



introduction to

DATA MINING

SECOND EDITION

Pang-Ning Tan

Michael Steinbach

Vipin Kumar

Anuj Karpatne



Pearson



introduction to

DATA MINING

SECOND EDITION

Pang-Ning Tan Michael Steinbach Vipin Kumar Anuj Karpatne



Pearson

INTRODUCTION TO DATA MINING

INTRODUCTION TO DATA MINING

SECOND EDITION

PANG-NING TAN

Michigan State University

MICHAEL STEINBACH

University of Minnesota

ANUJ KARPATNE

University of Minnesota

VIPIN KUMAR

University of Minnesota



Pearson

330 Hudson Street, NY NY 10013

Director, Portfolio Management: Engineering, Computer Science & Global Editions: Julian Partridge

Specialist, Higher Ed Portfolio Management: Matt Goldstein

Portfolio Management Assistant: Meghan Jacoby

Managing Content Producer: Scott Disanno

Content Producer: Carole Snyder

Web Developer: Steve Wright

Rights and Permissions Manager: Ben Ferrini

Manufacturing Buyer, Higher Ed, Lake Side Communications Inc (LSC):
Maura Zaldivar-Garcia

Inventory Manager: Ann Lam

Product Marketing Manager: Yvonne Vannatta

Field Marketing Manager: Demetrius Hall

Marketing Assistant: Jon Bryant

Cover Designer: Joyce Wells, jWellsDesign

Full-Service Project Management: Chandrasekar Subramanian, SPi Global

Copyright ©2019 Pearson Education, Inc. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions department, please visit www.pearsonhighed.com/permissions/.

Many of the designations by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Library of Congress Cataloging-in-Publication Data on File

Names: Tan, Pang-Ning, author. | Steinbach, Michael, author. | Karpatne, Anuj, author. | Kumar, Vipin, 1956- author.

Title: *Introduction to Data Mining* / Pang-Ning Tan, Michigan State University, Michael Steinbach, University of Minnesota, Anuj Karpatne, University of Minnesota, Vipin Kumar, University of Minnesota.

Description: Second edition. | New York, NY : Pearson Education, [2019] | Includes bibliographical references and index.

Identifiers: LCCN 2017048641 | ISBN 9780133128901 | ISBN 0133128903

Subjects: LCSH: Data mining.

Classification: LCC QA76.9.D343 T35 2019 | DDC 006.3/12–dc23 LC record available at <https://lccn.loc.gov/2017048641>



1 18

ISBN-10: 0133128903

ISBN-13: 9780133128901

To our families ...

Preface to the Second Edition

Since the first edition, roughly 12 years ago, much has changed in the field of data analysis. The volume and variety of data being collected continues to increase, as has the rate (velocity) at which it is being collected and used to make decisions. Indeed, the term, Big Data, has been used to refer to the massive and diverse data sets now available. In addition, the term data science has been coined to describe an emerging area that applies tools and techniques from various fields, such as data mining, machine learning, statistics, and many others, to extract actionable insights from data, often big data.

The growth in data has created numerous opportunities for all areas of data analysis. The most dramatic developments have been in the area of predictive modeling, across a wide range of application domains. For instance, recent advances in neural networks, known as deep learning, have shown impressive results in a number of challenging areas, such as image classification, speech recognition, as well as text categorization and understanding. While not as dramatic, other areas, e.g., clustering, association analysis, and anomaly detection have also continued to advance. This new edition is in response to those advances.

Overview

As with the first edition, the second edition of the book provides a comprehensive introduction to data mining and is designed to be accessible and useful to students, instructors, researchers, and professionals. Areas

covered include data preprocessing, predictive modeling, association analysis, cluster analysis, anomaly detection, and avoiding false discoveries. The goal is to present fundamental concepts and algorithms for each topic, thus providing the reader with the necessary background for the application of data mining to real problems. As before, classification, association analysis and cluster analysis, are each covered in a pair of chapters. The introductory chapter covers basic concepts, representative algorithms, and evaluation techniques, while the more following chapter discusses advanced concepts and algorithms. As before, our objective is to provide the reader with a sound understanding of the foundations of data mining, while still covering many important advanced topics. Because of this approach, the book is useful both as a learning tool and as a reference.

To help readers better understand the concepts that have been presented, we provide an extensive set of examples, figures, and exercises. The solutions to the original exercises, which are already circulating on the web, will be made public. The exercises are mostly unchanged from the last edition, with the exception of new exercises in the chapter on avoiding false discoveries. New exercises for the other chapters and their solutions will be available to instructors via the web. Bibliographic notes are included at the end of each chapter for readers who are interested in more advanced topics, historically important papers, and recent trends. These have also been significantly updated. The book also contains a comprehensive subject and author index.

What is New in the Second Edition?

Some of the most significant improvements in the text have been in the two chapters on classification. The introductory chapter uses the decision tree classifier for illustration, but the discussion on many topics—those that apply

across all classification approaches—has been greatly expanded and clarified, including topics such as overfitting, underfitting, the impact of training size, model complexity, model selection, and common pitfalls in model evaluation. Almost every section of the advanced classification chapter has been significantly updated. The material on Bayesian networks, support vector machines, and artificial neural networks has been significantly expanded. We have added a separate section on deep networks to address the current developments in this area. The discussion of evaluation, which occurs in the section on imbalanced classes, has also been updated and improved.

The changes in association analysis are more localized. We have completely reworked the section on the evaluation of association patterns (introductory chapter), as well as the sections on sequence and graph mining (advanced chapter). Changes to cluster analysis are also localized. The introductory chapter added the K-means initialization technique and an updated the discussion of cluster evaluation. The advanced clustering chapter adds a new section on spectral graph clustering. Anomaly detection has been greatly revised and expanded. Existing approaches—statistical, nearest neighbor/density-based, and clustering based—have been retained and updated, while new approaches have been added: reconstruction-based, one-class classification, and information-theoretic. The reconstruction-based approach is illustrated using autoencoder networks that are part of the deep learning paradigm. The data chapter has been updated to include discussions of mutual information and kernel-based techniques.

The last chapter, which discusses how to avoid false discoveries and produce valid results, is completely new, and is novel among other contemporary textbooks on data mining. It supplements the discussions in the other chapters with a discussion of the statistical concepts (statistical significance, p-values, false discovery rate, permutation testing, etc.) relevant to avoiding spurious results, and then illustrates these concepts in the context of data

mining techniques. This chapter addresses the increasing concern over the validity and reproducibility of results obtained from data analysis. The addition of this last chapter is a recognition of the importance of this topic and an acknowledgment that a deeper understanding of this area is needed for those analyzing data.

The data exploration chapter has been deleted, as have the appendices, from the print edition of the book, but will remain available on the web. A new appendix provides a brief discussion of scalability in the context of big data.

