x})$.

Using the preceding definition and after some rearrangements, $J(\boldsymbol{W})$ becomes

$$J(\boldsymbol{W}) = \frac{1}{2} \sum_{i=1}^{N} \left(\sum_{j=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{w}_j\|^{-2} \right)^{-1} \tag{15.12}$$

The gradient of $J(\boldsymbol{W})$ with respect to $\boldsymbol{w}_k$, $\partial J / \partial \boldsymbol{w}_k$, after some algebra, becomes

$$\frac{\partial J}{\partial \boldsymbol{w}_k} = -\sum_{i=1}^{N} z_k^2(\boldsymbol{x}_i)(\boldsymbol{x}_i - \boldsymbol{w}_k) \tag{15.13}$$

In the context of the gradient descent algorithms and using the "instantaneous" value of the gradient, as it was the case with the backpropagation algorithm, the following updating algorithm results.

$$\boldsymbol{w}_k(t) = \boldsymbol{w}_k(t-1) + \eta(t) z_k^2(\boldsymbol{x})(\boldsymbol{x} - \boldsymbol{w}_k(t-1)), \quad k = 1, \ldots, m \tag{15.14}$$

where $\boldsymbol{x}$ is the vector currently presented to the algorithm.

Notice that in this scheme *all* representatives are updated in proportion to their distance from $\boldsymbol{x}$. Thus, by smoothing $z_k(\boldsymbol{x})$, we end up with algorithms

that are competitive in a wider sense, for which general tools may apply for the establishment of their convergence properties.

Alternative choices of $z_j(\boldsymbol{x})$ and $J(\boldsymbol{W})$, leading to more general algorithmic schemes, are given in [Masu 93].

15.3.5 Self-Organizing Maps

So far, we have implicitly assumed that the representatives $\boldsymbol{w}_i$ are not interrelated. We will now remove this assumption. Specifically, for each representative $\boldsymbol{w}_j$ we define a neighborhood of representatives $Q_j(t)$, centered at $\boldsymbol{w}_j$ (Figure 15.8a and b). As t increases, $Q_j(t)$ shrinks and concentrates around $\boldsymbol{w}_j$. *The neighborhood is defined with respect to the indices j and it is independent of the distances between representatives in the vector space.* This situation is illustrated for $l = 1$ in Figure 15.8c, where it is assumed that the neighborhood of the jth representative contains the $j-1$ and the $j+1$ representatives, $j = 2, \ldots, 6$. Note that, although the second and fourth representatives are the neighbors of the third, the points $\boldsymbol{w}_6$ and $\boldsymbol{w}_7$ are closer in terms of the geometrical distance to $\boldsymbol{w}_3$.

If $\boldsymbol{w}_j$ wins on the current input vector $\boldsymbol{x}$ *all* the representatives in $Q_j(t)$ will be updated. This is the well-known Kohonen *self-organizing mapping (SOM) scheme* [Koho 89, Koho 95].

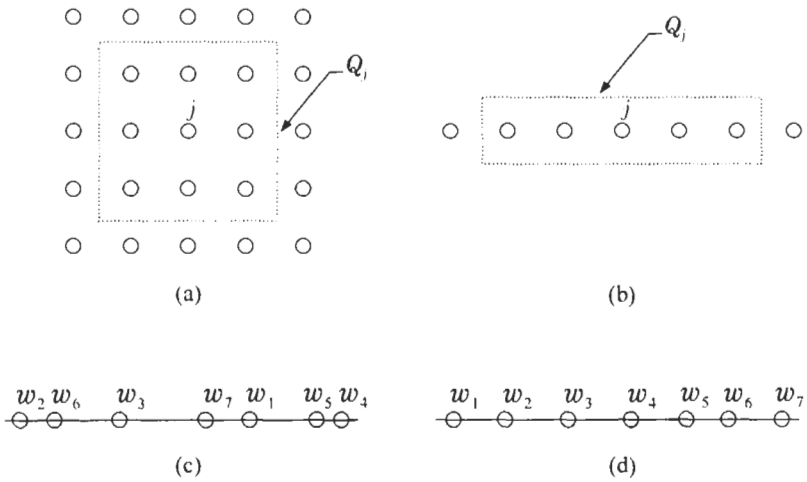


FIGURE 15.8: (a) A square-shaped 3×3 neighborhood. (b) A square-shaped 5×1 neighborhood. (c) The initial positions of the representatives on the real line ($l = 1$). (d) The final positions of the representatives after the application of the SOM algorithm.

In its simplest form, SOM may be viewed as a special case of the generalized competitive learning scheme (GCLS). Precisely, it is the same as the basic competitive learning algorithm as far as parts (A), (B), and (C) are concerned. However, part (D) is different. If $\boldsymbol{w}_j$ wins on $\boldsymbol{x}$, this part becomes

$$\boldsymbol{w}_k(t) = \begin{cases} \boldsymbol{w}_k(t-1) + \eta(t)(\boldsymbol{x} - \boldsymbol{w}_k(t-1)), & \text{if } \boldsymbol{w}_k \in Q_j(t) \\ \boldsymbol{w}_k(t-1), & \text{otherwise} \end{cases} \tag{15.15}$$

where $\eta(t)$ is a variable learning rate, which is chosen to satisfy the conditions in Section 15.3.1. The choices of $\eta(t)$ and $Q_j(t)$ are crucial for the convergence of the algorithm. *After convergence, the representatives $\boldsymbol{w}_i$ are topographically ordered and in a way representative of the distribution of the data. That is, neighboring representatives also lie "close" in terms of their distance in the vector space* (see Figure 15.8d). A suboptimal method for selecting the winner node that may lead to computational savings is presented in [Vish 00].

15.3.6 Supervised Learning Vector Quantization

A supervised variant of the competitive schemes has been suggested and extensively used in the context of VQ [Koho 89], [Kosk 92]. In this case, each cluster is treated as a class and the available vectors have known class labels. In this framework, let m be the number of classes. Supervised VQ *uses a set of m representatives, one for each class, and tries to place them in such a way that each class is "optimally" represented.* The simplest version of the supervised VQ (also called LVQ1 [Tsyp 73]) may be derived from generalized competitive learning schemes by keeping parts (A), (B), and (C) the same as in the basic competitive learning scheme and modifying part (D) to

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta(t)(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ correctly wins on } \boldsymbol{x} \\ \boldsymbol{w}_j(t-1) - \eta(t)(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ wrongly wins on } \boldsymbol{x} \\ \boldsymbol{w}_j(t-1), & \text{otherwise} \end{cases} \tag{15.16}$$

It is clear that the information related to the known class labels determines the direction in which $\boldsymbol{w}_j$ is moved. Specifically, we move $\boldsymbol{w}_j$ (a) toward $\boldsymbol{x}$ if $\boldsymbol{w}_j$ wins and $\boldsymbol{x}$ belongs to the jth class and (b) away from $\boldsymbol{x}$ if $\boldsymbol{w}_j$ wins and $\boldsymbol{x}$ does not belong to the jth class. In addition, all other representatives remain unaltered. Such algorithms have been used in speech recognition and OCR applications.

A variant of this scheme, where more than one representative is used to represent each class, is discussed in [Koho 89].

15.4 BRANCH AND BOUND CLUSTERING ALGORITHMS

As already stated in Chapter 5, *branch and bound* methods compute the globally optimal solution to combinatorial problems, according to a prespecified criterion (cost) function J, overcoming the need for exhaustive search.[4] They are applicable to monotonic criteria. That is, if k vectors of X have been assigned to clusters, the assignment of an extra vector to a cluster does not decrease the value of J.

We will first attempt to gain some insight into these methods by considering an example. Let us assume that our goal is to find the best way (with respect to a criterion J) in which three vectors can be clustered in two clusters. To this end, we construct the *classification tree* of Figure 15.9. Each node is characterized by a string of three symbols, namely 1, 2, and x. For example, the string "122" means that the first vector is assigned to cluster 1 while the other two are assigned to cluster 2. Also, the string "1xx" means that the first vector is assigned to cluster 1 while the other two remain unassigned. The first vector is always assigned to the first cluster. Note that each leaf corresponds to an actual clustering and there are as many leaves as the possible clusterings of the three vectors in two clusters. All the other nodes correspond to the so-called *partial clusterings*, that is, to clusterings where not all the vectors of X have been assigned yet to a cluster.

We are now ready to see where the computational saving comes from. Let us assume that at an iteration step of the algorithm, the best computed value for the criterion J is B. Then, if at a node the corresponding value of J is greater than B,

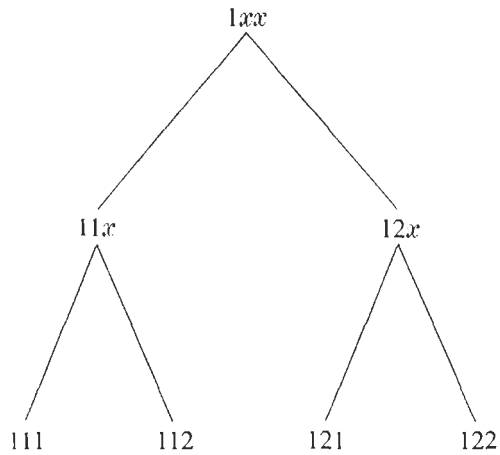


FIGURE 15.9: The classification tree corresponding to the grouping of three vectors in two clusters.

[4] In the sequel we consider only the minimization problem.

no further search is performed for all subsequent descendants springing from this node. This is because of the monotonicity of the criterion, which ensures that all descendants will result in values of J no less than B.

More formally, let $\mathcal{C}_r = [c_1, \ldots, c_r]$, $1 \leq r \leq N$, denote a partial clustering, where $c_i \in \{1, 2, \ldots, m\}$, $c_i = j$ if the vector $\boldsymbol{x}_i$ belongs to cluster C_j, and the vectors $\boldsymbol{x}_{r+1}, \ldots, \boldsymbol{x}_N$ are not yet assigned to any cluster.

In the sequel, we focus on compact clusters and we give a simple branch and bound algorithm [Koon 75]. We assume that the number of clusters, m, is fixed. The criterion function employed is defined as

$$J(\mathcal{C}_r) = \sum_{i=1}^{r} \|\boldsymbol{x}_i - \boldsymbol{m}_{c_i}(\mathcal{C}_r)\|^2 \tag{15.17}$$

where $\boldsymbol{m}_{c_i}$ is the mean vector of the cluster C_{c_i}, that is,

$$\boldsymbol{m}_j(\mathcal{C}_r) = \frac{1}{n_j(\mathcal{C}_r)} \sum_{\{q=1,\ldots,r: c_q = j\}} \boldsymbol{x}_q, \qquad j = 1, \ldots, m \tag{15.18}$$

with $n_j(\mathcal{C}_r)$ being the number of vectors $\boldsymbol{x} \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r\}$ that belong to cluster C_j. Note that the computation of the mean vectors of the clusters takes into account only the first r vectors. We assume that $J(\mathcal{C}_1) = 0$. One can easily verify that

$$J(\mathcal{C}_{r+1}) = J(\mathcal{C}_r) + \Delta J(\mathcal{C}_r) \tag{15.19}$$

with $\Delta J(\mathcal{C}_r) \geq 0$. In words, $\Delta J(\mathcal{C}_r)$ denotes the increase in the value of J when the next vector is assigned to a cluster. More precisely, assuming that the $r + 1$ vector is assigned to the cluster C_j, it can be shown that

$$\Delta J(\mathcal{C}_r) = \frac{n_j(\mathcal{C}_r)}{n_j(\mathcal{C}_r) + 1} \|\boldsymbol{x}_{r+1} - \boldsymbol{m}_j(\mathcal{C}_r)\|^2 \tag{15.20}$$

Let $\mathcal{C}_N^* = [c_1^*, \ldots, c_N^*]$ denote the optimal clustering. In the sequel, the index r denotes the vector that is currently considered for cluster assignment. Then the algorithm may be stated as follows.

Branch and Bound Clustering (BBC) Algorithm

$r = 1$

$B = +\infty$

While $r \neq 0$ do

- If $(J(C_r) < B)$ AND $(r < N)$ then
  - —$r = r + 1$
  - —Among all possible assignments, c_r, of $\boldsymbol{x}_r$ that have not been tested yet, choose the one that minimizes the value of $\Delta J(C_r)$.[5]
- End {If}
- If $(J(C_r) < B)$ AND $(r = N)$ then
  - —$C_N^* = C_N$
  - —$B = J(C_N)$
- End {If}
- If $(J(C_r) \leq B)$ AND $(r = N))$ OR $(J(C_r) > B))$ then
  - —(A) $r = r - 1$
  - —If $(r = 0)$ then
    - * Stop
  - —Else
    - * If all possible clusterings that branch from this node have been exhausted for the rth vector then
      - Go to (A)
    - * Else
      - $r = r + 1$
      - Among all possible c_r's that have not been tested yet, choose another path, the one that minimizes the value of $\Delta J(C_r)$.
    - * End {If}
  - —End {If}
- End {If}

End {While}

The algorithm starts from the initial node of the tree and goes down until either (i) a leaf or (ii) a node q with cost function value greater than B is encountered. In case (i) if the cost for the clustering that corresponds to that leaf is less than B, then this cost becomes the new bound B, and the clustering is the best clustering found so far. In case (ii) all subsequent clusterings branching from q are not considered any further and we say that they are exhausted. The algorithm, then, backtracks to the parent node of q in order to span a different path. If all paths branching from the parent of q have already been considered, we move to the grandparent of q. The algorithm terminates when all possible paths have been considered explicitly or implicitly (via the aforementioned case (ii)). Clearly, in the beginning, the BBC algorithm spans first a whole path from the initial node of the tree down to a leaf. The cost function of the clustering corresponding to that leaf is the new value of B.

[5] If more than one c_r's minimize $\Delta J(C_r)$, choose the smallest one.

It is clear that the tighter the upper bound B, the more paths are rejected without explicit consideration. Variations of this algorithm that use better estimates of B are discussed in [Koon 75]. Moreover, the substitution of the values $J(C_r)$ with tighter lower bounds of the optimal value of J is also suggested in [Koon 75]. This leads to the rejection of many more clusterings without considering them explicitly.

The major disadvantage of this algorithm is the excessive amount of computational time it requires even for moderate values of N.[6] In addition, this time is unpredictable. One way to face this problem is to run the algorithm for a prechosen time and use the best clustering found so far. It is clear that in this case, the algorithm can no longer guarantee the determination of the globally optimal clustering.

15.5 BINARY MORPHOLOGY CLUSTERING ALGORITHMS (BMCAs)

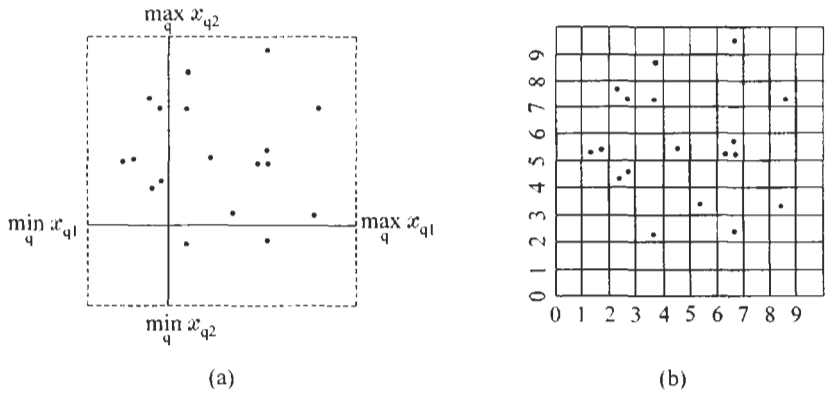
Algorithms of this type are suitable for cases in which clusters are not properly represented by a single representative ([Post 93], [Mora 00]). The idea here is to map X to a discrete set S that facilitates the clustering procedure and then use the identified clusters in S as a guide for the identification of the clusters in X. In the sequel, we describe such an algorithm, called the binary morphology clustering algorithm [Post 93]. The BMCA involves four main stages. During the first stage the data set is discretized and a new set is derived. This is the so-called *discrete binary (DB) set*. During the second stage, the basic morphological operators (opening and closing) are applied on the DB set, giving rise to a new discrete set. The third stage reveals the clusters formed in the last set. Finally, the last stage is responsible for the identification of the clusters formed by the original vectors of X, using as guide the clusters discovered during the third stage. Before we present the algorithm, let us first recall some basic tools and definitions.

15.5.1 Discretization

During the first step of the discretization stage, we normalize the vectors $\boldsymbol{x} \in X$ so that all their coordinates lie in the range $[0, r-1]$, where r is a user-defined parameter. This is achieved via the following transformation:

$$y_{ij} = \frac{x_{ij} - \min_{q=1,\ldots,N} x_{qj}}{\max_{q=1,\ldots,N} x_{qj} - \min_{q=1,\ldots,N} x_{qj}}(r-1), \quad i = 1, \ldots, N, j = 1, \ldots, l \tag{15.21}$$

[6] This is a common feature for the methods performing global optimization.



(a) (b)

FIGURE 15.10: (a) The original data set X. (b) Normalization of the data set X in $[0, r-1]^l$, with $r = 10$, and discretization. The nonempty hypercubes define the discrete binary set.

where x_{ij} denotes coordinate j of vector i. Let us denote the resulting set by X', that is,

$$X' = \{\mathbf{y}_i \in [0, r-1]^l, i = 1, \ldots, N\}$$

Next, we discretize $[0, r-1]^l$ into r^l hypercubes. This is achieved by segmenting the $[0, r-1]$ interval, for each coordinate, into r subintervals (see Figure 15.10). Each hypercube is identified by the coordinates of its lower left corner. The parameter r defines the "resolution" of $[0, r-1]^l$.

In the sequel, we identify the hypercube where each vector $\mathbf{y}_i \in X'$ lies. This can be accomplished by simply taking the integer part of each coordinate of $\mathbf{y}_i$. The resulting vector, $\mathbf{z}_i$, will be the identity label of a hypercube in the defined grid. More specifically,

$$z_{ij} = [y_{ij}], \qquad i = 1, \ldots, N, \quad j = 1, \ldots, l$$

where $[x]$ denotes the integer part of x. Let S be the set containing the new vectors $\mathbf{z}_i$, after removing all duplicates. Thus, each element of S corresponds to a nonempty hypercube, and S is the discrete binary set.

15.5.2 Morphological Operations

These operations are applied only to sets with discrete-valued vectors. The simplest operations of this kind are *dilation* and *erosion*. Based on these two operations, *opening* and *closing* operations are defined.

Let Y and T be subsets of Z^l, where Z is the set of integers and s a vector in Z^l. The *translation of Y by s* is defined as

$$Y_s = \{t \in Z^l : t = x + s,\ x \in Y\} \tag{15.22}$$

Definition 1. *The dilation of Y by T, denoted by $Y \oplus T$, is defined as*

$$Y \oplus T = \{e \in Z^l : e = x + s, x \in Y,\ s \in T\} \tag{15.23}$$

Equivalently, the set $Y \oplus T$ is determined by translating Y by all elements of T and taking the union of the resulting sets [Gonz 93].

Definition 2. *The erosion of Y by T is denoted by $Y \ominus T$ and is defined as*

$$Y \ominus T = \{f \in Z^l : x = f + s, x \in Y,\ \forall s \in T\} \tag{15.24}$$

Equivalently, the set $Y \ominus T$ is determined by translating Y by all elements of T and taking the intersection of the resulting sets [Gonz 93].

In both of these cases, T is called the *structuring element*. Usually, it has a hypercubical shape (see Figure 15.11) but other choices, such as hyperspherical shape, are also possible.

Example 15.3. Let us consider a two-dimensional normal density function with mean $\mu = [0, 0]^T$ and covariance matrix $\Sigma = 3I$, where I is the 2×2 identity matrix. Let X be a set containing 200 vectors stemming from this distribution (Figure 15.12a). We apply the discretization process on X, with $r = 20$, in order to obtain the corresponding discrete

(a) (b)

FIGURE 15.11: (a) A squared 3×3 structuring element. (b) A squared 5×5 structuring element.

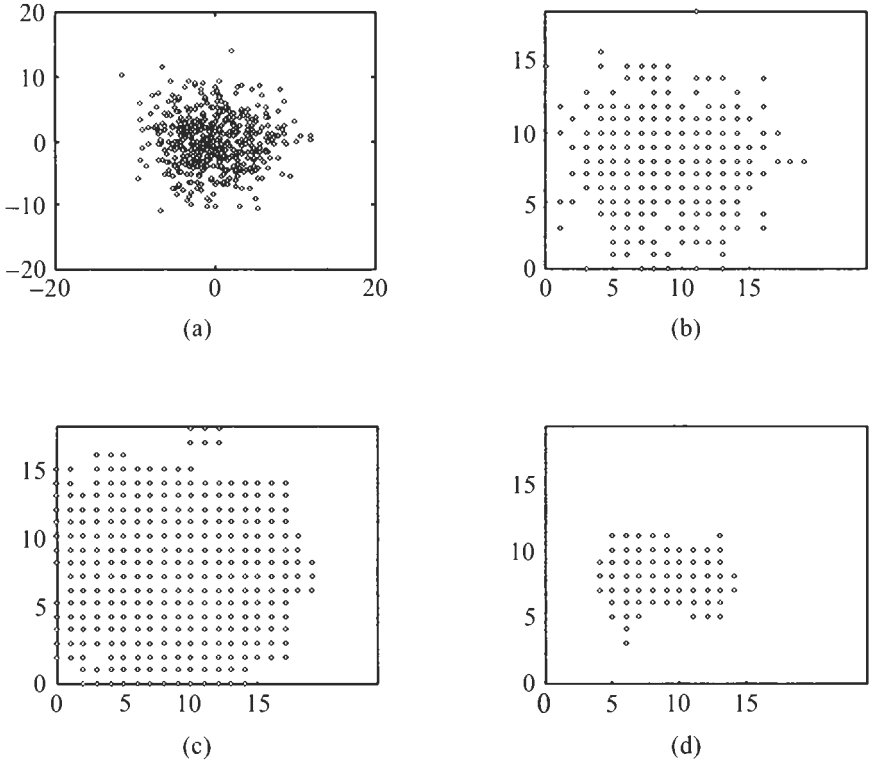


FIGURE 15.12: (a) The original data set X given in Example 15.3. (b) The discrete binary set Y derived from X. (c) The set Y dilated by T. (d) The set Y eroded by T.

binary set, which is denoted by Y (Figure 15.12b). Let T be the 3×3 structuring element shown in Figure 15.11a, which consists of the points

$$T = \{t_1, t_2, \ldots, t_9\}$$
$$= \{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)\}$$

In order to derive the dilation of Y by T, we compute the sets Y_i, $i = 1, \ldots, 9$, produced by the translation of Y by each element t_i of T, $i = 1, \ldots, 9$, and in the sequel, we take the union of all Y_i's. This is the dilation of Y by T (Figure 15.12c). The erosion of Y by T is computed by taking the intersection of all Y_i's defined above. The result of this operation is shown in Figure 15.12d.

Opening and *closing* are two additional basic operations that are defined in terms of the dilation and the erosion.

Definition 3. *The opening of Y by T is denoted by Y_T and is defined as*

$$Y_T = (Y \ominus T) \oplus T \tag{15.25}$$

that is, the opening of Y by T is the erosion of Y by T followed by the dilation of the resulting $Y \ominus T$ by T.

Definition 4. *The closing of Y by T is denoted by Y^T and is defined as*

$$Y^T = (Y \oplus T) \ominus T \tag{15.26}$$

that is, the closing of Y by T is the dilation of Y by T followed by the erosion of the resulting $Y \oplus T$ by T.

In general, Y is different from Y_T and Y^T. Note that the opening operation smooths out the boundary of Y by discarding irrelevant details of it. On the other hand, the closing operation fills the gaps in the set Y. These observations show that *opening and closing tend to produce new sets with simpler shapes than the original ones.* As pointed out in [Post 93], *"Opening and closing seem to be very effective to eliminate isolated groups of set points and holes, provided that these details do not exceed the size of the structuring element."* The following example shows how the opening and closing operations work.

Example 15.4. Let us consider the discrete binary set Y (see Figure 15.12b) and the structuring element T of Example 15.3. We derive first the opening of Y by T. The result is shown in Figure 15.13a. As one can observe, the resulting set retains the basic shape of Y, while irrelevant details of the boundary of Y have been discarded. The action of closing Y by T is shown in Figure 15.13b.

The above arguments indicate that the structuring element T plays an important role in the outcome of the above operations. Unfortunately, there are no general guidelines for choosing the appropriate T.

15.5.3 Determination of the Clusters in a Discrete Binary Set

We begin with a description of a rather simple algorithm suitable for clusters formed by the points of a discrete-valued data set $S \subset \{0, 1, \ldots, r-1\}^l$. Let us first define the neighborhood, $V(\boldsymbol{x})$, of a point $\boldsymbol{x} \in S$ as

$$V(\boldsymbol{x}) = \{\boldsymbol{y} \in S - \{\boldsymbol{x}\}: d(\boldsymbol{x}, \boldsymbol{y}) \leq d_q\}$$

where d may be any distance measure between two points (see Chapter 11) and d_q is a distance threshold. Also, let θ be a threshold of the density of the

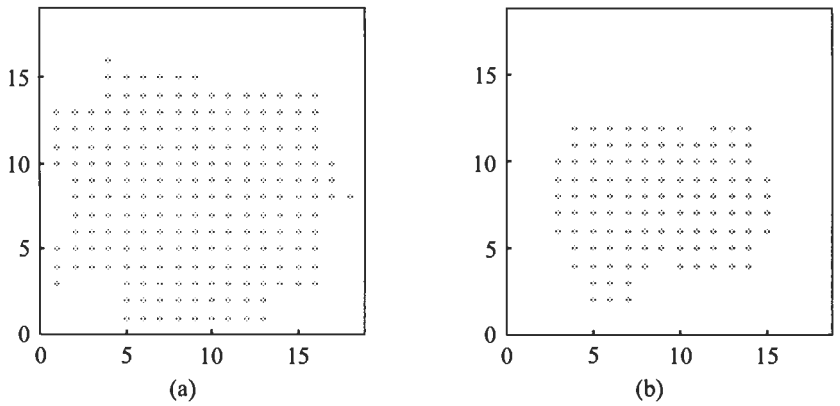


FIGURE 15.13: (a) The set Y given in Example 15.3, opened by T. (b) The set Y given in Example 15.3, closed by T.

neighborhood $V(\boldsymbol{x})$ of a point $\boldsymbol{x}$. That is, if $V(\boldsymbol{x})$ contains at least θ points of S, it is considered "dense." These are user-defined parameters.

Let $U(\boldsymbol{x})$ be the immediate neighborhood of $\boldsymbol{x}$, that is, the set that contains all points lying at a (Euclidean) distance less than or equal to $\sqrt{l}$ from $\boldsymbol{x}$.

Cluster Detection Algorithm for Discrete-Valued Sets (CDADV)

- Initially no vector is considered as processed.
- Repeat
  - —Choose a nonprocessed point $\boldsymbol{x}$ of S.
  - —Determine the neighborhood $V(\boldsymbol{x})$.
  - —If $V(\boldsymbol{x})$ contains at least θ points then
    - * Create a new cluster that includes:
      - The point $\boldsymbol{x}$
      - All points $\boldsymbol{y} \in S$ for which there exists a sequence of points $\boldsymbol{y}_{j_s} \in S$, $s = 1, \ldots, q_y$, such that $\boldsymbol{y} \in U(\boldsymbol{y}_{j_1})$, $\boldsymbol{y}_{j_s} \in U(\boldsymbol{y}_{j_{s+1}})$, $s = 1, \ldots, q_y - 1$, and $\boldsymbol{y}_{j_{q_y}} \in U(\boldsymbol{x})$.
    - * The defined points are considered as processed.
  - —Else
    - * Consider $\boldsymbol{x}$ as processed
  - —End {if}
- Until all points of S have been processed.

Example 15.5. Consider the setup of Figure 15.14a. We choose $d_q = \sqrt{2}$ and $\theta = 1$ (in this case $V(\boldsymbol{x}) \equiv U(\boldsymbol{x})$). Also, the sides of the squares depicted in Figure 15.14a are of unit length. The CDADV considers first $\boldsymbol{x}_1$. Since it is "dense" (i.e., its neighborhood contains

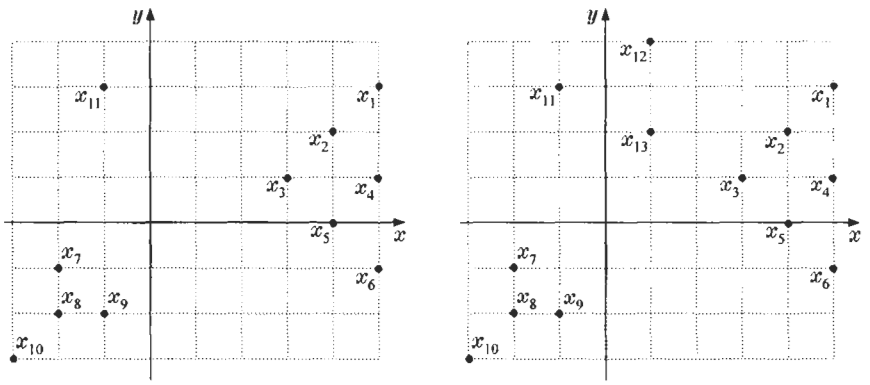


FIGURE 15.14: (a) The setup of Example 15.5. (b) A data set containing outliers.

at least one point of S apart from $\boldsymbol{x}_1$), a new cluster is created. $\boldsymbol{x}_2$ also belongs to this cluster because $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$. Moreover, $\boldsymbol{x}_3, \boldsymbol{x}_4$ belong to this cluster because $\boldsymbol{x}_3, \boldsymbol{x}_4 \in U(\boldsymbol{x}_2)$ and $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$. In addition, $\boldsymbol{x}_5$ belongs to this cluster because $\boldsymbol{x}_5 \in U(\boldsymbol{x}_3)$, $\boldsymbol{x}_3 \in U(\boldsymbol{x}_2)$, and $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$. Finally, since $\boldsymbol{x}_6 \in U(\boldsymbol{x}_5)$, $\boldsymbol{x}_5 \in U(\boldsymbol{x}_4)$, $\boldsymbol{x}_4 \in U(\boldsymbol{x}_2)$, and $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$, $\boldsymbol{x}_6$ also belongs to this cluster. Working similarly, we find that $\boldsymbol{x}_7, \boldsymbol{x}_8, \boldsymbol{x}_9$, and $\boldsymbol{x}_{10}$ form a second cluster, while no action is taken for $\boldsymbol{x}_{11}$.

In the preceding scheme, all points of S are processed by the algorithm, regardless of their density. In fact, CDADV works well when the points of S form well-separated clusters. However, if this is not the case, for example, when we have a small number of outliers in S lying between its clusters (Figure 15.14b), we may proceed as follows. As it is expected, the neighborhood of an outlier is rather "sparse", and we first define a lower threshold θ_1 ($\leq\theta$) for the density of the neighborhood of a point and we consider the set S' of all points of S whose neighborhoods have density at least θ_1. That is, we exclude the outliers from S'. Then we run the CDADV on S' using θ, and, after its completion, we assign each vector of $S - S'$ to the cluster where its nearest point in S' belongs. Note that the distance function between a point and a cluster that we employ for this stage should not involve cluster representatives, since S is a discrete-valued set.

15.5.4 Assignment of Feature Vectors to Clusters

This subsection deals with the final stage of the algorithmic procedure. Let us recall that S is the discrete-valued set obtained from X, after applying the opening and closing transformations. Let $C'_1, \ldots, C'_m$ be the clusters formed in S, determined by the previously discussed CDADV algorithm. The aim of the current task is to

determine m clusters in X, denoted by $C_1, \ldots, C_m$, that correspond to the clusters C_i', $i = 1, \ldots, m$.

The algorithm assigns the vectors of X to clusters in two steps. During the first one, all vectors $\boldsymbol{x} \in X$ such that $[\boldsymbol{y}] \in S$[7] are considered. In the sequel, the algorithm assigns a vector $\boldsymbol{x}$ to the cluster C_i if $[\boldsymbol{y}]$ belongs to C_i'. Let X' be the set of the vectors of X that have been assigned to clusters during this step. At the second stage, the algorithm assigns each of the points in $X - X'$ to its closest C_j, $j = 1, \ldots, m$.

15.5.5 The Algorithmic Scheme

Having described the steps, we may proceed now to the description of the BMCA.

Binary Morphology–Based Clustering Algorithm (BMCA)

- *1st stage.* Discretize the data set X and let S be the resulting discrete binary set.
- *2nd stage.*
  - —(a) Apply the opening transformation on S using a preselected structuring element T, to obtain S_T.
  - —(b) Apply the closing transformation on S_T using T. Let $S_1 = (S_T)^T$ be the set obtained.
- *3rd stage.* Determine the underlying clusters of S_1 using the CDADV algorithm.
- *4th stage.* Based on the clusters formed in S_1, determine the underlying clusters of X.

It should be noted here that different choices of morphological operators can be used at the second stage of the algorithm. Thus, for example, one may use either the opening or the closing operator or both of them in the reverse order.

BMCA is sensitive to the parameter r and the structuring element T. These parameters may cause overestimation or underestimation of the true number of clusters underlying X. However, it is expected that when X contains clusters, their number remains unchanged for a significant range of values of the parameters involved (a similar situation has been met earlier in Chapter 12). Based on this assumption, we run the first three stages of the algorithm for various values of r and different T (for simplicity we may assume that T has a hypercubic scheme and, thus, its only parameter that is subject to change is the length of its side a). Then we plot the number of the resulting clusters versus r and a and we consider the widest area in the (r, a) plane, for which the number of clusters remains unchanged. The

[7] By $[\boldsymbol{y}]$ we denote the l-dimensional vector whose ith coordinate is the integer part of the ith coordinate of the l-dimensional vector $\boldsymbol{y}$.

final values for r and a are chosen to be those corresponding to the middle point of the area. Using these values, we run the BMCA algorithmic scheme in order to determine the clusters of X.

A major drawback of the procedure is that it requires intensive computations, since many combinations of the values for r and a have to be considered. A way to reduce the required computations is to fix one of the two parameters to a reasonable value (if this is possible) and to apply the procedure only to the other parameter.

An important observation is that when the underlying clusters of X are compact and well separated, algorithms such as the isodata give better results than BMCA (see Problem 15.6). However, the situation is reversed when this is not the case. Let us consider the following example.

Example 15.6. Let X be a data set consisting of 1000 vectors. The first 500 of them, (x_{i_1}, x_{i_2}), are defined as

$$x_{i_1} = (i-1)\frac{2s}{500}$$

$$x_{i_2} = \sqrt{s^2 - x_{i_1}^2} + z_i$$

where $s = 10$ and z_i stems from a Gaussian distribution with zero mean and unit variance, $i = 1, \ldots, 500$. Similarly, the remaining vectors (x_{i_1}, x_{i_2}) are defined as

$$x_{i_1} = b_1 + (i-501)\frac{2s}{500}$$

$$x_{i_2} = b_2 + \sqrt{s^2 - (x_{i_1} - b_1)^2} + z_i'$$

where $b_1 = -10$, $b_2 = 3$, $s = 10$, and z_i' is normally distributed with zero mean and unit variance, $i = 501, \ldots, 1000$. It is not difficult to realize that the first 500 feature vectors spread around the upper half-circle with radius 10 centered at $(0, 0)$. Similarly, the rest of the 500 vectors spread around the lower half-circle with radius 10, centered at the point $(-10, 3)$ (see Figure 15.15a).

Clearly, two clusters are formed in X and each of them cannot be represented satisfactorily by a single point representative. In our simulations we use the 3×3 structuring element defined in Example 15.3 and $r = 25$. Also, the Euclidean distance between two vectors is adopted. The discrete binary set S, resulting from the discretization process, is shown in Figure 15.15b. Figure 15.15c shows the result of the opening of S by T, and Figure 15.15d shows the result of the closing of S_T by T. The two clusters involved in the last set are well separated. Application of the third stage of the BMCA algorithmic scheme reveals these two clusters. Finally, application of the fourth stage of the algorithm determines the clusters formed in X. The results obtained are excellent. Only two of the first 500 vectors were misclassified, and only 1 of the remaining 500 vectors was misclassified. Thus, 99.7% of the vectors of X were correctly classified. In contrast, the results obtained with the isodata algorithm were much inferior to these results.

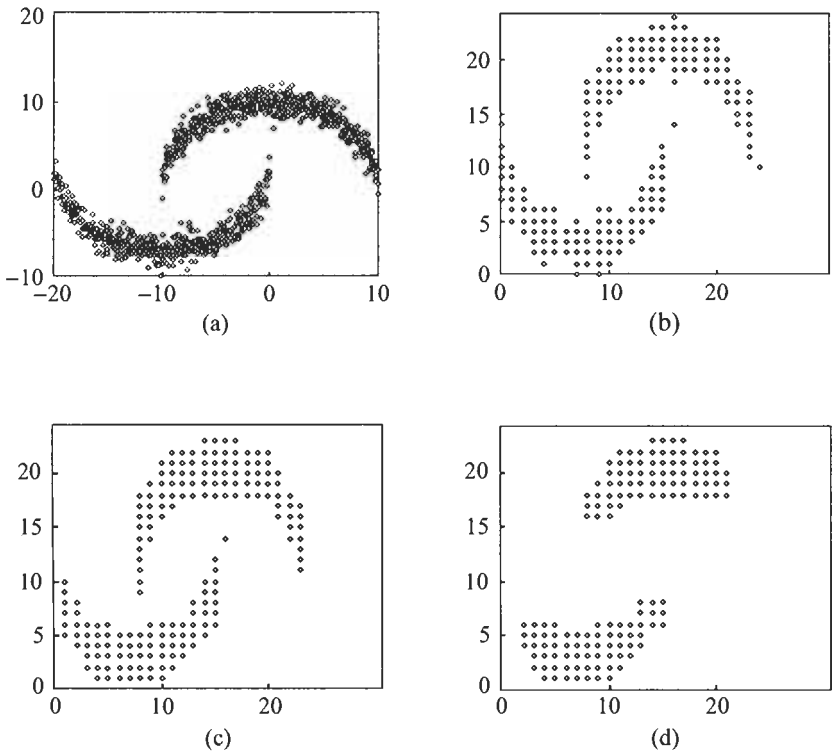


FIGURE 15.15: (a) The original data set X. (b) The set S resulting from the discretization of X. (c) The set resulting from opening of S by T. (d) The set resulting from closing of S_T by T.

15.6 BOUNDARY DETECTION ALGORITHMS

Most of the algorithms discussed so far determine clusters based on either the distance between vectors and clusters or the distance between clusters. In this section, a different rationale is discussed. Clusters are formed via the estimation of the boundary surfaces that separate them [Atiy 90]. This approach is well suited when the underlying clusters are compact. The idea is rather simple. The compact clusters are viewed as dense regions, in the l-dimensional space, separated by regions sparse in data vectors. Therefore, it suffices to begin with an initial estimate of the boundary and move it iteratively to regions that are sparse in vectors.