To the Instructor

As a textbook, this book is suitable for a wide range of students at the advanced undergraduate or graduate level. Since students come to this subject with diverse backgrounds that may not include extensive knowledge of statistics or databases, our book requires minimal prerequisites. No database knowledge is needed, and we assume only a modest background in statistics or mathematics, although such a background will make for easier going in some sections. As before, the book, and more specifically, the chapters covering major data mining topics, are designed to be as self-contained as possible. Thus, the order in which topics can be covered is quite flexible. The core material is covered in chapters 2 (data), 3 (classification), 5 (association analysis), 7 (clustering), and 9 (anomaly detection). We recommend at least a cursory coverage of Chapter 10 (Avoiding False Discoveries) to instill in students some caution when interpreting the results of their data analysis. Although the introductory data chapter (2) should be covered first, the basic classification (3), association analysis (5), and clustering chapters (7), can be covered in any order. Because of the relationship of anomaly detection (9) to classification (3) and clustering (7), these chapters should precede Chapter 9.

Various topics can be selected from the advanced classification, association analysis, and clustering chapters (4, 6, and 8, respectively) to fit the schedule and interests of the instructor and students. We also advise that the lectures be augmented by projects or practical exercises in data mining. Although they are time consuming, such hands-on assignments greatly enhance the value of the course.

Support Materials

Support materials available to all readers of this book are available at
<http://www-users.cs.umn.edu/~kumar/dmbook>.

- PowerPoint lecture slides
- Suggestions for student projects
- Data mining resources, such as algorithms and data sets
- Online tutorials that give step-by-step examples for selected data mining techniques described in the book using actual data sets and data analysis software

Additional support materials, including solutions to exercises, are available only to instructors adopting this textbook for classroom use. The book's resources will be mirrored at www.pearsonhighered.com/cs-resources. Comments and suggestions, as well as reports of errors, can be sent to the authors through dmbook@cs.umn.edu.

Acknowledgments

Many people contributed to the first and second editions of the book. We begin by acknowledging our families to whom this book is dedicated. Without their patience and support, this project would have been impossible.

We would like to thank the current and former students of our data mining groups at the University of Minnesota and Michigan State for their contributions. Eui-Hong (Sam) Han and Mahesh Joshi helped with the initial data mining classes. Some of the exercises and presentation slides that they created can be found in the book and its accompanying slides. Students in our data mining groups who provided comments on drafts of the book or who contributed in other ways include Shyam Boriah, Haibin Cheng, Varun Chandola, Eric Eilertson, Levent Ertöz, Jing Gao, Rohit Gupta, Sridhar Iyer, Jung-Eun Lee, Benjamin Mayer, Aysel Ozgur, Uygar Oztekin, Gaurav Pandey, Kashif Riaz, Jerry Scripps, Gyorgy Simon, Hui Xiong, Jieping Ye, and Pusheng Zhang. We would also like to thank the students of our data mining classes at the University of Minnesota and Michigan State University who worked with early drafts of the book and provided invaluable feedback. We specifically note the helpful suggestions of Bernardo Craemer, Arifin Ruslim, Jamshid Vayghan, and Yu Wei.

Joydeep Ghosh (University of Texas) and Sanjay Ranka (University of Florida) class tested early versions of the book. We also received many useful suggestions directly from the following UT students: Pankaj Adhikari, Rajiv Bhatia, Frederic Bosche, Arindam Chakraborty, Meghana Deodhar, Chris Everson, David Gardner, Saad Godil, Todd Hay, Clint Jones, Ajay Joshi, Joonsoo Lee, Yue Luo, Anuj Nanavati, Tyler Olsen, Sunyoung Park, Aashish Phansalkar, Geoff Prewett, Michael Ryoo, Daryl Shannon, and Mei Yang.

Ronald Kostoff (ONR) read an early version of the clustering chapter and offered numerous suggestions. George Karypis provided invaluable LATEX assistance in creating an author index. Irene Moultsas also provided

assistance with LATEX and reviewed some of the appendices. Musetta Steinbach was very helpful in finding errors in the figures.

We would like to acknowledge our colleagues at the University of Minnesota and Michigan State who have helped create a positive environment for data mining research. They include Arindam Banerjee, Dan Boley, Joyce Chai, Anil Jain, Ravi Janardan, Rong Jin, George Karypis, Claudia Neuhauser, Haesun Park, William F. Punch, György Simon, Shashi Shekhar, and Jaideep Srivastava. The collaborators on our many data mining projects, who also have our gratitude, include Ramesh Agrawal, Maneesh Bhargava, Steve Cannon, Alok Choudhary, Imme Ebert-Uphoff, Auroop Ganguly, Piet C. de Groen, Fran Hill, Yongdae Kim, Steve Klooster, Kerry Long, Nihar Mahapatra, Rama Nemani, Nikunj Oza, Chris Potter, Lisiane Pruinelli, Nagiza Samatova, Jonathan Shapiro, Kevin Silverstein, Brian Van Ness, Bonnie Westra, Nevin Young, and Zhi-Li Zhang.

The departments of Computer Science and Engineering at the University of Minnesota and Michigan State University provided computing resources and a supportive environment for this project. ARDA, ARL, ARO, DOE, NASA, NOAA, and NSF provided research support for Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar. In particular, Kamal Abdali, Mitra Basu, Dick Brackney, Jagdish Chandra, Joe Coughlan, Michael Coyle, Stephen Davis, Frederica Darema, Richard Hirsch, Chandrika Kamath, Tsengdar Lee, Raju Namburu, N. Radhakrishnan, James Sidoran, Sylvia Spengler, Bhavani Thuraisingham, Walt Tiernin, Maria Zemankova, Aidong Zhang, and Xiaodong Zhang have been supportive of our research in data mining and high-performance computing.

It was a pleasure working with the helpful staff at Pearson Education. In particular, we would like to thank Matt Goldstein, Kathy Smith, Carole Snyder,

and Joyce Wells. We would also like to thank George Nichols, who helped with the art work and Paul Anagnostopoulos, who provided LATEX support.

We are grateful to the following Pearson reviewers: Leman Akoglu (Carnegie Mellon University), Chien-Chung Chan (University of Akron), Zhengxin Chen (University of Nebraska at Omaha), Chris Clifton (Purdue University), Joydeep Ghosh (University of Texas, Austin), Nazli Goharian (Illinois Institute of Technology), J. Michael Hardin (University of Alabama), Jingrui He (Arizona State University), James Hearne (Western Washington University), Hillol Kargupta (University of Maryland, Baltimore County and Agnik, LLC), Eamonn Keogh (University of California-Riverside), Bing Liu (University of Illinois at Chicago), Mariofanna Milanova (University of Arkansas at Little Rock), Srinivasan Parthasarathy (Ohio State University), Zbigniew W. Ras (University of North Carolina at Charlotte), Xintao Wu (University of North Carolina at Charlotte), and Mohammed J. Zaki (Rensselaer Polytechnic Institute).

Over the years since the first edition, we have also received numerous comments from readers and students who have pointed out typos and various other issues. We are unable to mention these individuals by name, but their input is much appreciated and has been taken into account for the second edition.

Contents

Preface to the Second Edition v □

1 Introduction 1 □

1.1 What Is Data Mining? 4 □

1.2 Motivating Challenges 5 □

1.3 The Origins of Data Mining 7 □

1.4 Data Mining Tasks 9 □

1.5 Scope and Organization of the Book 13 □

1.6 Bibliographic Notes 15 □

1.7 Exercises 21 □

2 Data 23 □

2.1 Types of Data 26 □

2.1.1 Attributes and Measurement 27 □

2.1.2 Types of Data Sets 34 □

2.2 Data Quality 42 □

2.2.1 Measurement and Data Collection Issues 42 □

2.2.2 Issues Related to Applications 49 □

2.3 Data Preprocessing 50 □

2.3.1 Aggregation 51 □

2.3.2 Sampling 52 □

2.3.3 Dimensionality Reduction 56 □

2.3.4 Feature Subset Selection 58 □

2.3.5 Feature Creation 61 □

2.3.6 Discretization and Binarization 63 □

2.3.7 Variable Transformation 69 □

2.4 Measures of Similarity and Dissimilarity 71 □

2.4.1 Basics 72 □

2.4.2 Similarity and Dissimilarity between Simple Attributes 74 □

2.4.3 Dissimilarities between Data Objects 76 □

2.4.4 Similarities between Data Objects 78 □

2.4.5 Examples of Proximity Measures 79 □

2.4.6 Mutual Information 88 □

2.4.7 Kernel Functions* 90 □

2.4.8 Bregman Divergence* 94 □

2.4.9 Issues in Proximity Calculation 96 □

2.4.10 Selecting the Right Proximity Measure 98 □

2.5 Bibliographic Notes 100 □

2.6 Exercises 105 □

3 Classification: Basic Concepts and Techniques 113 □

- 3.1 Basic Concepts 114** □
- 3.2 General Framework for Classification 117** □
- 3.3 Decision Tree Classifier 119** □
 - 3.3.1 A Basic Algorithm to Build a Decision Tree 121** □
 - 3.3.2 Methods for Expressing Attribute Test Conditions 124** □
 - 3.3.3 Measures for Selecting an Attribute Test Condition 127** □
 - 3.3.4 Algorithm for Decision Tree Induction 136** □
 - 3.3.5 Example Application: Web Robot Detection 138** □
 - 3.3.6 Characteristics of Decision Tree Classifiers 140** □
- 3.4 Model Overfitting 147** □
 - 3.4.1 Reasons for Model Overfitting 149** □
- 3.5 Model Selection 156** □
 - 3.5.1 Using a Validation Set 156** □
 - 3.5.2 Incorporating Model Complexity 157** □
 - 3.5.3 Estimating Statistical Bounds 162** □
 - 3.5.4 Model Selection for Decision Trees 162** □
- 3.6 Model Evaluation 164** □
 - 3.6.1 Holdout Method 165** □
 - 3.6.2 Cross-Validation 165** □
- 3.7 Presence of Hyper-parameters 168** □
 - 3.7.1 Hyper-parameter Selection 168** □

3.7.2 Nested Cross-Validation 170 □

3.8 Pitfalls of Model Selection and Evaluation 172 □

3.8.1 Overlap between Training and Test Sets 172 □

3.8.2 Use of Validation Error as Generalization Error 172 □

3.9 Model Comparison* 173 □

3.9.1 Estimating the Confidence Interval for Accuracy 174 □

3.9.2 Comparing the Performance of Two Models 175 □

3.10 Bibliographic Notes 176 □

3.11 Exercises 185 □

4 Classification: Alternative Techniques 193 □

4.1 Types of Classifiers 193 □

4.2 Rule-Based Classifier 195 □

4.2.1 How a Rule-Based Classifier Works 197 □

4.2.2 Properties of a Rule Set 198 □

4.2.3 Direct Methods for Rule Extraction 199 □

4.2.4 Indirect Methods for Rule Extraction 204 □

4.2.5 Characteristics of Rule-Based Classifiers 206 □

4.3 Nearest Neighbor Classifiers 208 □

4.3.1 Algorithm 209 □

4.3.2 Characteristics of Nearest Neighbor Classifiers 210 □

4.4 Naïve Bayes Classifier 212 

4.4.1 Basics of Probability Theory 213 

4.4.2 Naïve Bayes Assumption 218 

4.5 Bayesian Networks 227 

4.5.1 Graphical Representation 227 

4.5.2 Inference and Learning 233 

4.5.3 Characteristics of Bayesian Networks 242 

4.6 Logistic Regression 243 

4.6.1 Logistic Regression as a Generalized Linear Model 244 

4.6.2 Learning Model Parameters 245 

4.6.3 Characteristics of Logistic Regression 248 

4.7 Artificial Neural Network (ANN) 249 

4.7.1 Perceptron 250 

4.7.2 Multi-layer Neural Network 254 

4.7.3 Characteristics of ANN 261 

4.8 Deep Learning 262 

4.8.1 Using Synergistic Loss Functions 263 

4.8.2 Using Responsive Activation Functions 266 

4.8.3 Regularization 268 

4.8.4 Initialization of Model Parameters 271 

4.8.5 Characteristics of Deep Learning 275 

- 4.9 Support Vector Machine (SVM) 276** 

 - 4.9.1 Margin of a Separating Hyperplane 276** 
 - 4.9.2 Linear SVM 278** 
 - 4.9.3 Soft-margin SVM 284** 
 - 4.9.4 Nonlinear SVM 290** 
 - 4.9.5 Characteristics of SVM 294** 

- 4.10 Ensemble Methods 296** 

 - 4.10.1 Rationale for Ensemble Method 297** 
 - 4.10.2 Methods for Constructing an Ensemble Classifier 297** 
 - 4.10.3 Bias-Variance Decomposition 300** 
 - 4.10.4 Bagging 302** 
 - 4.10.5 Boosting 305** 
 - 4.10.6 Random Forests 310** 
 - 4.10.7 Empirical Comparison among Ensemble Methods 312** 

- 4.11 Class Imbalance Problem 313** 

 - 4.11.1 Building Classifiers with Class Imbalance 314** 
 - 4.11.2 Evaluating Performance with Class Imbalance 318** 
 - 4.11.3 Finding an Optimal Score Threshold 322** 
 - 4.11.4 Aggregate Evaluation of Performance 323** 