Let us consider first the case in which two clusters are present. Let $g(\boldsymbol{x}; \boldsymbol{\theta})$ be the function describing the decision boundary between the two clusters, where $\boldsymbol{\theta}$ is the unknown parameter vector describing the surface. If, for a specific $\boldsymbol{x}$, $g(\boldsymbol{x}; \boldsymbol{\theta}) > 0$, then $\boldsymbol{x}$ belongs to the first cluster, denoted by C^+. Otherwise, $\boldsymbol{x}$ belongs to the

second cluster, denoted by C^-. The goal is to determine the unknown parameter vector $\boldsymbol{\theta}$. The situation looks similar to the supervised case where we identify the decision boundary between classes, utilizing the labelling information of the feature vectors. However, no such information is available here. In the present case, *the adjustment of $\boldsymbol{\theta}$ relies exclusively on the distances of the vectors of X from the decision boundary.*

To this end, we define a cost function J, whose maximization will lead to locally optimal values for $\boldsymbol{\theta}$. Let J be defined as

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} f^2(g(\boldsymbol{x}_i;\boldsymbol{\theta})) - \left(\frac{1}{N}\sum_{i=1}^{N} f(g(\boldsymbol{x}_i;\boldsymbol{\theta}))\right)^{2q} \tag{15.27}$$

where q is a constant positive integer and $f(x)$ is a monotonically increasing symmetric squashing function with

$$\lim_{x\to+\infty} f(x) = 1, \qquad \lim_{x\to-\infty} f(x) = -1, \quad \text{and} \quad f(0) = 0 \tag{15.28}$$

A common choice for such a function is the hyperbolic tangent

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Each of the two terms in (15.27) has a maximum value of 1. Also, $J(\boldsymbol{\theta})$ is nonnegative since

$$J(\boldsymbol{\theta}) \geq \frac{1}{N}\sum_{i=1}^{N}\left(f(g(\boldsymbol{x}_i;\boldsymbol{\theta})) - \frac{1}{N}\sum_{k=1}^{N} f(g(\boldsymbol{x}_k;\boldsymbol{\theta}))\right)^2 \geq 0 \tag{15.29}$$

One can easily observe that the first term in Eq. (15.27) is maximized when all $\boldsymbol{x} \in X$ lie away from the boundary. In this case $f^2(g(\boldsymbol{x}_i;\boldsymbol{\theta})) \to 1$ and the first term attains values close to 1. However, this argument holds also true if *all* vectors of X lie on the same side of the boundary and away from it. The role of the second term is to discourage such trivial solutions. Indeed, in such cases $(\frac{1}{N}\sum_{i=1}^{N} f(g(\boldsymbol{x}_i;\boldsymbol{\theta})))^{2q} \to 1$ and therefore $J(\boldsymbol{\theta})$ approaches zero, its minimum value. The role of q is to control the impact of the second term on the cost function J.

Let us now consider an intermediate case in which the boundary lies between two dense regions. In such cases, the contribution of the second term to J is small. Let us demonstrate it via a simplified example. Assume that the decision surface is a hyperplane H and that at the positive (negative) side of H we have k points lying at distance a $(-a)$ away from it. Then it is not difficult to show that the second term becomes zero while the first equals $f^2(a\|\boldsymbol{\theta}\|)$.

In the sequel we adopt a steepest ascent scheme in order to determine the optimal value for $\boldsymbol{\theta}$. Let θ_j be a coordinate of $\boldsymbol{\theta}$. Then

$$\theta_j(t+1) = \theta_j(t) + \mu \left. \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} \right|_{\theta_j = \theta_j(t)}$$

or

$$\theta_j(t+1) = \theta_j(t) + \mu \left(\frac{2}{N} \sum_{i=1}^{N} f(g(\boldsymbol{x}_i; \boldsymbol{\theta})) \frac{\partial f(g(\boldsymbol{x}_i; \boldsymbol{\theta}))}{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})} \frac{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})}{\partial \theta_j} \right.$$
$$\left. - \frac{2q}{N} \left(\frac{1}{N} \sum_{i=1}^{N} f(g(\boldsymbol{x}_i; \boldsymbol{\theta})) \right)^{2q-1} \sum_{i=1}^{N} \frac{\partial f(g(\boldsymbol{x}_i; \boldsymbol{\theta}))}{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})} \frac{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})}{\partial \theta_j} \right) \Bigg|_{\theta_j = \theta_j(t)} \tag{15.30}$$

For the simple case that $g(\boldsymbol{x}; \boldsymbol{\theta})$ is a hyperplane we can write

$$g(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

where $\boldsymbol{\theta} \equiv [\boldsymbol{w}\ w_0]^T$. The updating equation for the parameters follows directly from Eq. (15.30) if we notice that

$$\frac{\partial g(\boldsymbol{x}; \boldsymbol{\theta})}{\partial w_j} = \begin{cases} x_j, & j = 1, \ldots, l \\ 1, & j = 0 \end{cases}$$

The resulting algorithm is rather simple in its formulation and may be stated as follows

Boundary Detection Algorithm (BDA)

- Choose an initial value $\boldsymbol{\theta}(0)$ for the parameter vector.
- Compute $J(\boldsymbol{\theta}(0))$ using Eq. (15.27).
- $t = 0$
- Repeat
  - —$t = t + 1$
  - —Compute $\boldsymbol{\theta}(t)$ using Eq. (15.30).
  - —Compute $J(\boldsymbol{\theta}(t))$ using Eq. (15.27).
- Until $\left| \frac{J(\boldsymbol{\theta}(t)) - J(\boldsymbol{\theta}(t))}{J(\boldsymbol{\theta}(t)) - J(\boldsymbol{\theta}(t-1))} \right| < \varepsilon$.

We note here that the coordinates of $\boldsymbol{\theta}$ should not grow in an unbounded way. In the case that $g(\boldsymbol{x}; \boldsymbol{\theta})$ corresponds to a hyperplane, a bounded condition for the coordinates of $\boldsymbol{\theta}$ could be $\|\boldsymbol{\theta}\| \leq a$, where a is a user-defined parameter.

Let us now consider the case in which more than two clusters underlie X. In this case, we follow a hierarchical procedure. First we divide X into two clusters X^+ and X^- using the boundary detection algorithm. Then, using the algorithm again, we further divide X^+ (X^-) and obtain X^{+-} and X^{++} (X^{--} and X^{-+}). This procedure is then applied iteratively to the resulting clusters until a specific termination criterion is met. This procedure reminds us of the neural network design discussed in Chapter 4 [Atiy 90].

One can easily observe that if no sparse regions exist in a formed cluster C, the division of C will result in a low value of $J(\boldsymbol{\theta})$. Thus, an appropriate criterion that may be used to check whether C contains more clusters is

$$J(\boldsymbol{\theta}) \leq b$$

where b is a user-defined threshold. If this happens, the division of C is not acceptable. Otherwise, the division of C is accepted and we proceed with C^+ and C^-. It is clear that the smaller the value of b, the more clusters will be defined. On the other hand, higher values of b result in the acceptance of fewer clusters with well-defined borders among them. Thus, b should be chosen with care.

15.7 VALLEY-SEEKING CLUSTERING ALGORITHMS

The method discussed here is in the same spirit as that of the previous section. Let $p(\boldsymbol{x})$ be the density function describing the distribution of the vectors in X. An alternative way to attack the clustering problem is to view the clusters as peaks of $p(\boldsymbol{x})$ separated by valleys. Inspired by this consideration, one can search to identify such valleys, and try to move and place the borders of the clusters in these valleys.

In the sequel, we discuss an iterative and computationally effective algorithm based on this idea [Fuku 90]. Once more, let $V(\boldsymbol{x})$ be the local region of $\boldsymbol{x}$, that is,

$$V(\boldsymbol{x}) = \{\boldsymbol{y} \in X - \{\boldsymbol{x}\}: d(\boldsymbol{x}, \boldsymbol{y}) \leq a\} \tag{15.31}$$

where a is a user-defined parameter. The distance $d(\boldsymbol{x}, \boldsymbol{y})$ can be taken to be

$$d(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{y} - \boldsymbol{x})^T A(\boldsymbol{y} - \boldsymbol{x}) \tag{15.32}$$

where A is a symmetric positive definite matrix. Also, let k_j^i denote the number of vectors of the j cluster that belong to $V(\boldsymbol{x}_i)$, excluding $\boldsymbol{x}_i$. Also, let $c_i \in \{1, \ldots, m\}$ denote the cluster to which $\boldsymbol{x}_i$ belongs, $i = 1, \ldots, N$. Then the algorithm is stated as follows.

Valley-Seeking Algorithm

- Fix a.
- Fix the number of clusters m.
- Define an initial clustering of X.
- Repeat
  - —For $i = 1$ to N
    - * Find j: $k_j^i = \max_{q=1,\ldots,m} k_q^i$ [8]
    - * Set $c_i = j$
  - —End {For}
  - —For $i = 1$ to N
    - * Assign $\boldsymbol{x}_i$ to cluster C_{c_i}.
  - —End {For}
- Until no reclustering of vectors occurs.

Remarks

- Observe that the preceding algorithm has close similarities to the Parzen windows method, for pdf estimation, discussed in Chapter 2. Indeed, all it does is to move a window $d(\boldsymbol{x}, \boldsymbol{y}) \leq a$ at $\boldsymbol{x}$ and to count points from different clusters. Then it assigns $\boldsymbol{x}$ to the cluster with the larger number of points in the window. That is, to the cluster with the highest local pdf density. This is equivalent with moving the boundary away from the "winning" cluster.
- The preceding is a mode seeking algorithm. That is, if more than enough clusters are initially appointed, some of them may be empty.

Example 15.7. (a) Consider Figure 15.16a. X consists of the following 10 vectors: $\boldsymbol{x}_1 = [0, 1]^T$, $\boldsymbol{x}_2 = [1, 0]^T$, $\boldsymbol{x}_3 = [1, 2]^T$, $\boldsymbol{x}_4 = [2, 1]^T$, $\boldsymbol{x}_5 = [1, 1]^T$, $\boldsymbol{x}_6 = [5, 1]^T$, $\boldsymbol{x}_7 = [6, 0]^T$, $\boldsymbol{x}_8 = [6, 2]^T$ $\boldsymbol{x}_9 = [7, 1]^T$, $\boldsymbol{x}_{10} = [6, 1]^T$. The squared Euclidean distance is employed.

The initial clustering consists of two clusters as shown in Figure 15.16a. Also, a decision line, b_1, separating the two clusters is shown (Figure 15.16a). Let $a = 1.415$.[9] After the first iteration of the algorithm, $\boldsymbol{x}_4$ is assigned to the cluster denoted by "x." This is equivalent to moving the decision curve separating the two clusters to the valley between the two high density areas.

(b) Now consider Figure 15.16b. The set X remains unchanged. Also, the initial clustering remains the same except that $\boldsymbol{x}_6$ is assigned to the cluster denoted by "x." Three curves, b_1, b_2, and b_3, can now be used to separate the clusters. After the first iteration of the algorithm,

[8] If ties occur, that is, more than one maximum are encountered, we choose the one with the smallest index.

[9] This number is slightly greater than $\sqrt{2}$.

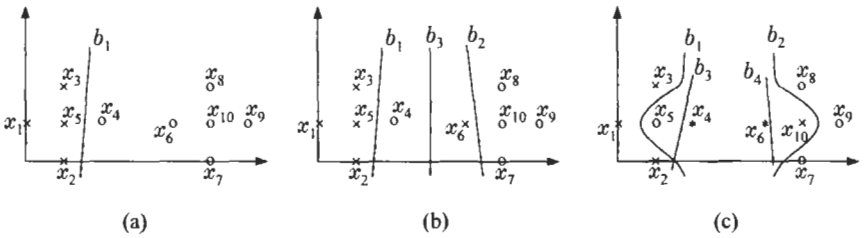


FIGURE 15.16: (a) The setup of Example 15.7(a). (b) The setup of Example 15.7(b). (c) The setup of Example 15.7(c).

x_4 is assigned to the cluster denoted "x" and x_6 is assigned to the cluster denoted by "o." Equivalently, b_1, b_2 can be thought to move to the valley between the two peaks in the place of b_3.

(c) Finally, let us consider Figure 15.16c. Again X remains unchanged. However, the initial clustering consists of three clusters, denoted by "x," "o," and "*." Also, the initial clustering is the same as that of (b) except that x_4 and x_6 are assigned to the cluster denoted by "*," x_5 is assigned to the cluster denoted by "o," and, finally, x_{10} is assigned to the cluster denoted by "x." The decision surfaces consist of the curves b_i, $i = 1, \ldots, 4$. After the first iteration of the algorithm, x_4 and x_5 are assigned to the cluster denoted by "x." Likewise, x_6 and x_{10} are assigned to the cluster denoted by "o." Finally, the rest vectors remain in the clusters where they were initially assigned. The important point here is that, although we initially considered three clusters, the algorithm ends up with two. This is because only two peaks are present. Moreover, in all cases, the decision surface is moved to the valley between the two peaks.

It should be emphasized here that the algorithm is sensitive to the value of the parameter a. Thus, one should run the algorithm several times for different values of a and interpret the results very carefully.

An alternative algorithm based on similar ideas is discussed in [Fuku 90]. It identifies the underlying clusters of X by moving the $\boldsymbol{x}_i \in X$ toward the direction of $\partial p(\boldsymbol{x})/\partial \boldsymbol{x}$, computed at $\boldsymbol{x}_i$, by $\eta \partial p(\boldsymbol{x})/\partial \boldsymbol{x}$, where η is a user-defined parameter. Iterating this procedure, points of the same cluster converge towards the same point in space (a method for the estimation of the gradient of $p(\boldsymbol{x})$ is given in [Fuku 90]). Finally, other related algorithms can be found in [Touz 88], [Chow 97].

15.8 CLUSTERING VIA COST OPTIMIZATION (REVISITED)

In this section we present two optimization methods that have been used successfully in many fields of application.

15.8.1 Simulated Annealing

This is a global optimization algorithm. More specifically, under certain conditions, it guarantees, in probability, the computation of the globally optimal solution of the problem at hand via the minimization of a cost function J. This algorithm has been proposed by Kirkpatrick *et al.* [Kirk 83] (see also [Laar 87]) and it is inspired by the problem of condensed matter in physics.[10] In contrast to the algorithms that allow corrections of the unknown parameters only to directions that reduce the cost function J, simulated annealing allows moves that, temporarily, may increase the value of J. The rationale is that, by allowing such moves, we may escape from the region of attraction of a local minimum.

A very important parameter of this method is the so-called temperature T, which is the analog of the temperature in physical systems. The algorithm starts with a high temperature, which is reduced gradually. A *sweep* is the time that has to be spent at a given temperature so that the system can enter the "thermal equilibrium" state. Let T_{max} and $\mathcal{C}_{init}$ denote the initial value of the temperature, T, and the initial clustering, respectively. Also, $\mathcal{C}$ denotes the current clustering and t the current sweep. The general scheme of simulated annealing, in the clustering context, is the following.

Simulated Annealing for Clustering

- Set $T = T_{max}$ and $\mathcal{C} = \mathcal{C}_{init}$.
- $t = 0$
- Repeat
  - $t = t + 1$
  - Repeat
    - Compute $J(\mathcal{C})$.
    - Produce a new clustering, $\mathcal{C}'$, by assigning a randomly chosen vector from X to a different cluster.
    - Compute $J(\mathcal{C}')$.
    - If $\Delta J = J(\mathcal{C}') - J(\mathcal{C}) < 0$ then
      - (A) $\mathcal{C} = \mathcal{C}'$
    - Else
      - (B) $\mathcal{C} = \mathcal{C}'$, with probability $P(\Delta J) = e^{-\Delta J/T}$.
    - End if
  - Until an equilibrium state is reached at this temperature.
  - $T = f(T_{max}, t)$
- Until a predetermined value T_{min} for T is reached.

[10] Also, it shares many common features with the Metropolis algorithm [Metr 53].

It is clear that high values of the temperature imply that almost all movements of vectors between clusters are allowed, since, as $T \to +\infty$, $P(\Delta J) \simeq 1$. On the other hand, for low values of T, fewer moves of the (B) type are allowed and, finally, as $T \to 0$, the probability of such moves tends to zero. Thus, as T is lowered, it becomes more probable that clusterings that correspond to lower values of J will be reached. On the other hand, by keeping T positive, we ensure a nonzero probability for escaping from a local minimum.

A difficulty with this algorithm is the determination of the equilibrium state at a specific temperature. One heuristic rule for this case is to consider that the equilibrium state has been reached if for k successive random reassignments of patterns, C remains unchanged (typically k is of the order of a few thousand). Further discussion of this direction is provided in [Klei 89]. Also, another crucial point is the schedule for lowering T. It has been shown that if

$$T = T_{max}/\ln(1 + t) \tag{15.33}$$

this scheme converges to the global minimum with probability 1 [Gema 84]. However, this schedule is too slow. A faster schedule for lowering T is discussed in [Szu 86]. Despite this, the main disadvantage of this algorithm remains the vast amount of computations required.

Finally, in [Al-S 93] simulated annealing is used in terms of fuzzy clustering. Experiments with simulated annealing in clustering problems are presented in [Klei 89], [Brow 92].

15.8.2 Deterministic Annealing

This is a hybrid parametric scheme combining the advantages of simulated annealing and the deterministic clustering methods. In contrast to simulated annealing, where successive clusterings are obtained by randomly disturbing the current one, no random disturbances occur here. On the other hand, the cost function is changed slightly, in order to accommodate the parameter $\beta = 1/T$, where T is defined as in the simulated annealing methods.[11] In contrast to simulated annealing, deterministic annealing is the counterpart of the phase transition phenomenon that is observed when the temperature of a material changes [Rose 91].

In this framework, a set of representatives $\boldsymbol{w}_j$, $j = 1, \ldots, m$ (m is fixed), is adopted and our goal is to locate them in appropriate positions so that a distortion function is minimized. To this end, the following "effective" cost function J is constructed [Rose 91]:

$$J = -\frac{1}{\beta} \sum_{i=1}^{N} \ln \left(\sum_{j=1}^{m} e^{-\beta d(\boldsymbol{x}_i, \boldsymbol{w}_j)} \right) \tag{15.34}$$

[11] We choose to work with β instead of T, because this notation is generally used for this algorithm.

where m is the number of clusters. Differentiating J with respect to the representative $\boldsymbol{w}_r$ and setting it equal to 0, we obtain

$$\frac{\partial J}{\partial \boldsymbol{w}_r} = \sum_{i=1}^{N} \left(\frac{e^{-\beta d(\boldsymbol{x}_i, \boldsymbol{w}_r)}}{\sum_{j=1}^{m} e^{-\beta d(\boldsymbol{x}_i, \boldsymbol{w}_j)}} \right) \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_r)}{\partial \boldsymbol{w}_r} = \mathbf{0} \tag{15.35}$$

It is clear that the ratio in parentheses takes values in [0, 1] and, in addition, all these terms for a specific $\boldsymbol{x}_i$ sum up to 1. Thus, it may be interpreted as the probability, P_{ir}, that $\boldsymbol{x}_i$ belongs to C_r, $r = 1, \ldots, m$. Then Eq. (15.35) can be written as

$$\sum_{i=1}^{N} P_{ir} \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_r)}{\partial \boldsymbol{w}_r} = \mathbf{0} \tag{15.36}$$

In the sequel, we assume that $d(\boldsymbol{x}, \boldsymbol{w})$ is a convex function of $\boldsymbol{w}$ for fixed $\boldsymbol{x}$. Note that for $\beta = 0$, all P_{ij}'s are equal to $1/m$, for all $\boldsymbol{x}_i$'s, $i = 1, \ldots, N$. Thus, in this case Eq. (15.36) becomes

$$\sum_{i=1}^{N} \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_r)}{\partial \boldsymbol{w}_r} = \mathbf{0} \tag{15.37}$$

Since $d(\boldsymbol{x}, \boldsymbol{w})$ is a convex function, $\sum_{i=1}^{N} d(\boldsymbol{x}_i, \boldsymbol{w}_r)$ is also a convex function and, thus, it has a unique minimum, which may be captured by any gradient descent scheme. Thus, in this case, *all the resulting representatives coincide with this unique global minimum.* That is, all data belong to a single cluster. As the value of β increases, it reaches a critical value where a phase transition occurs (alternatively, the probabilities P_{ir} "depart sufficiently" from the uniform model); that is, the clusters are no longer optimally represented by a single representative. Thus, the representatives split up in order to provide an optimal representation of the data set at the new phase. Further increase of β causes a new phase transition and the available representatives are further split up. By choosing m to be greater than the "actual" number of clusters, we ensure the ability of the algorithm to represent the data set properly. In the worst case, some of the representatives will coincide.

Note that as β increases, the probabilities P_{ij} depart from the uniform model and approach the hard clustering model; that is, for all $\boldsymbol{x}_i$, $P_{ir} \simeq 1$ for some r, and $P_{ij} \simeq 0$ for $j \neq r$.

Thus, the requirement for each vector to be assigned to a specific cluster with probability close to unity may serve as a termination criterion for the algorithm.

Schedules for the increase of β are discussed in [Rose 91]. Although simulation results show satisfactory performance of the algorithm, it is not guaranteed that it reaches the globally optimum clustering. Other applications of deterministic annealing to clustering are discussed in [Hofm 97], [Beni 94].

15.9 CLUSTERING USING GENETIC ALGORITHMS

Genetic algorithms have been inspired by the natural selection mechanism introduced by Darwin. They apply certain operators to a population of solutions of the problem at hand, in such a way that the new population is improved compared with the previous one according to a prespecified criterion function J. This procedure is applied for a preselected number of iterations and the output of the algorithm is the best solution found in the last population or, in some cases, the best solution found during the evolution of the algorithm.

In general, the solutions of the problem at hand are coded[12] and the operators are applied to the coded versions of the solutions. The way the solutions are coded plays an important role in the performance of a genetic algorithm. Inappropriate coding may lead to poor performance.

The operators used by genetic algorithms simulate the way natural selection is carried out. The most well-known operators used are the *reproduction*, *crossover*, and *mutation* operators applied in that order to the current population. The reproduction operator ensures that, in probability, the better (worse) a solution in the current population is, the more (less) replicates it has in the next population. The crossover operator, which is applied to the temporary population produced after the application of the reproduction operator, selects pairs of solutions randomly, splits them at a random position, and exchanges their second parts. Finally, the mutation operator, which is applied after the application of the reproduction and crossover operators, selects randomly an element of a solution and alters it with some probability. The last operator may be viewed as a way out of getting stuck in local minima. Apart from these three operators, many others have been proposed in the literature (e.g., [Mich 94]).

Besides the coding of the solutions, other parameters, such as the number of solutions in a population, p,[13] the probability with which we select two solutions for crossover, and the probability with which an element of a solution is mutated, play very important roles in the performance of the algorithm.

Several genetic algorithms with application to clustering have been proposed (e.g., [Bhan 91], [Andr 94], [Sche 97], [Maul 00], [Tsen 00]). In the sequel, we briefly discuss a simple parametric one that is suitable for hard clustering. We assume that the number of clusters, m, is fixed. As stated before, the first thing we have to decide is how to code a solution. A simple (but not unique) way to achieve this is to use the representatives in order to form the following string:

$$[\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m] \tag{15.38}$$

[12]Binary representations as well as more general ones are possible.

[13]This may be fixed or varied.

or, in more detail,

$$[w_{11}, w_{12}, \ldots, w_{1l}, w_{21}, w_{22}, \ldots, w_{2l}, \ldots, w_{m1}, w_{m2}, \ldots, w_{ml}] \tag{15.39}$$

The cost function we use is

$$J = \sum_{i=1}^{N} u_{ij} d(\boldsymbol{x}_i, \boldsymbol{w}_j) \tag{15.40}$$

where

$$u_{ij} = \begin{cases} 1, & d(\boldsymbol{x}_i, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d(\boldsymbol{x}_i, \boldsymbol{w}_k) \\ 0, & \text{otherwise} \end{cases} \quad i = 1, \ldots, N \tag{15.41}$$

The allowable cut points for the crossover operator are between different representatives. Also, in this case the mutation operator selects randomly a coordinate of a vector of a solution and decides randomly to add a small random number to it.

An alternative to this algorithm is the following. Before we apply the reproduction operator to the current population, we run the hard clustering algorithm, described in Chapter 14, p times, each time using a different solution of the current population as the initial state. The p solutions produced after the convergence of the hard clustering algorithm constitute the population to which the reproduction operator will be applied. It is expected that this modification will give better results, as it is likely that the resulting p solutions are local minima of the cost function. This modified algorithm has been reported to give satisfactory results in a color image quantization application, [Sche 97].

15.10 OTHER CLUSTERING ALGORITHMS

A clustering algorithm that is based on the so-called *tabu search method* is presented in [Al-S 95]. Its initial state is an arbitrarily chosen clustering. The algorithm proceeds as follows. Based on the current state of the algorithm, a set of candidate clusterings, A, is created. The next state is chosen to be the "best" element of A, according to some criterion function. Certain criteria are used to prevent the algorithm from returning to recently visited states. The procedure is repeated for a prespecified number of iterations. Preliminary results reported in [Al-S 95] show that this algorithm compares favorably with the hard clustering and the simulated annealing algorithms. A recent tabu search based heuristic scheme for clustering is presented in [Sung 00].

A method that directly relates clusters to peak values of the pdf has been suggested in [Tou 74], where the estimation of the pdf is achieved via Parzen windows.

A related method is the *mountain method* (see, e.g., [Dave 97]). The idea is to assign to each vector, $\boldsymbol{x}$, an energy source. The generated potential has a peak at $\boldsymbol{x}$ and rapidly decays as we move away from it. The total potential function at a

specific point in the vector space is the summation of the potentials produced by all the vectors of X. We compute the value of this function at each data point and form the array $\boldsymbol{v}$ of the N resulting values. The maximum one, which corresponds to the highest peak, is identified and the corresponding vector is considered as the representative of the first cluster. Then we remove the largest value of $\boldsymbol{v}$ and update the rest of the components appropriately. This procedure is repeated until a specific termination criterion is met.

An algorithm that combines ideas from both the fuzzy clustering schemes and the agglomerative algorithms (Chapter 13) is discussed in [Frig 97]. This scheme produces clusterings that minimize the following cost function

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^2 d(\boldsymbol{x}_i, C_j) - a \sum_{j=1}^{m} \left[\sum_{i=1}^{N} u_{ij}^2 \right] \tag{15.42}$$

where now m varies. Clearly, the first term in the above equation is minimized when $m = N$, and the second term is maximized when $m = 1$. It is worth mentioning that the hierarchies of clusterings produced do not necessarily possess the nested property.

Besides the preceding algorithms, many other clustering algorithms based on very different ideas have been proposed. For example, in [Matt 91] a scheme that does not use the concept of the distance between vectors is proposed. Also, in [Kodr 88] a clustering technique based on a conceptual distance is presented.

Another technique combining supervised and unsupervised methods has been proposed in [Pedr 97]. The latter may be useful in applications in which only a fraction of the data have a class label for the training.

In [Robe 00], a different clustering method is discussed. The clustering problem is stated in information theoretic terms and it is shown that minimization of the entropy can be used in order to estimate the clustering structure underlying the data set X. This is equivalent to obtaining the structure associated with maximum certainty. Another algorithm based on information theoretic criteria has been suggested in [Goks 02]. It is a valley-seeking algorithm and builds upon Renyi's entropy estimator.

Problems

15.1 Consider the set $X = \{\boldsymbol{x}_i, i = 1, \ldots, 7\}$, where $\boldsymbol{x}_1 = [1, 1]^T$, $\boldsymbol{x}_2 = [1, 2]^T$, $\boldsymbol{x}_3 = [2, 1]^T$, $\boldsymbol{x}_4 = [3, 1]^T$, $\boldsymbol{x}_5 = [6, 1]^T$, $\boldsymbol{x}_6 = [7, 1]^T$, $\boldsymbol{x}_7 = [6, 2]^T$.

(a) Determine the value of q (Section 15.2.1) for which the MST clustering algorithm gives two clusters.

(b) Apply the algorithms that are based on the idea of regions of influence when these regions are defined by Eqs. (15.2)–(15.5).

(c) Run the directed tree-based clustering algorithm and determine the values of θ for which it gives two clusters.

15.2 Consider the basic competitive learning algorithm with $\eta = 0.2$. Let $X = \{x_i, i = 1, \ldots, 4\}$, where $x_1 = -3$, $x_2 = -2$, $x_3 = 2$, $x_4 = 3$. Also, let $m = 2$ and $w_1 = -1$ and $w_2 = 1$. Assume that the vectors of X are presented to the algorithm in the same order, $x_1, x_2, x_3 x_4, x_1, x_2, x_3, x_4, \ldots$. Let us call the time required for the consideration of the feature vectors of X once the *updating circle*.

(a) Show that w_1 (w_2) always wins on the first (last) two feature vectors when the squared Euclidean distance is in use.

(b) Will w_1 and w_2 converge to the values -2.5 and 2.5, respectively, after an infinite number of updating circles? Give intuitive arguments.

15.3 What would the behavior of the leaky learning algorithm be if $\eta_w = \eta_l$?

15.4 *von der Malsburg learning rule.* Assume that the data set X consists of N binary-valued feature vectors, and for each of the m available representatives, $\boldsymbol{w}_j$, we have $\sum_{k=1}^{l} w_{jk} = 1$, where w_{jk} is the kth coordinate of $\boldsymbol{w}_j$. The rule may be stated as follows:

- Present an input vector $\boldsymbol{x} \in X$.
- Determine the winner, $\boldsymbol{w}_j$, of the competition for $\boldsymbol{x}$.
- Update the representatives

$$w_{jk}^{new} = \begin{cases} w_{jk} + \eta \left(\frac{x_k}{n_x} - w_{jk} \right), & \text{if } \boldsymbol{w}_j \text{ wins on } \boldsymbol{x} \\ w_{jk}, & \text{if } \boldsymbol{w}_j \text{ loses on } \boldsymbol{x} \end{cases} \tag{15.43}$$

where x_k is the kth coordinate of $\boldsymbol{x}$. In the last equation $n_x = \sum_{k=1}^{l} x_k$, that is, it is equal to the number of 1's contained in $\boldsymbol{x}$, and η, the learning rate, takes values in $[0, 1]$. The updating rule may be stated in words as follows: "If a representative wins, each of its coordinates gives up some proportion η that in the sequel is equally distributed among the coordinates w_{ji} that correspond to $x_k = 1$. All the remaining representatives do not change."

(a) Verify that this statement is equivalent to the updating rule given by Eq. (15.43).

(b) Prove that $\sum_{k=1}^{l} w_{jk}^{new} = 1$, $j = 1, \ldots, m$.

15.5 Prove Eq. (15.20).

15.6 Consider three 2-dimensional Gaussian probability density functions with means $\boldsymbol{\mu_1} = [-1, -1]^T$, $\boldsymbol{\mu_2} = [6, 3]^T$, $\boldsymbol{\mu_3} = [-0.7, 7]^T$ and covariance matrices $\Sigma_1 = \Sigma_2 = \Sigma_3 = 2I$, respectively, where I is the 2×2 identity matrix. Draw 200 points from each distribution and form a data set X with the resulting 600 points.

(a) Run the Binary Morphology Clustering Algorithm (BMCA) when T is the 3×3 square structuring element given in Example 15.3. For each case use different values of r and proceed to the third stage of the algorithm with the best one of them.

(b) Run the Generalized Hard Clustering Scheme (GHAS) algorithm from Chapter 14, using the squared Euclidean distance as the dissimilarity function between two vectors, for the optimum number of clusters derived by the previous procedure. Compare the results of the two algorithms.

15.7 For the data set X given in Example 15.6, run the isodata algorithm assuming that the number of clusters is 2. Compare the results obtained in Example 15.6 and those obtained with isodata. Give a qualitative explanation of the differences that may be observed.

15.8 Verify Eq. (15.29) and (15.30).

15.9 Derive the updating equation for the coordinates of the parameter vector $\boldsymbol{\theta}$ when $g(\boldsymbol{x}; \boldsymbol{\theta})$, defined in Section 15.6, is a quadratic function of $\boldsymbol{\theta}$.

Hint: In this case $g(\boldsymbol{x}; \boldsymbol{\theta}) = w_0 + \sum_{i=1}^{l} w_i x_i + \sum_{s=1}^{l} \sum_{r=1}^{l} w_{sr} x_s x_r$.

15.10 Consider two 2-dimensional Gaussian distributions with means $\boldsymbol{\mu}_1 = [0, 0]^T$ and $\boldsymbol{\mu}_2 = [3, 3]^T$ and covariance matrices $\Sigma_1 = I$ and $\Sigma_2 = 1.5I$, respectively, where I is the 2×2 identity matrix. Create a data set X such that 100 feature vectors stem from the first and another 100 feature vectors stem from the second Gaussian distribution.

(a) Run the Boundary Detection Algorithm (BDA) algorithm on X assuming that the decision boundary is a hyperplane. Use the hyperbolic tangent as f.

(b) Run the BDA algorithm on the X^+ and X^- stemming from the previous running. Comment on the results.

15.11 (a) What is the shape of $V(\boldsymbol{x})$, defined by Eq. (15.31) when $d(\boldsymbol{x}, \boldsymbol{y})$ is given as in Eq. (15.32)?

(b) How should $d(\boldsymbol{x}, \boldsymbol{y})$ be defined in order to have a hypercubical shape for $V(\boldsymbol{x})$?

(c) Does the shape of $V(\boldsymbol{x})$ affects the behavior of the valley-seeking clustering algorithm? Give an example.

15.12 Consider the set $X = \{\boldsymbol{x}_i, i = 1, \ldots, 10\}$, where $\boldsymbol{x}_1 = [0, 1]^T$, $\boldsymbol{x}_2 = [0, 2]^T$, $\boldsymbol{x}_3 = [0, 3]^T$, $\boldsymbol{x}_4 = [0, 4]^T$, $\boldsymbol{x}_5 = [1, 1]^T$, $\boldsymbol{x}_6 = [1, 2]^T$, $\boldsymbol{x}_7 = [1, 3]^T$, $\boldsymbol{x}_8 = [2, 1]^T$, $\boldsymbol{x}_9 = [2, 2]^T$, $\boldsymbol{x}_{10} = [2, 3]^T$. Initially, the first six of them belong to cluster C_1 and the next four belong to cluster C_2. Apply the valley-seeking algorithm to X and comment on the results.

15.13 If $T_{max} = 5$ and $T_{min} = 0.5$, estimate the number of sweeps required with the simulated annealing algorithm in order to determine (in probability) the clustering with the globally minimum value of J.

Hint: Use Eq. (15.33).

15.14 Modify the deterministic annealing algorithm so that the number of representatives is not fixed a priori but increases as β increases.

15.15 Consider the function

$$J = \sum_{i=1}^{N} d(\boldsymbol{x}_i, C_{\boldsymbol{x}_i}) \tag{15.44}$$

where $C_{\boldsymbol{x}_i}$ is the cluster to which $\boldsymbol{x}_i$ belongs and $d(\boldsymbol{x}_i, C_{\boldsymbol{x}_i})$ is a distance between a point and a set using no representative for the set (e.g., Chapter 11). Propose a coding of the solutions for a genetic algorithm that uses this function. Discuss the merits and the disadvantages of the proposed coding.

References

[Ahal 90] Ahalt S.C., Krishnamurthy A.K., Chen P., Melton D.E. "Competitive learning algorithms for vector quantization," *Neural Networks*, Vol. 3, pp. 277–290, 1990.

[Al-S 93] Al-Sultan K.S., Selim S.Z. "A global algorithm for the fuzzy clustering problem," *Pattern Recognition*, Vol. 26(9), pp. 1357–1361, 1993.

[Al-S 95] Al-Sultan K.S. "A tabu search to the clustering problem," *Pattern Recognition*, Vol. 28(9), pp. 1443–1451, 1995.

[Andr 94] Andrey P., Tarroux P. "Unsupervised image segmentation using a distributed genetic algorithm," *Pattern Recognition*, Vol. 27(5), pp. 659–673, 1994.

[Atiy 90] Atiya A.F. "An unsupervised learning technique for artificial neural networks," *Neural Networks*, Vol. 3, pp. 707–711, 1990.

[Banz 90] Banzhaf W., Haken H. "Learning in a competitive network," *Neural Networks*, Vol. 3, pp. 423–435, 1990.

[Beni 94] Beni G., Liu X. "A least biased fuzzy clustering method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, pp. 954–960, September 1994.

[Benv 87] Benveniste A., Metivier M., Priouret P. *Adaptive Algorithms and Stochastic Approximation*, Springer-Verlag, 1987.

[Bhan 91] Bhanu B., Lee S., Ming J. "Self-optimizing image segmentation system using a genetic algorithm," *Proceedings, Fourth International Conference on Genetic Algorithms*, pp. 362–369, 1991.

[Brow 92] Brown D.E., Huntley C.L. "A practical application of simulated annealing to clustering," *Pattern Recognition*, Vol. 25(4), pp. 401–412, 1992.

[Burr 91] Burrascano P. "Learning vector quantization for the probabilistic neural network," *IEEE Transactions on Neural Networks*, Vol. 2(4), pp. 458–461, 1991.

[Butl 96] Butler D., Jiang J. "Distortion equalized fuzzy competitive learning for image data vector quantization," *Proceedings of ICAPPS'96*, pp. 3390–3393, 1996.

[Chen 94] Chen L., Chang S. "An adaptive conscientious competitive learning algorithm and its applications," *Pattern Recognition*, Vol. 27(12), pp. 1787–1813, 1994.

[Chou 97] Chou C.S., Siu W. "Distortion sensitive competitive learning for vector quantizer design," *IEEE Proc.*, pp. 3405–3408, 1997.

[Chow 97] Chowdhury N., Murthy C.A. "Minimal spanning tree based clustering technique: relationship with Bayes classifier," *Pattern Recognition*, Vol. 30(11), pp. 1919–1929, 1997.

[Dave 97] Dave R.N., Krishnapuram R. "Robust clustering methods: A unified view," *IEEE Transactions on Fuzzy Systems*, Vol. 5(2), pp. 270–293, May 1997.

[Dela 80] Delattre M., Hansen P. "Bicriterion cluster analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 2(4), pp. 277–291, July 1980.

[Frig 97] Frigui H., Krishnapuram R., "Clustering by competitive agglomeration," *Pattern Recognition*, Vol. 30(7), pp. 1109–1119, 1997.

[Fu 93] Fu L., Yang M, Braylan R., Benson N. "Real-time adaptive clustering of flow cytometric data," *Pattern Recognition*, Vol. 26(2), pp. 365–373, 1993.

[Fuku 90] Fukunaga K. *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press, 1990.

[Gabr 69] Gabriel K.R., Sokal R.R. "A new statistical approach to geographic variation analysis," *Syst. Zool.* Vol. 18, pp. 259–278, 1969.

[Gema 84] Geman S., Geman D. "Stochastic relaxation, Gibbs distribution and Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 721–741, 1984.

[Gers 79] Gersho A. "Asumptotically optimal block quantization," *IEEE Transactions on Information Theory*, Vol. 25(4), pp. 373–380, 1979.

[Gers 92] Gersho A., Gray R.M. *Vector Quantization and Signal Compression*, Kluwer Academic, 1992.

[Goks 02] Goksay E., Principe J.C., "Information theoretic clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24(2), pp. 158–171, 2002.

[Gonz 93] Gonzalez R.C., Woods R.E. *Digital image processing*, Addison Wesley, 1993.

[Gray 84] Gray R.M. "Vector quantization," *IEEE ASSP Magazine*, pp. 4–29, April 1984.

[Gros 76a] Grossberg S. "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," *Biological Cybernetics*, Vol. 23, pp. 121–134, 1976.

[Gros 76b] Grossberg S. "Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions," *Biological Cybernetics*, Vol. 23, pp. 187–202, 1976.

[Gros 87] Grossberg S. "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Science*, Vol. 11, pp. 23–63, 1987.

[Hech 88] Hecht-Nielsen R. "Applications of counter-propagation networks," *Neural Networks*, Vol. 1(2), pp. 131–141, 1988.

[Hoch 64] Hochberg J.E. *Perception*, Prentice-Hall, 1964.

[Hofm 97] Hofmann T., Buchmann J.M. "Pairwise data clustering by deterministic annealing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(1), pp. 1–14, 1997.

[Jarv 78] Jarvis R.A. "Shared nearest neighbor maximal spanning trees for cluster analysis," *Proceedings, 4th Joint Conference on Pattern Recognition*, Kyoto, Japan, pp. 308–313, 1978.

[Jayn 82] Jaynes E.T. "On the rationale of maximum-entropy methods," *Proc. IEEE*, Vol. 70(9), pp. 939–952, September 1982.