- 4.12 Multiclass Problem 330** 
- 4.13 Bibliographic Notes 333** 

4.14 Exercises 345 □

5 Association Analysis: Basic Concepts and Algorithms 357 □

5.1 Preliminaries 358 □

5.2 Frequent Itemset Generation 362 □

5.2.1 The *Apriori* Principle 363 □

5.2.2 Frequent Itemset Generation in the *Apriori* Algorithm 364 □

5.2.3 Candidate Generation and Pruning 368 □

5.2.4 Support Counting 373 □

5.2.5 Computational Complexity 377 □

5.3 Rule Generation 380 □

5.3.1 Confidence-Based Pruning 380 □

5.3.2 Rule Generation in *Apriori* Algorithm 381 □

5.3.3 An Example: Congressional Voting Records 382 □

5.4 Compact Representation of Frequent Itemsets 384 □

5.4.1 Maximal Frequent Itemsets 384 □

5.4.2 Closed Itemsets 386 □

5.5 Alternative Methods for Generating Frequent Itemsets* 389 □

5.6 FP-Growth Algorithm* 393 □

5.6.1 FP-Tree Representation 394 □

5.6.2 Frequent Itemset Generation in FP-Growth Algorithm 397 □

5.7 Evaluation of Association Patterns 401 □

5.7.1 Objective Measures of Interestingness 402 □

5.7.2 Measures beyond Pairs of Binary Variables 414 □

5.7.3 Simpson's Paradox 416 □

5.8 Effect of Skewed Support Distribution 418 □

5.9 Bibliographic Notes 424 □

5.10 Exercises 438 □

6 Association Analysis: Advanced Concepts 451 □

6.1 Handling Categorical Attributes 451 □

6.2 Handling Continuous Attributes 454 □

6.2.1 Discretization-Based Methods 454 □

6.2.2 Statistics-Based Methods 458 □

6.2.3 Non-discretization Methods 460 □

6.3 Handling a Concept Hierarchy 462 □

6.4 Sequential Patterns 464 □

6.4.1 Preliminaries 465 □

6.4.2 Sequential Pattern Discovery 468 □

6.4.3 Timing Constraints* 473 □

6.4.4 Alternative Counting Schemes* 477 □

6.5 Subgraph Patterns 479 □

6.5.1 Preliminaries 480 □

6.5.2 Frequent Subgraph Mining 483 □

6.5.3 Candidate Generation 487 □

6.5.4 Candidate Pruning 493 □

6.5.5 Support Counting 493 □

6.6 Infrequent Patterns* 493 □

6.6.1 Negative Patterns 494 □

6.6.2 Negatively Correlated Patterns 495 □

6.6.3 Comparisons among Infrequent Patterns, Negative Patterns, and Negatively Correlated Patterns 496 □

6.6.4 Techniques for Mining Interesting Infrequent Patterns 498 □

6.6.5 Techniques Based on Mining Negative Patterns 499 □

6.6.6 Techniques Based on Support Expectation 501 □

6.7 Bibliographic Notes 505 □

6.8 Exercises 510 □

7 Cluster Analysis: Basic Concepts and Algorithms 525 □

7.1 Overview 528 □

7.1.1 What Is Cluster Analysis? 528 □

7.1.2 Different Types of Clusterings 529 □

7.1.3 Different Types of Clusters 531 □

7.2 K-means 534 □

7.2.1 The Basic K-means Algorithm 535 □

7.2.2 K-means: Additional Issues 544 □

7.2.3 Bisecting K-means 547 □

7.2.4 K-means and Different Types of Clusters 548 □

7.2.5 Strengths and Weaknesses 549 □

7.2.6 K-means as an Optimization Problem 549 □

7.3 Agglomerative Hierarchical Clustering 554 □

7.3.1 Basic Agglomerative Hierarchical Clustering Algorithm 555 □

7.3.2 Specific Techniques 557 □

7.3.3 The Lance-Williams Formula for Cluster Proximity 562 □

7.3.4 Key Issues in Hierarchical Clustering 563 □

7.3.5 Outliers 564 □

7.3.6 Strengths and Weaknesses 565 □

7.4 DBSCAN 565 □

7.4.1 Traditional Density: Center-Based Approach 565 □

7.4.2 The DBSCAN Algorithm 567 □

7.4.3 Strengths and Weaknesses 569 □

7.5 Cluster Evaluation 571 □

7.5.1 Overview 571 □

7.5.2 Unsupervised Cluster Evaluation Using Cohesion and Separation 574 □

7.5.3 Unsupervised Cluster Evaluation Using the Proximity Matrix 582 □

7.5.4 Unsupervised Evaluation of Hierarchical Clustering 585 □

7.5.5 Determining the Correct Number of Clusters 587 □

7.5.6 Clustering Tendency 588 □

7.5.7 Supervised Measures of Cluster Validity 589 □

7.5.8 Assessing the Significance of Cluster Validity Measures 594 □

7.5.9 Choosing a Cluster Validity Measure 596 □

7.6 Bibliographic Notes 597 □

7.7 Exercises 603 □

8 Cluster Analysis: Additional Issues and Algorithms 613 □

8.1 Characteristics of Data, Clusters, and Clustering Algorithms 614 □

8.1.1 Example: Comparing K-means and DBSCAN 614 □

8.1.2 Data Characteristics 615 □

8.1.3 Cluster Characteristics 617 □

8.1.4 General Characteristics of Clustering Algorithms 619 □

8.2 Prototype-Based Clustering 621 □

8.2.1 Fuzzy Clustering 621 □

8.2.2 Clustering Using Mixture Models 627 □

8.2.3 Self-Organizing Maps (SOM) 637 

8.3 Density-Based Clustering 644 

8.3.1 Grid-Based Clustering 644 

8.3.2 Subspace Clustering 648 

8.3.3 DENCLUE: A Kernel-Based Scheme for Density-Based Clustering 652 

8.4 Graph-Based Clustering 656 

8.4.1 Sparsification 657 

8.4.2 Minimum Spanning Tree (MST) Clustering 658 

8.4.3 OPOSSUM: Optimal Partitioning of Sparse Similarities Using METIS 659 

8.4.4 Chameleon: Hierarchical Clustering with Dynamic Modeling 660 

8.4.5 Spectral Clustering 666 

8.4.6 Shared Nearest Neighbor Similarity 673 

8.4.7 The Jarvis-Patrick Clustering Algorithm 676 

8.4.8 SNN Density 678 

8.4.9 SNN Density-Based Clustering 679 

8.5 Scalable Clustering Algorithms 681 

8.5.1 Scalability: General Issues and Approaches 681 

8.5.2 BIRCH 684 

8.5.3 CURE 686 

8.6 Which Clustering Algorithm? 690 □

8.7 Bibliographic Notes 693 □

8.8 Exercises 699 □

9 Anomaly Detection 703 □

9.1 Characteristics of Anomaly Detection Problems 705 □

9.1.1 A Definition of an Anomaly 705 □

9.1.2 Nature of Data 706 □

9.1.3 How Anomaly Detection is Used 707 □

9.2 Characteristics of Anomaly Detection Methods 708 □

9.3 Statistical Approaches 710 □

9.3.1 Using Parametric Models 710 □

9.3.2 Using Non-parametric Models 714 □

9.3.3 Modeling Normal and Anomalous Classes 715 □

9.3.4 Assessing Statistical Significance 717 □

9.3.5 Strengths and Weaknesses 718 □

9.4 Proximity-based Approaches 719 □

9.4.1 Distance-based Anomaly Score 719 □

9.4.2 Density-based Anomaly Score 720 □

9.4.3 Relative Density-based Anomaly Score 722 □

9.4.4 Strengths and Weaknesses 723 □

9.5 Clustering-based Approaches 724 □

9.5.1 Finding Anomalous Clusters 724 □

9.5.2 Finding Anomalous Instances 725 □

9.5.3 Strengths and Weaknesses 728 □

9.6 Reconstruction-based Approaches 728 □

9.6.1 Strengths and Weaknesses 731 □

9.7 One-class Classification 732 □

9.7.1 Use of Kernels 733 □

9.7.2 The Origin Trick 734 □

9.7.3 Strengths and Weaknesses 738 □

9.8 Information Theoretic Approaches 738 □

9.8.1 Strengths and Weaknesses 740 □

9.9 Evaluation of Anomaly Detection 740 □

9.10 Bibliographic Notes 742 □

9.11 Exercises 749 □

10 Avoiding False Discoveries 755 □

10.1 Preliminaries: Statistical Testing 756 □

10.1.1 Significance Testing 756 □

10.1.2 Hypothesis Testing 761 □

10.1.3 Multiple Hypothesis Testing 767 □

10.1.4 Pitfalls in Statistical Testing 776 □

10.2 Modeling Null and Alternative Distributions 778 □

10.2.1 Generating Synthetic Data Sets 781 □

10.2.2 Randomizing Class Labels 782 □

10.2.3 Resampling Instances 782 □

10.2.4 Modeling the Distribution of the Test Statistic 783 □

10.3 Statistical Testing for Classification 783 □

10.3.1 Evaluating Classification Performance 783 □

10.3.2 Binary Classification as Multiple Hypothesis Testing 785 □

10.3.3 Multiple Hypothesis Testing in Model Selection 786 □

10.4 Statistical Testing for Association Analysis 787 □

10.4.1 Using Statistical Models 788 □

10.4.2 Using Randomization Methods 794 □

10.5 Statistical Testing for Cluster Analysis 795 □

10.5.1 Generating a Null Distribution for Internal Indices 796 □

10.5.2 Generating a Null Distribution for External Indices 798 □

10.5.3 Enrichment 798 □

10.6 Statistical Testing for Anomaly Detection 800 □

10.7 Bibliographic Notes 803 □

10.8 Exercises 808 □

Author Index 816 □

Subject Index 829 

Copyright Permissions 839 

1 Introduction

*Rapid advances in data collection and storage technology, coupled with the ease with which data can be generated and disseminated, have triggered the explosive growth of data, leading to the current age of **big data**. Deriving actionable insights from these large data sets is increasingly important in decision making across almost all areas of society, including business and industry; science and engineering; medicine and biotechnology; and government and individuals.*

However, the amount of data (volume), its complexity (variety), and the rate at which it is being collected and processed (velocity) have simply become too great for humans to analyze unaided. Thus, there is a great need for automated tools for extracting useful information from the big data despite the challenges posed by its enormity and diversity.

Data mining blends traditional data analysis methods with sophisticated algorithms for processing this abundance of data. In this introductory chapter, we present an overview of data mining and outline the key topics to be covered in this book. We start with a

description of some applications that require more advanced techniques for data analysis.

Business and Industry Point-of-sale data collection (bar code scanners, radio frequency identification (RFID), and smart card technology) have allowed retailers to collect up-to-the-minute data about customer purchases at the checkout counters of their stores. Retailers can utilize this information, along with other business-critical data, such as web server logs from e-commerce websites and customer service records from call centers, to help them better understand the needs of their customers and make more informed business decisions.

Data mining techniques can be used to support a wide range of business intelligence applications, such as customer profiling, targeted marketing, workflow management, store layout, fraud detection, and automated buying and selling. An example of the last application is high-speed stock trading, where decisions on buying and selling have to be made in less than a second using data about financial transactions. Data mining can also help retailers answer important business questions, such as “Who are the most profitable customers?” “What products can be cross-sold or up-sold?” and “What is the revenue outlook of the company for next year?” These questions have inspired the development of such data mining techniques as association analysis ([Chapters 5](#) and [6](#)).

As the Internet continues to revolutionize the way we interact and make decisions in our everyday lives, we are generating massive amounts of data about our online experiences, e.g., web browsing, messaging, and posting on social networking websites. This has opened several opportunities for business applications that use web data. For example, in the e-commerce sector, data about our online viewing or shopping preferences can be used to

provide personalized recommendations of products. Data mining also plays a prominent role in supporting several other Internet-based services, such as filtering spam messages, answering search queries, and suggesting social updates and connections. The large corpus of text, images, and videos available on the Internet has enabled a number of advancements in data mining methods, including deep learning, which is discussed in [Chapter 4](#). These developments have led to great advances in a number of applications, such as object recognition, natural language translation, and autonomous driving.

Another domain that has undergone a rapid big data transformation is the use of mobile sensors and devices, such as smart phones and wearable computing devices. With better sensor technologies, it has become possible to collect a variety of information about our physical world using low-cost sensors embedded on everyday objects that are connected to each other, termed the Internet of Things (IOT). This deep integration of physical sensors in digital systems is beginning to generate large amounts of diverse and distributed data about our environment, which can be used for designing convenient, safe, and energy-efficient home systems, as well as for urban planning of smart cities.

Medicine, Science, and Engineering Researchers in medicine, science, and engineering are rapidly accumulating data that is key to significant new discoveries. For example, as an important step toward improving our understanding of the Earth's climate system, NASA has deployed a series of Earth-orbiting satellites that continuously generate global observations of the land surface, oceans, and atmosphere. However, because of the size and spatio-temporal nature of the data, traditional methods are often not suitable for analyzing these data sets. Techniques developed in data mining can aid Earth scientists in answering questions such as the following: "What is the relationship between the frequency and intensity of ecosystem disturbances

such as droughts and hurricanes to global warming?" "How is land surface precipitation and temperature affected by ocean surface temperature?" and "How well can we predict the beginning and end of the growing season for a region?"