[Kara 96] Karayiannis N.B., Pai P. "Fuzzy algorithms for learning vector quantization," *IEEE Transactions on Neural Networks*, Vol. 7(5), pp. 1196–1211, 1996.

[Kirk 83] Kirkpatrick S., Gelatt C.D. Jr., Vecchi M.P. "Optimization by simulated annealing," *Science*, Vol. 220, pp. 671–680, 1983.

[Klei 89] Klein R.W., Dubes R.C. "Experiments in projection and clustering by simulated annealing," *Pattern Recognition*, Vol. 22(2), pp. 213–220, 1989.

[Kodr 88] Kodratoff Y., Tecuci G. "Learning based on conceptual distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10(6), pp. 897–909, 1988.

[Koho 89] Kohonen T. *Self-Organization and Associative Memory*, 2nd ed., Springer-Verlag, 1989.

[Koho 95] Kohonen T. *Self-Organizing Maps*, Springer-Verlag, 1995.

[Koon 75] Koontz W.L.G., Narendra P.M., Fukunaga K. "A branch and bound clustering algorithm," *IEEE Transactions on Computers*, Vol. 24(9), pp. 908–914, September 1975.

[Koon 76] Koontz W.L.G., Narendra P.M., Fukunaga K. "A graph-theoretic approach to nonparametric cluster analysis," *IEEE Transactions on Computers*, Vol. 25(9), pp. 936–944, September 1976.

[Kosk 91] Kosko B. "Stochastic competitive learning," *IEEE Transactions on Neural Networks*, Vol. 2(5), pp. 522–529, 1991.

[Kosk 92] Kosko B. *Neural Networks for Signal Processing*, Prentice Hall, 1992.

[Kotr 92] Kotropoulos C., Auge E., Pitas I. "Two-layer learning vector quantizer for color image quantization," *Signal Processing VI*, pp. 1177–1180, 1992.

[Kout 95] Koutroumbas K. "Hamming neural networks, architecture design and applications," Ph.D. dissertation, Department of Informatics, University of Athens, 1995 (in Greek).

[Laar 87] van Laarhoven P.J.M., Aarts E.H.L. *Simulated Annealing: Theory and Applications*, Reidel, Hingham, MA, 1987.

[Likh 97] Likhovidov V. "Variational approach to unsupervised learning algorithms of neural networks," *Neural Networks*, Vol. 10(2), pp. 273–289, 1997.

[Lind 80] Linde Y., Buzo A., Gray R.M. "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, Vol. 28(1), pp. 84–95, 1980.

[Luen 84] Luenberger D.G. *Linear and Nonlinear Programming*, Addison Wesley, 1984.

[Mals 73] von der Maslburg, "Self-organization sensitive cells in the striate cortex," *Kybernetic*, Vol. 14, pp. 85–100, 1973.

[Mara 80] Maragos P., Schafer R.W. "Morphological systems for multidimensional signal processing," *Proc. IEEE*, Vol. 78(4), pp. 690–710, April 1980.

[Mart 93] Martinetz T.M., Berkovich S.G., Schulten K.J. "Neural-gas network for vector quantization and its application to time-series prediction," *IEEE Transactions on Neural Networks*, Vol. 4(4), pp. 558–569, July 1993.

[Masu 93] Masuda T. "Model of competitive learning based upon a generalized energy function," *Neural Networks*, Vol. 6, pp. 1095–1103, 1993.

[Matt 91] Matthews G., Hearne J. "Clustering without a metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13(2), pp. 175–184, 1991.

[Maul 00] Maulik U., Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern Recognition*, Vol. 33, pp. 1455–1465, 2000

[Metr 53] Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., Teller E. "Equations of state calculations by fast computing machines," *Journal of Chemical Physics*, Vol. 21, pp. 1087–1092, 1953.

[Mich 94] Michalevitz Z. *Genetic Algorithms + Data Structures = Evolutionary Programming*, 2nd ed., Springer-Verlag, 1994.

[Mora 00] Morales E., Shih F. Y., "Wavelet coefficients clustering using morphological operations and pruned quadtrees," *Pattern Recognition*, Vol. 33, pp. 1611–1620, 2000.

[Nyec 92] Nyeck A., Mokhtari H., Tosser-Roussey A. "An improved fast adaptive search algorithm for vector quantization by progressive codebook arrangement," *Pattern Recognition*, Vol. 25(8), pp. 799–802, 1992.

[Ozbo 95] Osbourn G.C., Martinez R.F. "Empirically defined regions of influence for cluster analysis," *Pattern Recognition*, Vol. 28(11), pp. 1793–1806, 1995.

[Pedr 97] Pedrycz W., Waletzky J. "Neural–network front ends in unsupervised learning," *IEEE Transactions on Neural Networks*, Vol. 8(2), pp. 390–401, March 1997.

[Post 93] Postaire J.G., Zhang R.D., Lecocq-Botte C. "Cluster analysis by binary morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(2), pp. 170–180, 1993.

[Robe 00] Roberts S.J., Everson R., Rezek I., "Maximum certainty data partitioning," *Pattern Recognition*, pp. 833–839, 2000.

[Rose 91] Rose K. "Deterministic annealing, clustering and optimization," Ph.D. dissertation, California Institute of Technology, 1991.

[Rose 93] Rose K., Gurewitz E., Fox G.C. "Constrained clustering as an optimization method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(8), pp. 785–794, 1993.

[Rume 86] Rumelhart D.E., Zipser D. "Feature discovery by competitive learning," *Cognitive Science*, Vol. 9, pp. 75–112, 1986.

[Rume 86] Rumelhart D.E., McLelland J.L. *Parallel Distributed Processing*, Cambridge, MA: MIT Press, 1986.

[Sche 97] Scheunders P. "A genetic c-means clustering algorithm applied to color image quantization," *Pattern Recognition*, Vol. 30(6), pp. 859–866, 1997.

[Sung 00] Sung C.S., Jin H.W., "A tabu-search-based heuristic for clustering," *Pattern Recognition*, Vol. 33, pp. 849–858, 2000.

[Szu 86] Szu H. "Fast simulated annealing," in *Neural Networks for Computing* (Denker J. S., ed.), American Institute of Physics, 1986.

[Tou 74] Tou J.T., Gonzales R.C. *Pattern Recognition Principles*, Addison-Wesley, 1974.

[Tous 80] Toussaint G.T. "The relative neighborhood graph of a finite planar set," *Pattern Recognition*, Vol. 12, pp. 261–268, 1980.

[Touz 88] Touzani A., Postaire J.G. "Mode detection by relaxation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10(6), pp. 970–977, 1988.

[Tsen 00] Tseng L.Y., Yang S.B., "A genetic clustering algorithm for data with non-spherical-shape clusters," *Pattern Recognition*, Vol. 33, pp. 1251–1259, 2000.

[Tsyp 73] Tsypkin Y.Z. *Foundations of the Theory of Learning Systems*, Academic Press, 1973.

[Uchi 94] Uchiyama T., Arbib M.A. "An algorithm for competitive learning in clustering problems," *Pattern Recognition*, Vol. 27(10), pp. 1415–1421, 1994.

[Ueda 94] Ueda N., Nakano R. "A new competitive learning approach based on an equidistortion principle for designing optimal vector quantizers," *Neural Networks*, Vol. 7(8), pp. 1211–1227, 1994.

[Urqu 82] Urquhart R. "Graph theoretical clustering based on limited neighborhood sets," *Pattern Recognition*, Vol. 15(3), pp. 173–187, 1982.

[Vish 00] Vishwanathan S.V.N., Murty M.N., "Kohonen's SOM with cashe," *Pattern Recognition*, Vol. 33, pp. 1927–1929, 2000.

[Xie 93] Xie Q., Laszlo A., Ward R.K. "Vector quantization technique for nonparametric classifier design," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(12), pp. 1326–1330, 1993.

[Yama 80] Yamada Y., Tazaki S., Gray R.M. "Asymptotic performance of block quantizers with difference distortion measures," *IEEE Transactions on Information Theory*, Vol. 26(1), pp. 6–14, 1980.

[Zahn 71] Zahn C.T. "Graph-theoretical methods for detecting and describing gestalt clusters," *IEEE Transactions on Computers*, Vol. 20(1), pp. 68–86, January 1971.

[Zhu 94] Zhu C., Li L., He Z., Wang J. "A new competitive learning learning algorithm for vector quantization," *Proceedings of ICASSP'94*, pp. 557–560, 1994.

CHAPTER 16

CLUSTER VALIDITY

16.1 INTRODUCTION

A common characteristic of the majority of the clustering algorithms, discussed in the previous chapters, is that they *impose* a clustering structure on the data set X, even though X may not possess such a structure. In the latter case, the results produced after the application of a clustering algorithm on X are not indicative of the structure of X. In other words, *cluster analysis is not a panacea*. That is, we must have an indication that the vectors of X form clusters before we apply a clustering algorithm. The problem of verifying whether X possesses a clustering structure, without identifying it explicitly, is known as *clustering tendency* and is discussed at the end of the chapter.

Let us now assume that X possesses a clustering structure and we want to unravel it. A different kind of problem is encountered now. Recall that all the clustering algorithms require knowledge of the values of specific parameters and, in addition, some of them impose restrictions on the shape of the clusters (e.g., compact, hyperellipsoidal). As already shown in the previous chapters, poor estimation of these parameters and inappropriate restrictions on the shape of the clusters (wherever such restrictions are required) may lead to incorrect conclusions about the clustering structure of X. Thus, the need for further evaluation of the results of a clustering algorithm is apparent.

In this chapter, we discuss methods suitable for quantitative evaluation of the results of a clustering algorithm. This task is known under the general term *cluster validity*. However, it must be emphasized that the results obtained by these methods are *only* tools at the disposal of the expert in order to evaluate the resulting clustering.

Let $\mathcal{C}$ denote the clustering structure resulting from the application of a clustering algorithm on X. This may be a hierarchy of clusterings, as is the case with the hierarchical algorithms, or a single clustering, as happens with all the other algorithms discussed in the previous chapters. Cluster validity can be approached in three possible directions. First, we may evaluate $\mathcal{C}$ in terms of an independently drawn structure, which is imposed on X a priori and reflects our intuition about the clustering structure of X. The criteria used for the evaluation of this

kind are called *external criteria*. In addition, external criteria may be used to measure the degree to which the available data confirm a prespecified structure, without applying any clustering algorithm to X. Second, we may evaluate $\mathcal{C}$ in terms of quantities that involve the vectors of X themselves, for example, the proximity matrix. The criteria used for this kind of evaluation are called *internal criteria*. Finally, we may evaluate $\mathcal{C}$ by comparing it with other clustering structures, resulting from the application of the same clustering algorithm, but with different parameter values, or of other clustering algorithms to X. Criteria of this kind are called *relative criteria*.

The cluster validation methods based on external or internal criteria rely on statistical hypothesis testing, which was introduced in Chapter 5. The following section contains some additional definitions to be used in this chapter.

16.2 HYPOTHESIS TESTING REVISITED

Let H_0 and H_1 be the null and alternative hypotheses, respectively,

$$H_1 : \boldsymbol{\theta} \neq \boldsymbol{\theta}_0$$
$$H_0 : \boldsymbol{\theta} = \boldsymbol{\theta}_0$$

Also let $\bar{D}_\rho$ be the critical interval corresponding to significance level ρ of a test statistic q, and Θ_1 the set of all possible values that $\boldsymbol{\theta}$ may take under hypothesis H_1. The *power function* of the test is defined as

$$W(\boldsymbol{\theta}) = P(q \in \bar{D}_\rho | \boldsymbol{\theta} \in \Theta_1) \tag{16.1}$$

For a specific $\boldsymbol{\theta} \in \Theta_1$, $W(\boldsymbol{\theta})$ is known as the *test power under the alternative* $\boldsymbol{\theta}$. In words, $W(\boldsymbol{\theta})$ is the probability that q lies in the critical region when the value of the parameter vector is $\boldsymbol{\theta}$. This is the probability of making the correct decision when H_0 is rejected. The power function can be used for the comparison of two different statistical tests. The test whose power under the alternative hypotheses is greater is always preferred.

There are two types of errors associated with a statistical test.

- Suppose that H_0 is true. If $q(\boldsymbol{x}) \in \bar{D}_\rho$, H_0 will be rejected even if it is true. This is called a type I error. The probability of such an error is ρ. The probability of accepting H_0 when it is true is $1 - \rho$.
- Suppose that H_0 is false. If $q(\boldsymbol{x}) \notin \bar{D}_\rho$, H_0 will be accepted even if it is false. This is called a type II error. The probability of such an error is $1 - W(\boldsymbol{\theta})$, and it depends on the specific value of $\boldsymbol{\theta}$.

In practice, the final decision to reject or accept H_0 is based partially on the preceding statements as well as on other factors, such as the cost of a

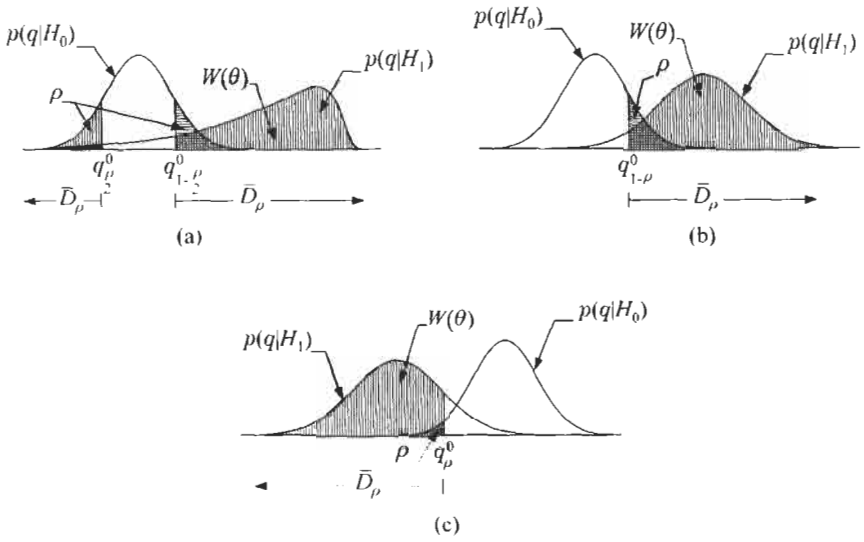


FIGURE 16.1: Critical regions of (a) a two-tailed test, (b) a right-tailed test, and (c) a left-tailed test. q_a^0 is the a percentile of q under H_0.[1]

wrong decision. Thus, the terms "accept" and "reject" H_0 must be interpreted accordingly.

The probability density function (pdf) of the statistic q, under H_0, for most of the statistics used in practice has a single maximum and the $\bar{D}_\rho$ region is either a half-line or the union of two half-lines. These assumptions have also been adopted here. Figure 16.1 shows the three possible cases for $\bar{D}_\rho$. In the first case, $\bar{D}_\rho$ is the union of two half-lines. Such a test is known as a *two-tailed statistical test*. The other two tests are called *one-tailed* statistical tests, because $\bar{D}_\rho$ consists of a single half-line. Figure 16.1a is an example of a two-tailed statistical test[2] and Figure 16.1b and 16.1c are examples of a right- and a left-tailed test, respectively.

In many practical cases the exact form of the pdf of a statistic q, under a given hypothesis, is not available and it is difficult to obtain. In the sequel, we discuss two methods for estimating pdf's via simulations.

- *Monte Carlo techniques* [Shre 64], [Sobo 84] rely on simulating the process at hand using a sufficient number of computer-generated data. For each of the, say r, data sets, X_i, we compute the value of q, denoted by q_i, and then we construct the corresponding histogram of these values. The unknown pdf

[1] The a percentile of q is the smallest number q_a such that $a = P(q \leq q_a)$.

[2] More general versions of a two-tailed statistical test are also possible (e.g., [Papo 91]).

can then be approximated by this histogram. Assume now that q corresponds to a right-tailed statistical test and a histogram is constructed using r values of q corresponding to the r data sets. For a given data set, if q is the corresponding value of the statistic, rejection (acceptance) of the null hypothesis is done on the basis of

$$\text{Reject (accept) } H_0 \text{ if } q \text{ is greater (smaller) than } (1-\rho)r \text{ of the } q_i \text{ values} \tag{16.2}$$

For a left-tailed test, rejection or acceptance of the null hypothesis is done on the basis of

$$\text{Reject (accept) } H_0 \text{ if } q \text{ is smaller (greater) than } \rho r \text{ of the } q_i \text{ values} \tag{16.3}$$

Finally, for a two-tailed test we have

$$\begin{aligned} &\text{Accept } H_0 \text{ if q is greater than } (\rho/2)\ r \text{ of the } q_i \text{ values and} \\ &\text{less than } (1-\rho/2)\ r \text{ of the } q_i \text{ values} \end{aligned} \tag{16.4}$$

- *Bootstrapping techniques* constitute an alternative way to cope with a limited amount of data. The idea here is to parameterize the unknown pdf in terms of an unknown parameter. To cope with the limited amount of data and in order to improve the accuracy of the estimate of the unknown pdf parameter, several "fake" data sets $X_1, \ldots, X_r$ are created by sampling X with replacement, as discussed in Chapter 5.

  Typically, good estimates are obtained if r is between 100 and 200. For a more detailed discussion and applications of the bootstrapping techniques see, for example, [Diac 83], [Efro 79], [Jain 87a], [Jain 87b].

16.3 HYPOTHESIS TESTING IN CLUSTER VALIDITY

In this framework, the null hypothesis H_0 will be expressed in a slightly different way. This is because our major concern is not to test a parameter against a specific value. In contrast, our concern here is to test whether the data of X possess a "random" structure or not. *Thus, in this case, the null hypothesis H_0 should be a statement of randomness concerning the structure of X.* Thus, our goal is now twofold.

- First, we must generate a reference *data population under the random hypothesis*, that is, a data population that models a random structure.
- Second, we must define an appropriate statistic, whose values are indicative of the structure of a data set, and compare the value that results from

our data set X against the value obtained from the reference (random) population.

There are three different ways to generate the reference population under the null (randomness) hypothesis, each being appropriate for different situations.

- *Random position hypothesis.* This hypothesis is appropriate for ratio data. It *requires* that *"All the arrangements of N vectors in a specific region of the l-dimensional space are equally likely to occur."* Such regions may be the H_l hypercube or the l-dimensional hypersphere. One way to produce such an arrangement is to insert each point randomly in this region of the l-dimensional space, according to the uniform distribution. The random position hypothesis can be used with either external or internal criteria.
  - — *Internal criteria.* In this case, the statistic q is defined so as to measure the degree to which a clustering structure, produced by a clustering algorithm, matches the proximity matrix of the corresponding data set. Let X_i be a set of N vectors generated according to the random position hypothesis and P_i be the corresponding proximity matrix. In the sequel, we apply the same clustering algorithm to each X_i and to our data set X and let $\mathcal{C}_i$ and $\mathcal{C}$ be the resulting clustering structures, respectively. For each case, the value of the statistic q is computed. The random hypothesis, H_0, is then rejected if the value q, resulting from X lies in the critical interval $\bar{D}_\rho$ of the statistic pdf of the reference population (i.e., under H_0), that is, if q is unusually small or large.
  - — *External criteria.* The statistic q is defined so as to measure the degree of correspondence between a *prespecified structure* $\mathcal{P}$ imposed on X and the clustering that results after the application of a specific clustering algorithm to X. Then, the value of q corresponding to the clustering $\mathcal{C}$ resulting from the data set X is tested against the q_i's, corresponding to the clusterings resulting from the reference population generated under the random position hypothesis. Once more, the random hypothesis is rejected if q is unusually large or small .
- *Random graph hypothesis.* It is usually adopted when only internal information (i.e., information that concerns only the vectors themselves or their relationships) is available. It is appropriate when ordinal proximities between vectors are used. Before we proceed, let us define the ordinal, or rank order, $N \times N$ matrix A as a symmetric matrix with zero diagonal elements (provided that dissimilarity measures are used) and with its upper diagonal elements being integers in the range $[1, N(N-1)/2]$. The entry $A(i, j)$ of A provides only qualitative information about the dissimilarity between the corresponding vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. If, for example, $A(2, 3) = 3$ and $A(2, 5) = 5$, we can only conclude that $\boldsymbol{x}_3$ is more similar to $\boldsymbol{x}_2$ than $\boldsymbol{x}_5$. That is, in this context, comparing dissimilarities is

meaningless (recall the comments made in Chapter 11, concerning ordinal type data).

Let A_i be an $N \times N$ rank order proximity matrix with no ties; that is, all entries in the upper diagonal are different from each other. Under the random graph hypothesis, the reference population consists of such matrices A_i each one generated by inserting randomly the integers in the range $[1, N(N-1)/2]$, in its upper diagonal entries. Let P be the ordinal proximity matrix associated with the given data set X and C be the clustering structure produced by the application of a specific algorithm to P. Finally, let C_i be the clustering structure produced when the same algorithm is applied to A_i. We may now proceed as in the previous case and define a statistic q that measures the agreement between a rank order (proximity) matrix and the corresponding clustering structure. If the value of q, corresponding to P and C, is unusually large or small, the random hypothesis is rejected.

It must be emphasized that the random graph hypothesis is not appropriate for ratio-scaled data. Let us take, for example, the case where the Euclidean distance is in use and $l \leq N-2$ and consider the points $x_1 = 0$, $x_2 = 1$, $x_3 = 3$ on the real line. It is clear that the distance between x_1 and x_3 cannot be smaller than the distance between x_2 and x_3. That is, the matrix

$$A = \begin{bmatrix} 0 & 2 & 1 \\ 2 & 0 & 3 \\ 1 & 3 & 0 \end{bmatrix}$$ is not a valid proximity matrix for these ratio-scaled data.

- *Random label hypothesis.* Let us consider all possible partitions, $\mathcal{P}'$, of X into m groups. Each partition may be defined in terms of a mapping g from X to $\{1, \ldots, m\}$. The random label hypothesis assumes that *all* possible mappings are *equally likely*. The statistic q can be defined so as to measure the degree to which information inherent in the data set X, such as the proximity matrix P, matches a specific partition. The statistic q is then used to test the degree of match between P and an externally imposed partition $\mathcal{P}$, against the q_i's corresponding to the random partitions generated under the random label hypothesis. Once more, H_0 is then rejected if q is unusually large or small.

In the sequel, we give a number of statistic indices appropriate for external and, then, for internal criteria.

16.3.1 External Criteria

External criteria are used either (a) for the comparison of a clustering structure C, produced by a clustering algorithm, with a partition $\mathcal{P}$ of X drawn independently from C or (b) for measuring the degree of agreement between a predetermined partition $\mathcal{P}$ and the proximity matrix of X, P.

Comparison of $\mathcal{P}$ with a Clustering $\mathcal{C}$

In this case, $\mathcal{C}$ may be either a specific hierarchy of clusterings or a specific clustering. The latter may be produced either by cutting the dendrogram produced by a hierarchical algorithm at a given level (see Chapter 13) or by any other algorithm discussed in the previous chapters. However, a prespecified hierarchy of partitions is rarely available in practice. Thus, the problem of validating hierarchies of clusterings is of limited practical interest.

In the sequel, we consider the validation task concerning a clustering, $\mathcal{C}$, resulting from a specific clustering algorithm, in terms of an independently drawn partition $\mathcal{P}$ of X. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ and $\mathcal{P} = \{P_1, \ldots, P_s\}$. Note that the number of clusters in $\mathcal{C}$ need not be the same as the number of groups in $\mathcal{P}$. Our goal is to define appropriate statistical indices to be used for the hypothesis test.

Let n_{ij} denote the number of vectors that belong to C_i and P_j simultaneously. Also let $n_i^C = \sum_{j=1}^{s} n_{ij}$; that is, n_i^C is the number of vectors that belong to C_i. Similarly, we define the number of vectors that belong to P_j as $n_j^P = \sum_{i=1}^{m} n_{ij}$.

Consider a pair of vectors $(\boldsymbol{x}_v, \boldsymbol{x}_u)$. We refer to it as (a) SS if both vectors belong to the same cluster in $\mathcal{C}$ and to the same group in $\mathcal{P}$, (b) DD if both vectors belong to different clusters in $\mathcal{C}$ and to different groups in $\mathcal{P}$, (c) SD if the vectors belong to the same cluster in $\mathcal{C}$ and to different groups in $\mathcal{P}$, and (d) DS if the vectors belong to different clusters in $\mathcal{C}$ and to the same group in $\mathcal{P}$. Let a, b, c, and d be the number of SS, SD, DS, and DD pairs of vectors of X, respectively. Then $a+b+c+d = M$, where M is the total number of possible pairs in X, that is, $M = N(N-1)/2$.

Example 16.1. Let $X = \{\boldsymbol{x}_i, i = 1, \ldots, 6\}$, $\mathcal{C} = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}, \{\boldsymbol{x}_4, \boldsymbol{x}_5\}, \{\boldsymbol{x}_6\}\}$, and $\mathcal{P} = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}, \{\boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\}\}$. The following table shows the type of all pairs of vectors in X.

	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$	$\boldsymbol{x}_6$
$\boldsymbol{x}_1$		SS	SS	DD	DD	DD
$\boldsymbol{x}_2$			SS	DD	DD	DD
$\boldsymbol{x}_3$				DD	DD	DD
$\boldsymbol{x}_4$					SS	DS
$\boldsymbol{x}_5$						DS
$\boldsymbol{x}_6$						

From this table we obtain $a = 4$, $b = 0$, $c = 2$ and $d = 9$.

Let $m_1 = a+b$ be the number of pairs of vectors that belong to the same cluster in $\mathcal{C}$ and $m_2 = a+c$ be the number of pairs of vectors that belong to the same group in $\mathcal{P}$. Using the preceding definitions, we can define statistical indices (statistics) in order to measure the degree to which $\mathcal{C}$ matches $\mathcal{P}$. Such statistical indices are

the following:

- Rand statistic

$$R = (a + d)/M \tag{16.5}$$

- Jaccard coefficient

$$J = a/(a + b + c) \tag{16.6}$$

- Fowlkes and Mallows index

$$FM = a/\sqrt{m_1 m_2} = \sqrt{\frac{a}{a+b}\frac{a}{a+c}} \tag{16.7}$$

The term $a + d$ is the number of SS pairs of vectors plus the number of DD pairs. Thus, the Rand statistic measures the fraction of the total number of pairs that are either SS or DD. The Jaccard coefficient follows the same philosophy as the Rand statistic, except that it excludes d. The values of these two statistics are between 0 and 1. However, a prerequisite for achieving the maximum value is to have $m = s$, which, in general, is not always the case.

For all the above defined indices, it is clear that the larger their value (1), the higher the agreement between $\mathcal{C}$ and $\mathcal{P}$, i.e., *all the corresponding statistical tests are right tailed.*

Another very popular statistic that is frequently used in conjunction with external criteria is Hubert's Γ statistic (e.g., [Hube 76],[Mant 67], [Bart 62]). It measures the correlation between two matrices, X and Y, of dimension $N \times N$, drawn independently of each other. For symmetric matrices this can be written as

- Hubert's Γ statistic

$$\Gamma = (1/M) \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} X(i, j) Y(i, j) \tag{16.8}$$

where $X(i, j)$ and $Y(i, j)$ are the (i, j) elements of the matrices X and Y, respectively. High values of Γ indicate close agreement between X and Y. The normalized version of the Γ statistic, denoted by $\hat{\Gamma}$, is also used.

- Normalized Γ statistic

$$\hat{\Gamma} = \frac{(1/M) \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (X(i, j) - \mu_X)(Y(i, j) - \mu_Y)}{\sigma_X \sigma_Y} \tag{16.9}$$

where μ_X, μ_Y, σ_X^2, and σ_Y^2 are the respective means and variances, that is, $\mu_X = (1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} X(i, j)$, $\sigma_X^2 = (1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} X(i, j)^2 - \mu_X^2$ (similarly we define μ_Y and σ_Y^2). The values of $\hat{\Gamma}$ are between -1 and 1.

Let us set $X(i, j)$ equal to 1 if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same cluster in $\mathcal{C}$ and 0 otherwise, and $Y(i, j)$ equal to 1 if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same group in $\mathcal{P}$ and 0 otherwise. It can then be shown (see Problem 16.2) that in this case the $\hat{\Gamma}$ statistic becomes equal to

$$\hat{\Gamma} = (Ma - m_1 m_2)/\sqrt{m_1 m_2 (M - m_1)(M - m_2)} \tag{16.10}$$

Unusually large absolute values of $\Gamma(\hat{\Gamma})$ suggest that $\mathcal{C}$ and $\mathcal{P}$ agree with each other.

As almost always happens in practice, the exact computation of the pdf of all these indices, under the null hypothesis, is very difficult. Thus, we use Monte Carlo techniques for their estimation. In the sequel, we discuss such a procedure, which is based on the random position hypothesis. Data are assumed to be ratio scaled.

- For $i = 1$ to r
  - — Generate a data set X_i of N vectors in the area of interest of X, so that the vectors are uniformly distributed in it.
  - — Assign each vector $\boldsymbol{y}_j^i \in X_i$ to the group where the $\boldsymbol{x}_j \in X$ belongs, according to the structure imposed by $\mathcal{P}$.
  - — Run the same clustering algorithm, used for obtaining $\mathcal{C}$, on X_i and let $\mathcal{C}_i$ be the resulting clustering.
  - — Compute the value $q(\mathcal{C}_i)$ of the corresponding statistical index q for $\mathcal{P}$ and $\mathcal{C}_i$.
- End {For}
- Create the histogram of $q(\mathcal{C}_i)$'s.

The following example demonstrates how this methodology can be used in practice.

Example 16.2. (a) Consider a data set X of 100 vectors in the H_3 hypercube. The data are generated to form four groups, each consisted of 25 vectors. Each group is generated by a normal distribution. The first group of 25 vectors of X is generated from the first distribution while the second, third, and fourth groups of 25 vectors are generated from the second, the third, and the fourth distribution, respectively. The covariance matrices of all distributions are equal to $0.2I$, where I is the 3×3 identity matrix. The mean vectors for the four distributions are $[0.2, 0.2, 0.2]^T$, $[0.5, 0.2, 0.8]^T$, $[0.5, 0.8, 0.2]^T$, and $[0.8, 0.8, 0.8]^T$, respectively. If a distribution generates a vector that is outside the unit hypercube, it is ignored and replaced by another that lies inside H_3. It is not difficult to realize that the points of X form four compact and well-separated clusters.

We assume that the external information is: "The vectors of X belong to four different groups P_1, P_2, P_3, and P_4, such that P_1 contains the first 25 vectors of X and P_2, P_3, and P_4 contain the second, third, and fourth groups of 25 vectors of X, respectively."

We run the isodata algorithm for $m = 4$ and let C be the resulting clustering. We compute the values of the Rand, R, the Jaccard, J, the Fowlkes and Mallows, FM, and the $\hat{\Gamma}$ statistics for C and $\mathcal{P}$. These are 0.91, 0.68, 0.81 and 0.75, respectively. Next, we estimate the distribution of these statistics using the procedure described before. Specifically, 100 data sets $X_i, i = 1, \ldots, 100$, are generated, each of them consisting of 100 randomly selected vectors in H_3, following the uniform distribution. According to the $\mathcal{P}$ defined earlier, we assign the first 25 of them to P_1 and the second, third, and fourth groups of 25 vectors to P_2, P_3, and P_4, respectively. For each X_i we run the isodata algorithm for $m = 4$ and we produce the clustering $C_i, i = 1, \ldots, 100$. Then we compute the values of the four statistics, R_i, J_i, FM_i, and $\hat{\Gamma}_i$ for each C_i and $\mathcal{P}$, $i = 1, \ldots, 100$. We set the significance level at $\rho = 0.05$. Then, in terms of a given statistic, we accept or reject the null hypothesis (i.e., the random hypothesis) according to the conditions given in section 16.2. In our case R is greater than all R_i's. Similarly, J, FM, and $\hat{\Gamma}$ are greater than all J_i's, FM_i's, and $\hat{\Gamma}_i$'s, respectively. Thus, all statistics reject the null hypothesis at significance level $\rho = 0.05$.

(b) Now let X' be a data set constructed as X, but with the covariance matrices of the normal distributions equal to $0.6I$. In this case, the vectors of X form weak clusters, that is, clusters that exhibit "large" spread around their mean vector. The values of the four statistics in this case are $R = 0.64$, $J = 0.15$, $FM = 0.27$, and $\hat{\Gamma} = 0.03$. R is greater than 99 of the R_i's. Similarly, J, FM, and $\hat{\Gamma}$ are greater than 94 J_i's, 94 FM_i's, and 98 $\hat{\Gamma}_i$'s, respectively. Thus, according to the Rand and $\hat{\Gamma}$ statistics, the null hypothesis is rejected at significance level $\rho = 0.05$. However, this is not the case for the other two indices.

This situation illustrates the fact that different statistics may lead to different conclusions when no clear-cut situations are considered (see also comparative studies in [Mill 80], [Mill 83], [Mill 85]).

(c) Let us now construct X'' by selecting the covariance matrices equal to $0.8I$. In this case, the vectors of X'' are so dispersed that, practically, X'' does not exhibit any clustering structure. The values of the four statistics in this case are $R = 0.63$, $J = 0.14$, $FM = 0.25$, and $\hat{\Gamma} = -0.01$. Specifically, R is greater than 62, from the total of 100, R_i's. Similarly, J, FM, and $\hat{\Gamma}$ are greater than 48 R_i's, 48 J_i's, and 55 $\hat{\Gamma}_i$'s, respectively. Thus, according to all statistics, the null hypothesis is not rejected at significance level $\rho = 0.05$.

Remark

- For each of these statistics, q, there exists a corresponding "corrected" statistic q', which is a normalized version of q and is defined as

$$q' = \frac{q - E(q)}{\max(q) - E(q)} \tag{16.11}$$

where $\max(q)$ is the maximum possible value of q and $E(q)$ is the mean value of q, under the null hypothesis. Its values are between 0 and 1. The maximum value is always achievable when a perfect match between C and $\mathcal{P}$

occurs and the minimum if $\mathcal{C}$ and $\mathcal{P}$ have been chosen by chance. The problem encountered here is the computation of $E(q)$ and $\max(q)$. This problem is attacked in [Hube 85], for the Rand statistic, under the assumption that the maximum value of the Rand statistic is 1. The same problem for the Fowlkes–Mallows index is treated in [Fowl 83].

Assessing the Agreement Between $\mathcal{P}$ and Proximity Matrix P

In this section, we show that the Γ statistic can be used to measure the degree to which the proximity matrix P of X matches a partition $\mathcal{P}$, which is imposed a priori on X. Recall that $\mathcal{P}$ may be viewed as a mapping g of X to $\{1, \ldots, m\}$. Let us consider the matrix Y whose (i, j) element, $Y(i, j)$, is defined as follows:

$$Y(i, j) = \begin{cases} 1, & \text{if } g(\boldsymbol{x}_i) \neq g(\boldsymbol{x}_j) \\ 0, & \text{otherwise} \end{cases} \tag{16.12}$$

for $i, j = 1, \ldots, N$. It is clear that Y is symmetric. Then, the Γ (or $\hat{\Gamma}$) statistic is applied to the proximity matrix P and Y. Its value is a measure of the degree to which Y matches P.

In order to estimate the pdf of Γ (or $\hat{\Gamma}$) under the *random label hypothesis*, we produce, say, r mappings g_i, $i = 1, \ldots, r$.[3] For each of them we form the corresponding Y_i matrix and we apply the Γ (or $\hat{\Gamma}$) statistic to P and Y_i, $i = 1, \ldots, r$. Then we proceed as usual for the acceptance or rejection of the random label hypothesis.

Example 16.3. We consider a data set X of 64 two-dimensional vectors. The first 16 of them spring out of a normal distribution with mean $[0.2, 0.2]^T$, and the remaining three groups of 16 vectors stem from three normal distributions with means $[0.2, 0.8]^T$, $[0.8, 0.2]^T$, and $[0.8, 0.8]^T$, respectively. The covariance matrices of all distributions are equal to $0.15I$. Let P be the proximity matrix of X when the squared Euclidean distance is in use. Also, we set the significance level at $\rho = 0.05$.

(a) Let $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$. Suppose that the first set of 16 vectors is assigned to P_1, the second is assigned to P_2, the third is assigned to P_3, and the last to P_4. Based on this information, we form Y as described before and we compute the value of $\hat{\Gamma}$ for P and Y, which is found to be 0.77. Then we generate random partitions $\mathcal{P}_i$, $i = 1, \ldots, 100$, we form the corresponding matrices Y_i, and we compute the values $\hat{\Gamma}_i$ between P and each of the Y_i's. It turns out that $\hat{\Gamma}$ is greater than all of these values. Thus, the null hypothesis is rejected at significance level ρ.

(b) Assume now that the external information $\mathcal{P}$ assigns randomly 16 vectors of X to each P_i. It is clear that the external information does not agree with the underlying structure

[3] Typically, $r = 100$.

of X. If we apply the same procedure as before, we find that $\hat{\Gamma} = -0.01$, which is less than 70 values of $\hat{\Gamma}_i$. Thus, the randomness hypothesis is accepted.

16.3.2 Internal Criteria

Our aim here is to verify whether the clustering structure produced by a clustering algorithm fits the data, using only information inherent in the data. In the sequel, unless otherwise stated, we consider the case in which the data are represented by their proximity matrix. Two cases are considered: (a) the clustering structure is a hierarchy of clusterings and (b) the clustering structure consists of a single clustering.

Validation of Hierarchies of Clusterings

We recall that the dendrogram produced by a hierarchical clustering algorithm may be represented by the respective cophenetic matrix, P_c. *We will define statistical indices that measure the degree of agreement between the cophenetic matrix, P_c, produced by a specific hierarchical clustering algorithm, with the proximity matrix P of X.* Because both matrices are symmetric and have their diagonal elements equal to 0,[4] we consider only the $M \equiv N(N-1)/2$ upper diagonal elements of P_c and P. Let d_{ij} and c_{ij} be the (i, j) element of P and P_c, respectively.

The first index, known as the *cophenetic correlation coefficient* (*CPCC*) measures the correlation between P_c and P and is used when the matrices are interval or ratio scaled. It is defined as

$$CPCC = \frac{(1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} d_{ij}c_{ij} - \mu_p\mu_c}{\sqrt{\left((1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} d_{ij}^2 - \mu_p^2\right)\left((1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} c_{ij}^2 - \mu_c^2\right)}} \tag{16.13}$$

where the corresponding mean values are defined as in Eq. (16.9). It can be shown that the values of the *CPCC* are between -1 and 1 (see Problem 16.4). The closer the *CPCC* index to 1, the better the agreement between the cophenetic and the proximity matrix. The *CPCC* statistic has been studied by various researchers (see, e.g., [Rolp 68], [Rolp 70], [Farr 69]). The major difficulty associated with it is that it depends on many parameters of the problem, such as the size of X, the clustering algorithm used and the employed proximity measure. Hence, the exact computation of its pdf under H_0 is very difficult. Once more, one is forced to use Monte Carlo techniques for the estimation of its distribution, under H_0. According to the random position hypothesis, we generate r sets X_i, whose vectors are randomly distributed according to the uniform distribution, and we apply to

[4] This implies that we use a dissimilarity measure.

each X_i the same hierarchical algorithm that has produced P_c. Then, we compute *CPCC* for the proximity matrix of X_i, P_i, and the resulting cophenetic matrix, P_{c_i} and we construct the corresponding histogram.

Interestingly enough, in [Rolp 70], it is stated that even high values of *CPCC* (near 0.9) should be handled with caution when the unweighted pair group method average (UPGMA) algorithm is in use (Chapter 13), as there are cases for which even such large values cannot guarantee close agreement between the cophenetic and the proximity matrix.

Another statistical index, which is suitable for cases in which P_c and P are ordinally scaled, is the γ statistic, which is described in the sequel. Let $\boldsymbol{v}_p$ and $\boldsymbol{v}_c$ be two vectors of dimension $N(N-1)/2$, each containing the upper diagonal elements of P and P_c, respectively, ordered by rows. Let (v_{p_i}, v_{p_j}) and (v_{c_i}, v_{c_j}) be two pairs of elements of $\boldsymbol{v}_p$ and $\boldsymbol{v}_c$, respectively. The following definitions are in order.