As another example, researchers in molecular biology hope to use the large amounts of genomic data to better understand the structure and function of genes. In the past, traditional methods in molecular biology allowed scientists to study only a few genes at a time in a given experiment. Recent breakthroughs in microarray technology have enabled scientists to compare the behavior of thousands of genes under various situations. Such comparisons can help determine the function of each gene, and perhaps isolate the genes responsible for certain diseases. However, the noisy, high-dimensional nature of data requires new data analysis methods. In addition to analyzing gene expression data, data mining can also be used to address other important biological challenges such as protein structure prediction, multiple sequence alignment, the modeling of biochemical pathways, and phylogenetics.

Another example is the use of data mining techniques to analyze electronic health record (EHR) data, which has become increasingly available. Not very long ago, studies of patients required manually examining the physical records of individual patients and extracting very specific pieces of information pertinent to the particular question being investigated. EHRs allow for a faster and broader exploration of such data. However, there are significant challenges since the observations on any one patient typically occur during their visits to a doctor or hospital and only a small number of details about the health of the patient are measured during any particular visit.

Currently, EHR analysis focuses on simple types of data, e.g., a patient's blood pressure or the diagnosis code of a disease. However, large amounts of

more complex types of medical data are also being collected, such as electrocardiograms (ECGs) and neuroimages from magnetic resonance imaging (MRI) or functional Magnetic Resonance Imaging (fMRI). Although challenging to analyze, this data also provides vital information about patients. Integrating and analyzing such data, with traditional EHR and genomic data is one of the capabilities needed to enable precision medicine, which aims to provide more personalized patient care.

1.1 What Is Data Mining?

Data mining is the process of automatically discovering useful information in large data repositories. Data mining techniques are deployed to scour large data sets in order to find novel and useful patterns that might otherwise remain unknown. They also provide the capability to predict the outcome of a future observation, such as the amount a customer will spend at an online or a brick-and-mortar store.

Not all information discovery tasks are considered to be data mining. Examples include queries, e.g., looking up individual records in a database or finding web pages that contain a particular set of keywords. This is because such tasks can be accomplished through simple interactions with a database management system or an information retrieval system. These systems rely on traditional computer science techniques, which include sophisticated indexing structures and query processing algorithms, for efficiently organizing and retrieving information from large data repositories. Nonetheless, data mining techniques have been used to enhance the performance of such systems by improving the quality of the search results based on their relevance to the input queries.

Data Mining and Knowledge Discovery in Databases

Data mining is an integral part of **knowledge discovery in databases (KDD)**, which is the overall process of converting raw data into useful information, as shown in [Figure 1.1](#). This process consists of a series of steps, from data preprocessing to postprocessing of data mining results.

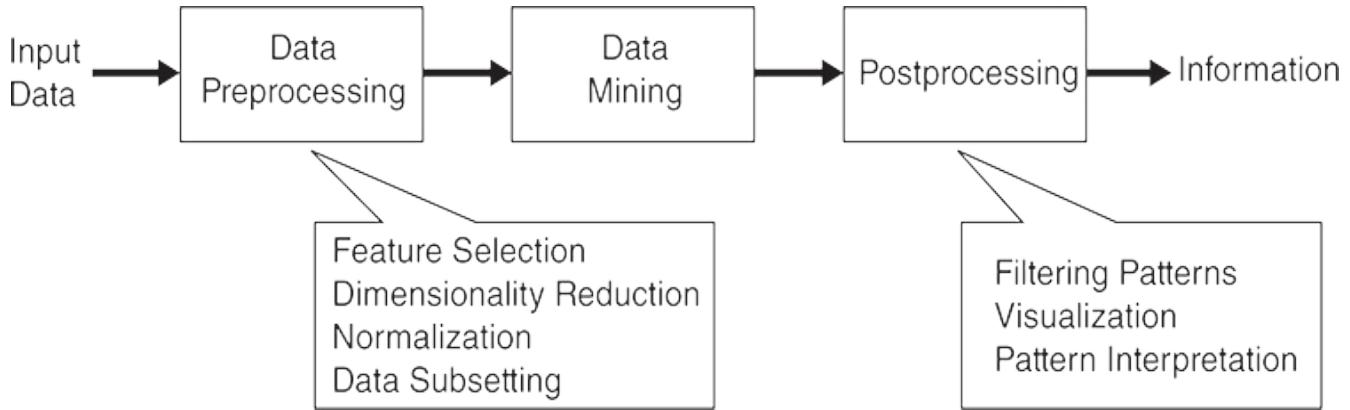


Figure 1.1.

The process of knowledge discovery in databases (KDD).

The input data can be stored in a variety of formats (flat files, spreadsheets, or relational tables) and may reside in a centralized data repository or be distributed across multiple sites. The purpose of **preprocessing** is to transform the raw input data into an appropriate format for subsequent analysis. The steps involved in data preprocessing include fusing data from multiple sources, cleaning data to remove noise and duplicate observations, and selecting records and features that are relevant to the data mining task at hand. Because of the many ways data can be collected and stored, data preprocessing is perhaps the most laborious and time-consuming step in the overall knowledge discovery process.

“Closing the loop” is a phrase often used to refer to the process of integrating data mining results into decision support systems. For example, in business applications, the insights offered by data mining results can be integrated with campaign management tools so that effective marketing promotions can be conducted and tested. Such integration requires a **postprocessing** step to ensure that only valid and useful results are incorporated into the decision support system. An example of postprocessing is visualization, which allows analysts to explore the data and the data mining results from a variety of viewpoints. Hypothesis testing methods can also be applied during

postprocessing to eliminate spurious data mining results. (See [Chapter 10](#).)

1.2 Motivating Challenges

As mentioned earlier, traditional data analysis techniques have often encountered practical difficulties in meeting the challenges posed by big data applications. The following are some of the specific challenges that motivated the development of data mining.

Scalability

Because of advances in data generation and collection, data sets with sizes of terabytes, petabytes, or even exabytes are becoming common. If data mining algorithms are to handle these massive data sets, they must be scalable. Many data mining algorithms employ special search strategies to handle exponential search problems. Scalability may also require the implementation of novel data structures to access individual records in an efficient manner. For instance, out-of-core algorithms may be necessary when processing data sets that cannot fit into main memory. Scalability can also be improved by using sampling or developing parallel and distributed algorithms. A general overview of techniques for scaling up data mining algorithms is given in Appendix F.