A set of pairs $\{(v_{p_i}, v_{p_j}), (v_{c_i}, v_{c_j})\}$ is called

- concordant if

$$\left((v_{p_i} < v_{c_i}) \,\&\, (v_{p_j} < v_{c_j})\right) \text{ or } \left((v_{p_i} > v_{c_i}) \,\&\, (v_{p_j} > v_{c_j})\right)$$

- discordant if

$$\left((v_{p_i} < v_{c_i}) \,\&\, (v_{p_j} > v_{c_j})\right) \text{ or } \left((v_{p_i} > v_{c_i}) \,\&\, (v_{p_j} < v_{c_j})\right)$$

Finally, a set of pairs is neither concordant nor discordant if $v_{p_i} = v_{c_i}$ or $v_{p_j} = v_{c_j}$. Let S_+ and S_- be the numbers of the concordant and discordant pairs, respectively. Then γ is defined as

$$\gamma = \frac{S_+ - S_-}{S_+ + S_-} \tag{16.14}$$

The γ statistic takes values between -1 and 1.

Example 16.4. Let $\boldsymbol{v}_p = [3, 2, 1, 5, 2, 6]^T$ and $\boldsymbol{v}_c = [2, 3, 5, 1, 6, 4]^T$. For all possible 16 pairs of pairs we have

Index	$\boldsymbol{v}_p$	$\boldsymbol{v}_c$		Index	$\boldsymbol{v}_p$	$\boldsymbol{v}_c$	
(1, 2)	(3, 2)	(2, 3)	dis.	(2, 6)	(2, 6)	(3, 4)	dis.
(1, 3)	(3, 1)	(2, 5)	dis.	(3, 4)	(1, 5)	(5, 1)	dis.
(1, 4)	(3, 5)	(2, 1)	con.	(3, 5)	(1, 2)	(5, 6)	con.
(1, 5)	(3, 2)	(2, 6)	dis.	(3, 6)	(1, 6)	(5, 4)	dis.
(1, 6)	(3, 6)	(2, 4)	con.	(4, 5)	(5, 2)	(1, 6)	dis.
(2, 3)	(2, 1)	(3, 5)	con.	(4, 6)	(5, 6)	(1, 4)	con.
(2, 4)	(2, 5)	(3, 1)	dis.	(5, 6)	(2, 6)	(6, 4)	dis.
(2, 5)	(2, 2)	(3, 6)	con.				

Thus, $S_+ = 6$, $S_- = 9$ and $\gamma = -1/5 = -0.2$.

The γ statistic depends on all the factors of the problem at hand and, as a consequence, the estimate of its pdf under the randomness hypothesis (H_0) is also difficult to derive. Thus, one has to use Monte Carlo techniques once again for the estimation of the pdf of γ under H_0. In this case, the random graph hypothesis is used. Specifically, we produce r random rank order proximity matrices P_i, with no ties, and we run the algorithm that produced P_c on each of them. Then we compute the value of γ for each P_i and its corresponding cophenetic matrix P_{c_i} and we form the histogram for the values of γ.

Remarks

- It has been conjectured [Hube 74] that when the single and the complete link algorithms are used, the statistic $N\gamma - a \ln N$ follows (approximately) the standard normal distribution. The constant a is set equal to 1.1 (1.8) when the single (complete) link algorithm is used. If we adopt this conjecture, it relieves us of the computational burden of the Monte Carlo method.
- The γ statistic may also be used to compare the results for two different hierarchies of clusterings resulting from two different clustering algorithms. (e.g., [Bake 74],[Hube 74], Problem 16.5).

Another measure that is suitable for ordinal-scaled P and P_c is Kudall's τ statistic [Cunn 72], which is defined as

$$\tau = \frac{S_+ - S_-}{N(N-1)/2} \tag{16.15}$$

The difference from the γ statistic is that the denominator here extends to all sets of pairs, whereas in the case of the γ statistic the sets of pairs that are neither concordant nor discordant are excluded.

Validation of Individual Clusterings

Our goal here is to investigate whether a given clustering $\mathcal{C}$, consisting of m clusters, matches information that is inherent in the data set X. In the sequel, we show that the Γ (or $\hat{\Gamma}$) statistic can be used in order to achieve this goal. Once again, we use the proximity matrix P as a measure representing the structural information inherent in the data. The (i, j) element of the matrix Y is defined as

$$Y(i, j) = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ and } \boldsymbol{x}_j \text{ belong to different clusters} \\ 0, & \text{otherwise} \end{cases} \tag{16.16}$$

for $i, j = 1, \ldots, N$. It is clear that Y is symmetric. Then the Γ (or $\hat{\Gamma}$) statistic is applied to P and Y. Its value is a measure of the degree of correspondence between P and Y.

The *random position hypothesis* is employed. For each of the resulting random data sets X_i, the proximity matrix P_i is computed. Then we apply, to each of them, the clustering algorithm used to produce $\mathcal{C}$. Let $\mathcal{C}_i$, $i = 1, \ldots, r$, be the resulting clusterings of m clusters. We compute Y_i and Γ_i. Finally, we decide for the rejection or acceptance of the null hypothesis at a given significance level ρ according to the conditions given in Section 16.2.

Example 16.5. Consider a data set X of 100 vectors in the H_2 hypercube. The vectors are generated to form four groups, each of 25 vectors. Each group is generated by a normal distribution. The corresponding covariance matrices are all equal to $0.1I$ and the mean vectors are $[0.2, 0.2]^T$, $[0.8, 0.2]^T$, $[0.2, 0.8]^T$, $[0.8, 0.8]^T$, respectively. We apply the isodata algorithm and let $\mathcal{C}$ be the resulting clustering. Computing the corresponding matrices Y and P, we obtain $\hat{\Gamma} = 0.5704$. Then we generate 100 data sets, X_i, whose vectors are randomly distributed in H_2, following the uniform distribution. The isodata algorithm is applied to each of them, and let $\mathcal{C}_i$, $i = 1, \ldots, 100$, be the resulting clusterings. Computing Y_i and P_i associated with the resulting clusterings for each X_i, it turns out that 99 of the corresponding $\hat{\Gamma}_i$ values are smaller than $\hat{\Gamma}$. Thus, the null hypothesis is rejected at significance level $\rho = 0.05$.

Repeating the experiment but with covariance matrices equal to $0.2I$, we find that $\hat{\Gamma}$ is greater than 86 of 100 $\hat{\Gamma}_i$ values. Thus, the null hypothesis is not rejected at significance level $\rho = 0.05$.

16.4 RELATIVE CRITERIA

So far, clustering validation has been performed on the basis of statistical tests. A major drawback of most of these techniques is their high computational demands, due to the required Monte Carlo methodology. In this section, a different approach is discussed that does not involve statistical tests. To this end, *a set of clusterings is considered and the goal is to choose the best one according to a prespecified criterion.* More specifically, let $\mathcal{A}$ be the set of parameters associated with a specific algorithm. For example, for the algorithms of Chapter 14, $\mathcal{A}$ contains the number of clusters, m, as well as the initial estimates of the parameter vectors associated with each cluster. The problem can be stated as follows:

"Among the clusterings produced by a specific clustering algorithm, for different values of the parameters in $\mathcal{A}$, choose the one that best fits the data set X."

We consider the following cases:

- *$\mathcal{A}$ does not contain the number of clusters, m*, as a parameter (such as the algorithms based on graph theory, the morphological clustering algorithm and the boundary detection algorithms).

  The choice of the "best" parameter values for this type of algorithm is based on the assumption that *if X possesses a clustering structure, this structure is captured for a "wide" range of values of the parameters* in $\mathcal{A}$

(e.g., [Post 93]). Based on this assumption, we proceed as follows. We run the algorithm for a wide range of values of its parameters and we choose the widest range for which, m, remains constant (typically $m << N$). Then we choose as appropriate values of the parameters of $\mathcal{A}$ the values that correspond to the middle of this range. Note that, implicitly, this procedure also identifies the number of clusters that underlie X.

Example 16.6. (a) We consider a data set X, consisted of three groups of 100 two-dimensional vectors. These groups are formed from normal distributions with means $[0, 0]^T$, $[8, 4]^T$, and $[8, 0]^T$, respectively, and covariance matrices equal to $1.5I$. As one can easily observe in Figure 16.2a, the three groups form three compact and well-separated clusters. We run the binary morphology clustering algorithm (BMCA), using the 3×3 structuring element (Figure 15.11a), with the resolution parameter r ranging from 1 to 77 and we plot the number of clusters versus r (Figure 16.2b). We observe that for any value of r between 27 and 67, the number of clusters remains contant and equal to 3. Taking into account that this range of values is the largest one, we choose $r = 47$, and we conclude that our data form three clusters.

(b) Generate another data set, as before, but with the covariance matrices equal to $2.5I$. This data set is depicted in Figure 16.3a. We observe that in this case the three groups are so dispersed that they practically cannot be distinguished from each other. We run BMCA once again, using the 3×3 structuring element, for r ranging from 1 to 77, with step 1, and we plot the number of clusters versus r (see Figure 16.3b). In this case, for $r = 7, \ldots, 46$, the number of clusters remains constant and the corresponding value of m is 1.

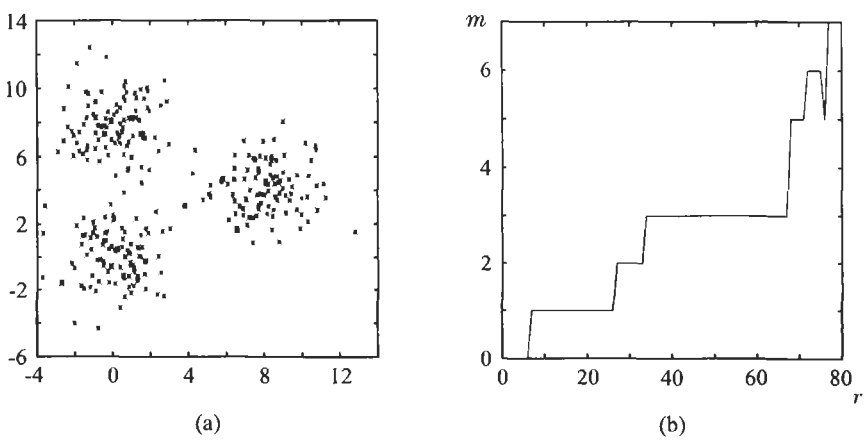


FIGURE 16.2: (a) Three well-separated clusters. (b) The plot of the number of clusters m versus the resolution parameter r, using the binary morphology clustering algorithm (BMCA).

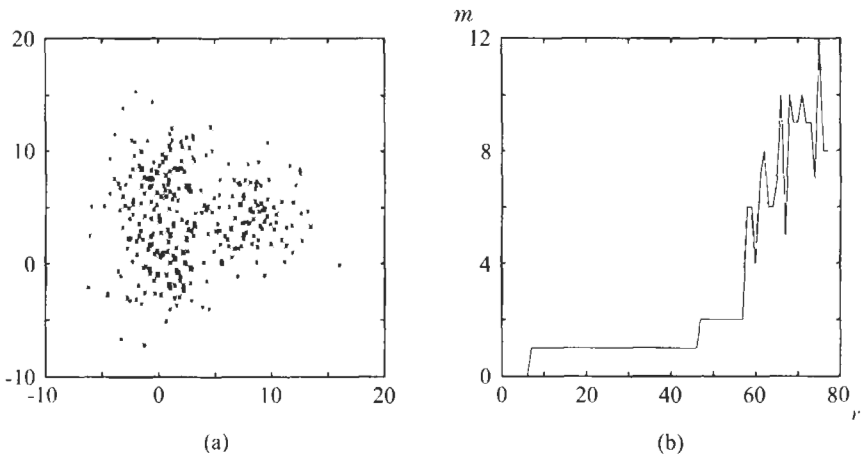


FIGURE 16.3: (a) Three overlapped clusters. (b) The plot of m versus r.

- *$\mathcal{A}$ contains m as a parameter* (such as the fuzzy and hard clustering algorithms discussed in Chapter 14). For this case, a different procedure is followed. We first select a suitable performance index q. The "best" clustering is identified, in terms of q, via the following procedure. We run the clustering algorithm at hand for all values of m between a minimum $m_{\min}$ and a maximum $m_{\max}$, where $m_{\min}$ and $m_{\max}$ are chosen a priori. For each value of m, we run the algorithm r times, using different sets of values for the other parameters of $\mathcal{A}$.[5] Then we plot the best values of q, obtained for each m, versus m and we seek the maximum or the minimum of this plot, according to whether large or small values of q indicate good clusterings. This procedure works well if q exhibits no trend with respect to m. However, as we will see, several of the commonly used indices q exhibit an increasing (decreasing) trend as m increases. Thus, locating the maximum (minimum) versus m is no longer indicative of a good clustering. For indices that exhibit such behavior, in the range $[m_{\min}, m_{\max}]$, we search for values of m at which a significant local change in the value of q occurs. This change appears in the plot as a significant "knee". *The presence of such a knee is an indication of the number of clusters underlying X. On the other hand, the absence of such a knee may be an indication that X possesses no clustering structure.*

  Another source of complication, associated with many of the indices used in this framework is that their behavior depends on many other factors such

[5] For example, if the isodata algorithm is used, we run it using different initial conditions.

as the number of vectors in X and their dimensionality. The situation is demonstrated via the following example.

Example 16.7. (a) In this example, we consider 16 different data sets with different numbers of vectors and dimensionalities. Specifically, we consider four 2-dimensional data sets of 50, 100, 150, and 200 vectors; four 4-dimensional data sets of 50, 100, 150, and 200 vectors; four 6-dimensional data sets of 50, 100, 150, and 200 vectors; and four 8-dimensional data sets of 50, 100, 150, and 200 vectors. The vectors of the data sets lie in the H_i hypercube, $i = 2, 4, 6, 8$, respectively. All the data sets contain four compact and well-separated clusters. All the clusters stem from normal distributions with means $\overbrace{[0.2, \ldots, 0.2]}^{i}{}^T$, $\overbrace{[0.2, \ldots, 0.2,}^{i/2}$ $\overbrace{0.8, \ldots, 0.8]}^{i/2}{}^T$, $\overbrace{[0.8, \ldots, 0.8,}^{i/2}$ $\overbrace{0.2, \ldots, 0.2]}^{i/2}{}^T$, $\overbrace{[0.8, \ldots, 0.8]}^{i}{}^T$, where i is the dimensionality, and covariance matrices $0.2I_i$, where I_i is the $i \times i$ identity matrix. For each of these data sets we run the isodata algorithm for $m = 1, \ldots, 10$, and we compute the corresponding values of the cost function J. For this case, only a single run is performed for each m. In Figure 16.4 we plot J versus the number of clusters, m, for the cases of 50, 100, 150, and 200 vectors and for different dimensionalities.

One can easily notice that the higher the dimensionality, the sharper the knee at $m = 4$. Moreover, as the size of the data set increases, the knee at $m = 4$ becomes sharper, even at lower dimensionalities. Rules for automatic identification of a knee are discussed in [Dube 87a].

(b) We again construct 16 data sets, but now the vectors in each of them are randomly distributed in the H_i hypercube, according to the uniform distribution. If we carry out the same procedure as before, we see in Figure 16.5 that there are no sharp knees in the plots. Thus, the absence of sharp knees in the plots *may be* an indication of the absence of clustering structure.

16.4.1 Hard Clustering

In this section we discuss indices that are suitable for hard clusterings. In the sequel unless otherwise stated, we consider only the case of compact clusters.

- *The modified Hubert* Γ *statistic.* Let $c_i = k$ if the vector $\boldsymbol{x}_i$ belongs to cluster C_k. Also let Q be the $N \times N$ matrix whose (i, j) element, $Q(i, j)$, is equal to the distance $d(\boldsymbol{w}_{c_i}, \boldsymbol{w}_{c_j})$ between the representatives of the clusters where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong. The modified Hubert Γ statistic is defined as in Eq. (16.8) and it is applied to the proximity matrix P of the data set X and the matrix Q (of course, the same distance measure must be used for both P and Q). Similarly, we can define the normalized modified Hubert Γ statistic. It is clear that if $d(\boldsymbol{w}_{c_i}, \boldsymbol{w}_{c_j})$ is close to $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, for $i, j=1, \ldots, N$, i.e., when compact clusters are encountered in X, P and Q will be in close agreement and the values of Γ and $\hat{\Gamma}$ will be high. Conversely, high values of $\Gamma(\hat{\Gamma})$

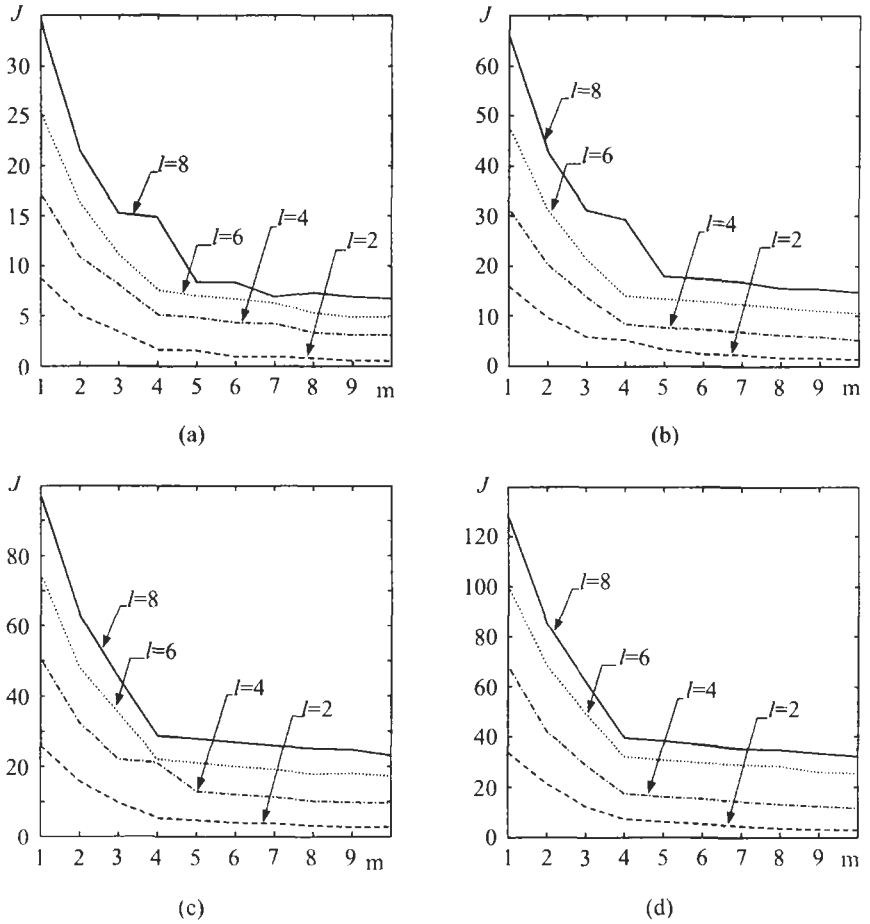


FIGURE 16.4: Plots of J versus m for (a) $N = 50$, (b) $N = 100$, (c) $N = 150$, (d) $N = 200$, for clustered data.

indicate the existence of compact clusters. If the opposite is true, the values of the modified Γ and $\hat{\Gamma}$ indices are expected to be low. Thus, in the plot of $\hat{\Gamma}$ versus m, we seek a significant knee that corresponds to a significant increase of $\hat{\Gamma}$. The value of m at which this knee occurs indicates the number of clusters that underlie X.

For $m = 1$ and $m = N$ the index is not defined. Also, this index tends to increase as m increases toward N (see Problem 16.6) for random data and tends to be flat for data sets that possess a clustering structure [Jain 88].

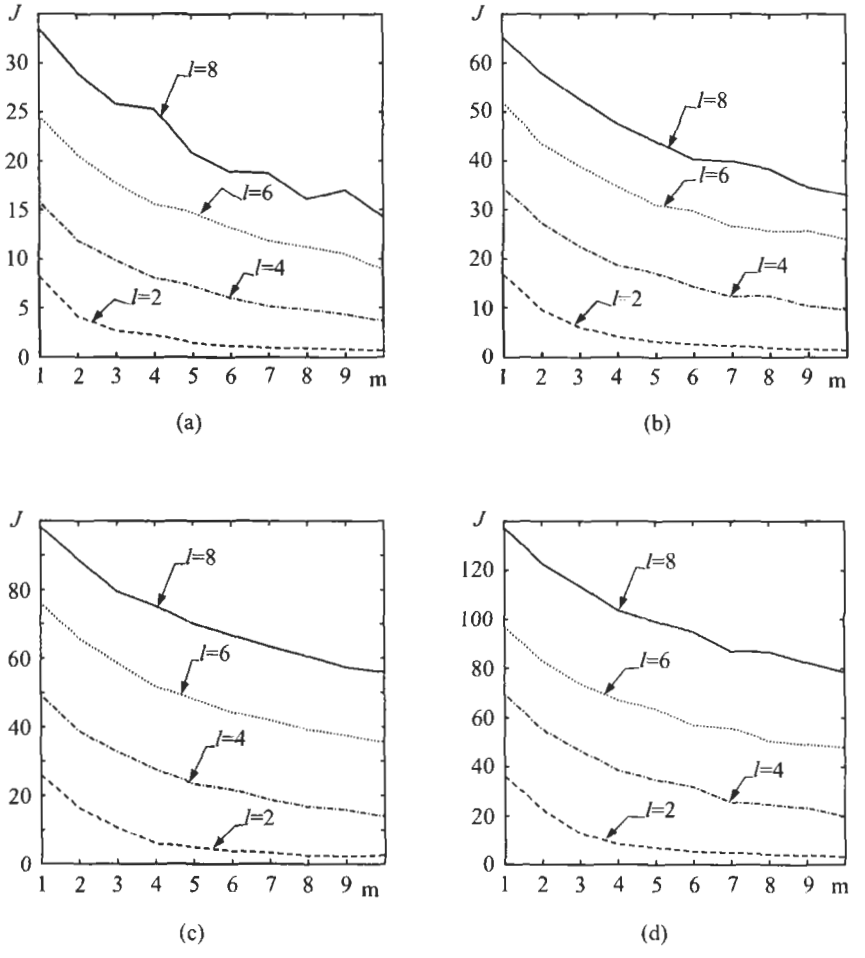


FIGURE 16.5: Plots of J versus m for (a) $N = 50$, (b) $N = 100$, (c) $N = 150$, (d) $N = 200$, for random data.

- *The Dunn and Dunn-like indices.* Let the dissimilarity function between two clusters C_i and C_j be (Chapter 11)

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \tag{16.17}$$

and define the *diameter of a cluster* C as

$$diam(C) = \max_{x, y \in C} d(x, y) \tag{16.18}$$

that is, the diameter of a cluster C is the distance of its two most distant vectors. $diam(C)$ may be viewed as a measure of dispersion of C. Then, the Dunn index for a specific m is defined as

$$D_m = \min_{i=1,\ldots,m} \left\{ \min_{j=i+1,\ldots,m} \left(\frac{d(C_i, C_j)}{\max_{k=1,\ldots,m} diam(C_k)} \right) \right\} \tag{16.19}$$

It is clear that if X contains compact and well-separated clusters, Dunn's index will be large, since the distance between the clusters is expected to be "large" and the diameter of the clusters is expected to be "small." Conversely, large values of Dunn's index indicate the presence of compact and well-separated clusters. The index D_m does not exhibit any trend with respect to m, hence the maximum in the plot of D_m versus m can be used to indicate the number of clusters that underlie X.

In [Dunn 74], it is shown that if $D_m > 1$ for a specific clustering, then this clustering contains compact and well-separated clusters.

A disadvantage of the Dunn index is the considerable amount of time required for its computation (see Problem 16.7). Moreover, Dunn's index is sensitive to the presence of noisy vectors in X, because these are likely to increase the value of the denominator of Eq. (16.19).

In [Pal 97] three Dunn-like indices are proposed that are more robust to the presence of noisy vectors. Furthermore, preliminary simulation results show that they may be used for cases in which shell-shaped clusters underlie X. These three indices are based on the concepts of the minimum spanning tree (MST), the relative neighborhood graph (RNG), and the Gabriel graph (GG), discussed in Chapter 15. Let us consider explicitly the index based on the MST concept. The other two are defined using similar arguments.

Consider a cluster C_i and the *complete* graph G_i having vertices that correspond to the vectors of C_i. The weight, w_e, of an edge, e, of this graph equals the distance between its two end points, $\boldsymbol{x}$ and $\boldsymbol{y}$, that is, $w_e = d(\boldsymbol{x}, \boldsymbol{y})$. Let E_i^{MST} be the set of edges of the MST of G_i and let e_i^{MST} be the edge in E_i^{MST} with the maximum weight. Then the diameter of C_i, $diam_i^{MST}$, is defined as the weight of e_i^{MST} (see Figure 16.6).

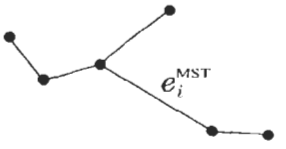


FIGURE 16.6: A minimum spanning tree.

The dissimilarity between two clusters is defined as the distance of their mean vectors, $d(C_i, C_j) = d(\boldsymbol{m}_i, \boldsymbol{m}_j)$. Then, the Dunn-like index, based on the concept of the MST, is defined as

$$D_m^{MST} = \min_{i=1,\ldots,m} \left\{ \min_{j=i+1,\ldots,m} \left(\frac{d(C_i, C_j)}{\max_{k=1,\ldots,m} diam_k^{MST}} \right) \right\} \tag{16.20}$$

The maximum in the plot of D_m^{MST} versus m indicates the underlying number of clusters in X. Similar arguments are followed to define Dunn-like indices for GG and RNG graphs (see Problem 16.8).

- *The Davies–Bouldin (DB) and DB-like indices.* Let s_i be a measure of dispersion of a cluster C_i (i.e., a measure of its spread around its mean vector) and $d(C_i, C_j) \equiv d_{ij}$ the dissimilarity between two clusters, using an appropriate dissimilarity measure. Based on these, a similarity index R_{ij} between C_i and C_j is defined to satisfy the following conditions [Davi 79]:

  (C1) $R_{ij} \geq 0$.
  (C2) $R_{ij} = R_{ji}$.
  (C3) If $s_i = 0$ and $s_j = 0$ then $R_{ij} = 0$.
  (C4) If $s_j > s_k$ and $d_{ij} = d_{ik}$ then $R_{ij} > R_{ik}$.
  (C5) If $s_j = s_k$ and $d_{ij} < d_{ik}$ then $R_{ij} > R_{ik}$.

  These conditions state that R_{ij} is nonnegative and symmetric. If both clusters, C_i and C_j, collapse to a single point, then $R_{ij} = 0$. A cluster C_i with the same distance from two other clusters, C_j, C_k, is more similar to the cluster with the largest dispersion (condition (C4)). For the case of equal dispersions and different dissimilarity levels, the cluster C_i is more similar to the closer of the two (condition (C5)).

  A (simple) choice for an R_{ij} that satisfies these conditions is the following [Davi 79]:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{16.21}$$

provided that d_{ij} is symmetric.

Also let R_i be defined as

$$R_i = \max_{j=1,\ldots,m, j \neq i} R_{ij}, \quad i = 1, \ldots, m \tag{16.22}$$

Then the DB index is defined as

$$DB_m = \frac{1}{m} \sum_{i=1}^{m} R_i \tag{16.23}$$

That is, DB_m is the average similarity between each cluster C_i, $i = 1, \ldots, m$, and its most similar one. As it is desirable for the clusters to have the minimum possible similarity to each other, we seek clusterings that minimize DB. On the other hand, small values of DB are indicative of the presence of compact and well-separated clusters. The DB_m index exhibits no trends with respect to m [Davi 79], thus we seek the minimum value of DB_m, in the plot of DB_m versus m.

In [Davi 79], the dissimilarity $d(C_i, C_j)$ between two clusters is defined as

$$d_{ij} = \|\boldsymbol{w}_i - \boldsymbol{w}_j\|_q = \left(\sum_{k=1}^{l} |w_{ik} - w_{jk}|^q \right)^{1/q} \tag{16.24}$$

Also, the dispersion of a cluster C_i is defined as

$$s_i = \left(\frac{1}{n_i} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{w}_i\|^r \right)^{1/r} \tag{16.25}$$

where n_i is the number of vectors in C_i. (Compare this definition with that of the diameter of a cluster defined earlier.)

In [Pal 97] three variants of the DB index, based again on the MST, RNG, and GG concepts, are proposed. We focus on the MST case. Let s_i^{MST} be the $diam_i^{MST}$, as defined in the Dunn-like index, and let d_{ij} be the distance between the mean vectors of C_i, C_j. Then, we define

$$R_{ij}^{MST} = \frac{s_i^{MST} + s_j^{MST}}{d_{ij}} \tag{16.26}$$

It is easy to show that R_{ij}^{MST} satisfies the conditions (C1)–(C5) (see Problem 16.10). Defining $R_i^{MST} = \max_{j=1,\ldots,m, j \neq i} R_{ij}^{MST}$, the MST DB index is defined as

$$DB_m^{MST} = \frac{1}{m} \sum_{i=1}^{m} R_i^{MST} \tag{16.27}$$

The minimum in the plot of DB_m^{MST} versus m is an indication of the number of clusters that underlie X.

Using arguments similar to these, we may define DB_m^{RNG} and DB_m^{GG}.

- *Information Theory based criteria.* A different philosophy that may be used for the estimation of the number of clusters m relies on the determination of a model that best fits the available data, without having any knowledge of their true distribution (see, for example, [Lu 00]).

Let us define the following criterion function:

$$C(\boldsymbol{\theta}, K) = -2\ln(L(\boldsymbol{\theta})) + \phi(K) \tag{16.28}$$

where $\boldsymbol{\theta}$ is the parameter vector of the model, $L(\boldsymbol{\theta})$, is the loglikelihood function (see eq. (2.50)), K is the order of the model, that is, the dimensionality of $\boldsymbol{\theta}$, and ϕ is an increasing function of K. Typical choices of ϕ are $\phi(K) = 2K$ (Akaike Information Criterion, AIC [Akai 85]), $\phi(K) = \frac{2KN}{N-K-1}$ (Consistent AIC [Hurv 89]), $\phi(K) = K \log N$ (Minimum Description Length (MDL) Criterion [Riss 78], [Riss 89] and Bayesian Information Criterion (BIC) [Schw 76], [Fral 98]). Note that K is a strictly increasing function of the number of clusters, m, since the higher the m, the larger the dimensionality of $\boldsymbol{\theta}$. For example, in the case where $p(\boldsymbol{x}; \boldsymbol{\theta})$ is a weighted summation of m l-dimensional Gaussian distributions, each one corresponding to a cluster, $\boldsymbol{\theta}$ consists of the ml parameters associated with the mean values of the distributions, plus $m\frac{l(l+1)}{2}$ parameters associated with the covariance matrices of the distributions plus the $m - 1$ weighting parameters. Thus, $K = (l + \frac{l(l+1)}{2} + 1)m - 1$. In words, K is an increasing linear function of m.

The aim is to minimize C with respect to $\boldsymbol{\theta}$ and K. We proceed as follows. First, the set of candidate models is fixed, involving models of similar structure but of different orders. Let $m \in [m_{\min}, m_{\max}]$ for the models of the above set. Then for each value of $m_i \in [m_{\min}, m_{\max}]$, we optimize $C(\boldsymbol{\theta}, m_i)$ with respect to $\boldsymbol{\theta}$, that is, we determine the maximum likelihood estimation $\boldsymbol{\theta}_i$. Then, among all pairs $(\boldsymbol{\theta}_i, m_i)$, we choose the one, say $(\boldsymbol{\theta}_j, m_j)$, that minimizes C. Thus, the estimated number of clusters is m_j. In the case where it is desirable to choose the best among models of different structure, we first identify all the subsets, each one containing similar models of differing order. Then, we determine the best model of each subset as described above. Finally, among these models, we select the one that leads to the minimum value of C.

Other indices suitable for hard clusterings have also been proposed. For example, in [Mill 80] and [Mill 85] many indices of this kind are tested on specific data sets. Also, in [Kirl 00] two new indices are presented and their relation to the method discussed in Section 12.3, for estimating the number of clusters, is investigated.

16.4.2 Fuzzy Clustering

In this section we consider indices suitable for fuzzy clustering. In this context, we seek clusterings that are not very fuzzy, that is, those whose clusters exhibit small overlap. In other words, we seek clusterings where most of the vectors of X exhibit high grade of membership in only one cluster. Recall that a fuzzy

clustering is defined by the $N \times m$ matrix $U = [u_{ij}]$, where u_{ij} denotes the grade of membership of the vector $\boldsymbol{x}_i$ in the j-th cluster. Also, let $W = \{\boldsymbol{w}_j, j = 1, \ldots, m\}$ be the set of the cluster representatives.

The strategy followed for the hard clustering case is also adopted here. That is, we define an appropriate index q (not to be confused with the fuzzifier) and we search for the minimum or the maximum in the plot of q versus m. In the case where q exhibits a trend with respect to m in the range $[m_{min}, m_{max}]$, we seek a significant knee of decrease or increase of q.

Indices for Clusters with Point Representatives

A. Indices that Involve Only U

One such index is the *partition coefficient* [Bezd 74], which is defined as

$$PC = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^2 \tag{16.29}$$

where u_{ij}'s are the values obtained after the convergence of the adopted fuzzy clustering algorithm.

The range of values for PC is $[1/m, 1]$. This index is computed for values of m greater than 1, since for $m = 1$, it is trivially equal to 1. The closer to unity the PC, the harder the clustering is or, alternatively, the smaller the "sharing" of the vectors in X among different clusters. The lowest value of PC is obtained when all u_{ij}'s are equal, that is, $u_{ij} = 1/m$, $j = 1, \ldots, m$, $i = 1, \ldots, N$. Thus, the closer the value of PC to $1/m$, the fuzzier the clustering. A value close to $1/m$ indicates that either X possesses no clustering structure or the adopted clustering algorithm failed to unravel it [Pal 95].

Another index of this category is the *partition entropy coefficient* [Bezd 75], which is defined as

$$PE = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m} (u_{ij} \log_a u_{ij}) \tag{16.30}$$

where a is the base of the logarithm. This index is also computed for values of m greater than 1. Its minimum value equals 0 and its maximum $\log_a m$. The closer the value of PE to 0, the harder the clustering is. On the other hand, the closer the value of PE to $\log_a m$, the fuzzier the clustering is. As in the previous case, values close to $\log_a m$ indicate the absence of any clustering structure in X or the inability of the clustering algorithm to reveal it [Pal 95].

Both of these indices measure the amount of "overlap" among clusters, without utilizing any additional information concerning the positions of the data vectors and the cluster representatives in space.

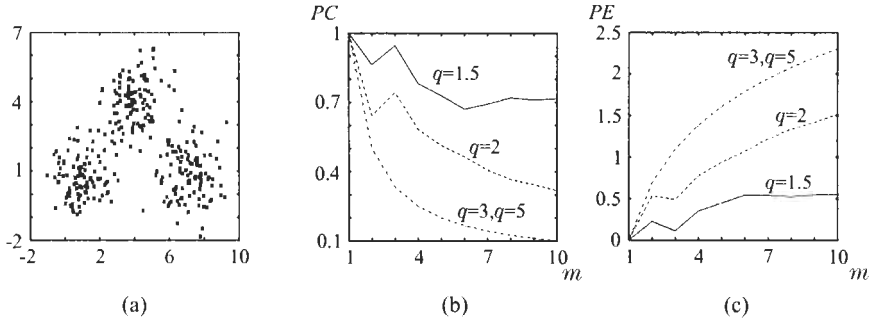


FIGURE 16.7: (a) The data set. (b) The plot of PC versus m. (c) The plot of PE versus m.

A disadvantage of both PC and PE indices is that they exhibit a dependence on m with a trend to increase or decrease, respectively, as m increases. Thus, one seeks significant knees of increase (for PC) or decrease (for PE) in the plot of the indices versus m. Moreover, they are also sensitive to the fuzzifier q. It can be shown (Problem 16.13) that as $q \to 1^+$,[6] both PC and PE give the same values for all m's; that is, they are unable to discriminate between different values of m. On the other hand, as $q \to \infty$, both PC and PE exhibit the most significant knee at $m = 2$ (see Problem 16.13). The behavior of PC and PE is illustrated via the following example.

Example 16.8. Let X be a data set that consists of three groups of two-dimensional vectors, each containing 100 vectors. The groups stem from normal distributions with means $[1, 1]^T$, $[4, 4]^T$, $[7, 1]^T$, respectively (see Figure 16.7a). All covariance matrices are equal to the identity 2×2 matrix. We run the fuzzy c-means algorithm for $q = 1.5, 2, 3, 5$ and $m = 1, \ldots, 10$. Figure 16.7b shows the behavior of the PC index. One can observe that for $q = 1.5$ and $q = 2$, the corresponding plots exhibit a significant knee at $m = 3$, which is the correct number of (the natural) clusters. The plots for $q = 3$ and $q = 5$ coincide. This implies that no significant change in the behavior of the index is expected for $q \geq 3$. Moreover, no peak is encountered; that is, no conjecture can be made for the number of clusters. Also, notice the general decreasing trend as m increases.

Figure 16.7c shows the behavior of the PE index. The plots corresponding to $q = 1.5$ and $q = 2$ exhibit a significant knee at $m = 3$. Also, as in the previous case, no significant change in the behavior of the index is expected for $q \geq 3$, and no minimum is encountered for $q \geq 3$. Finally, PE exhibits an increasing trend as m increases.

Other indices of this kind have also been proposed in the literature (e.g., [Wind 81]). We consider next indices that involve X, U, and W.

[6]This notation means that q tends to 1 from the right.

B. Indices Involving W, U, and the Data Set X

Let us define the so-called *cluster variation* as $\sigma_j^q = \sum_{i=1}^{N} u_{ij}^q \|\boldsymbol{x}_i - \boldsymbol{w}_j\|^2$ (compare this with the dispersion used in the DB index) and the *total variation* as $\sigma_q = \sum_{j=1}^{m} \sigma_j^q$. The parameter σ_q may be viewed as a measure of compactness of the specific clustering. Also let $d_{\min} = \min_{i,j=1,\ldots,m, i\neq j} \|\boldsymbol{w}_i - \boldsymbol{w}_j\|^2$ be a measure of separability of the clusters in X, where $\boldsymbol{w}_j$ denotes the representative of the j-th cluster $j=1, \ldots, m$. Then the *Xie–Beni index*, which is also called the *compactness and separation validity function*, is defined as

$$XB = \frac{\sigma_2/N}{d_{\min}} \tag{16.31}$$

This index is usually employed for the validation of clusterings produced by the fuzzy c-means algorithm when the Euclidean distance is in use. Note that despite the fact that the fuzzifier q in the fuzzy c-means may have any value greater than 1, *in the XB index the value of q involved in* σ_q *is restricted to* 2.

It is clear that for compact and well-separated clusters, small values of XB are expected. On the other hand, small values of XB indicate compact and well-separated clusters. As stated in [Xie 91], the XB index decreases monotonically as m gets very close to N. One way to handle this problem is to determine the starting point, $m_{\max}$, of the monotonicity behavior and to search for the minimum value of XB in the range $[2, m_{\max}]$.

Let

$$J_q = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q \|\boldsymbol{x}_i - \boldsymbol{w}_j\|^2 \tag{16.32}$$

(recall that this is the cost function minimized by the fuzzy c-means clustering algorithm when the squared Euclidean distance is in use). Then XB may be written in terms of J_2 as

$$XB = \frac{J_2}{N d_{\min}} \tag{16.33}$$

Thus, minimization of XB implies minimization of J_2.

Removal of the constraint $q=2$, used in the definition of XB, allows the definition of the generalized XB index as

$$XB_q = \frac{\sigma_q}{N d_{\min}} \tag{16.34}$$

It can be shown (Problem 16.14) that, as $q \to \infty$, XB tends to ∞ and XB_q becomes indeterminate.

Another index that combines X, W, and U is the *Fukuyama–Sugeno index* [Pal 95], which is defined as

$$FS_q = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q \left(\|\boldsymbol{x}_i - \boldsymbol{w}_j\|_A^2 - \|\boldsymbol{w}_j - \boldsymbol{w}\|_A^2 \right) \tag{16.35}$$

where $\boldsymbol{w}$ is the mean vector of X and A is an $l \times l$ positive definite, symmetric matrix. Recall that $\|\cdot\|_A$ is the A norm defined in Section 2.4. When $A = I$, the above distance becomes the squared Euclidean distance.

The first of the two terms in the parenthesis measures the compactness of the clusters, and the second one measures the distance of the cluster representatives, $\boldsymbol{w}_i$, from the overall mean vector $\boldsymbol{w}$. It is clear that for compact and well-separated clusters we expect small values for FS_q. Furthermore, small values of FS_q are indicative of compact and well-separated clusters. As far as the limiting behavior of FS_q is concerned, it can be shown (Problem 16.15) that (a) as $q \to 1^+$, FS_q behaves like $tr(S_w)$, the trace of the within scatter matrix (see Chapter 5), and (b) as $q \to \infty$, FS_q tends to 0.

In [Gath 89], three additional indices are proposed that are based on the concepts of hypervolume and density. Let us define the *fuzzy covariance matrix* of the j-th cluster as

$$\Sigma_j = \frac{\sum_{i=1}^{N} u_{ij}^q (\boldsymbol{x}_i - \boldsymbol{w}_j)(\boldsymbol{x}_i - \boldsymbol{w}_j)^T}{\sum_{i=1}^{N} u_{ij}^q} \tag{16.36}$$

The *fuzzy hypervolume* of the j-th cluster is defined as

$$V_j = |\Sigma_j|^{1/2} \tag{16.37}$$

where $|\Sigma_j|$ is the determinant of Σ_j. Note that this is a measure of compactness of the j-th cluster. The smaller the value of V_j, the more "compact" the j-th cluster is.

The *total fuzzy hypervolume* is defined as

$$FH = \sum_{j=1}^{m} V_j \tag{16.38}$$

Small values of FH indicate the existence of compact clusters.

Let $X_j = \{\boldsymbol{x} \in X : (\boldsymbol{x} - \boldsymbol{w}_j)^T \Sigma_j^{-1} (\boldsymbol{x} - \boldsymbol{w}_j) < 1\}$; that is, X_j contains all the vectors in X that are within a prespecified (small) region around $\boldsymbol{w}_j$. Also let $S_j = \sum_{\boldsymbol{x}_i \in X_j} u_{ij}$ be the so-called sum of central members of the j-th cluster.

The quantity S_j / V_j is called the *fuzzy density* of the j-th cluster. Then the *average partition density* is defined as

$$PA = \frac{1}{m} \sum_{j=1}^{m} \frac{S_j}{V_j} \tag{16.39}$$

A different measure is the *partition density* index and it is defined as

$$PD = \frac{S}{FH} \tag{16.40}$$

where $S = \sum_{j=1}^{m} S_j$.

"Compact" clusters lead to large values of PA and PD, and vice versa, large values of PA and PD are indications of "compact" clusters.

Note that all these indices, except PE and PC, may be used in the framework of hard clustering as well, by defining

$$u_{ij} = \begin{cases} 1, & \text{if } d(\boldsymbol{x}_i, C_j) = \min_{k=1,\ldots,m} d(\boldsymbol{x}_i, C_k) \\ 0, & \text{otherwise} \end{cases} \qquad i = 1, \ldots, N \tag{16.41}$$

Indices for Shell-Shaped Clusters

Let us now focus on the case of shell-shaped clusters (see Chapter 14). The PE and PC indices, discussed previously, may also be used in this case, since they involve no information concerning the geometrical characteristics of X.[7] However, the rest of the previously discussed indices need to be modified accordingly. Here, the representatives of each cluster, are shell shaped and they are denoted by β_j. The parameter vector $\boldsymbol{\theta}_j$ contains all the necessary parameters for the identification of β_j. For a vector $\boldsymbol{x}_i$, we define its distance from β_j in terms of

$$\tau_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j^i \tag{16.42}$$

where $\boldsymbol{x}_j^i$ is the point on β_j that is closer to $\boldsymbol{x}_i$. It is not difficult to show (Problem 16.16) that for clusters of spherical shape, where $\boldsymbol{\theta}_j = (\boldsymbol{c}_j, r_j)$, with $\boldsymbol{c}_j$ being the center and r_j being the radius of the corresponding sphere,

$$\tau_{ij} = (\boldsymbol{x}_i - \boldsymbol{c}_j) - r_j \frac{\boldsymbol{x}_i - \boldsymbol{c}_j}{\|\boldsymbol{x}_i - \boldsymbol{c}_j\|} \tag{16.43}$$

However, for general types of shells, computation of the τ_{ij}'s is not always an easy task. In such cases we resort to approximations of $\boldsymbol{x}_j^i$ ([Kris 95a]).

[7] Such characteristics may concern the shape of the clusters, the position of the representatives, etc.

The *fuzzy shell covariance matrix* for the j-th cluster is defined in accordance with Eq. (16.36) as

$$\Sigma_j^S = \frac{\sum_{i=1}^N u_{ij}^q \tau_{ij} \tau_{ij}^T}{\sum_{i=1}^N u_{ij}^q} \tag{16.44}$$

Then the *shell hypervolume* of a cluster is defined as

$$V_j^S = |\Sigma_j^s|^{1/2} \tag{16.45}$$

Let us define $X_j^S = \{\boldsymbol{x}_i : \tau_{ij}^T (\Sigma_j^S)^{-1} \tau_{ij} < 1\}$ and $S_j^S = \sum_{\boldsymbol{x}_i \in X_j} u_{ij}$. Then, the *fuzzy shell density*, the *average partition shell density*, and the *shell partition density* are defined as before.

Finally, another measure suitable for shell-shaped clusters is the *total fuzzy average shell thickness*, T^S [Kris 95b], which is defined as

$$T^S = \sum_{j=1}^m T_j^S \tag{16.46}$$

where T_j^S is the so-called *fuzzy average shell thickness* of the j-th cluster, defined as

$$T_j^S = \frac{\sum_{i=1}^N u_{ij}^q \|\tau_{ij}\|^2}{\sum_{i=1}^N u_{ij}^q} \tag{16.47}$$

It is clear that the "thicker" the clusters, the smaller the value of T^S. Furthermore, small values of T^S indicate "thick" clusters. However, T^S tends to decrease monotonically as the number of clusters increases.

The comments made for the total fuzzy hypervolume, the average partition density and the partition density indices are also valid here.

Note that *PA^S, PD^S, and T^S can be thought as measures of the density of the clusters formed by the vectors of X around their representatives.*

A few other indices of this kind have also been proposed and discussed in [Dave 90], [Kris 93]. A general comment applied to all these indices is that they are sensitive to the size and the density of the points in the clusters.

Finally, using Eq. (16.41), we obtain the shell density, the average partition shell density, and the shell partition density for the hard clustering case.

Remarks

- An alternative way of determining the number of clusters underlying in the data set X, is to perform the so called *progressive clustering method*

(e.g., [Kris95b]). According to this method we run first the clustering algorithm at hand for an overspecified number of clusters, m. Then, we remove spurious clusters, we merge compatible clusters and we identify the "good" clusters. Let k be the number of spurious and "good" clusters defined above. Then, we temporarily remove the vectors contained in the above clusters from the data set and we apply the algorithm on the reduced data set for $m-k$ clusters. This procedure is repeated until no "good" clusters can be removed anymore or no vectors are left in the data set. The output of the above method is the set of the "good" clusters determined above.

The advantage of this method is that, in general, it is not necessary to run the clustering algorithm for all values of m in a prespecified range. Also, this method is less influenced by the presence of noise. However, one must establish criteria concerning the merging and the removing operations involved in the above method as well as criteria for the identification of "good" clusters.

- A different philosophy for the determination of the number of clusters underlying in the data set X employs the idea of information criteria (IC), such as Akaike and the Minimum Description Length (MDL) criteria (see, e.g., [Sclo 87], [Lang 98]).

16.5 VALIDITY OF INDIVIDUAL CLUSTERS

There are two cases in which individual cluster validity may be of interest. One is when we want to test whether a given subset of X forms a "good" cluster. "Good" in this case is interpreted in terms of compactness, with respect to its own data, and isolation with respect to the other vectors of X. The other case concerns the validation of a cluster resulting from the application of a clustering algorithm. To this end, both external and internal criteria may be used.

16.5.1 External Criteria

In this section we consider hard clusters and ordinal-type proximity matrices [Bail 82], [Jain 88]. The goal is to test whether a given subset of X forms a compact and well-separated cluster. In [Bail 82], two indices are defined, one for compactness and one for isolation. Both are based on graph theory concepts. However, some necessary definitions must first be provided.

Let us consider the proximity graph $G(p)$, with p ($<N(N-1)/2$) edges, whose vertices correspond to the N vectors of X and whose edges correspond to the p smallest entries of the upper diagonal of the proximity matrix of X, P. In other words, a pair of vertices $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is connected with an edge if the dissimilarity $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is among the p smallest dissimilarity values of all possible

pairs of vectors in X (see Chapter 13). Also, let C be a predetermined subset of X, with k vectors. Our goal is to determine whether C is a good cluster. For the $G(p)$ and the given P, we define the sets $A_{in}(p)$, $A_{out}(p)$, and $A_{bet}(p)$ as follows: (a) $A_{in}(p)$ is the set of edges whose end points are vectors in C, (b) $A_{out}(p)$ is the set of edges whose end points are vectors in $X - C$, and (c) $A_{bet}(p)$ is the set of edges that connect vectors in C with vectors in $X - C$.

For a given $G(p)$, let $q_C(p)$ be the number of edges connecting vertices in C with vertices in $X - C$ and $r_C(p)$ be the set of edges connecting vertices in C. Clearly, these indices depend on p. It is easy to see that low values of $q_C(p)$ indicate a well-isolated cluster, and large values of $r_C(p)$ indicate a compact cluster. In order to extract conclusions about the compactness and isolation of C, we consider the behavior of these indices with respect to p. To this end, we plot the indices versus p. It is expected that an isolated and compact cluster will exhibit low values for $q_C(p)$ and high values for $r_C(p)$, for a "wide" range of values of p. The size of this range is application dependent.

A drawback of these indices is that they do not provide information with respect to a random population. To overcome this, an extension of the indices within the probabilistic framework is discussed in [Bail 82].

16.5.2 Internal Criteria

The aim here is to validate a single cluster that results from a clustering algorithm using only the information residing in the proximity matrix, P, of X.

- *Hard clustering case*
  - —*Ordinal proximity matrices.* A method for the evaluation of a cluster is given in [Ling 72] and [Ling 73]. This method is well suited for hierarchies of clusterings, produced by a hierarchical clustering algorithm. It relies on the lifetime, $L(C)$, of a cluster C, which is given by $L(C) = d_a(C) - d_f(C)$ where $d_f(C)$ is the level of hierarchy where C is formed and $d_a(C)$ is the level where C is absorbed in a larger cluster. The statistical index used is the so-called Ling index, which is defined as the probability of the lifetime of a randomly selected cluster exceeding $L(C)$. Finally, other methods in this category are the so-called *best case method* [Bail 82] and the *CM (clustering method) reachable method* [Bake 76].
  - —*Ratio-scaled proximity matrices.* In this case, we may adopt the hard hypervolume and the hard density (Section 16.4.2) when we seek compact clusters and the hard shell hypervolume and hard shell density when shell-shaped clusters are considered. Here, an empirically established threshold, ε, is used and, according to whether the value of the index is greater or less than ε, C is characterized as a "good" cluster or not.

- *Fuzzy clustering case.* We first focus on shell-shaped clusters. In this context, "good" clusters are those that are *"compact" around their shell representatives*. In this framework, one can use the shell hypervolume, the shell density indices, and the fuzzy average shell thickness, defined in Section 16.4. Based on these indices, a cluster is characterized as a good one according to whether the value of the corresponding index is greater or less than a prespecified threshold ε.

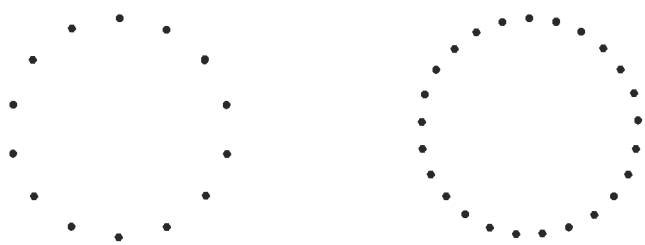
  All these indices do not take into account the fact that shell clusters lie in subspaces of the vector space [Kris 95b]. A criterion that takes this observation into account is the *surface density* criterion. We present the two-dimensional case. The criterion measures the number of points in a cluster per unit curve length. Let us define X' as the set of the vectors in C that lie at a distance smaller than or equal to τ_{max} from the shell representative β of C and let $S = \sum_{j:\boldsymbol{x}_j \in X'} u_j$. Then, the surface density δ of a cluster C is defined as

$$\delta = \frac{S}{2\pi r_{eff}} \tag{16.48}$$

  where r_{eff} is defined as

$$r_{eff} = \sqrt{\text{trace}\{\Sigma\}} \tag{16.49}$$

  where Σ is given in Eq. (16.44) and $tr(\Sigma)$ is the trace of Σ. The quantity $2\pi r_{eff}$ may be viewed as an estimate of the arc length of C (See problem 16.17). The higher the value of δ, the more dense the cluster is expected to be. Consider for example Figure 16.8. The clusters depicted there have a circular shape and their representative circles are of equal radius. Also, the one on the right is denser around its representative than the one on



(a) (b)

FIGURE 16.8: A sparse and a dense circular cluster.

the left. The value of δ for the right cluster is greater than that for the left cluster.

For compact clusters, indices such as the fuzzy hypervolume or the fuzzy density of a cluster can be employed.

16.6 CLUSTERING TENDENCY

As discussed in the introduction of the chapter, almost all the clustering algorithms introduced in the previous sections share an annoying feature. That is, they impose a clustering structure on a data set X even though the vectors of X do not exhibit such a structure. Thus, in order to prevent a misleading interpretation of the structure of the data set X, it would be more sensible to check first whether X possesses a clustering structure. If this is the case, then one may proceed by applying a clustering algorithm to X. Otherwise, cluster analysis is likely to lead to misleading results. The problem of determining the presence or the absence of a clustering structure in X is called *clustering tendency*. Usually, this task relies on statistical tests.

Clustering tendency methods have been applied in various application areas (e.g., [Digg 83], [Ripl 81]). However, most of these methods are suitable only for $l = 2$. In the sequel, we discuss the problem in the general $l \geq 2$ case. Furthermore, we focus on methods that are suitable for detecting compact clusters (if any).

In this framework, we test the randomness (null) hypothesis (H_0) against the clustering hypothesis and the regularity hypothesis. Let us define these terms more precisely.

- "The vectors of X are randomly distributed, according to the uniform distribution in the sampling window[8] of X" (H_0).
- "The vectors of X are regularly spaced in the sampling window." This implies that, they are not too close to each other.
- "The vectors of X form clusters."

If the randomness or the regularity hypothesis is accepted, methods alternative to clustering analysis should be used for the interpretation of the data set X.

There are two key points that have an important influence on the performance of many statistical tests used in clustering tendency. The first is the dimensionality of the data, l, which affects the performance in a non-obvious way. This dependence can be revealed through simulations [Pana 83].

The other key point is the sampling window. Apart from artificial experiments, in practice, we do not know the sampling window. One of the problems that this

[8] In [Smit 84] the sampling window is mathematically defined as the compact convex support set for the underlying distribution of the vectors of the data set X.

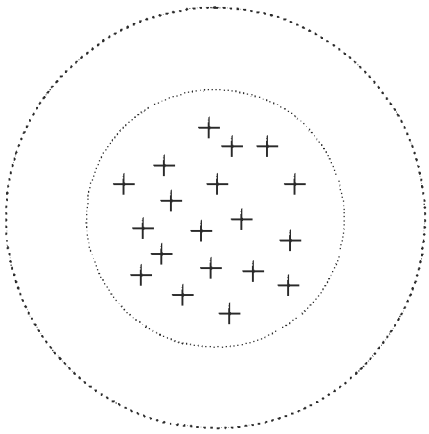


FIGURE 16.9: See text for explanation.

may cause is demonstrated in Figure 16.9. The vectors in the dashed circle are uniformly distributed in it. Thus, we expect that tests for randomness will identify this situation. However, if we use as sampling window the region surrounded by the dash-dotted line (for the same data set), the vectors are no longer uniformly distributed and the tests for randomness may fail to accept H_0. Moreover, due to the finite extent of the window, the statistical characteristics of the data are different near the edges of the sampling window than they are in its center. For example, the distribution of the distances of a vector $\boldsymbol{x} \in X$ from the rest of the vectors of X is different when $\boldsymbol{x}$ is in the center than when it is near the border of the sampling window. One way to overcome this situation is to use a periodic extension of the sampling window. Another popular technique is to consider data in a smaller area inside the sampling window, known as *sampling frame*. With this method, we overcome the boundary effects in the sampling frame by considering points outside it and inside the sampling window, for the estimation of statistical properties.

Example 16.9. Consider a data set X that consists of 100 vectors uniformly distributed in the H_2 hypercube (see Figure 16.10a). Figure 16.10b shows the distribution of the distances between the point $\boldsymbol{x} = [0.5045, 0.4764]^T$ and each of the points of $X - \{\boldsymbol{x}\}$. Also, Figure 16.10c shows the distribution of the distances between the point $\boldsymbol{y} = [0.0159, 0.8089]^T$ and each of the points of $X - \{\boldsymbol{y}\}$. Note that $\boldsymbol{x}$ lies close to the center of H_2 and $\boldsymbol{y}$ lies close to its border.

A method for estimating the sampling window is to use the convex hull of the vectors in X. However, the distributions for the tests, derived using this sampling

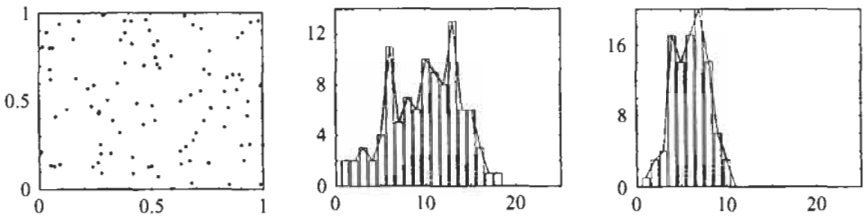


FIGURE 16.10: (a) The data set X. (b) The distribution of the distances of the point [0.5045, 0.4764] from the remaining points in X. (c) The distribution of the distances of the point $[0.0159, 0.8089]^T$ from the remaining points in X.

window, depend on the specific data at hand. A second drawback associated with this approach is the high computational cost for computing the convex hull of X. An alternative [Zeng 85], [Dube 87b] that seems to work well in practice is to define the sampling window as the hypersphere centered at the mean point of X and including half of its vectors. The fact that half of the vectors are discarded is not so crucial, because in the current framework we want to test only whether the vectors of X possess a clustering structure. If this is the case, then the clusters will be identified by applying a clustering algorithm to all the data of X. Notice the similarity to the sampling frame technique discussed earlier.

In the sequel, we define various test statistics, q, suitable for the detection of clustering tendency. Recall that a crucial quantity we have to determine is $p(q|H_0)$. Moreover, in order to measure the power of q against the regularity and the clustering tendency hypotheses, we also need to determine the respective pdf's under these hypotheses. In the sequel, we provide general guidelines on how to generate clustered and regularly spaced data sets. This is required in order to estimate the pdf's of q under regularity and clustering tendency hypotheses, via Monte Carlo simulations. Randomly spaced data sets may be generated by inserting vectors in the sampling window, according to the uniform distribution.

- *Generation of clustered data.* A well-known procedure for generating (compact) clustered data is the Neyman–Scott procedure [Neym 72]. This procedure assumes that the sampling window is known. It produces a random number of compact clusters, formed at random positions in the sampling window and each consisting of a random number of points. The number of points in each cluster follows the Poisson distribution (Appendix A). The technique requires as inputs the total number of points N of the set, the intensity of the Poisson process λ, and the spread parameter σ that controls the spread of each cluster around its center. According to this procedure, we randomly insert a point $\boldsymbol{y}_i$ in the sampling window, following the uniform distribution.

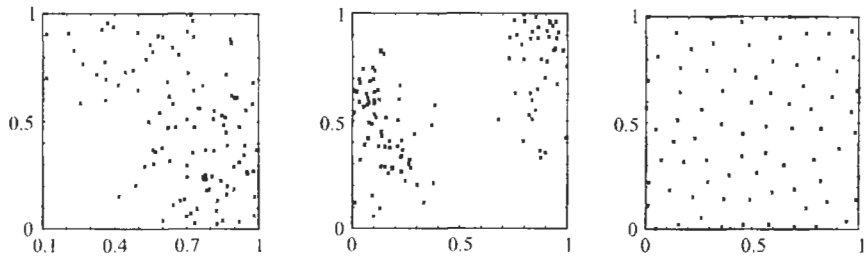


FIGURE 16.11: (a) and (b) Clustered data sets produced by the Neyman–Scott process. (c) Regularly spaced data produced by the SSI model.

This point serves as the center of the ith cluster, and we determine its number of vectors, n_i, using the Poisson distribution. Then the n_i points around $\boldsymbol{y}_i$ are generated according to the normal distribution with mean $\boldsymbol{y}_i$ and covariance matrix $\sigma^2 I$. If a point turns out to be outside the sampling window, we ignore it and another one is generated. This procedure is repeated until N points have been inserted in the sampling window (see Figure 16.11a and b). In some cases, $\boldsymbol{y}_i$'s are also included as vectors in the set.

- *Generation of regularly spaced data.* Perhaps the simplest way to produce regularly spaced points is to define a lattice in the convex hull of X and to place the vectors at its vertices. An alternative procedure, known as *simple sequential inhibition (SSI)* (see, e.g., [Jain 88], [Zeng 85]), is the following. The points $\boldsymbol{y}_i$ are inserted in the sampling window one at a time. For each point we define a hypersphere of radius r centered at $\boldsymbol{y}_i$. The next point can be placed anywhere in the sampling window in such a way that its hypersphere does not intersect with any of the hyperspheres defined by the previously inserted points. The procedure stops when a predetermined number of points have been inserted in the sampling window, or when no more points can be inserted in the sampling window, after say a few thousand trials (see Figure 16.11c). A variation of this model allows intersection of these hyperspheres up to a certain degree. A measure of the degree of fulfillment of the sampling window is the so-called *packing density*, which is defined as

$$\rho = \frac{L}{V} V_r \tag{16.50}$$

where L/V is the average number of points per unit volume and V_r is the volume of a hypersphere of radius r. V_r can be written as

$$V_r = Ar^l \tag{16.51}$$

where A is the volume of the l-dimensional hypersphere with unit radius, which is given by

$$A = \frac{\pi^{l/2}}{\Gamma(l/2+1)} \tag{16.52}$$

and $\Gamma(\cdot)$ is the gamma function (Appendix A).

16.6.1 Tests for Spatial Randomness

Several tests for spatial randomness have been proposed in the literature. All of them assume knowledge of the sampling window. The *scan test* ([Naus 82], [Cono 79]), the *quadrat analysis* [Grei 64], [Piel 69], [Mead 74], the *second moment structure* [Ripl 77], and the *interpoint distances* [Ripl 78], [Silv 78], [Stra 75] provide us with tests for clustering tendency that have been extensively used when $l = 2$. In the sequel, we discuss three methods for determining clustering tendency that are well suited for the general $l \geq 2$ case. All these methods require knowledge of the sampling window.

Tests Based on Structural Graphs

In this section, we discuss a test for testing randomness, that is based on the idea of the minimum spanning tree (MST) ([Smit 84]). First, we determine the convex region where the vectors of X lie. Then, we generate M vectors that are uniformly distributed over a region that approximates the convex region found before (usually M = N). These vectors constitute the set X'. Next we find the MST of $X \cup X'$ and we determine the number of edges, q, that connect vectors of X with vectors of X'. This number is used as the statistic index. If X contains clusters, then we expect q to be small. Conversely, small values of q indicate the presence of clusters. On the other hand, large values of q indicate a regular arrangement of the vectors of X.

Let e be the number of pairs of the MST edges that share a node. In [Frie 79], the following expressions for the mean value of q and the variance of q under the null (randomness) hypothesis, conditioned on e, are derived:

$$E(q|H_0) = \frac{2MN}{M+N} \tag{16.53}$$

and

$$var(q|e, H_0) = \frac{2MN}{L(L-1)}\left[\frac{2MN-L}{L} + \frac{e-L+2}{(L-2)(L-3)}[L(L-1)-4MN+2]\right] \tag{16.54}$$

where $L = M + N$. Moreover, it can be shown [Frie 79] that if $M, N \to \infty$ and M/N is away from 0 and ∞, the pdf of the statistic

$$q' = \frac{q - E(q|H_0)}{\sqrt{var(q|e, H_0)}} \tag{16.55}$$

is approximately given by the standard normal distribution. Thus, we reject H_0 at significance level ρ if q' is less than the ρ-percentile of the standard normal distribution. This test exhibits high power against clustering tendency and little power against regularity [Jain 88].

Tests Based on Nearest Neighbor Distances

Two tests of this kind are the Hopkins test [Hopk 54] and the Cox–Lewis test [Cox 76], [Pana 83]. The tests rely on the distances between the vectors of X and a number of vectors which are randomly placed in the sampling window.

The Hopkins test

Let $X' = \{y_i, i = 1, \ldots, M\}$, $M \ll N$,[9] be a set of vectors that are randomly distributed in the sampling window, following the uniform distribution. Also let $X_1 \subset X$ be a set of M randomly chosen vectors of X. Let d_j be the distance from $y_j \in X'$ to its closest vector in X_1, denoted by x_j, and δ_j be the distance from x_j to its closest vector in $X_1 - \{x_j\}$. Then the Hopkins statistic involves the lth powers of d_j and δ_j and it is defined as [Jain 88]

$$h = \frac{\sum_{j=1}^{M} d_j^l}{\sum_{j=1}^{M} d_j^l + \sum_{j=1}^{M} \delta_j^l} \tag{16.56}$$

This statistic compares the nearest neighbor distribution of the points in X_1 with that from the points in X'. When X contains clusters, the distances between nearest neighbor points in X_1 are expected to be small, on the average, and, thus, large values of h are expected. Furthermore, large values of h indicate the presence of a clustering structure in X. When the points in X are regularly distributed in the sampling window, it is expected that, on the average, the term $\sum_{j=1}^{M} d_j^l$ is smaller than $\sum_{j=1}^{M} \delta_j^l$, thus leading to small values of h. Also, small values of h indicate the presence of regularly spaced points. Finally, a value around $1/2$ is an indication that the vectors of X are randomly distributed over the sampling window. It can be shown (e.g., [Jain 88]) that if the generated vectors are distributed according to a Poisson random process (hypothesis of randomness) and all nearest neighbor

[9] Typically $M = 0.1N$.

distances are statistically independent, h (under H_0) follows a beta distribution, with (M, M) parameters (Appendix A).

Simulation results [Zeng 85] show that this test exhibits high power against regularity for a hypercubic sampling window and periodic boundaries, for $l = 2, \ldots, 5$. However, its power is limited against clustering tendency.

The Cox–Lewis test

This test is less intuitive than the previous one. It was first proposed in [Cox 76] for the two-dimensional case and it has been extended to the general $l \geq 2$ dimensional case in [Pana 83]. It follows the setup of the previous test with the exception that X_1 need not be defined. For each $\boldsymbol{y}_j \in X'$, we determine its closest vector in X, say $\boldsymbol{x}_j$, and then we determine the vector closest to $\boldsymbol{x}_j$ in $X - \{\boldsymbol{x}_j\}$, say $\boldsymbol{x}_i$. Let d_j be the distance between $\boldsymbol{y}_j$ and $\boldsymbol{x}_j$ and δ_j the distance between $\boldsymbol{x}_j$ and $\boldsymbol{x}_i$. We consider all $\boldsymbol{y}_j$'s for which $2d_j/\delta_j$ is greater than or equal to one. Let M' be the number of such $\boldsymbol{y}_j$'s. Then, an appropriate function R_j of $2d_j/\delta_j$ (see [Pana 83]) is defined for these $\boldsymbol{y}_j$'s. Finally, we define the statistic

$$R = \frac{1}{M'} \sum_{j=1}^{M'} R_j \tag{16.57}$$

It can be shown [Pana 83] that R, under H_0, has an approximately normal distribution with mean $1/2$ and variance $12M'$. Small values of R indicate the presence of a clustering structure in X, and large values indicate a regular structure in X. Finally, values around the mean of R indicate that the vectors of X are randomly arranged in the sampling window. Simulation results [Zeng 85] show that the Cox–Lewis test exhibits inferior performance compared with the Hopkins test against the clustering alternative. However, this is not the case against the regularity hypothesis.

Two additional tests are the so called T-squared sampling tests, introduced in [Besa 73]. However, simulation results [Zeng 85] show that the these two tests exhibit rather poor performance compared with the Hopkins and Cox–Lewis tests.

A Sparse Decomposition Technique

This technique begins with the data set X and sequentially removes vectors from it until no vectors are left [Hoff 87]. Before we proceed further, some definitions are needed. A *sequential decomposition* D of X is a partition of X into $L_1, \ldots, L_k$ sets, such that the order of their formation matters. L_i's are also called *decomposition layers*.

We denote by $MST(X)$ the MST corresponding to X. Let $S(X)$ be the set derived from X according to the following procedure. Initially, $S(X) = \emptyset$. We move an

end point x of the longest edge, e, of the $MST(X)$ to $S(X)$. Also, we mark this point and all points that lie at a distance less than or equal to b from x, where b is the length of e. Then, we determine the unmarked point, $y \in X$, that lies closer to $S(X)$ and we move it to $S(X)$. Also, we mark all the unmarked vectors that lie at a distance no greater than b from y. We apply the same procedure for all the unmarked vectors of X. The procedure terminates when all vectors are marked.

Let us define $R(X) \equiv X - S(X)$. Setting $X = R^0(X)$, we define

$$L_i = S(R^{i-1}(X)), \qquad i = 1, \ldots, k \tag{16.58}$$

where k is the smallest integer such that $R^k(X) = \emptyset$. The index i denotes the so-called *decomposition layer*. Intuitively speaking, the procedure sequentially "peels" X until all of its vectors have been removed.

The information that becomes available to us after the application of the decomposition procedure is (a) the number of decomposition layers k, (b) the decomposition layers L_i, (c) the cardinality, l_i, of the L_i decomposition layer, $i = 1, \ldots, k$, and (d) the sequence of the longest MST edges used in deriving the decomposition layers. The decomposition procedure gives different results when the vectors of X are clustered and when they are regularly spaced or randomly distributed in the sampling window. Based on this observation we may define statistical indices utilizing the information associated with this decomposition procedure. For example, it is expected that the number of decomposition layers, k, is smaller for random data than it is for clustered data. Also, it is smaller for regularly spaced data than for random data (see Problem 16.20). This situation is illustrated in the following example.

Example 16.10. (a) We consider a data set X_1 of 60 two-dimensional points in the unit square. The first 15 points stem from a normal distribution, with mean $[0.2, 0.2]^T$ and covariance matrix $0.15I$. The second, the third, and the fourth group of 15 points also stem from normal distributions with means $[0.2, 0.8]^T$, $[0.8, 0.2]^T$, and $[0.8, 0.8]^T$, respectively. Their covariance matrices are also equal to $0.15I$. Applying the sparse decomposition technique on X_1, we obtain 15 decomposition layers.

(b) We consider another data set X_2 of 60 two-dimensional points, which are now randomly distributed in the unit square. The sparse decomposition technique in this case gives 10 decomposition layers.

(c) Finally, we generate a data set X_3 of 60 two-dimensional points regularly distributed in the unit square, using the simple sequential inhibition (SSI) procedure. The sparse decomposition technique gives seven decomposition layers in this case.

Figures 16.12, 16.13, and 16.14 show the first four decomposition layers for clustered, random, and regularly spaced data. It is clear that the rate of point removal is much slower for the clustered data and much faster for the regular data.

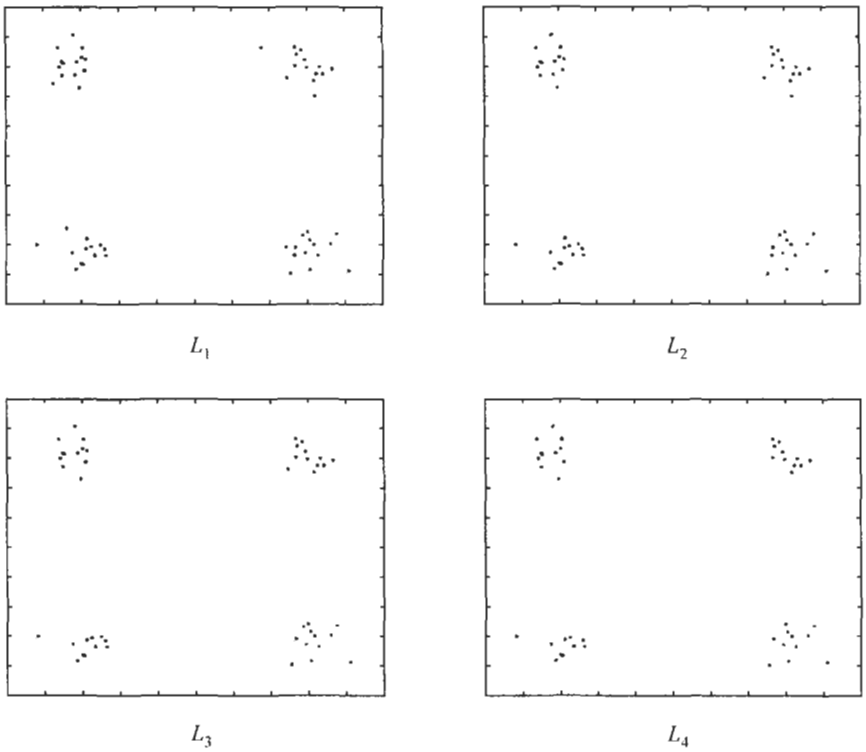


FIGURE 16.12: The first four decomposition layers for clustered data in the unit square (Example 16.10(a)).

Several tests that rely on the preceding information are discussed in [Hoff 87]. One such statistic that exhibits good performance is the so-called *P statistic*, which is defined as follows:

$$P = \prod_{i=1}^{k} \frac{l_i}{n_i - l_i} \tag{16.59}$$

where n_i is the number of points in $R^{i-1}(X)$. In words, each factor of P is the ratio of the removed to the remaining points at each decomposition stage.

Preliminary simulation results show high power of P against the clustering alternative. The required pdf's of P under H_0, H_1, and H_2 are estimated using Monte Carlo techniques, since it is difficult to derive theoretical results [Hoff 87].

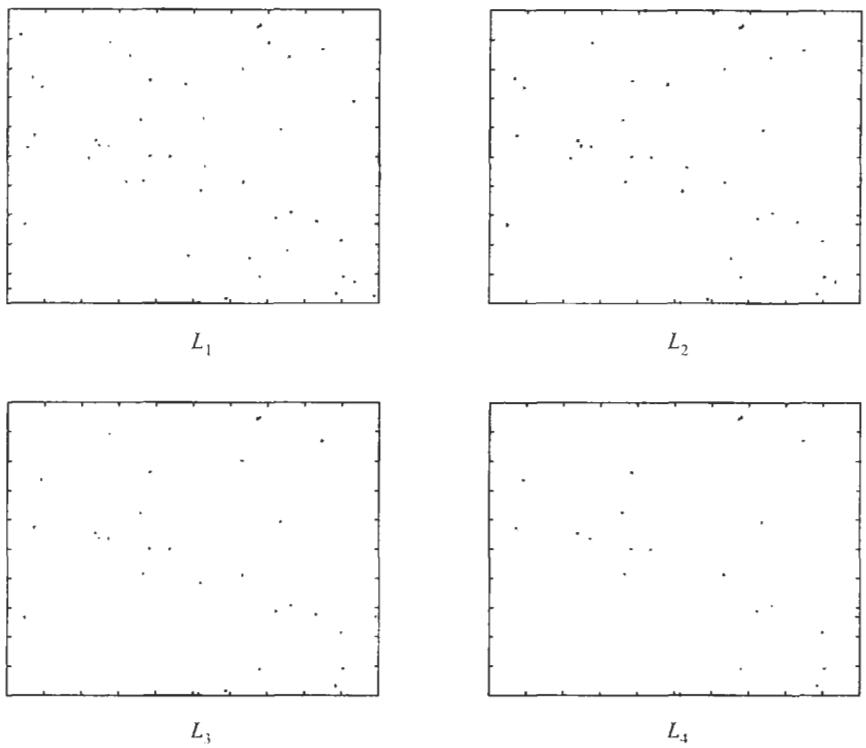


FIGURE 16.13: The first four decomposition layers for randomly distributed data in the unit square (Example 16.10(b)).

Finally, tests for clustering tendency for the cases in which ordinal proximity matrices are in use have also been proposed (e.g. [Fill 71],[Dube 79]). Most of them are based on graph theory concepts. Let $G_N(v)$ be a threshold graph with N vertices, one for each vector of X (Chapter 13). Then, graph properties, such as the node degree and the number of edges needed for $G_N(v)$ to be connected, are used in order to investigate the clustering tendency of X. Specifically, suppose that we use the number of edges n needed to make $G_N(v)$ connected. Obviously, n depends directly on v. That is, increasing v, we also increase n. Let v^* be the smallest value of v for which$G_N(v^*)$ becomes connected, for the given proximity matrix. Let V be the random variable that models v. Also, let $P(V \leq v|N)$ be the probability that a graph with N nodes and v randomly inserted edges is connected (this is provided from tables in [Ling 76]). Then, for the specific v^*, we determine $P(V \leq v^*|N)$. Very high values of $P(V \leq v^*|N)$ indicate that the proximity matrix was not chosen at random. This is because the within-cluster edges will tend to

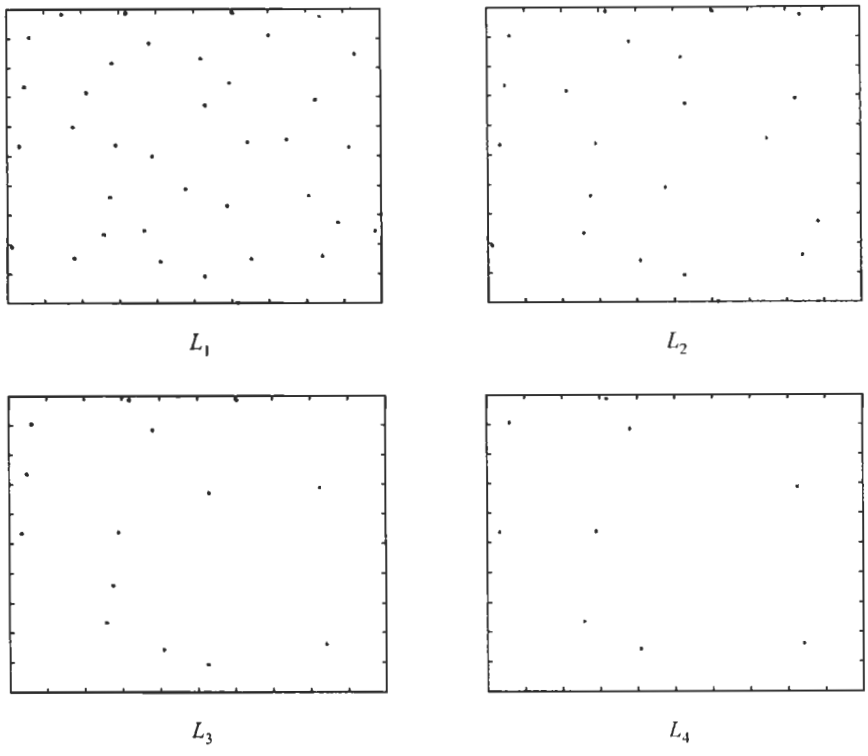


FIGURE 16.14: The first four decomposition layers for regularly spaced data in the unit square (Example 16.10(c)).

occur before the between-cluster edges when the data are clustered, thus, delaying the formation of a connected graph.