High Dimensionality

It is now common to encounter data sets with hundreds or thousands of attributes instead of the handful common a few decades ago. In bioinformatics, progress in microarray technology has produced gene expression data involving thousands of features. Data sets with temporal or spatial components also tend to have high dimensionality. For example,

consider a data set that contains measurements of temperature at various locations. If the temperature measurements are taken repeatedly for an extended period, the number of dimensions (features) increases in proportion to the number of measurements taken. Traditional data analysis techniques that were developed for low-dimensional data often do not work well for such high-dimensional data due to issues such as curse of dimensionality (to be discussed in [Chapter 2](#)). Also, for some data analysis algorithms, the computational complexity increases rapidly as the dimensionality (the number of features) increases.

Heterogeneous and Complex Data

Traditional data analysis methods often deal with data sets containing attributes of the same type, either continuous or categorical. As the role of data mining in business, science, medicine, and other fields has grown, so has the need for techniques that can handle heterogeneous attributes. Recent years have also seen the emergence of more complex data objects.

Examples of such non-traditional types of data include web and social media data containing text, hyperlinks, images, audio, and videos; DNA data with sequential and three-dimensional structure; and climate data that consists of measurements (temperature, pressure, etc.) at various times and locations on the Earth's surface. Techniques developed for mining such complex objects should take into consideration relationships in the data, such as temporal and spatial autocorrelation, graph connectivity, and parent-child relationships between the elements in semi-structured text and XML documents.

Data Ownership and Distribution

Sometimes, the data needed for an analysis is not stored in one location or owned by one organization. Instead, the data is geographically distributed among resources belonging to multiple entities. This requires the development

of distributed data mining techniques. The key challenges faced by distributed data mining algorithms include the following: (1) how to reduce the amount of communication needed to perform the distributed computation, (2) how to effectively consolidate the data mining results obtained from multiple sources, and (3) how to address data security and privacy issues.

Non-traditional Analysis

The traditional statistical approach is based on a hypothesize-and-test paradigm. In other words, a hypothesis is proposed, an experiment is designed to gather the data, and then the data is analyzed with respect to the hypothesis. Unfortunately, this process is extremely labor-intensive. Current data analysis tasks often require the generation and evaluation of thousands of hypotheses, and consequently, the development of some data mining techniques has been motivated by the desire to automate the process of hypothesis generation and evaluation. Furthermore, the data sets analyzed in data mining are typically not the result of a carefully designed experiment and often represent opportunistic samples of the data, rather than random samples.

1.3 The Origins of Data Mining

While data mining has traditionally been viewed as an intermediate process within the KDD framework, as shown in [Figure 1.1](#), it has emerged over the years as an academic field within computer science, focusing on all aspects of KDD, including data preprocessing, mining, and postprocessing. Its origin can be traced back to the late 1980s, following a series of workshops organized on the topic of knowledge discovery in databases. The workshops brought together researchers from different disciplines to discuss the challenges and opportunities in applying computational techniques to extract actionable knowledge from large databases. The workshops quickly grew into hugely popular conferences that were attended by researchers and practitioners from both the academia and industry. The success of these conferences, along with the interest shown by businesses and industry in recruiting new hires with data mining background, have fueled the tremendous growth of this field.

The field was initially built upon the methodology and algorithms that researchers had previously used. In particular, data mining researchers draw upon ideas, such as (1) sampling, estimation, and hypothesis testing from statistics and (2) search algorithms, modeling techniques, and learning theories from artificial intelligence, pattern recognition, and machine learning. Data mining has also been quick to adopt ideas from other areas, including optimization, evolutionary computing, information theory, signal processing, visualization, and information retrieval, and extending them to solve the challenges of mining big data.

A number of other areas also play key supporting roles. In particular, database systems are needed to provide support for efficient storage, indexing, and query processing. Techniques from high performance (parallel) computing are

often important in addressing the massive size of some data sets. Distributed techniques can also help address the issue of size and are essential when the data cannot be gathered in one location. [Figure 1.2](#) shows the relationship of data mining to other areas.

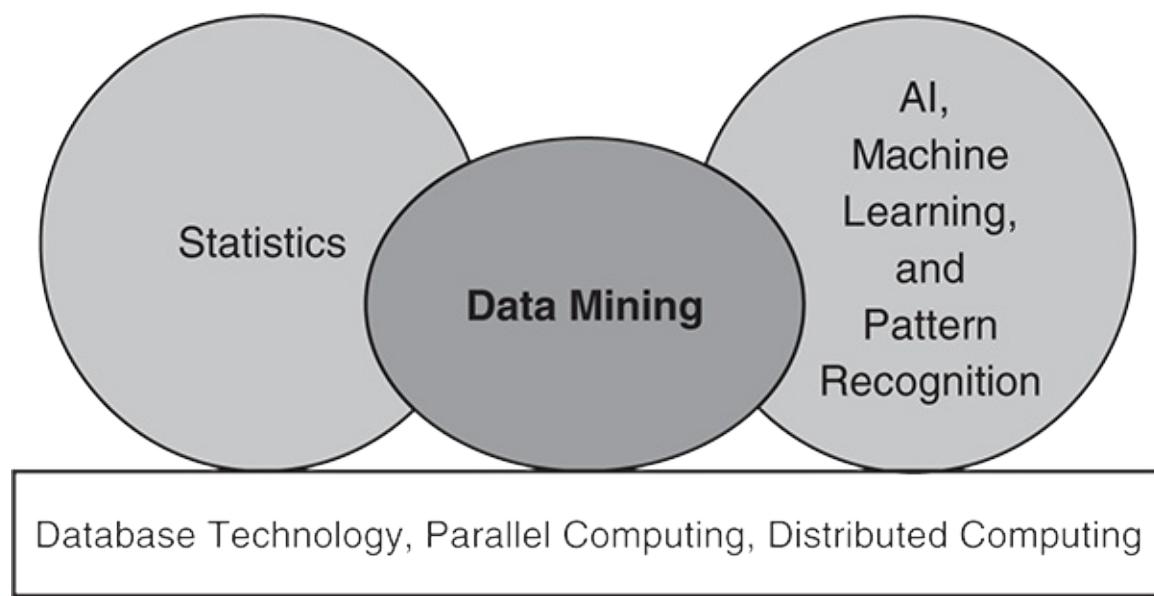


Figure 1.2.

Data mining as a confluence of many disciplines.

Data Science and Data-Driven Discovery

Data science is an interdisciplinary field that studies and applies tools and techniques for deriving useful insights from data. Although data science is regarded as an emerging field with a distinct identity of its own, the tools and techniques often come from many different areas of data analysis, such as data mining, statistics, AI, machine learning, pattern recognition, database technology, and distributed and parallel computing. (See [Figure 1.2](#).)