Problems

16.1 Let X be a set of vectors. Show that if the number of clusters in a clustering $\mathcal{C}$ of X is m and the number of groups in a partition $\mathcal{P}$ of X is $q \neq m$, then the maximum values of the Rand, the Jaccard, and the Fowlkes and Mallows statistics are less than 1.

16.2 Prove Eq. (16.10).

16.3 (a) Repeat Example 16.2 with two-dimensional vectors steming from the normal distributions with means $[0.2, 0.2]^T$, $[0.2, 0.8]^T$, $[0.8, 0.2]^T$, $[0.8, 0.8]^T$ and covariance matrices $0.2^2 I$.
(b) Repeat the experiment when all covariance matrices are equal to $0.5^2 I$.

16.4 Prove that the values of the *CPCC* in Section 16.3.2 lie in the interval $[-1, 1]$.

16.5 Consider a data set X of six vectors, whose (ordinal) proximity matrix is

$$P = \begin{bmatrix} 0 & 1 & 5 & 7 & 8 & 9 \\ 1 & 0 & 3 & 6 & 10 & 11 \\ 5 & 3 & 0 & 12 & 13 & 14 \\ 7 & 6 & 12 & 0 & 6 & 4 \\ 8 & 10 & 13 & 6 & 0 & 2 \\ 9 & 11 & 14 & 4 & 2 & 0 \end{bmatrix}$$

Run the single and the complete link algorithms on X and compare the resulting dendrograms, using the γ statistic. Comment on the results.

16.6 Let $X = \{\boldsymbol{x}_i, i = 1, \ldots, 12\}$, with $\boldsymbol{x}_1 = [-4, 0]^T$, $\boldsymbol{x}_2 = [-3, 1]^T$, $\boldsymbol{x}_3 = [-3, -1]^T$, $\boldsymbol{x}_4 = [-2, 0]^T$, $\boldsymbol{x}_5 = [2, 0]^T$, $\boldsymbol{x}_6 = [3, 1]^T$, $\boldsymbol{x}_7 = [4, 0]^T$, $\boldsymbol{x}_8 = [3, -1]^T$, $\boldsymbol{x}_9 = [-1, 7]^T$, $\boldsymbol{x}_{10} = [0, 8]^T$, $\boldsymbol{x}_{11} = [1, 7]^T$, $\boldsymbol{x}_{12} = [0, 6]^T$.

(a) Let $m = 2$. Consider the vectors $\boldsymbol{w}_1 = [0, 0]^T$ and $\boldsymbol{w}_2 = [0, 7]^T$, such that the first one represents the points $\boldsymbol{x}_1$ through $\boldsymbol{x}_8$ and the second one represents the rest of the points in X. Compute the values of Γ and $\hat{\Gamma}$ (Section 16.4.1).

(b) Let $m = 3$. Consider the vectors $\boldsymbol{w}_1 = [-3, 0]^T$ $\boldsymbol{w}_2 = [3, 0]^T$, and $\boldsymbol{w}_3 = [0, 7]^T$, so that the first one represents the points $\boldsymbol{x}_1$ through $\boldsymbol{x}_4$, the second represents the points $\boldsymbol{x}_5$ through $\boldsymbol{x}_8$, and the third represents the rest of the points of X. Compute the values of Γ and $\hat{\Gamma}$.

(c) Let $m = 4$. Define $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ as in the previous case. Also, define $\boldsymbol{w}_3 = [-0.5, 7.5]^T$ and $\boldsymbol{w}_4 = [0.5, 6.5]^T$, so that the first represents $\boldsymbol{x}_9$ and $\boldsymbol{x}_{10}$, while the second represents $\boldsymbol{x}_{11}$ and $\boldsymbol{x}_{12}$. Compute the values of Γ and $\hat{\Gamma}$.

(d) What conclusions can you draw from the comparison of the values of Γ and $\hat{\Gamma}$ obtained from the preceding three cases?

16.7 Estimate the number of operations required for the computation of Dunn's index, D_m, given by Eq. (16.19). What is the total number of computations required for the computation of D_m, for $m = 1, \ldots, N$?

16.8 Define explicitly $diam_i^{GG}$ and $diam_i^{RNG}$ that are involved in the definitions of the GG and the RNG Dunn-like indices. Using these definitions derive explicitly the GG and the RNG Dunn-like indices.

16.9 Show that $D_m^{GG} \leq D_m^{RNG} \leq D_m^{MST}$.

Hint: Use the fact that for a cluster C_i, $E_i^{MST} \subseteq E_i^{RNG} \subseteq E_i^{GG}$.

16.10 (a) Show that the R_{ij}^{MST} given by Eq. (16.26) satisfies the conditions (C1)–(C5).

(b) Taking into account the definition of R_{ij}^{MST}, define R_{ij}^{GG} and R_{ij}^{RNG} and show that they also satisfy the conditions (C1)–(C5).

16.11 Show that $DB_m^{GG} \geq DB_m^{RNG} \geq DB_m^{MST}$.

Hint: Use the fact that for a cluster C_i, $E_i^{MST} \subseteq E_i^{RNG} \subseteq E_i^{GG}$.

16.12 Explain intuitively why the MST DB is more robust to the presence of noisy vectors than the original DB.

16.13 (a) Prove that, as $q \to 1^+$, PC and PE tend to 1 and 0, respectively.

(b) Prove that, as $q \to \infty$, PC and PE tend to $1/m$ and $\log_a m$, respectively.

(c) Show that in the latter case, in the plots of PC and PE, the most significant knee is exhibited at $m = 2$.

16.14 Prove that, as $q \to \infty$, the XB index tends to ∞, while XB_q becomes indeterminate.
Hint: Use the following facts: (a) $\lim_{q\to\infty} \boldsymbol{w}_i = \boldsymbol{w}$, where $\boldsymbol{w}$ is the mean vector of all vectors in X, $i = 1, \ldots, m$, and (b) for $q \to \infty$, $u_{ij} = 1/m$.

16.15 (a) Prove that $\lim_{q\to 1+} FS_q = 2N\text{trace}(S_w) - N\text{trace}(S_m)$, where S_w and S_m are the within and the mixture scatter matrices defined in Chapter 5.
(b) Prove that $\lim_{q\to\infty} FS_q = 0$.
Hint: Use the hints given in the previous problem.

16.16 Prove that the distance of a point $\boldsymbol{x}_i$ from a sphere with center $\boldsymbol{c}_j$ and radius r_j is given by Eq. (16.43).

16.17 Consider a cluster C that consists of the points of a circular arc of radius r, subtending an angle ϕ (of course, this case is of theoretical interest, since the number of vectors in C would be infinite). The covariance matrix of this arc is

$$\Sigma = \frac{1}{L_\phi} \int_{-\phi/2}^{\phi/2} \boldsymbol{x}\boldsymbol{x}^T dl - \boldsymbol{m}\boldsymbol{m}^T \tag{16.60}$$

where $\boldsymbol{x} = [r\cos\theta, r\sin\theta]^T$ is a point on the arc, $dl = rd\theta$, and L_ϕ is the arc length. Prove that

$$\delta = \frac{\phi}{2\pi\sqrt{1 - \frac{4sin^2(\phi/2)}{\phi^2}}} \tag{16.61}$$

What is the value of δ when $\phi = 2\pi$?
(b) Repeat for the case where the length of the cluster is a line segment of length L.

16.18 Consider a square of side a. Consider a grid of horizontal and vertical lines in the square so that the distance between two adjacent parallel lines is r. Place in it $(a/2r)^2$ vectors (of course, $a/2r$ is assumed to be an integer) such that each of them lies at an intersection point of a grid and the circles of radius r centered at these points do not intersect at more than one point. (a) Compute the packing density of the square. Repeat the above for the case where r is replaced by $r/2$.
(b) Assuming that no circle is allowed to have a part of it outside the square, is it possible to determine an arrangement of points in the square that results in a higher packing density?

16.19 Sometimes we say that the Hopkins test includes "first-order" information on the data set X and the Cox–Lewis test "second-order" information. Can you justify this proposition?

16.20 Repeat Example 16.10 and explain why (a) the number of decomposition layers is greater in clustered data than in random data and (b) the number of decomposition layers is greater in random data than in regular data.

References

[Akai 85] Akaike H., "Prediction and Entropy," in *A celebration of statistics* (A. C. Atkinson, S. E. Fienberg), Sprieger-Verlag, New York, pp. 1–24, 1985.

[Back 81] Backer E., Jain A.K. "A clustering performance measure based on fuzzy set decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 3(1), pp. 66–75, 1981.

[Bail 82] Bailey T.A., Dubes R.C. "Cluster validity profiles," *Pattern Recognition*, Vol. 15, pp. 61–83, 1982.

[Bake 74] Baker F.B. "Stability of two hierarchical grouping techniques—Case 1. Sensitivity to data errors," *Journal of the American Statistical Association*, Vol. 69, 440–445, 1974.

[Bake 76] Baker F.B., Hubert L.J. "A graph-theoretic approach to goodness of fitting complete-link hierarchical clustering," *Journal of the American Statistical Association*, Vol. 71, pp. 870–878, 1976.

[Bart 62] Barton D.E., David F.N. "Randomization basis for multivariate tests in the bivariate case—randomness of points in the plane," *Bulletin of the International Statistical Institute*, Vol. 39, pp. 455–467, 1962.

[Beni 94] Beni G., Liu X. "A least biased fuzzy clustering method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16(9), pp. 954–960, 1994.

[Besa 73] Besag J.E., Gleaves J.T. "On the detection of spatial pattern in plant communities," *Bulletin of International Statistics Institute*, Vol. 45, p. 153, 1973.

[Bezd 74] Bezdek J.C. "Cluster validity with fuzzy sets," *Journal of Cybernetics*, Vol. 3(3), pp. 58–72, 1974.

[Bezd 75] Bezdek J.C. "Mathematical models for systematics and taxonomy," in *Proc. 8th Int. Conf. in Numerical Taxonomy* (Estarook G., ed.), Freeman, San Francisco pp. 143–166, 1975.

[Cono 79] Conover W.J., Benent T.R., Iman R.L. "On a method for detecting clusters of possible uranium deposits," *Technometrics*, Vol. 21, pp. 277–282, 1979.

[Cox 76] Cox T.F., Lewis T. "A conditioned distance ratio method for analyzing spatial patterns," *Biometrika*, Vol. 63, p. 483, 1976.

[Cunn 72] Cunningham K.M., Ogilvie J.C. "Evaluation of hierarchical grouping techniques: A preliminary study," *Computer Journal*, Vol. 15, pp. 209–213, 1972.

[Dave 90] Dave R.N., Patel K.J. "Progressive fuzzy clustering algorithms for characteristic shape recognition," *Proc. North American Fuzzy Inf. Process. Soc. Workshop*, Toronto, pp. 121–124, 1990.

[Davi 79] Davies D.L., Bouldin D.W. "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 1(2), pp. 224–227 1979.

[Diac 83] Diaconis P., Efron B. "Computer-intensive methods in statistics," *Scientific American*, May, pp. 116–130, 1983.

[Digg 83] Diggle P.J. *Statistical Analysis of Spatial Point Patterns*, Academic Press, 1983.

[Dube 79] Dubes R.C., Jain A.K. "Validity studies in clustering methodologies," *Pattern Recognition*, Vol. 11, pp. 235–254, 1979.

[Dube 87a] Dubes R.C. "How many clusters are best? An experiment," *Pattern Recognition*, Vol. 20(6), pp. 645–663, 1987.

[Dube 87b] Dubes R.C., Zeng G. "A test for spatial homogeneity in cluster analysis," *Journal of Classification*, Vol. 4, pp. 33–56, 1987.

[Duda 73] Duda R.O., Hart P.E. *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[Dunn 74] Dunn J.C. "Well separated clusters and optimal fuzzy partitions", *Journal of Cybernetics*, Vol. 4, pp. 95–104, 1974.

[Dunn 76] Dunn J.C. "Indices of partition fuzziness and the detection of clusters in large data sets," *in Fuzzy Automata and Decision Processes* (Gupta M.M., ed.), Elsevier, 1976.

[Efro 79] Efron B. "Bootstrap methods: Another look at jackknife," *Applied Statistics*, Vol. 7, pp. 1–26, 1979.

[Farr 69] Farris J.S. "On the cophenetic correlation coefficient," *Systematic Zoology*, Vol. 18, pp. 279–285, 1969.

[Fill 71] Fillenbaum S., Rapoport A. *Structures in the Subjective Lexicon*, Academic Press, 1971.

[Fowl 83] Fowlkes E.B., Mallows C.L. "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, Vol. 78, pp. 553–569, 1983.

[Fral 98] Fraley C., Raftery A. E., "How many clusters? Which clustering method? Answers via model-based cluster analysis," *The Computer Journal*, Vol. 41, No. 8, pp. 578–588, 1998.

[Frie 79] Friedman J.H., Rafsky L.C. "Multivariate generalization of the Wald–Wolfowitz and Smirnov two-sample tests," *Annual Statistics*, Vol. 7, pp. 697–717, 1979.

[Gath 89] Gath I., Geva A.B. "Unsupervised optimal fuzzy clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11(7), pp. 773–781, 1989.

[Grei 64] Greig-Smith P. *Quantitative Plant Ecology*, 2nd ed., Butterworth, 1964.

[Hall 73] Hall D.J., Duda R.O., Huffman D.A., Wolf E.E. "Development of new pattern recognition methods," Aerospace Research Labs., AD-7726141, 1973.

[Hart 75] Hartigan J.A. *Clustering Algorithms*, John Wiley & Sons, 1975.

[Hoff 87] Hoffman R.L., Jain A.K. "Sparse decompositions for exploratory pattern analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 9(4), pp. 551–560, 1987.

[Hopk 54] Hopkins B. "A new method for determining the type of distribution of plant-individuals," *Annals of Botany*, Vol. 18, pp. 213–226, 1954.

[Hube 74] Hubert L.J. "Approximate evaluation techniques for the single-link and complete-link hierarchical clustering procedures," *Journal of the American Statistical Association*, Vol. 69, pp. 698–704, 1974.

[Hube 76] Hubert L.J., Schultz J. "Quadratic assignment as a general data-analysis strategy," *British Journal of Mathemetical and Statistical Psychology*, Vol. 29, pp. 190–241, 1976.

[Hube 85] Hubert L.J., Arabie P. "Comparing partitions," *Journal of Classification*, Vol. 2, pp. 193–218, 1985.

[Hurv 89] Hurvich C. M., Tsai C-L, "Regression and time series model selection in small samples," *Biometrika* Vol. 76, pp. 297–307, 1989.

[Ivch 1991] Ivchenko G., Medvedev Y., Chistyakov A. *Problems in Mathematical Statistics*, Mir Publishers Moscow, 1991.

[Jain 87a] Jain A.K., Dubes R., Chen C.C. "Bootstrapping techniques for error estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 9, pp. 628–633, 1987.

[Jain 87b] Jain A.K., Moreau J.V. "Bootstrap technique in cluster analysis," *Pattern Recognition*, Vol. 20(5), pp. 547–568, 1987.

[Jain 88] Jain A.K., Dubes R.C. *Algorithms for Clustering Data*, Prentice Hall, 1988.

[Kirl 00] Kirlin R. L., Dizaji R. M., "Cluster order using clustering performance index rate, CPIR," NORSIG 2000, Kolmarden, Sweden, June 2000.

[Kris 93] Krishnapuram R., Frigui H., Nasraoui O. "Quadratic shell clustering algorithms and the detection of second-degree curves," *Pattern Recognition Letters*, Vol. 14(7), pp. 545–552, July 1993.

[Kris 95a] Krishnapuram R., Frigui H., Nasraoui O. "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation—Part I," *IEEE Transactions on Fuzzy Systems*, Vol. 3(1), pp. 29–43, 1995.

[Kris 95b] Krishnapuram R., Frigui H., Nasraoui O. "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation—Part II," *IEEE Transactions on Fuzzy Systems*, Vol. 3(1), pp. 44–60, 1995.

[Lang 98] Langan D.A., Modestino J.W., Zhang J. "Cluster validation for unsupervised stochastic model-based image segmentation," *IEEE Transactions on Image Processing*, Vol. 7(2), pp. 180–195, 1998.

[Ling 72] Ling R.F. "On the theory and construction of k-clusters," *Computer Journal*, Vol. 15, pp. 326–332, 1972.

[Ling 73] Ling R.F. "Probability theory of cluster analysis," *Journal of the American Statistical Association*, Vol. 68, pp. 159–164, 1973.

[Ling 76] Ling R.F., Killough G.S. "Probability tables for cluster analysis based on a theory of random graphs," *Journal of the American Statistical Association*, Vol. 71, pp. 293–300, 1976.

[Lu 00] Lu X. "Comparisons among information-based criteria, a novel modification thereof, and the Monte Carlo Markov chain method," *MSc Thesis*, University of Victoria, British Columbia, Canada, July 2000.

[Mant 67] Mantel N. "The detection of disease clustering and a generalized regression approach," *Cancer Research*, Vol. 27, pp. 209–220, 1967.

[Mead 74] Mead R. "A test for spatial pattern at several scales using data from a grid of contiguous quadrats," *Biometrics*, Vol. 30, pp. 295–308, 1974.

[Mill 80] Milligan G.W. "An examination of the effect of six types of error perturbation on fifteen clustering algorithms," *Psychometrica*, Vol. 45, pp. 325–342, 1980.

[Mill 83] Milligan G.W., Soon S.C., Sokol L.M. "The effect of cluster size, dimensionality, and the number of clusters on recovery of true cluster structure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 5, pp. 40–47, 1983.

[Mill 85] Milligan G.W., Cooper M.C. "An examination of procedures for determining the number of clusters in a data set," *Psychometrika*, Vol. 50, pp. 159–179, 1985.

[Naus 82] Naus J.J. "Approximations for distributions of scan statistics," *Journal of the American Statistical Association*, Vol. 77, pp. 177–183, 1982.

[Neym 72] Neyman J., Scott E.L. "Processes of clustering and applications," in *Stochastic Point Processes: Statistical Analysis, Theory and Applications* (Lewis P.A.W., ed.), John Wiley & Sons, 1972.

[Pal 95] Pal N.R., Bezdek J.C. "On cluster validity for the fuzzy c-means model," *IEEE Transactions on Fuzzy Systems*, Vol. 3(3), pp. 370–379, 1995.

[Pal 97] Pal N.R., Biswas J. "Cluster validation using graph theoretic concepts," *Pattern Recognition*, Vol. 30(6), pp. 847–857, 1997.

[Pana 83] Panayirci E., Dubes R.C. "A test for multidimensional clustering tendency," *Pattern Recognition*, Vol. 16(4), pp. 433–444, 1983.

[Papo 91] Papoulis A. *Probability, Random Variables and Stochastic Processes*, 3rd ed., McGraw-Hill, 1991.

[Piel 69] Pielou E.C. *An Introduction to Mathematical Ecology*, John Wiley & Sons, 1969.

[Post 93] Postaire J.G., Zhang R.D., Lecocq-Botte C. "Cluster analysis by binary morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(2), pp. 170–180, 1993.

[Ripl 77] Ripley B.D. "Modelling spatial patterns," *Journal of the Royal Statistical Society*, Vol. B39, pp. 172–212, 1977.

[Ripl 78] Ripley B.D., Silverman B.W. "Quick tests for spatial interaction," *Biometrika*, Vol. 65, pp. 641–642, 1978.

[Ripl 81] Ripley B.D. *Spatial Statistics*, John Wiley & Sons, 1981.

[Riss 78] Rissanen J. "Modeling by shortest data description," *Automatica* 14, pp. 465–471, 1978.

[Riss 89] Rissanen J. "Stochastic complexity in statistical enquiry," *Series in computer science*, 15, World Scientific, Singapore, 1989.

[Rolp 68] Rolph F.J., Fisher D.R. "Tests for hierarchical structure in random data sets," *Systematic Zoology*, Vol. 17, pp. 407–412, 1968.

[Rolp 70] Rolph F.J. "Adaptive hierarchical clustering schemes," *Systematic Zoology*, Vol. 19, pp. 58–82, 1970.

[Schw 76] Schwarz G. "Estimating the dimension of a model," *Annals of Statistics* Vol. 6, pp. 461–464, 1976.

[Sclo 87] Sclove S.L. "Application of model-selection criteria to some problems in multivariate analysis," *Psychometrika*, Vol. 52, pp. 333–343, 1987.

[Shre 64] Shreider Y.A. *Method of Statistical Testing: Monte Carlo Method*, Elsevier North-Holland, 1964.

[Silv 78] Silverman B., Brown T. "Short distances, flat triangles and Poisson limits," *Journal of Applied Probability*, Vol. 15, pp. 815–825, 1978.

[Smit 84] Smith S.P., Jain A.K. "Testing for uniformity in multidimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 73–81, 1984.

[Snea 77] Sneath P.H.A. "A significance test for clusters in UPGMA phenograms obtained from squared Euclidean distance," *Classification Soc. Bulletin*, Vol. 4, pp. 2–14, 1977.

[Sobo 84] Sobol I.M. *The Monte Carlo Method*, Mir Publishers, Moscow, 1984.

[Stra 75] Strauss D.J. "Model for clustering," *Biometrika*, Vol. 62, pp. 467–475, 1975.

[Wind 81] Windham A.P. "Cluster validity for fuzzy clustering algorithms," *Fuzzy Sets and Systems*, Vol. 5, pp. 177–185, 1981.

[Wind 82] Windham M.P. "Cluster validity for the fuzzy c-means clustering algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 4(4), pp. 357–363, July 1982.

[Xie 91] Xie X.L., Beni G. "A validity measure for fuzzy clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13(8), pp. 841–846, 1991.

[Yarm 87] Yarman-Vural F., Ataman E. "Noise, histogram and cluster validity for Gaussian mixtured data," *Pattern Recognition*, Vol. 20(4), pp. 385–401, 1987.

[Zeng 85] Zeng G., Dubes R.C. "A comparison of tests for randomness," *Pattern Recognition*, Vol. 18(2), pp. 191–198, 1985.

APPENDIX A

HINTS FROM PROBABILITY AND STATISTICS

A.1 TOTAL PROBABILITY AND THE BAYES RULE

Let $\mathcal{A}_i, i = 1, 2, \ldots, M$, be M events so that $\sum_{i=1}^{M} P(\mathcal{A}_i) = 1$. Then the probability of an arbitrary event $\mathcal{B}$ is given by

$$P(\mathcal{B}) = \sum_{i=1}^{M} P(\mathcal{B}|\mathcal{A}_i) P(\mathcal{A}_i) \tag{A.1}$$

where $P(\mathcal{B}|\mathcal{A}_i)$ denotes the conditional probability of $\mathcal{B}$ assuming $\mathcal{A}_i$, which is defined as

$$P(\mathcal{B}|\mathcal{A}) = \frac{P(\mathcal{B}, \mathcal{A})}{P(\mathcal{A})} \tag{A.2}$$

and $P(\mathcal{B}, \mathcal{A})$ is the joint probability of the two events. Equation (A.1) is known as the *total probability theorem*. From the definition in (A.2) the Bayes rule is readily available

$$P(\mathcal{B}|\mathcal{A}) P(\mathcal{A}) = P(\mathcal{A}|\mathcal{B}) P(\mathcal{B}) \tag{A.3}$$

These are easily extended to random variables or vectors described by probability density functions and we have

$$p(\boldsymbol{x}|\mathcal{A}) P(\mathcal{A}) = P(\mathcal{A}|\boldsymbol{x}) p(\boldsymbol{x}) \tag{A.4}$$

and

$$p(\boldsymbol{x}|\boldsymbol{y}) p(\boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x}) \tag{A.5}$$

and finally

$$p(\boldsymbol{x}) = \sum_{i=1}^{M} p(\boldsymbol{x}|\mathcal{A}_i) P(\mathcal{A}_i) \tag{A.6}$$

A.2 MEAN AND VARIANCE

Let $p(x)$ be the probability density function (pdf) describing the random variable x. Its mean and variance are defined as

$$E[x] = \int_{-\infty}^{+\infty} xp(x)\,dx, \quad \sigma_x^2 = \int_{-\infty}^{+\infty} (x - E[x])^2 p(x)\,dx \tag{A.7}$$

A.3 STATISTICAL INDEPENDENCE

Two (or more) random variables x and y are statistically independent if and only if

$$p(x, y) = p_x(x)p_y(y) \tag{A.8}$$

It turns out that in this case $E[xy] = E[x]E[y]$. These are generalized to more than two variables.

A.4 CHARACTERISTIC FUNCTIONS

Let $p(x)$ be the probability density function of a random variable x. The associated *characteristic function* is by definition the integral

$$\Phi(\Omega) = \int_{-\infty}^{+\infty} p(x)\exp(j\Omega x)\,dx \equiv E[\exp(j\Omega x)] \tag{A.9}$$

If $j\Omega$ is changed into s, the resulting integral becomes

$$\Phi(s) = \int_{-\infty}^{+\infty} p(x)\exp(sx)\,dx \equiv E[\exp(sx)] \tag{A.10}$$

and it is known as the *moment generating function.*

The function

$$\Psi(\Omega) = \ln \Phi(\Omega) \tag{A.11}$$

is known as the *second characteristic function* of x.

The joint characteristic function of l random variables is defined by

$$\Phi(\Omega_1, \Omega_2, \ldots, \Omega_l) = \int_{-\infty}^{+\infty} p(x_1, x_2, \ldots, x_l)\exp\left(j\sum_{i=1}^{l}\Omega_i x_i\right)d\boldsymbol{x} \tag{A.12}$$

The logarithm of the above is the second joint characteristic function of the l random variables.

A.5 MOMENTS AND CUMULANTS

Taking the derivative of $\Phi(s)$ in Eq. (A.10) we obtain

$$\frac{d^n \Phi(s)}{ds^n} \equiv \Phi^{(n)}(s) = E[x^n \exp(sx)] \tag{A.13}$$

and hence for $s = 0$

$$\Phi^{(n)}(0) = E[x^n] \equiv m_n \tag{A.14}$$

where m_n is known as the nth-order moment of x. If the moments of all orders are finite, the Taylor series expansion of $\Phi(s)$ near the origin exists and is given by

$$\Phi(s) = \sum_{n=0}^{+\infty} \frac{m_n}{n!} s^n \tag{A.15}$$

Similarly, the Taylor expansion of the second generating function results in

$$\Psi(s) = \sum_{n=1}^{+\infty} \frac{\kappa_n}{n!} s^n \tag{A.16}$$

where

$$\kappa_n \equiv \frac{d^n \Psi(0)}{ds^n} \tag{A.17}$$

and are known as the *cumulants* of the random variable x. It is not difficult to show that $\kappa_0 = 0$. For a zero mean random variable, it turns out that

$$\kappa_1(x) = E[x] = 0 \tag{A.18}$$

$$\kappa_2(x) = E[x^2] = \sigma^2 \tag{A.19}$$

$$\kappa_3(x) = E[x^3] \tag{A.20}$$

$$\kappa_4(x) = E[x^4] - 3\sigma^4 \tag{A.21}$$

That is, the first three cumulants are equal to the corresponding moments. The fourth-order cumulant is also known as *kurtosis. For a Gaussian process all cumulants of order higher than two are zero.* The kurtosis is commonly used as a measure of the non-Gaussianity of a random variable. For random variables described by (unimodal) pdfs with spiky shape and heavy tails, known as leptokurtic or super-Gaussian, κ_4 is positive, whereas for random variables associated with pdfs with a flatter shape, known as platykurtic or sub-Gaussian, κ_4 is negative. Gaussian variables have zero kurtosis. The opposite is not always true, in the sense

that there exist non-Gaussian random variables with zero kurtosis; however, this can be considered rare.

Similar arguments hold for the expansion of the joint charcteristic functions for multivariate pdf's. For zero mean random variables, $x_i, i = 1, 2, \ldots, l$, the cumulants of order up to four are given by

$$\kappa_1(x_i) = E[x_i] = 0 \tag{A.22}$$

$$\kappa_2(x_i, x_j) = E[x_i x_j] \tag{A.23}$$

$$\kappa_3(x_i, x_j, x_k) = E[x_i x_j x_k] \tag{A.24}$$

$$\kappa_4(x_i, x_j, x_k, x_r) = E[x_i x_j x_k x_r] - E[x_i x_j]E[x_k x_r] \tag{A.25}$$

$$- E[x_i x_k]E[x_j x_r] - E[x_i x_r]E[x_j x_k] \tag{A.26}$$

Thus, once more, the cumulants of the first three orders are equal to the corresponding moments. If all variables coincide, we talk about *auto-cumulants*, and otherwise about *cross-cumulants*, i.e.,

$$\kappa_4(x_i, x_i, x_i, x_i) = \kappa_4(x_i)$$

that is, the auto-cumulant of x_i is identical to its kurtosis. It is not difficult to see that if the zero mean random variables are mutually independent, their cross-cumulants are zero. *This is also true for the cross-cumulants of all orders.*

A.6 EDGEWORTH EXPANSION OF A PDF

Taking into account the expansion in Eq. (A.16), the definition given in Eq. (A.11), and taking the inverse Fourier of $\Phi(\Omega)$ in Eq. (A.9) we can obtain the following expansion of $p(x)$ for a zero mean unit variance random variable x:

$$p(x) = g(x)\left(1 + \frac{1}{3!}\kappa_3(x)H_3(x) + \frac{1}{4!}\kappa_4(x)H_4(x) + \frac{10}{6!}\kappa_3^2(x)H_6(x)\right.$$
$$\left. + \frac{1}{5!}\kappa_5(x)H_5(x) + \frac{35}{7!}\kappa_3(x)\kappa_4(x)H_7(x) + \ldots\right) \tag{A.27}$$

where $g(x)$ is the unit variance and zero mean normal pdf, and $H_k(x)$ is the Hermite polynomial of degree k. The rather strange ordering of terms is the outcome of a specific reordering in the resulting expansion, so that the successive coefficients in the series decrease uniformly. This is very important when truncation of the series is required. The Hermite polynomials are defined as

$$H_k(x) = (-1)^k \exp(x^2/2)\frac{d^k}{dx^k}\exp(-x^2/2) \tag{A.28}$$

and they form a complete orthogonal basis set in the real axis, i.e.,

$$\int_{-\infty}^{+\infty} \exp(-x^2/2) H_n(x) H_m(x)\, dx = \begin{cases} n!\sqrt{2\pi} & \text{if } n = m \\ 0 & \text{if } n \neq m \end{cases} \tag{A.29}$$

The expansion of $p(x)$ in Eq. (A.27) is known as the *Edgeworth expansion*, and it is actually an expansion of a pdf around the normal pdf [Papo 91].

A.7 KULLBACK–LEIBLER DISTANCE

The Kullback–Leibler (KL) distance is a measure of the distance between two probability density functions $p(\boldsymbol{x})$ and $\hat{p}(\boldsymbol{x})$ and is defined as

$$L = -\int p(\boldsymbol{x}) \ln \frac{\hat{p}(\boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{x} \tag{A.30}$$

Sometimes it is referred to as *cross or relative entropy*. The KL distance can be shown to be always nonnegative but it is not a true distance measure, from a mathematical point of view, since it is not symmetric. Sometimes it is referred as the KL divergence.

The KL distance is closely related to the *mutual information* measure, I, between l scalar random variables, x_i, $i = 1, 2, \ldots, l$. Indeed, let us compute the KL distance between the joint pdf $p(\boldsymbol{x})$ and the pdf resulting from the product of the corresponding marginal probability densities, i.e.,

$$\begin{aligned} L &= -\int p(\boldsymbol{x}) \ln \frac{\prod_{i=1}^{l} p_i(x_i)}{p(\boldsymbol{x})} d\boldsymbol{x} \\ &= \int p(\boldsymbol{x}) \ln p(\boldsymbol{x}) d\boldsymbol{x} - \sum_{i=1}^{l} \int p(\boldsymbol{x}) \ln p_i(x_i) d\boldsymbol{x} \\ &= -H(\boldsymbol{x}) - \sum_{i=1}^{l} \int p(\boldsymbol{x}) \ln p_i(x_i) d\boldsymbol{x} \end{aligned} \tag{A.31}$$

Carrying out the integrations on the right-hand side it is straightforward to see the KL distance is equal to the mutual information, I, defined as

$$I(x_1, x_2, \ldots, x_l) = -H(\boldsymbol{x}) + \sum_{i=1}^{l} H(x_i) \tag{A.32}$$

where $H(x_i)$ is the associated entropy of x_i, defined as ([Papo 91])

$$H(x_i) = -\int p_i(x_i) \ln p_i(x_i)\, dx_i \tag{A.33}$$

It is now easy to see that if the variables $x_i, i = 1, 2, \ldots, l$, are statistically independent their mutual information I is zero. Indeed, in this case $\Pi_{i=1}^{l} p_i(x_i) = p(\boldsymbol{x})$, hence $L = I(x_1, x_2, \ldots, x_l) = 0$.

A.8 MULTIVARIATE GAUSSIAN OR NORMAL PROBABILITY DENSITY FUNCTION

This is defined as a generalization of the univariate normal pdf

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{l/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{A.34}$$

where $\boldsymbol{\mu}$ is the mean vector, that is, $E\left[[x_1, x_2, \ldots, x_l]^T\right] = [\mu_1, \mu_2, \ldots, \mu_l]^T$ and Σ the covariance matrix

$$\Sigma = E[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T] \tag{A.35}$$

and we say that $\boldsymbol{x}$ is normally distributed as $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$. For the one dimensional, $l = 1$, case the covariance matrix becomes the variance σ^2 and the Gaussian density function takes the form

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Figure A.1 shows the plots of two Gaussians for the same mean and different variances. For the general l dimensional case the covariance matrix has the form

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1l} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2l} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{l1} & \sigma_{l2} & \cdots & \sigma_l^2 \end{bmatrix} \tag{A.36}$$

where $\sigma_i^2 = E[(x_i - \mu_i)^2]$, $\sigma_{ij} = \sigma_{ji} = E[(x_i - \mu_i)(x_j - \mu_j)]$. Thus, the main matrix diagonal consists of the respective variances of the elements of the random vector and the off-diagonal elements are the respective covariances between the elements of the random vector. Note that if the random variables x_i are statistically independent, then the mean of the product equals the product of the means, that is, $E[(x_i-\mu_i)(x_j-\mu_j)] = E[(x_i-\mu_i)]E[(x_j-\mu_j)] = 0$, and the covariance matrix is diagonal. However, a diagonal covariance matrix does not, in general, mean that the variables are statistically independent. In the case, though, of multivariate

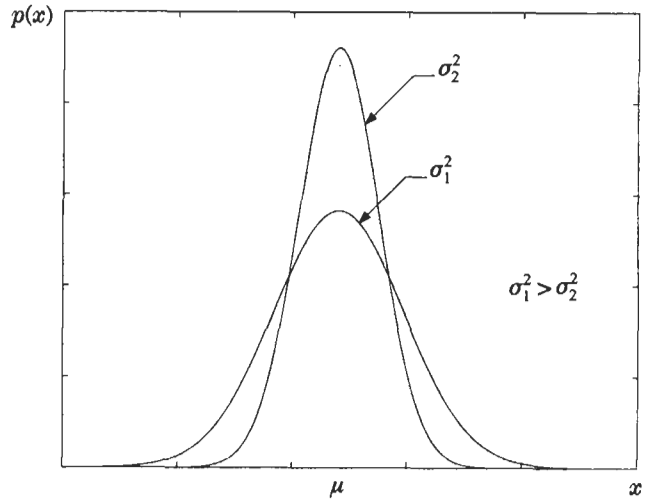


FIGURE A.1: Two Gaussians with the same mean μ and different variances.

Gaussian densities the opposite is also valid. Indeed, if the covariance matrix is diagonal then it is straightforward to see that

$$p(\boldsymbol{x}) = \prod_{i=1}^{l} p_i(x_i) \tag{A.37}$$

where

$$p_i(x_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right)$$

which is the univariate Gaussian describing the ith variable (why?). Thus, the joint probability density is the product of the individual (marginal) ones, which is the definition of statistical independence.

A.9 THE CRAMER–RAO LOWER BOUND

Let $p(\boldsymbol{x}; \boldsymbol{\theta})$ be the pdf of a random vector, parameterized in terms of an r-dimensional vector parameter $\boldsymbol{\theta}$. If X is the set of N observations $\boldsymbol{x}_i$, $i = 1, 2, \ldots, N$, the log likelihood function is the logarithm of the joint pdf of the observations $\ln p(X; \boldsymbol{\theta}) = L(\boldsymbol{\theta})$. The Fisher matrix is defined so that its (i, j)

element equals

$$J_{ij} = -E\left[\frac{\partial^2 L(\boldsymbol{\theta})}{\partial\theta_i \partial\theta_j}\right], \quad i, j = 1, 2, \ldots, r \tag{A.38}$$

It can be shown that the ith element of *any* unbiased estimate $\hat{\boldsymbol{\theta}}$, of the parameter $\boldsymbol{\theta}$, based on the observations set X satisfies

$$E[(\hat{\theta}_i - \theta_i)^2] \geq J_{ii}^{-1} \tag{A.39}$$

In other words, its variance is *lower bounded by the* (i, i) *element of the inverse Fisher matrix*. This is known as the Cramer–Rao bound. If the relation is valid with equality, the corresponding estimator is called *efficient*.

A.10 CENTRAL LIMIT THEOREM

Let $x_1, x_2, \ldots, x_N$ be N independent random variables, with mean and variances μ_i, σ_i^2, respectively. We form the new random variable

$$z = \sum_{i=1}^{N} x_i \tag{A.40}$$

Its mean and variance are given by $\mu = \sum_{i=1}^{N} \mu_i$ and $\sigma^2 = \sum_{i=1}^{N} \sigma_i^2$. The central limit theorem states that as $N \to \infty$ and under certain general conditions the pdf of the variable

$$q = \frac{z - \mu}{\sigma} \tag{A.41}$$

approaches $\mathcal{N}(0, 1)$, irrespective of the pdf's of the summands [Papo 91]. Thus in practice, for large enough N we can consider z as approximately Gaussian with mean μ and variance σ^2.

A.11 CHI-SQUARE DISTRIBUTION

Let x_i, $i = 1, 2, \ldots, N$ be samples of a Gaussian $\mathcal{N}(0, 1)$ random variable x. The sum of squares variable

$$\chi^2 = x_1^2 + x_2^2 + \cdots + x_N^2 \equiv y \tag{A.42}$$

is a chi-square distributed variable with N degrees of freedom. Its probability density function is given by [Papo 91]

$$p_y(y) = \frac{1}{2^{N/2}\Gamma(N/2)} y^{N/2-1} \exp(-y/2)u(y) \tag{A.43}$$

where

$$\Gamma(b+1) = \int_0^\infty y^b \exp(-y)\, dy \quad b > -1 \tag{A.44}$$

where $u(y)$ is the step function (1 for $y > 0$ and 0 for $y < 0$). Recalling the respective definitions, it is easy to show that $E[y] = N$, $\sigma_y^2 = 2N$.

The chi-square distribution possesses the *additive property*. Let χ_1^2 and χ_2^2 be independent random variables of chi-square distribution with N_1, N_2 degrees of freedom, respectively. Then the random variable

$$\chi^2 = \chi_1^2 + \chi_2^2 \tag{A.45}$$

is a chi-square variable with $N_1 + N_2$ degrees of freedom. Based on these properties we can show that the variance estimate of Equation (5.5) is described by a chi-square distribution with $N - 1$ degrees of freedom, provided x is Gaussian and the samples x_i are independent. The proof is simple and interesting [Fras 58]. Define the following transformation:

$$y_1 = \sqrt{N}\bar{x} = \frac{x_1 + \cdots + x_n}{\sqrt{N}}$$

$$y_2 = \frac{1}{\sqrt{2}}(x_2 - x_1)$$

$$\vdots$$

$$y_n = \frac{1}{\sqrt{n(n-1)}}[(n-1)x_n - (x_1 + \cdots + x_{n-1})], \quad n = 2, 3, \ldots, N$$

It is easy to show that this transformation is an orthogonal one (Problem 5.3). Thus the random variables y_i are also Gaussian, statistically independent, and with the same variance σ^2 as x (Problem 5.4). This transformation easily results in

$$\sum_{i=1}^{N} y_i^2 = \sum_{i=1}^{N} x_i^2 \tag{A.46}$$

and of course

$$y_1^2 = N\bar{x}^2 \tag{A.47}$$

Subtracting the two, we obtain

$$\sum_{i=2}^{N} y_i^2 = \sum_{i=1}^{N} (x_i - \bar{x})^2 \equiv (N-1)\hat{\sigma}^2 \tag{A.48}$$

Furthermore, $E[y_i] = 0, i = 2, \ldots, N$. Thus the variable

$$z = \frac{N-1}{\sigma^2}\hat{\sigma}^2 = \sum_{i=2}^{N} \frac{y_i^2}{\sigma^2} \tag{A.49}$$

is a chi-square with $N - 1$ degrees of freedom.

A.12 t-DISTRIBUTION

Let x and z be two independent random variables with x being $\mathcal{N}(0, 1)$ and z a chi-square with N degrees of freedom. Then it can be shown [Papo 91] that the variable

$$q = \frac{x}{\sqrt{z/N}} \tag{A.50}$$

is a so-called t-distributed variable with probability density function given by

$$p_q(q) = \frac{\gamma_1}{\sqrt{(1 + q^2/N)^{N+1}}}, \quad \gamma_1 = \frac{\Gamma((N+1)/2)}{\sqrt{\pi N}\Gamma(N/2)}$$

where $\Gamma(\cdot)$ was defined in Eq. (A.44). Thus, from the test statistic in Equation (5.6) and (A.49) we have

$$q = \frac{\bar{x} - \mu}{\hat{\sigma}/\sqrt{N}} = \frac{\frac{\bar{x}-\mu}{\sigma/\sqrt{N}}}{\sqrt{z/N-1}} \tag{A.51}$$

Since z is a chi-square with $N - 1$ degrees of freedom, q is a t-distributed variable with $N - 1$ degrees of freedom. In a similar way we can show that the test statistic in Equation (5.10) is t-distributed with $2N - 2$ degrees of freedom.

A.13 BETA DISTRIBUTION

A random variable follows the Beta distribution with parameters a and $b (a, b > 0)$, if its probability density function is defined as

$$p(x) = \begin{cases} \frac{x^{a-1}(1-x)^{b-1}}{B(a,b)}, & 0 < x < 1 \\ 0, & \text{Otherwise} \end{cases} \tag{A.52}$$

where

$$B(a, b) = \int_0^1 u^{a-1}(1-u)^{b-1}\, du \tag{A.53}$$

Its mean and variance are equal to $a/(a+b)$ and $ab/((a+b)^2(a+b+1))$, respectively.

A.14 POISSON DISTRIBUTION

A Poisson distributed random variable X with parameter a takes the values $k = 0, 1, 2, \ldots$, with probabilities

$$P(X = k) = e^{-a}\frac{a^k}{k!} \tag{A.54}$$

A *Poisson process* scatters vectors in a Euclidean space in such a way that the random variable X, denoting the number of vectors in a region of volume V, has a Poisson distribution with parameter λV, i.e.,

$$P(X = k) = e^{-\lambda V}\frac{(\lambda V)^k}{k!}, \quad k = 0, 1, 2, \ldots \tag{A.55}$$

The parameter λ is called the *intensity* of the process and equals the expected number of vectors per unit volume.

References

[Fras 58] Fraser D.A.S. *Statistics: An Introduction*, John Wiley, 1958.

[Digg 83] Diggle P.J. *Statistical analysis of spatial point processes*, Academic Press, 1983.

[Papo 91] Papoulis A. *Probability random variables and stochastic processes*, 3rd Ed. McGraw Hill, 1991.

[Spie 75] Spiegel M.R. *Schaum's outline of theory and problems of probability and statistics*, McGraw Hill, 1975.

APPENDIX **B**

LINEAR ALGEBRA BASICS

B.1 POSITIVE DEFINITE AND SYMMETRIC MATRICES

- An $l \times l$ real matrix A is called *positive definite* if for *every* nonzero vector $\boldsymbol{x}$ the following is true:

$$\boldsymbol{x}^T A \boldsymbol{x} > 0 \tag{B.1}$$

  If equality with zero is allowed, A is called *nonnegative or positive semidefinite*.
- It is easy to show that all eigenvalues of such a matrix are positive. Indeed, let λ_i be one eigenvalue and $\boldsymbol{v}_i$ the corresponding unit norm eigenvector ($\boldsymbol{v}_i^T \boldsymbol{v}_i = 1$). Then by the respective definitions

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \quad \text{or} \tag{B.2}$$

$$0 < \boldsymbol{v}_i^T A \boldsymbol{v}_i = \lambda_i \tag{B.3}$$

  Since the determinant of a matrix is equal to the product of its eigenvalues, we conclude that the determinant of a positive definite matrix is also positive.
- Let A be an $l \times l$ symmetric matrix, $A^T = A$. Then the eigenvectors corresponding to distinct eigenvalues are orthogonal. Indeed, let $\lambda_i \neq \lambda_j$ be two such eigenvalues. From the definitions we have

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \tag{B.4}$$

$$A\boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j \tag{B.5}$$

  Multiplying (B.4) on the left by $\boldsymbol{v}_j^T$ and the transpose of (B.5) on the right by $\boldsymbol{v}_i$, we obtain

$$\boldsymbol{v}_j^T A \boldsymbol{v}_i - \boldsymbol{v}_j^T A \boldsymbol{v}_i = 0 = (\lambda_i - \lambda_j) \boldsymbol{v}_j^T \boldsymbol{v}_i \tag{B.6}$$

  Thus, $\boldsymbol{v}_j^T \boldsymbol{v}_i = 0$. Furthermore, it can be shown that even if the eigenvalues are not distinct, we can still find a set of orthogonal eigenvectors. The same is true for Hermitian matrices, in case we deal with more general complex-valued matrices.

- Based on this, it is now straightforward to show that a symmetric matrix A can be diagonalized by the similarity transformation

$$\Phi^T A \Phi = \Lambda \tag{B.7}$$

where matrix Φ has as its columns the unit eigenvectors ($\boldsymbol{v}_i^T \boldsymbol{v}_i = 1$) of A, that is,

$$\Phi = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_l] \tag{B.8}$$

and Λ is the diagonal matrix with elements the corresponding eigenvalues of A. From the orthonormality of the eigenvectors it is obvious that $\Phi^T \Phi = I$, that is, Φ is a unitary matrix, $\Phi^T = \Phi^{-1}$. The proof is similar for Hermitian complex matrices as well.

B.2 CORRELATION MATRIX DIAGONALIZATION

Let $\boldsymbol{x}$ be a random vector in the l-dimensional space. Its correlation matrix is defined as $R = E[\boldsymbol{x}\boldsymbol{x}^T]$. Matrix R is readily seen to be positive semidefinite. For our purposes we will assume that it is positive definite, thus invertible. Moreover, it is symmetric, and hence it can always be diagonalized

$$\Phi^T R \Phi = \Lambda \tag{B.9}$$

where Φ is the matrix consisting of the (orthogonal) eigenvectors and Λ the diagonal matrix with the corresponding eigenvalues on its diagonal. Thus, we can always transform $\boldsymbol{x}$ into another random vector whose elements are uncorrelated. Indeed

$$\boldsymbol{x}_1 \equiv \Phi^T \boldsymbol{x} \tag{B.10}$$

Then the new correlation matrix is $R_1 = \Phi^T R \Phi = \Lambda$. Furthermore, if $\Lambda^{1/2}$ is the diagonal matrix whose elements are the square roots of the eigenvalues of R ($\Lambda^{1/2}\Lambda^{1/2} = \Lambda$), then it is readily shown that the transformed random vector

$$\boldsymbol{x}_1 \equiv \Lambda^{-1/2} \Phi^T \boldsymbol{x} \tag{B.11}$$

has uncorrelated elements with unit variance. $\Lambda^{-1/2}$ denotes the inverse of $\Lambda^{1/2}$. It is now easy to see that if the correlation matrix of a random vector is the identity matrix I, then this is invariant under any unitary transformation $A^T \boldsymbol{x}$, $A^T A = I$. That is, the transformed variables are also uncorrelated with unit variance. A useful byproduct of this is the following lemma.

Lemma. Let $\boldsymbol{x}$, $\boldsymbol{y}$ be two random vectors with correlation matrices R_x, R_y, respectively. Then there is a linear transformation that diagonalizes both matrices simultaneously.

Proof. Let Φ be the eigenvector matrix diagonalizing R_x. Then the transformation

$$\boldsymbol{x}_1 \equiv \Lambda^{-1/2}\Phi^T\boldsymbol{x} \tag{B.12}$$

$$\boldsymbol{y}_1 \equiv \Lambda^{-1/2}\Phi^T\boldsymbol{y} \tag{B.13}$$

generates two new random vectors with correlation matrices $R_x^1 = I$, R_y^1, respectively. Now let Ψ be the eigenvector matrix diagonalizing R_y^1. Then the random vectors generated by the unitary transformation ($\Psi^T\Psi = I$)

$$\boldsymbol{x}_2 \equiv \Psi^T\boldsymbol{x}_1 \tag{B.14}$$

$$\boldsymbol{y}_2 \equiv \Psi^T\boldsymbol{y}_1 \tag{B.15}$$

have correlation matrices $R_x^2 = I$, $R_y^2 = D$, where D is the diagonal matrix with elements the eigenvalues of R_y^1. Thus, the linear transformation of the original vectors by the matrix

$$A^T = \Psi^T\Lambda^{-1/2}\Phi^T \tag{B.16}$$

diagonalizes both correlation matrices simultaneously (one to an identity matrix). All these are obviously valid for covariance matrices as well. ■

APPENDIX C

COST FUNCTION OPTIMIZATION

In this appendix we review a number of optimization schemes that have been encountered throughout the book.

Let $\boldsymbol{\theta}$ be an unknown parameter vector and $J(\boldsymbol{\theta})$ the corresponding cost function to be minimized. Function $J(\boldsymbol{\theta})$ is assumed to be differentiable.

C.1 GRADIENT DESCENT ALGORITHM

The algorithm starts with an initial estimate $\boldsymbol{\theta}(0)$ of the minimum point and the subsequent algorithmic iterations are of the form

$$\boldsymbol{\theta}(new) = \boldsymbol{\theta}(old) + \Delta\boldsymbol{\theta} \tag{C.1}$$

$$\Delta\boldsymbol{\theta} = -\mu \left. \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}(old)} \tag{C.2}$$

where $\mu > 0$. If a maximum is sought, the method is known as *gradient ascent* and the minus sign in (C.2) is neglected.

Figure C.1 shows the geometric interpretation of the scheme. The new estimate $\boldsymbol{\theta}(new)$ is chosen in the direction that decreases $J(\boldsymbol{\theta})$. The parameter μ is very important and it plays a crucial role in the convergence of the algorithm. If it is too small, the corrections $\Delta\boldsymbol{\theta}$ are small and the convergence to the optimum point is very slow. On the other hand, if it is too large, the algorithm may oscillate around the optimum value and convergence is not possible. However, if the parameter is properly chosen, the algorithm converges to a stationary point of $J(\boldsymbol{\theta})$, which can be either, a local minimum ($\boldsymbol{\theta}_1^0$) or a global minimum ($\boldsymbol{\theta}^0$) or a saddle point ($\boldsymbol{\theta}_2^0$). In other words, it converges to a point where the gradient becomes zero (see Figure C.2). To which of the stationary points the algorithm will converge depends on the position of the initial point, relative to the stationary points. Furthermore, the convergence speed depends on the form of the cost $J(\boldsymbol{\theta})$. Figure C.3 shows the constant $J(\boldsymbol{\theta}) = c$ curves, for two cases and for different values of c, in the

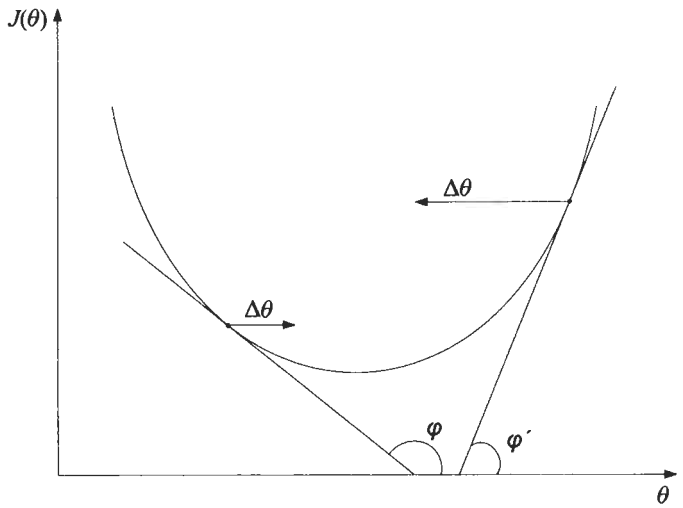


FIGURE C.1: In the gradient descent scheme, the correction of the parameters takes place in the direction that decreases the value of the cost function.

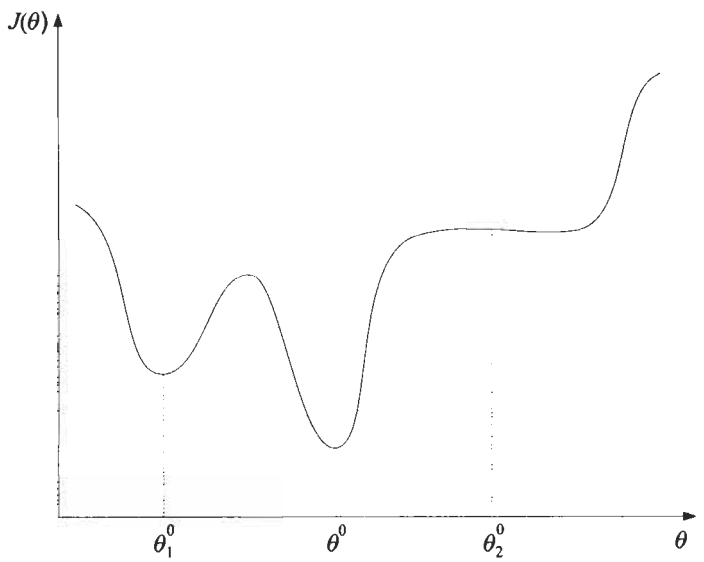


FIGURE C.2: A local minimum, a global minimum and a saddle point of $J(\theta)$.

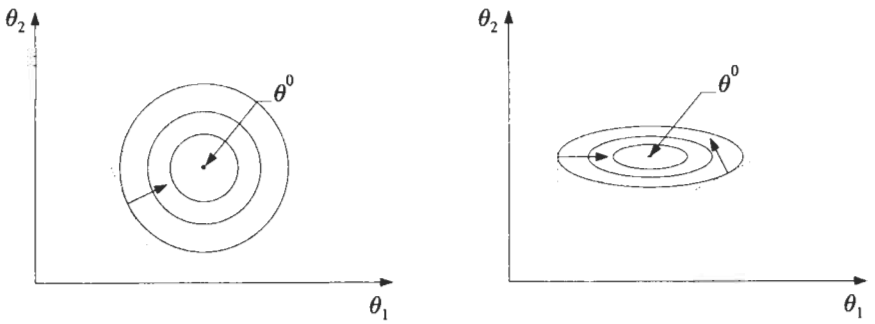


FIGURE C.3: Curves of constant cost values.

two-dimensional space, that is, $\boldsymbol{\theta} = [\theta_1, \theta_2]^T$. The optimum $\boldsymbol{\theta}^0$ is located at the center of the curves. Recall that the gradient $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ is always vertical to the tangent to the constant J curves. Indeed, if $J(\boldsymbol{\theta}) = c$, then

$$dc = 0 = \frac{\partial J(\boldsymbol{\theta})^T}{\partial \boldsymbol{\theta}} d\boldsymbol{\theta} \Longrightarrow \frac{J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \perp d\boldsymbol{\theta} \tag{C.3}$$

Hence, in the case of Figure C.3a the gradient, that is, the correction term, always points to the optimum point. In principle, in such cases, convergence can be achieved in a single step. The scenario is different for the case of Figure C.3b. There, the gradient points to the center at only very few places. Thus, convergence in this case can be quite slow and $\Delta\boldsymbol{\theta}$ can oscillate back and forth following a zigzag path until it rests at the optimum.

- *Quadratic surface:* Let $J(\boldsymbol{\theta})$ be of a quadratic form, that is,

$$J(\boldsymbol{\theta}) = b - \boldsymbol{p}^T\boldsymbol{\theta} + \frac{1}{2}\boldsymbol{\theta}^T R\boldsymbol{\theta} \tag{C.4}$$

  where R is assumed to be positive definite, in order (C.4) to have a (single) minimum (why?). Then,

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = R\boldsymbol{\theta} - \boldsymbol{p} \tag{C.5}$$

  Thus, the optimum value is given by

$$R\boldsymbol{\theta}^0 = \boldsymbol{p} \tag{C.6}$$

  The tth iteration step in (C.1) then becomes

$$\boldsymbol{\theta}(t) = \boldsymbol{\theta}(t-1) - \mu\,(R\boldsymbol{\theta}(t-1) - \boldsymbol{p}) \tag{C.7}$$

Subtracting $\boldsymbol{\theta}^0$ from both sides and taking into account (C.6), (C.7) becomes

$$\tilde{\boldsymbol{\theta}}(t) = \tilde{\boldsymbol{\theta}}(t-1) - \mu R\tilde{\boldsymbol{\theta}}(t-1) = (I - \mu R)\tilde{\boldsymbol{\theta}}(t-1) \tag{C.8}$$

where $\tilde{\boldsymbol{\theta}}(t) \equiv \boldsymbol{\theta}(t) - \boldsymbol{\theta}^0$. Now let R be a symmetric matrix. Then, as we know from Appendix B, it can be diagonalized, that is,

$$R = \Phi^T \Lambda \Phi \tag{C.9}$$

where Φ is the orthogonal matrix with columns the orthonormal eigenvectors of R and Λ the diagonal matrix having the corresponding eigenvalues on its diagonal. Incorporating (C.9) into (C.8) we obtain

$$\hat{\boldsymbol{\theta}}(t) = (I - \mu\Lambda)\hat{\boldsymbol{\theta}}(t-1) \tag{C.10}$$

where $\hat{\boldsymbol{\theta}}(t) \equiv \Phi\tilde{\boldsymbol{\theta}}(t)$. Matrix $I - \mu\Lambda$ is now diagonal, and (C.10) is equivalent to

$$\hat{\theta}_i(t) = (1 - \mu\lambda_i)\hat{\theta}_i(t-1) \tag{C.11}$$

where $\hat{\boldsymbol{\theta}} \equiv [\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_l]^T$. Considering (C.11) for successive iteration steps we obtain

$$\hat{\theta}_i(t) = (1 - \mu\lambda_i)^t \hat{\theta}_i(0) \tag{C.12}$$

which converges to

$$\lim_{t\to\infty} \hat{\theta}_i(t) = 0, \Longrightarrow \lim_{t\to\infty} \theta_i(t) = \theta_i^0, \quad i = 1, 2, \ldots, l \tag{C.13}$$

provided that $|1 - \mu\lambda_i| < 1, i = 1, 2, \ldots, l$. Thus, we can conclude that

$$\boldsymbol{\theta} \longrightarrow \boldsymbol{\theta}^0, \quad \text{if } \mu < \frac{2}{\lambda_{\max}} \tag{C.14}$$

where $\lambda_{\max}$ is the maximum eigenvalue of R (which is positive since R is positive definite). Thus, the convergence speed of the gradient descent algorithm is controlled by the ratio $\lambda_{\min}/\lambda_{\max}$ as (C.12) and (C.14) suggest.

- *Nonquadratic cost functions:* If $J(\boldsymbol{\theta})$ is not quadratic, we can mobilize Taylor's theorem and assume that at some step near a stationary point, $\boldsymbol{\theta}^0$, $J(\boldsymbol{\theta})$ can be written approximately as

$$J(\boldsymbol{\theta}) = J(\boldsymbol{\theta}^0) + (\boldsymbol{\theta} - \boldsymbol{\theta}^0)^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^0)^T H(\boldsymbol{\theta} - \boldsymbol{\theta}^0) \tag{C.15}$$

where $\boldsymbol{g}$ is the gradient at $\boldsymbol{\theta}^0$ and H is the corresponding Hessian matrix, that is,

$$\boldsymbol{g} = \left.\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0}, \quad H(i, j) = \left.\frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \tag{C.16}$$

Thus, in the neighborhood of $\boldsymbol{\theta}^0$, $J(\boldsymbol{\theta})$ is given approximately by a quadratic form and the convergence of the algorithm is controlled by the eigenvalues of the Hessian matrix.

C.2 NEWTON'S ALGORITHM

The problems associated with the dependence of the convergence speed on the eigenvalue spread can be overcome by using Newton's iterative scheme, where the correction in (C.2) is defined by

$$\Delta\boldsymbol{\theta} = -H^{-1}(old)\left.\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}(old)} \tag{C.17}$$

where $H(old)$ is the Hessian matrix computed at $\boldsymbol{\theta}(old)$. Newton's algorithm converges much faster than the gradient descent method and, practically, its speed is independent of the eigenvalue spread. Faster convergence can be demonstrated by looking at the approximation in (C.15). Taking the gradient results in

$$\frac{\partial J(\boldsymbol{\theta})}{\partial(\boldsymbol{\theta})} = \left.\frac{\partial J(\boldsymbol{\theta})}{\partial(\boldsymbol{\theta})}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} + H(\boldsymbol{\theta} - \boldsymbol{\theta}^0) \tag{C.18}$$

Thus, the gradient is a linear function of $\boldsymbol{\theta}$ and hence the Hessian is constant, that is H. Having assumed that $\boldsymbol{\theta}^0$ is a stationary point, the first term on the right-hand side becomes zero. Now let $\boldsymbol{\theta} = \boldsymbol{\theta}(old)$. Then, according to Newton's iteration

$$\boldsymbol{\theta}(new) = \boldsymbol{\theta}(old) - H^{-1}(H(\boldsymbol{\theta}(old) - \boldsymbol{\theta}^0)) = \boldsymbol{\theta}^0 \tag{C.19}$$

Thus, the minimum is found in a single iteration. Of course, in practice, this is not true, as the approximations are not exactly valid. It is true, however, for quadratic costs.

Following a more formal proof (e.g., [Luen 84]), it can be shown that the convergence of Newton's algorithm is quadratic (i.e., the error at one step is proportional to the square of the previous step) while that of the gradient descent is linear. This speedup in convergence is achieved at increased computational cost, since Newton's algorithm requires the computation and then inversion of the Hessian matrix. Furthermore, numerical issues concerning the invertibility of H arise.

C.3 CONJUGATE-GRADIENT METHOD

Discussing the gradient descent method, we saw that, in general, a zigzag path is followed from the initial estimate to the optimum. This drawback is overcome by the following scheme, which results in improved convergence speed with respect to the gradient descent method. Compute the correction term according to the following rule:

$$\Delta\boldsymbol{\theta}(t) = \boldsymbol{g}(t) - \beta(t)\Delta\boldsymbol{\theta}(t-1) \tag{C.20}$$

where

$$\boldsymbol{g}(t) = \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}(t)} \tag{C.21}$$

and

$$\beta(t) = \frac{\boldsymbol{g}^T(t)\boldsymbol{g}(t)}{\boldsymbol{g}^T(t-1)\boldsymbol{g}(t-1)} \tag{C.22}$$

or

$$\beta(t) = \frac{\boldsymbol{g}^T(t)\,(\boldsymbol{g}(t) - \boldsymbol{g}(t-1))}{\boldsymbol{g}^T(t-1)\boldsymbol{g}(t-1)} \tag{C.23}$$

The former is known as the Fletcher–Reeves and the latter as the Polak–Ribiere formula.

For a more rigorous treatment of the topic the reader is referred to [Luen 84]. Finally, it must be stated that a number of variants of these schemes have appeared in the literature.

C.4 OPTIMIZATION FOR CONSTRAINED PROBLEMS

C.4.1 Equality Constraints

We will first focus on linear equality constraints and then generalize to the nonlinear case. Although the philosophy for both cases is the same, it is easier to grasp the

basics when linear constraints are involved. Thus the problem is cast as

$$\begin{array}{ll} \text{minimize} & J(\boldsymbol{\theta}) \\ \text{subject to} & A\boldsymbol{\theta} = \boldsymbol{b} \end{array}$$

where A is an $m \times l$ matrix and $\boldsymbol{b}$, $\boldsymbol{\theta}$ are $m \times 1$ and $l \times 1$ vectors, respectively. It is assumed that the cost function $J(\boldsymbol{\theta})$ is twice continuously differentiable and it is, in general, a nonlinear function. Furthermore, we assume that the rows of A are linearly independent, hence A has full row rank. This assumption is known as the *regularity assumption*.

Let $\boldsymbol{\theta}_*$ be a local minimizer of $J(\boldsymbol{\theta})$ over the set $\{\boldsymbol{\theta}: A\boldsymbol{\theta} = \boldsymbol{b}\}$. Then it is not difficult to show (e.g., [Nash 96]) that, at this point, the gradient of $J(\boldsymbol{\theta})$ is given by

$$\frac{\partial}{\partial \boldsymbol{\theta}}(J(\boldsymbol{\theta}))|_{\boldsymbol{\theta}=\boldsymbol{\theta}_*} = A^T\boldsymbol{\lambda} \tag{C.24}$$

where $\boldsymbol{\lambda} \equiv [\lambda_1, \ldots, \lambda_m]^T$. Taking into account that

$$\frac{\partial}{\partial \boldsymbol{\theta}}(A\boldsymbol{\theta}) = A^T \tag{C.25}$$

Eq. (C.24) states that, at a constrained minimum, the gradient of the cost function is a linear combination of the gradients of the constraints. This is quite natural. Let us take a simple example involving a single linear constraint, i.e.,

$$\boldsymbol{a}^T\boldsymbol{\theta} = b$$

Equation (C.24) then becomes

$$\frac{\partial}{\partial \boldsymbol{\theta}}(J(\boldsymbol{\theta}_*)) = \lambda \boldsymbol{a}$$

where the parameter λ is now a scalar. Figure C.4 shows an example of isovalue contours of $J(\boldsymbol{\theta}) = c$ in the two-dimensional space ($l = 2$). The constrained minimum coincides with the point where the straight line "meets" the isovalue contours for the first time, as one moves from small to large values of c. This is the point where the line is tangent to an isovalue contour; hence at this point the gradient of the cost function is in the direction of $\boldsymbol{a}$ (see Chapter 3.)

Let us now define the function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = J(\boldsymbol{\theta}) - \boldsymbol{\lambda}^T(A\boldsymbol{\theta} - \boldsymbol{b}) \tag{C.26}$$

$$= J(\boldsymbol{\theta}) - \sum_{i=1}^{m} \lambda_i(\boldsymbol{a}_i^T\boldsymbol{\theta} - b_i) \tag{C.27}$$

where $\boldsymbol{a}_i^T$, $i = 1, 2, \ldots, m$, are the rows of A. $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ is known as the *Langrangian function* and the coefficients, λ_i, $i = 1, 2, \ldots, m$, as the *Langrange multipliers*.

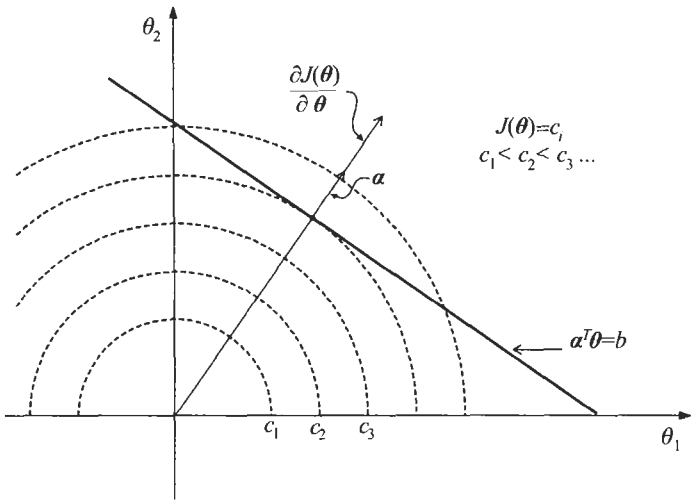


FIGURE C.4: At the minimizer, the gradient of the cost function is in the direction of the gradient of the constraint function.

The optimality condition (C.24), together with the constraints, which the minimizer has to satisfy, can now be written in a compact form as

$$\nabla \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathbf{0} \tag{C.28}$$

where ∇ denotes the gradient operation with respect to both $\boldsymbol{\theta}$ and $\boldsymbol{\lambda}$. Indeed, equating with zero the derivatives of the Langrangian with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\lambda}$ gives, respectively,

$$\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = A^T \boldsymbol{\lambda}$$

$$A\boldsymbol{\theta} = \boldsymbol{b}$$

The above is a set of $m + l$ unknowns, i.e., $(\theta_1, \ldots, \theta_l, \lambda_1, \ldots, \lambda_m)$, with $m + l$ equations, whose solution provides the minimizer $\boldsymbol{\theta}_*$ and the corresponding Langrange multipliers. Similar arguments hold for nonlinear equation constraints. Let us consider the problem

$$\begin{array}{ll} \text{minimize} & J(\boldsymbol{\theta}) \\ \text{subject to} & f_i(\boldsymbol{\theta}) = 0, \quad i = 1, 2, \ldots, m \end{array}$$

The minimizer is again a *stationary point* of the corresponding Langrangian

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = J(\boldsymbol{\theta}) - \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{\theta})$$

and results from the solution of the set of $m + l$ equations

$$\nabla \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathbf{0}$$

The regularity condition for nonlinear constraints requires the gradients of the constraints $\frac{\partial}{\partial \boldsymbol{\theta}}(f_i(\boldsymbol{\theta}))$ to be linearly independent.

C.4.2 Inequality Constraints

The general problem can be cast as follows:

$$\begin{array}{ll} \text{minimize} & J(\boldsymbol{\theta}) \\ \text{subject to} & f_i(\boldsymbol{\theta}) \geq 0, \quad i = 1, 2, \ldots, m \end{array} \tag{C.29}$$

Each one of the constraints defines a region in $\mathcal{R}^l$. The intersection of all these regions defines the area in which the constrained minimum, $\boldsymbol{\theta}_*$, must lie. This is known as the *feasible region* and the points in it (candidate solutions) as *feasible points*. The type of the constraints control the type of the feasible region, i.e., whether it is convex or concave. At this point, it will not harm us to recall a few definitions.

Convex functions. A function $f(\boldsymbol{\theta})$

$$f: S \subseteq \mathcal{R}^l \longrightarrow \mathcal{R}$$

is called convex in S, if for every $\boldsymbol{\theta}$ and $\boldsymbol{\theta}' \in S$

$$f(\lambda\boldsymbol{\theta} + (1 - \lambda)\boldsymbol{\theta}') \leq \lambda f(\boldsymbol{\theta}) + (1 - \lambda) f(\boldsymbol{\theta}')$$

for every $\lambda \in [0, 1]$. If strict inequality holds, we say that the function is strict convex.

Concave functions. A function $f(\boldsymbol{\theta})$ is called concave, if for every $\boldsymbol{\theta}, \boldsymbol{\theta}' \in S$

$$f(\lambda\boldsymbol{\theta} + (1 - \lambda)\boldsymbol{\theta}') \geq \lambda f(\boldsymbol{\theta}) + (1 - \lambda) f(\boldsymbol{\theta}')$$

for every $\lambda \in [0, 1]$. For strict inequality, the function is known as strict concave.

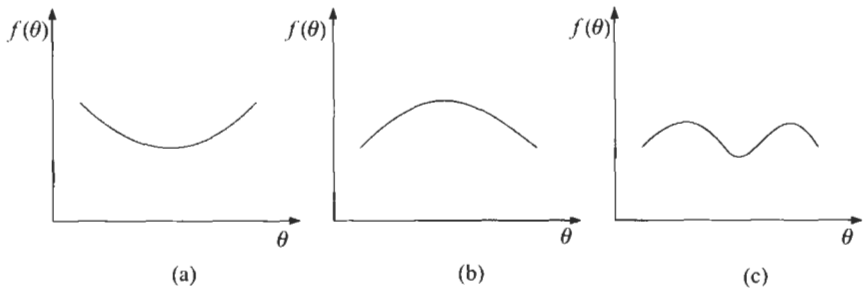


FIGURE C.5: (a) A convex function, (b) a concave function, and (c) a function that is neither convex nor concave.

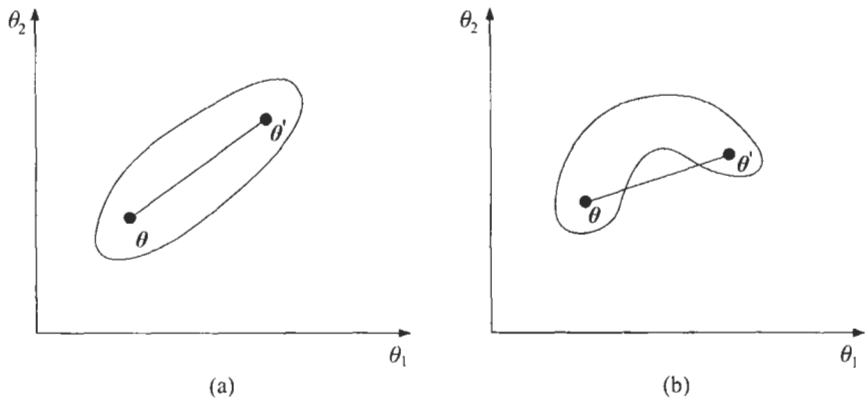


FIGURE C.6: (a) A convex set and (b) a concave set of points.

Figure C.5 shows three functions, one convex, one concave, and one which is neither convex nor concave.

Convex sets. A set $S \subseteq \mathcal{R}^l$ is called convex, if for every pair of points $\boldsymbol{\theta}, \boldsymbol{\theta}' \in S$, the line segment joining these points also belongs to the set. In other words, all points $\lambda\boldsymbol{\theta} + (1 - \lambda)\boldsymbol{\theta}'$, $\lambda \in [0, 1]$ belong to the set. Figure C.6 shows two sets, one convex and one nonconvex.

Remarks

- If $f(\boldsymbol{\theta})$ is convex then $-f(\boldsymbol{\theta})$ is concave and vice versa. Furthermore, if $f_i(\boldsymbol{\theta})$, $i = 1, 2, \ldots, m$, are convex, so is the sum $\sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{\theta})$, $\lambda_i \geq 0$. Similarly, if $f_i(\boldsymbol{\theta})$ are concave, so is their summation.

- If a function $f(\boldsymbol{\theta})$ is convex, it can be shown that *a local minimum is also a global one. This can be easily checked from the graph of Figure C.5. Furthermore, if the function is strict convex then this minimum is unique.* For concave functions, the above also hold true but for points where a maximum occurs.
- A direct consequence of the respective definitions is that if $f(\boldsymbol{\theta})$ is convex then the set

$$X = \{\boldsymbol{\theta} | f(\boldsymbol{\theta}) \leq b,\ b \in \mathcal{R}\}$$

  is convex. Also, if $f(\boldsymbol{\theta})$ is concave then the set

$$X = \{\boldsymbol{\theta} | f(\boldsymbol{\theta}) \geq b,\ b \in \mathcal{R}\}$$

  is also convex.
- The intersection of convex sets is also a convex set.

From the above remarks one can easily conclude that, if each one of the functions in the constraints in (C.29) is concave, then the feasible region is a convex one. This is also valid if the constraints are linear, since a linear function can be considered either as convex or concave. For more on these issues, the interested reader may refer, for example, to [Baza 79].

The Karush–Kuhn–Tucker (KKT) Conditions

This is a set of *necessary* conditions, which a local minimizer $\boldsymbol{\theta}_*$ of the problem given in (C.29) has to satisfy. If $\boldsymbol{\theta}_*$ is a point that satisfies the regularity condition, then there exists a vector $\boldsymbol{\lambda}$ of Langrange multipliers so that the following are valid:

$$\begin{aligned} &(1) \quad \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}) = \mathbf{0} \\ &(2) \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, m \\ &(3) \quad \lambda_i f_i(\boldsymbol{\theta}_*) = 0, \quad i = 1, 2, \ldots, m \end{aligned} \tag{C.30}$$

Actually, there is a fourth condition concerning the Hessian of the Langrangian function, which is not of interest to us. The above set of equations is also part of the sufficiency conditions; however, in this case, there are a few subtle points and the interested reader is referred to more specialized textbooks, e.g., [Baza 79, Flet 87, Bert 95, Nash 96].

Conditions (3) in (C.30) are known as *complementary slackness conditions*. They state that at least one of the terms in the products is zero. In the case where, in each one of the equations, only one of the two terms is zero, i.e., either λ_i or $f_i(\boldsymbol{\theta}_*)$, we talk about *strict complementarity*.

Remarks

- The first condition is most natural. It states that the minimum must be a stationary point of the Langrangian, with respect to $\boldsymbol{\theta}$.
- A constraint, $(f_i(\boldsymbol{\theta}_*))$, is called *inactive* if the corresponding Langrange multiplier is zero. This is because this constraint does not affect the problem. A constrained minimizer $\boldsymbol{\theta}_*$ can lie either in the interior of the feasible region or on its boundary. In the former case, the problem is equivalent to an unconstrained one. Indeed, if it happens that a minimum is located within the feasible region, then the value of the cost function in a region around this point will increase (or remain the same) as one moves away from this point. Hence, this point will be a stationary point of the cost function $J(\boldsymbol{\theta})$. Thus in this case, the constraints are redundant and do not affect the problem. In words, the constraints are inactive and this is equivalent to setting the Langrange multipliers equal to zero. The nontrivial constrained optimization task is when the (unconstrained) minimum of the cost function is located outside the feasible region. In this case, the constrained minimum will be located on the boundary of the feasible region. In other words, in this nontrivial case, there will be one or more of the constraints for which $f_i(\boldsymbol{\theta}_*) = 0$. These constitute the *active constraints*. The rest of the constraints will be inactive with the corresponding Langrange multipliers being zero.

  Figure C.7 illustrates a simple case with the following constraints:

$$f_1(\boldsymbol{\theta}) = \theta_1 + 2\theta_2 - 2 \geq 0$$
$$f_2(\boldsymbol{\theta}) = \theta_1 - \theta_2 + 2 \geq 0$$
$$f_3(\boldsymbol{\theta}) = -\theta_1 + 2 \geq 0$$

  The (unconstrained) minimum of the cost function is located outside the feasible region. The dotted lines are the isovalue curves $J(\boldsymbol{\theta}) = c$, with $c_1 < c_2 < c_3$. The constrained minimum coincides with the point where an isovalue curve "touches" the boundary of the feasible region for the first time (smallest value of c). This point may belong to more than one of the constraints, e.g., a corner point of the boundary.
- The Langrange multipliers of the active constraints are *nonnegative*. To understand why this is so, let us consider for simplicity the case of linear constraints $A\boldsymbol{\theta} \geq \boldsymbol{b}$, where A includes the active constraints only. If $\boldsymbol{\theta}_*$ is a minimizer lying on the active constraints, then any other feasible point can be written as

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_* + \boldsymbol{p}$$
$$A\boldsymbol{p} \geq \boldsymbol{0}$$

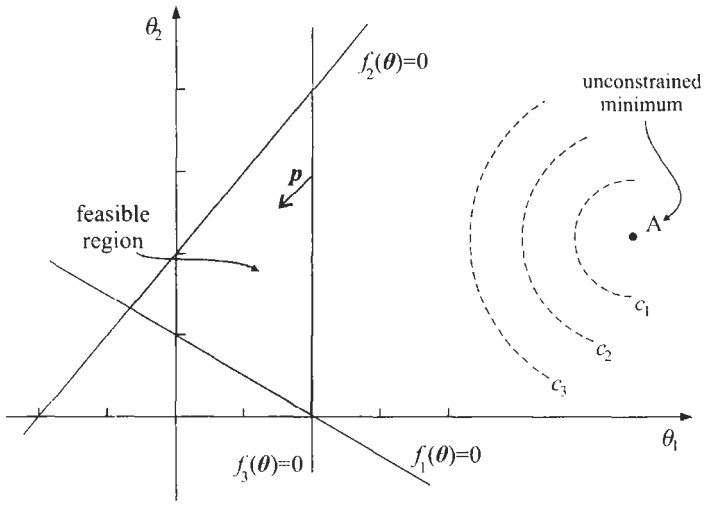


FIGURE C.7: An example of the nontrivial case, where the unconstrained minimum lies outside the feasible region.

since this guarantees that $A\hat{\boldsymbol{\theta}} \geq \boldsymbol{b}$. If the direction $\boldsymbol{p}$ points into the feasible region (Figure C.7) then $A\boldsymbol{p} \neq \mathbf{0}$, that is, some of its components are strictly positive. Since $\boldsymbol{\theta}_*$ is a minimizer, from condition (1) in (C.30) we have that

$$\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}_*) = A^T \boldsymbol{\lambda}$$

The change of the cost function along the direction of $\boldsymbol{p}$ is proportional to

$$\boldsymbol{p}^T \frac{\partial}{\partial \boldsymbol{\theta}} (J(\boldsymbol{\theta})) = \boldsymbol{p}^T A^T \boldsymbol{\lambda}$$

and since $\boldsymbol{\theta}_*$ is a minimizer, this must be a direction of ascent at $\boldsymbol{\theta}_*$. Thus $\boldsymbol{\lambda}$ must be nonnegative to guarantee that $\boldsymbol{p}^T A^T \boldsymbol{\lambda} \geq 0$ for any $\boldsymbol{p}$ pointing into the feasible region. An active constraint whose corresponding Langrange multiplier is zero is known as *degenerate*.

- It can be shown that, if the cost function is convex and the feasible region is also convex, then a local minimum is also a global one. A little thought (and a look at Figure C.7) suffices to see why this is so.

Having now discussed all these nice properties, the major question arises: how can one compute a constrained (local) minimum? Unfortunately, this is not always an easy task. A straightforward approach would be to assume that some of the

constraints are active and some inactive, and check if the resulting Langrange multipliers of the active constraints are nonnegative. If not, then choose another combination of constraints and repeat the procedure until one ends up with non-negative multipliers. However, in practice, this may require a prohibitive amount of computation. Instead, a number of alternative approaches have been proposed. In the sequel, we will review some basics from Game Theory and use these to reformulate the KKT conditions. This new setup can be useful in a number of cases in practice.

Min-Max Duality

Let us consider two players, namely X and Y, playing a game. Player X will choose a strategy, say, x and simultaneously player Y will choose a strategy y. As a result, X will pay to Y the amount $\mathcal{F}(x, y)$, which can also be negative, i.e., X wins. Let us now follow their thinking, prior to their final choice of strategy, assuming that the players are good professionals.

X: If Y knew that I am going to choose x, then, since he/she is a clever player, he/she would choose y to make his/her profit maximum, i.e.,

$$\mathcal{F}^*(x) = \max_y \mathcal{F}(x, y)$$

Thus, in order to make my *worst-case payoff* to Y minimum, I have to choose x so as to minimize $\mathcal{F}^*(x)$, i.e.,

$$\min_x \mathcal{F}^*(x)$$

This problem is known as the *min-max* problem since it seeks the value

$$\min_x \max_y \mathcal{F}(x, y)$$

Y: X is a good player, so if he/she knew that I am going to play y, he/she would choose x so that to make his/her payoff minimum, i.e.,

$$\mathcal{F}_*(y) = \min_x \mathcal{F}(x, y)$$

Thus, in order to make my *worst-case profit* maximum I must choose y that maximizes $\mathcal{F}_*(y)$, i.e.,

$$\max_y \mathcal{F}_*(y)$$

This is known as the *max-min* problem, since it seeks the value

$$\max_y \min_x \mathcal{F}(x, y)$$

The two problems are said to be *dual to each other*. The first is known to be the *primal*, whose objective is to minimize $\mathcal{F}^*(x)$ and the second is the *dual* problem with the objective to maximize $\mathcal{F}_*(y)$.

For any x and y, the following is valid:

$$\mathcal{F}_*(y) \equiv \min_x \mathcal{F}(x, y) \leq \mathcal{F}(x, y) \leq \max_y \mathcal{F}(x, y) \equiv \mathcal{F}^*(x) \tag{C.31}$$

which easily leads to

$$\max_y \min_x \mathcal{F}(x, y) \leq \min_x \max_y \mathcal{F}(x, y) \tag{C.32}$$

Saddle Point Condition

Let $\mathcal{F}(\boldsymbol{x}, \boldsymbol{y})$ be a function of two vector variables with $\boldsymbol{x} \in X \subseteq \mathcal{R}^l$ and $\boldsymbol{y} \in Y \subseteq \mathcal{R}^l$. If a pair of points $(\boldsymbol{x}_*, \boldsymbol{y}_*)$, with $\boldsymbol{x}_* \in X$, $\boldsymbol{y}_* \in Y$ satisfies the condition

$$\mathcal{F}(\boldsymbol{x}_*, \boldsymbol{y}) \leq \mathcal{F}(\boldsymbol{x}_*, \boldsymbol{y}_*) \leq \mathcal{F}(\boldsymbol{x}, \boldsymbol{y}_*) \tag{C.33}$$

for every $\boldsymbol{x} \in X$ and $\boldsymbol{y} \in Y$, we say that it satisfies the *saddle point condition*. It is not difficult to show (e.g., [Nash 96]) that a pair $(\boldsymbol{x}_*, \boldsymbol{y}_*)$ satisfies the saddle point conditions *if and only if*

$$\max_{\boldsymbol{y}} \min_{\boldsymbol{x}} \mathcal{F}(\boldsymbol{x}, \boldsymbol{y}) = \min_{\boldsymbol{x}} \max_{\boldsymbol{y}} \mathcal{F}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{F}(\boldsymbol{x}_*, \boldsymbol{y}_*) \tag{C.34}$$

Langrangian Duality

We will now use all the above in order to formulate our original cost function minimization problem as a min-max task of the corresponding Langrangian function. Under certain conditions, this formulation can lead to computational savings when computing the constrained minimum. The optimization task of our interest is

$$\begin{array}{ll} \text{minimize} & J(\boldsymbol{\theta}) \\ \text{subject to} & f_i(\boldsymbol{\theta}) \geq 0, \quad i = 1, 2, \ldots, m \end{array}$$

The Langrangian function is

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = J(\boldsymbol{\theta}) - \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{\theta}) \tag{C.35}$$

Let

$$\mathcal{L}^*(\boldsymbol{\theta}) = \max_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \tag{C.36}$$

However, since $\boldsymbol{\lambda} \geq \mathbf{0}$ and $f_i(\boldsymbol{\theta}) \geq 0$, the maximum value of the Langrangian occurs if the summation in (C.35) is zero (either $\lambda_i = 0$ or $f_i(\boldsymbol{\theta}) = 0$ or both) and

$$\mathcal{L}^*(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) \tag{C.37}$$

Therefore our original problem is equivalent with

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\lambda} \geq \mathbf{0}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \tag{C.38}$$

As we already know, the dual problem of the above is

$$\max_{\boldsymbol{\lambda} \geq \mathbf{0}} \min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \tag{C.39}$$

Convex Programming

A large class of practical problems obeys the following two conditions:

$$(1) \quad J(\boldsymbol{\theta}) \text{ is convex} \tag{C.40}$$

$$(2) \quad f_i(\boldsymbol{\theta}) \text{ are concave} \tag{C.41}$$

This class of problems turns out to have a very useful and mathematically tractable property.

Theorem. *Let $\boldsymbol{\theta}_*$ be a minimizer of such a problem, which is also assumed to satisfy the regularity condition. Let $\boldsymbol{\lambda}_*$ be the corresponding vector of Langrange multipliers. Then $(\boldsymbol{\theta}_*, \boldsymbol{\lambda}_*)$ is a saddle point of the Langrangian function, and as we know this is equivalent to*

$$\mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}_*) = \max_{\boldsymbol{\lambda} \geq \mathbf{0}} \min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\lambda} \geq \mathbf{0}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \tag{C.42}$$

Proof. Since $f_i(\boldsymbol{\theta})$ are concave, $-f_i(\boldsymbol{\theta})$ are convex, so the Langrangian function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = J(\boldsymbol{\theta}) - \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{\theta})$$

for $\lambda_i \geq 0$, is also convex. Note, now, that for concave function constraints of the form $f_i(\boldsymbol{\theta}) \geq 0$, the feasible region is convex (see remarks above). The function

$J(\boldsymbol{\theta})$ is also convex. Hence, as already stated in the remarks, every local minimum is also a global one; thus for any $\boldsymbol{\theta}$

$$\mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}_*) \leq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}_*) \tag{C.43}$$

Furthermore, the complementary slackness conditions suggest that

$$\mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}_*) = J(\boldsymbol{\theta}_*) \tag{C.44}$$

and for any $\boldsymbol{\lambda} \geq \mathbf{0}$

$$\mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}) \equiv J(\boldsymbol{\theta}_*) - \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{\theta}_*) \leq J(\boldsymbol{\theta}_*) = \mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}_*) \tag{C.45}$$

Combining (C.43) and (C.45) we obtain

$$\mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}) \leq \mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}_*) \leq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}_*) \tag{C.46}$$

In other words, *the solution $(\boldsymbol{\theta}_*, \boldsymbol{\lambda}_*)$ is a saddle point.* ■

This is a very important theorem and it states that the constrained minimum of a convex programming problem can also be obtained as a maximization task applied on the Langrangian. This leads us to the following very useful formulation of the optimization task.

Wolfe Dual Representation

A convex programming problem is equivalent to

$$\max_{\boldsymbol{\lambda} \geq \mathbf{0}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \tag{C.47}$$

$$\text{subject to} \quad \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathbf{0} \tag{C.48}$$

The last equation guarantees that $\boldsymbol{\theta}$ is a minimum of the Langrangian.

Example C.1. Consider the quadratic problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} \\ \text{subject to} \quad & A\boldsymbol{\theta} \geq \boldsymbol{b} \end{aligned}$$

This is a convex programming problem; hence the Wolfe dual representation is valid:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} - \boldsymbol{\lambda}^T(A\boldsymbol{\theta} - \boldsymbol{b}) \\ \text{subject to} \quad & \boldsymbol{\theta} - A^T\boldsymbol{\lambda} = \mathbf{0} \end{aligned}$$

For this example, the equality constraint has an analytic solution (this is not, however, always possible). Solving with respect to $\boldsymbol{\theta}$, we can eliminate it from the maximizing

function and the resulting dual problem involves only the Langrange multipliers,

$$\max_{\boldsymbol{\lambda}} \left\{ -\frac{1}{2}\boldsymbol{\lambda}^T A A^T \boldsymbol{\lambda} + \boldsymbol{\lambda}^T b \right\}$$

$$\text{subject to } \boldsymbol{\lambda} \geq \mathbf{0}$$

This is also a quadratic problem but the set of constraints is now simpler.

References

[Baza 79] Bazaraa M.S., Shetty C.M. *Nonlinear Programming: Theory and Algorithms*, John Wiley, 1979.

[Bert 95] Bertsekas, D.P., Belmont, M.A. *Nonlinear Programming*, Athenas Scientific, 1995.

[Flet 87] Fletcher, R. *Practical Methods of Optimization*, 2nd edition, John Wiley, 1987.

[Luen 84] Luenberger D.G. *Linear and nonlinear programming*, Addison Wesley, 1984.

[Nash 96] Nash S.G., Sofer A. *Linear and Nonlinear Programming*, McGraw-Hill, 1996.

APPENDIX D

BASIC DEFINITIONS FROM LINEAR SYSTEMS THEORY

D.1 LINEAR TIME INVARIANT (LTI) SYSTEMS

A discrete linear time-invariant system is characterized uniquely by its *impulse response sequence*, $h(n)$. This is the output of the system when its input is excited by the impulse sequence, $\delta(n)$, that is,

$$\delta(n) = \begin{cases} 1 & \text{for } n = 0 \\ 0 & \text{for } n \neq 1 \end{cases} \tag{D.1}$$

When its input is excited by a sequence $x(n)$, its output sequence is given by the *convolution* of $x(n)$ with $h(n)$, defined as

$$y(n) = \sum_{k=-\infty}^{+\infty} h(k)x(n-k) = \sum_{k=-\infty}^{+\infty} x(k)h(n-k) \equiv h(n) * x(n) \tag{D.2}$$

For continuous time systems the convolution becomes an integral, that is,

$$\begin{aligned} y(t) &= \int_{-\infty}^{+\infty} h(\tau)x(t-\tau)\,d\tau \\ &= \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)\,d\tau \equiv x(t) * y(t) \end{aligned} \tag{D.3}$$

where $h(t)$ is the impulse response of the system, that is, the output when its input is excited by the Dirac delta function $\delta(t)$, defined by

$$\delta(t) = 0, \quad \text{for } t \neq 0, \quad \text{and} \quad \int_{-\infty}^{+\infty} \delta(t)dt = 1 \tag{D.4}$$

Linear time-invariant systems can be:

- *Causal:* Their impulse response is zero for $n < 0$. Otherwise, they are known as *noncausal*. Observe that only causal systems can be realized in real time. This is because for noncausal systems, the output at time n would require knowledge of future samples $x(n+1), x(n+2), \ldots$, which in practice is not possible.
- *Finite impulse response (FIR):* The corresponding impulse response is of finite extent. If this is not the case, the systems are known as *infinite impulse response (IIR)* systems. For a causal FIR system the input–output relation becomes

$$y(n) = \sum_{k=0}^{L-1} h(k)x(n-k) \tag{D.5}$$

where L is the length of the impulse response. When a system is FIR but noncausal, it can become causal by delaying its output. Take for example the system with impulse response $0, \ldots, 0, h(-2), h(-1), h(0), h(1), h(2), 0, \ldots$. Then,

$$\begin{aligned} y(n-2) = {} & h(-2)x(n) + h(-1)x(n-1) + h(0)x(n-2) \\ & + h(1)x(n-3) + h(2)x(n-4) \end{aligned} \tag{D.6}$$

That is, at time "n" the output corresponds to the delayed time "$n-2$." The delay is equal to the maximum negative index of nonzero impulse coefficient.

D.2 TRANSFER FUNCTION

The z-transform of the impulse response, defined as

$$H(z) = \sum_{n=-\infty}^{+\infty} h(n)z^{-n} \tag{D.7}$$

is known as the *transfer function* of the system. The free parameter z is a complex variable. The definition in (D.7) is meaningful, provided that the series converges. For most of the sequences of our interest this is true for some region in the complex plane. It can easily be shown that for causal and FIR systems the region of convergence is of the form

$$|z| > |R|, \quad \text{for some } |R| < 1 \tag{D.8}$$

that is, it is the exterior of a circle in the complex plane, centered at the origin, and it contains the unit circle ($|z| = 1$). Let $X(z)$ and $Y(z)$ be the z-transforms of the

input and output sequences of a linear time-invariant system. Then (D.2) is shown to be equivalent to

$$Y(z) = H(z)X(z) \tag{D.9}$$

If the unit circle is in the region of convergence of the respective z-transforms (for example, for causal FIR systems), then for $z = \exp(-j\omega)$ we obtain the equivalent Fourier transform and

$$Y(\omega) = H(\omega)X(\omega) \tag{D.10}$$

If the impulse response of a linear time-invariant system is delayed by r samples, for example, to make it causal in case it is noncausal, the transfer function of the delayed system is given by $z^{-r}H(z)$.

D.3 SERIAL AND PARALLEL CONNECTION

Consider two LTI systems with responses $h_1(n)$ and $h_2(n)$, respectively. Figure D.1a shows the two systems connected in serial and Figure D.1b in parallel.

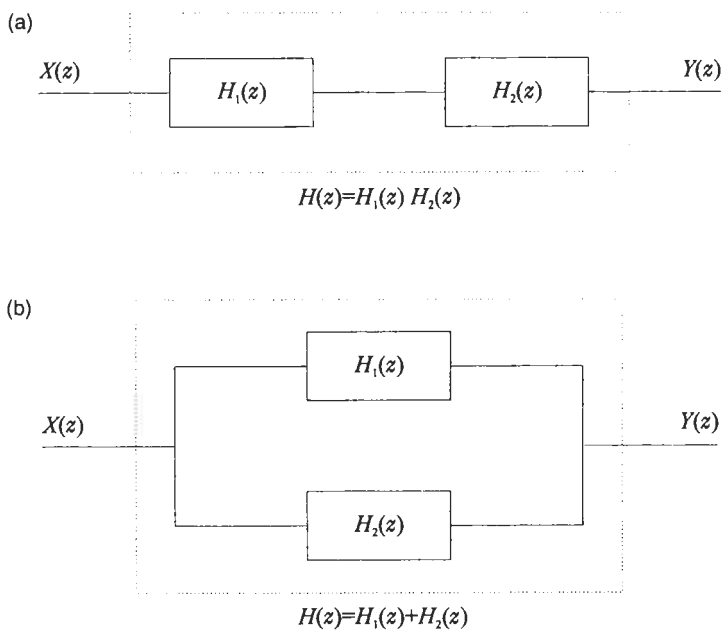


FIGURE D.1: Serial and Parallel connections of LTI systems.

The overall impulse responses are easily shown to be

$$\text{Serial} \quad h(n) = h_1(n) * h_2(n) \tag{D.11}$$

$$\text{Parallel} \quad h(n) = h_1(n) + h_2(n) \tag{D.12}$$

D.4 TWO-DIMENSIONAL GENERALIZATIONS

A two-dimensional linear time-invariant system is also characterized by its two-dimensional impulse response sequence $H(m, n)$, which in the case of images is known as a *point spread function*. On filtering an input image array $X(m, n)$ by $H(m, n)$ the resulting image array is given by the two-dimensional convolution

$$\begin{aligned} Y(m, n) &= \sum_k \sum_l H(m - k, n - l)X(k, l) \equiv H(m, n) * *X(m, n) \\ &= \sum_k \sum_l H(k, l)X(m - k, n - l) \end{aligned} \tag{D.13}$$

INDEX

A

Absolute moments, 272
Acceptance interval, 169
Adaline, 70
Adaptive fuzzy C-shells (AFCS) algorithm, 512
Adaptive momentum, 125
Adaptive resonance theory (ART2), 435
Agglomerative hierarchical algorithms, 450
Akaike Information Criterion (AIC), 522, 614
Algebraic distance, 508. *See also* distance between a point and a quadratic surface
Alternative hypothesis, 166
Alternating Cluster Estimation, 522
Alternating Optimization, 504
Angular second moment, 274
Any path method, 363
Autoregressive–moving average (ARMA(p, m)), 287
Autoregressive processes, 287
Average expected quantization error, 534
Average partition density (PA), 619
Average partition shell density, 620
Average Risk, 17

B

Backpropagation algorithm, 110, 104
Basic competitive learning algorithm, 554
Basic sequential algorithmic scheme (BSAS), 433
Basis images, 209
Basis images (matrices), 209
Basis sequences, 229, 243
Basis vectors, 208
Baum–Welch reestimation, 367
Bayes classification rule, 14
Bayes decision theory, 13
Bayes rule, 13
Bayesian Inference, 32
Bayesian Information Criterion (BIC), 614
Bellman's optimality principle, 324
Bending energy, 302
Best path method, 365
Between-class scatter matrix, 179
Bhattacharyya distance, 177
Bias–variance dilemma, 76
Binary morphology based clustering algorithm (BMCA), 571
Biorthogonal expansion, 243
Biorthogonality condition, 245
Boosting; *See* Combining classifiers
Bootstrapping techniques, 594
Boundary detection algorithms (BDA), 432
Branch and bound, 186
Branch and bound clustering (BBC) algorithm, 562, 432
Branch and bound methods, 561

C

c-means algorithm. *See* isodata algorithm
Center of mass, 301
Central moments, 271

Centroid condition, 535
Chain codes, 298
Chaining effect, 457
Channel equalization, 356
Characteristic functions, 404, 644
Chernoff bound, 177
Circular backpropagation model, 126
Classification error probability, 16
Classification tree, 561
Classifiers, 3
Classifier combining; *See* Combining classifiers
Closing, 567
Cluster, 397, 402
  compact, 489
  linear-shaped, 489
  ring-shaped, 489, 497
  shell-shaped, 489
  spherical, 517
Cluster detection algorithm for discrete valued sets (CDADV), 569
Cluster validity, 591
Cluster variation, 617
Clustering, 402–404, 429
Clustering criterion, 397, 399
Clustering hypothesis, 575
Clustering tendency, 399, 591, 624
Co-occurrence, 273
Co-occurrence matrices, 272
Code vector, 533. *See also* reproduction vector
Combining classifiers, 150
Compactness and separation validity function, 617, *See also* Xie–Beni index
Competitive learning algorithms, 552–560
Competitive learning associated with cost functions, 558
Complete link, 455, 469
Computer-aided diagnosis, 2
Computer storage utilization, 533
Concordant pair, 603
Confusion matrix, 496, 506
Conjugate gradient, 664
Conscientious competitive learning algorithms, 556
Constrained optimization, 664
Constraints, 335
Constructive techniques, 102
Context-dependent classification, 351
Contingency table, 410
Continuous observation HMM, 370
Continuous speech recognition, 329
Contrast, 274
Convex hull, 625
Convex functions, 667
Convex programming, 674
Convex sets, 668
Cophenetic correlation coefficient (CPCC), 602
Cophenetic distance, 476
Cophenetic matrix, 477
"Corrected" statistic, 600
Correlation, 66
Cost function, 489
Covariance matrix, 20
Cox–Lewis test, 630
Crisp clustering algorithms, 432. *See* hard clustering algorithms
Critical interval, 166
Cross-correlation, 66
Cross-correlation coefficient, 338
Cross-entropy, 115, 372, 647
Crossover, 461
Cumulants, 221, 645
Curse of dimensionality, 43, 134
Curvature features, 300

D

Data compression, 400, 533
Data normalization, 165
Data reduction, 400
Davies–Bouldin (DB) index, 612–613
Davies–Bouldin-like indices, 612–613
Decision surfaces, 19
Decision trees, 143
Decomposition layers, 630
Deformable template matching, 343

Delta-bar-delta, 113
Delta-delta rule, 113
Dendrogram, 452. *See also* threshold dendrogram
Deterministic annealing, 580–581
Diameter of a cluster, 610
Dilation, 565
Directed graphs, 464
Directed path, 550
Directed tree, 550
Direction length features, 299
Discordant pair, 603
Discrete binary (DB) set, 564
Discrete cosine, 230
Discrete cosine transform (DCT), 230
Discrete fourier transform, 226
Discrete observation HMM models, 366
Discrete sine transform (DST), 231
Discrete time wavelet coefficients, 243
Discrete time wavelet transform, 239
Discrete wavelet frame, 260
Discriminant functions, 19
Dispersion of a cluster, 612
Dissimilarity matrix, 451
Dissimilarity measure
  between points, 423, 425
  between sets, 406, 423
Distance between a point and a quadratic surface
  algebraic distance, 508
  normalized radial distance, 510
  perpendicular distance, 508
  radial distance, 509
Distortion function, 534
Distortion measure, 534
Divergence, 174
Divisive hierarchical algorithm, 477, 450, 480
Dunn index, 610
Dunn-like indices, 610–611
Dynamic programming, 186, 324
Dynamic similarity measures, 413
Dynamic time warping, 329

E

Eccentricity, 301
Edge connectivity, 467
Edgeworth expansion, 223, 646
Edit distance, 325
EM-algorithm, 491
Empirical classification error, 193
End point constraints, 333
Entropy, 34, 213, 272, 275
Erosion, 565
Error counting approach, 385
Euclidean distance, 25, 405
Expectation maximization (EM) Algorithm, 36
External criteria, 592, 595–601

F

Feature selection, 163, 398
Features, 3
  interval scaled, 401
  nominal, 401
  ordinal, 401
  ratio scaled, 401
Feature vectors, 3
Finite state automation, 361
First-order statistics features, 270
Fisher's discriminant ratio, 181
Fisher's linear discriminant, 190
Floating search methods, 185
Fourier descriptors, 296
Fowlkes and Mallows index, 598
Fractal dimension, 303
Fractals, 303
Fractional Brownian motion sequences, 307
Frobenius norm, 217
Fukuyamma–Sugeno index (FS), 618
Fuzzifier, 501
Fuzzy approaches, 490. *See also* fuzzy clustering algorithms
Fuzzy average shell thickness, 620
Fuzzy C ellipsoidal shells (FCES) algorithm, 515
Fuzzy c-Means (FCM) algorithm, 505

Fuzzy c + 2 means algorithm, 522
Fuzzy C plano-quadric shells (FCPQS), 517
Fuzzy C quadric shells (FCQS) algorithm, 516
Fuzzy c-varieties (FCV) algorithm
Fuzzy covariance matrix, 618
Fuzzy density, 619
Fuzzy hypervolume, 618
Fuzzy measures, 415
Fuzzy shell clustering algorithms, 512–517
  adaptive fuzzy C-shells (AFCS) algorithm, 512–513
  fuzzy C ellipsoidal shells (FCES) algorithm, 512, 515, 541
  fuzzy C plano-quadric shells (FCPQS), 517
  fuzzy C quadric shells (FCQS) algorithm, 515–516, 541
  modified fuzzy C quadric shells (MFCQS) algorithm, 515–516, 541
Fuzzy shell covariance matrix, 620
Fuzzy shell density, 620

G

Γ statistic, 598, 604
γ statistic, 603
Gabor filter, 260
Gabriel graphs (GG), 550, 611, 613
Gauss–Newton method, 506
Generalized agglomerative scheme (GAS), 450
Generalized competitive learning scheme (GCLS), 554
Generalized divisive scheme (GDS), 478
Generalized fuzzy algorithmic scheme (GFAS), 504
Generalized hard algorithmic scheme (GHAS), 530
Generalized linear classifiers, 127
Generalized mixture decomposition algorithmic scheme (GMDAS), 492
Generalized possibilistic algorithmic scheme (GPAS), 525
Generalized XB index, 617
Genetic algorithms, 432, 545, 582
  crossover, 582
  mutation, 582
  reproduction, 582
Geometric moments, 281
Gibbs random fields, 377
Global constraints, 325, 333
Global convergence theorem, 522
Grade of membership, 403
Gradient descent algorithm, 57, 106, 659
Graph
  complete, 546
  edges, 464, 546
  inconsistent edges, 546
  vertices, 464
Graph theory–based algorithmic scheme (GTAS), 467
Gray level run lengths, 275
Gustafson–Kessel (G–K) algorithm, 517

H

Haar transform, 233, 249
Hadamard transform, 231
Hamming distance, 410–411
Hard clustering, 491
Hard clustering algorithms, 529
Hidden Markov models, 361
Hierarchical clustering algorithms, 431
  agglomerative algorithms, 431
  divisive algorithms, 431
Hierarchical search, 341
Higher order, 126
Holdout method, 388
Hopkins test, 629
Hubert's Γ statistic, 598. *See also* Γ statistic
Hughes phenomenon, 196
Hurst parameter, 308
Hypercube, 521
Hypersphere, 595

Hypothesis generation, 400
Hypothesis testing, 400, 592–605

I

Images, 207
Incomplete data set, 36
Independent component analysis (ICA), 219
Information theoretic clustering, 584
Information theory based criteria, 584, 613, 614
Inner product, 408
Internal criteria, 592, 595, 602–605
Interpoint distances, 628
Interpretation of clustering results, 399
Intersymbol interference, 356
Inverse difference moment, 275
Isodata algorithm, 532
Isolated word recognition, 329
Itakura constraints, 333, 334
Itakura–Saito distortion, 555

J

Jaccard coefficient, 598

K

Karhunen–Loéve transform, 210
Karush–Kuhn–Tucker (KKT) conditions, 84, 669
Kernels, 41
Kesler's construction, 64
k-means algorithm. *See* isodata algorithm
k Nearest Neighbor density estimation, 43
Kullback–Leibler distance, 176, 223, 647
Kurtosis, 226, 271, 645

L

Langrange multipliers, 665
LBG algorithm, 535
Leaky learning algorithm, 556
Learning subspace methods, 214
Least squares methods, 65
Leave-one-out method, 388
Levenberg–Marquardt (L–M) method, 506
Levinson's algorithm, 288
Lexicographic ordering, 208
Lifetime of a cluster, 480
Likelihood function, 20, 28
Likelihood ratio, 18
Likelihood ratio test, 44
Linear classifiers, 55
Linear dichotomies, 129
Lloyd's algorithm, 535
LMS algorithm, 68
l_p metric dissimilarity measures, 407–408
Local constraints, 325, 334
Local feature extractor, 279
Logarithmic search, 340
Logistic function, 104
Long run emphasis, 278
Loss matrix, 17

M

Machine vision, 1
Mahalanobis distance, 26
Markov chain, 352
Markov model, 352
Markov random fields, 292, 375
Matching score generator, 443
Matrix updating algorithmic scheme (MUAS), 455
Maximally complete subgraph, 465
Maximally connected subgraph, 465
Maximum a posteriori probability estimation, 31
Maximum entropy estimation, 34
Maximum likelihood, 28
MaxNet, 443, 444
Mean center, 421
Mean square error estimation, 65
Mean square error regression, 72
Mean vector, 420
Median center, 421
Mellin transforms, 339
Membership functions, 403

Merging procedure, 441
Minimum description length (MDL), 614
Minimum distance classifier, 25
Minimum spanning tree (MST), 546, 611, 613, 628, 630
Minimum variance algorithm, 458, *See also* Ward's algorithm
Minmax duality, 672
Minkowski distance, 505
Mixture decomposition, 490
Mixture models, 35
Mixture scatter matrix, 180
Mode-seeking property, 526
Modified BSAS, 437
Modified fuzzy C quadric shells (FCQS) algorithm, 515
Modified Hubert Γ statistic, 608
Moments, 270
Moments of Hu, 283
Momentum factor, 112
Monothetic algorithms, 479
Monotonicity, 334, 461
Monte Carlo techniques, 593
Morphological operations, 565–568
  closing, 568
  dilation, 566
  erosion, 566
  opening, 568
  translation, 566
Motion compensation, 337
Motion estimation, 337
Mountain method, 583
Multiresolution analysis, 251
Multiresolution decomposition, 238
Multiresolution interpretation, 250
Multispectral remote sensing, 7
Mutual information, 223, 372, 647

N

Natural gradient, 225
Nearest neighbor condition, 534
Nearest neighbor distances tests, 629
Nearest Neighbor rule, 44
Nested clusterings, 449
Network, 126
Neural networks, 373
Neyman–Scott procedure, 626
Newton's algorithm, 663
Noble identity, 236, 245
Node connectivity, 466
Node impurity, 146
Node degree, 467
Normalization, 565
Normalized Γ statistic, 598
Normalized central moments, 282
Normalized radial distance, 510, *See also* distance between a point and a quadratic surface
Null hypothesis, 166, 592, 594

O

Octave-band filter banks, 247
One-dimensional DFT, 227
One-tailed statistical test, 593
  left-tailed statistical test, 594
  right-tailed statistical test, 594
Opening, 567
Optical character recognition (OCR), 294
Optimization, 659
Ordinal proximity matrices, 621, 622
Orientation, 301
Outliers, 498, 522, 524
Overtraining, 119

P

P statistic, 632
Packing density, 627
Paraunitary, 244
Partial clusterings, 561
Partition algorithmic schemes, 526
Partition coefficient (PC), 619
Partition density (PD) index, 619
Partition entropy (PE), 615, 619
Partition shell density, 620
Parzen windows, 41, 577
Path of a graph, 464
Pattern mode algorithm, 553

Perceptron, 61
Perceptron algorithm, 57
Perceptron cost, 57
Perfect reconstruction, 242
Perpendicular distance, 508, *See also* distance between a point and a quardatic surface
pFCM, 506
Pocket algorithm, 63
Poisson distribution, 626
Poisson process, 626
Polynomial classifiers, 131
Polythetic algorithms, 479
Possibilistic algorithms, 432
Power function of a test, 592
Prediction based on groups, 401
Principle Component Analysis (PCA); *See* Karhunen–Loéve transform
Probabilistic clustering algorithms, 432
Projection pursuit, 220
Proximity dendrogram, 453
Proximity function
  between a point and a set
    average proximity function, 419
    maximum proximity function, 419
    minimum proximity function, 419
  between two sets
    average proximity function, 424
    maximum proximity function, 424
    mean proximity function, 424
    minimum proximity function, 424
Proximity graph, 466, 621
  dissimilarity graph, 466
  similarity graph, 466
Proximity matrix, 451
Proximity measure, 399
Pruning techniques, 120
Pseudolikelihood function, 378

Q

Quadrat analysis, 628
Quadric surface, 507
  hyperellipses, 507
  hyperparabolas, 507
Quickprop, 114

R

Radial basis function networks, 133
Radial distance, 509, *See also* distance between a point and a quadratic surface
Rand statistic, 598
Random field, 286
Random graph hypothesis, 595
Random label hypothesis, 596
Random position hypothesis, 595
Randomness hypothesis, 595, 624
Ratio scaled proximity matrices, 622
Reassignment procedure, 442
Receiver operating characteristics curve, 173
Recognition, 1
Recognition of handwritten, 256
Regions of influence, 549
Regularity hypothesis, 624
Relative criteria, 605–619
Relative edge consistence, 549
Relative neighborhood graph (RNG), 611, 613
Representatives
  hyperplane, 517
  point, 505–506
  quadratic surfaces, 507
Reproduction set, 534
Reproduction vector, 533
Resubstitution method, 387
Reward and punishment, 61
Right-tailed statistical test, 594
Risk or loss, 17
Robbins–Monro iteration, 556
Roundness ratio, 302
Run length nonuniformity, 278
Run percentage, 279

S

Sakoe and Chiba, 335
Saliency, 122
Sampling frame, 625
Sampling window, 624
Scan test, 628

Scatter matrices, 179
Second moment structure, 628
Segment modeling, 372
Self-affine, 307
Self-organizing maps (SOM), 559
Self-similarity, 303
Sequential backward selection, 184
Sequential clustering algorithms, 431, 433–440
Sequential decomposition, 630
Sequential forward selection, 185
Shape characterization, 294
Shell hypervolume, 620
Shell partition density, 620
Short run emphasis, 278
Short-time fourier transform, 251
Significance level, 167
Sigmoid functions, 104
Similarity matrix, 451
Similarity measure
  between points, 405
  between sets, 406
Simple sequential inhibition (SSI), 627
Simulated annealing, 579–580
  sweep, 579
Simulated annealing for clustering, 579–580
Sine transforms, 230
Single link, 455, 467
Singular value decomposition, 216
Skewness, 271
Softmax activation, 152
Spanning tree, 473
Sparse decomposition technique, 630–634
Spatial dependence matrix, 273
Speaker-dependent recognition, 329
Speaker-independent recognition, 329
Spectral representation, 216
Speech recognition, 560
Statistic, 592
Stirling numbers of the second kind, 430
Stochastic approximation, 68
Stochastic relaxation methods, 432
String patterns, 322
Structural graph tests, 628
Structural risk minimization, 196
Subgraph, 464, 465
  complete, 464
  connected, 464
Subsampling, 235
Subspace classification , 214
Sum of error squares estimation, 70
Supervised learning vector quantization, 560
Support vector machines, 77, 136, 139, 199
Surface density criterion, 623
System evaluation, 385

T

t-distribution, 170
t-test, 171
T-squared sampling tests, 630
Tabu search method, 187, 583
Tanimoto measure
  for continuous valued
  for discrete valued, 412
Template matching, 321
Templates, 321
Test statistic, 166
Texture, 259, 270
Tied-mixture densities, 372
Tiling algorithm, 102
Threshold dendrogram, 471
Threshold graph, 465
Topological ordering, 560
Total fuzzy average shell thickness, 620
Total variation, 617
Touching clusters, 547
Translation, 566
Tree-structured filter bank, 238
Triangular inequality, 405
Two-dimensional AR models, 289
Two-dimensional DFT, 229
Two-tailed statistical test, 593
Two-threshold sequential algorithmic scheme, 439

U

Uncertainty principle, 261
Undirected graphs, 464
Universal approximators, 137
Unsupervised learning, 397
Unweighted graphs, 464
Unweighted pair group method average (UPGMA), 457
Unweighted pair group method centroid (UPGMC), 457

V

Validation of clustering results, 399
Valley-seeking algorithms, 433, 576
Vapnik–Chernovenkis learning theory, 193
Variational similarity, 326
Vector quantization, 533
Vector quantizer, 534
  decoder, 535
  encoder, 534
Video coding, 321
Viterbi algorithm, 353
Viterbi reestimation, 369
Voronoi tessellation, 46

W

Ward's algorithm, 459
Wavelet packets, 252
Weighted graphs, 464
Weighted pair group method average (WPGMA), 457
Weighted pair group method centroid (WPGMC), 458
Weight sharing, 126
Well-formed functions, 115
Within-class scatter matrix, 179
Within scatter matrix, 532

X

Xie–Beni (XB) index, 617
XOR problem, 93

Z

Zernike moments, 284

Electrical Engineering / Computing /
Applied Mathematics / Pattern Recognition

PATTERN RECOGNITION

SECOND EDITION

SERGIOS THEODORIDIS • KONSTANTINOS KOUTROUMBAS

Pattern recognition is incredibly important in all automation, information handling and retrieval applications. The new edition of ***Pattern Recognition***, a text written by two of the field's leading experts, covers the entire spectrum of pattern recognition applications from an engineering perspective, examining topics from image analysis to speech recognition and communications. This thoroughly updated edition presents cutting-edge material on neural networks and highlights the latest developments in this growing field, including independent components and support vector machines. Developed through more than 10 years of teaching experience, ***Pattern Recognition*** is the most comprehensive reference available for both engineering students and practicing engineers.

Coverage Includes:

- Latest techniques in feature generation, including features based on Wavelets, Wavelet Packets, Fractals and a new section on Independent Component Analysis (ICA)
- All new sections on Support Vector Machines, Deformable Template Matching and a related appendix on Constrained Optimization
- Feature selection techniques
- Design of linear and non-linear classifiers, including Bayesian, Multilayer Perceptrons, Decision Trees and RBF networks
- Context-dependent classification, including Dynamic Programming and Hidden Markov Modeling techniques
- Classical approaches, as well as the most recent developments in clustering algorithms, such as fuzzy, possibilistic, morphological, genetic, and annealing techniques
- Coverage of numerous, diverse applications, including Image Analysis, Character Recognition, Medical Diagnosis, Speech Recognition, and Channel Equalization
- Numerous computer simulation examples, supporting the methods given in the book, available via the Web

Printed in the United States of America

ELSEVIER
ACADEMIC PRESS

www.academicpressbooks.com

ISBN 0-12-685875-6

90090

9 780126 858754