The emergence of data science as a new field is a recognition that, often, none of the existing areas of data analysis provides a complete set of tools for the data analysis tasks that are often encountered in emerging applications.

Instead, a broad range of computational, mathematical, and statistical skills is often required. To illustrate the challenges that arise in analyzing such data, consider the following example. Social media and the Web present new opportunities for social scientists to observe and quantitatively measure human behavior on a large scale. To conduct such a study, social scientists work with analysts who possess skills in areas such as web mining, natural language processing (NLP), network analysis, data mining, and statistics. Compared to more traditional research in social science, which is often based on surveys, this analysis requires a broader range of skills and tools, and involves far larger amounts of data. Thus, data science is, by necessity, a highly interdisciplinary field that builds on the continuing work of many fields.

The data-driven approach of data science emphasizes the direct discovery of patterns and relationships from data, especially in large quantities of data, often without the need for extensive domain knowledge. A notable example of the success of this approach is represented by advances in neural networks, i.e., deep learning, which have been particularly successful in areas which have long proved challenging, e.g., recognizing objects in photos or videos and words in speech, as well as in other application areas. However, note that this is just one example of the success of data-driven approaches, and dramatic improvements have also occurred in many other areas of data analysis. Many of these developments are topics described later in this book.

Some cautions on potential limitations of a purely data-driven approach are given in the Bibliographic Notes.

1.4 Data Mining Tasks

Data mining tasks are generally divided into two major categories:

Predictive tasks The objective of these tasks is to predict the value of a particular attribute based on the values of other attributes. The attribute to be predicted is commonly known as the **target** or **dependent variable**, while the attributes used for making the prediction are known as the **explanatory** or **independent variables**.

Descriptive tasks Here, the objective is to derive patterns (correlations, trends, clusters, trajectories, and anomalies) that summarize the underlying relationships in data. Descriptive data mining tasks are often exploratory in nature and frequently require postprocessing techniques to validate and explain the results.

Figure 1.3  illustrates four of the core data mining tasks that are described in the remainder of this book.

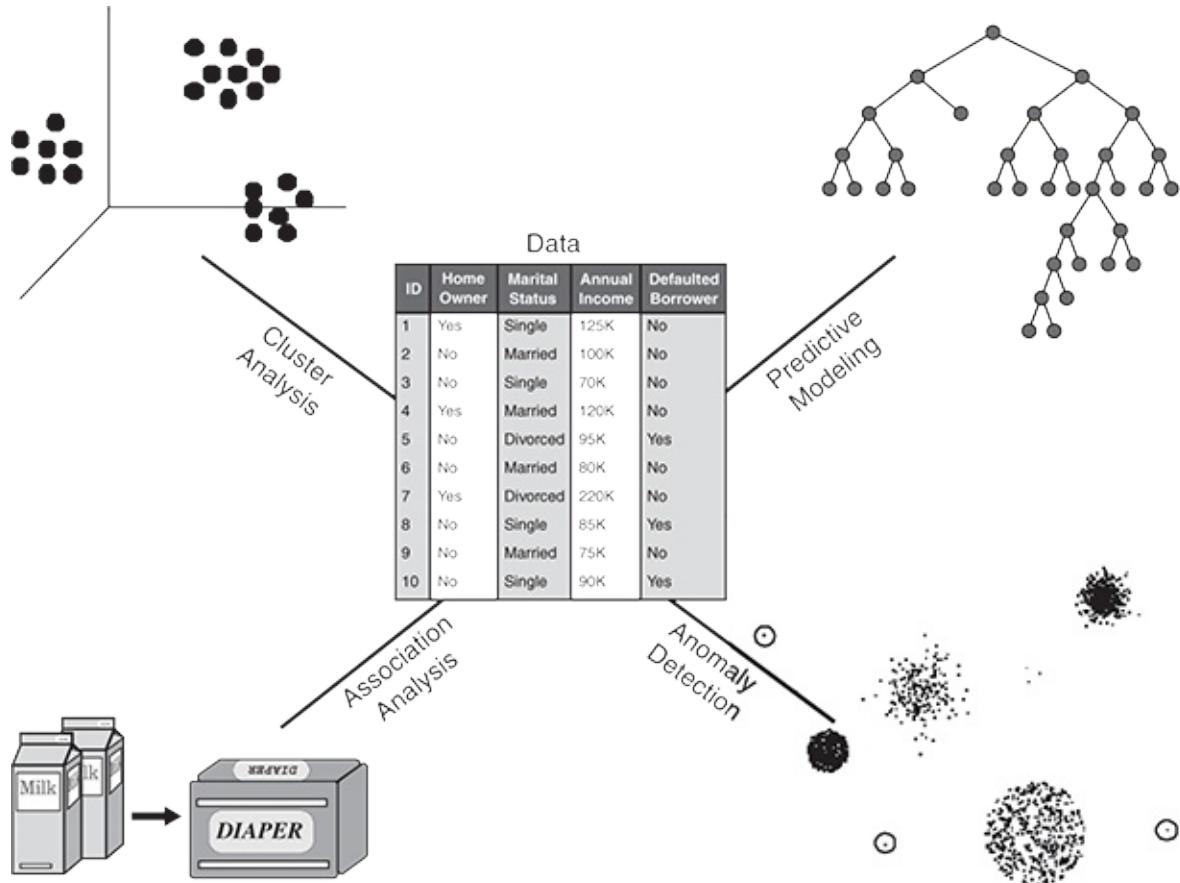


Figure 1.3.

Four of the core data mining tasks.

Predictive modeling refers to the task of building a model for the target variable as a function of the explanatory variables. There are two types of predictive modeling tasks: **classification**, which is used for discrete target variables, and **regression**, which is used for continuous target variables. For example, predicting whether a web user will make a purchase at an online bookstore is a classification task because the target variable is binary-valued. On the other hand, forecasting the future price of a stock is a regression task because price is a continuous-valued attribute. The goal of both tasks is to learn a model that minimizes the error between the predicted and true values of the target variable. Predictive modeling can be used to identify customers who will respond to a marketing campaign, predict disturbances in the Earth's

ecosystem, or judge whether a patient has a particular disease based on the results of medical tests.

Example 1.1 (Predicting the Type of a Flower).

Consider the task of predicting a species of flower based on the characteristics of the flower. In particular, consider classifying an Iris flower as one of the following three Iris species: Setosa, Versicolour, or Virginica. To perform this task, we need a data set containing the characteristics of various flowers of these three species. A data set with this type of information is the well-known Iris data set from the UCI Machine Learning Repository at <http://www.ics.uci.edu/~mlearn>. In addition to the species of a flower, this data set contains four other attributes: sepal width, sepal length, petal length, and petal width. [Figure 1.4](#) shows a plot of petal width versus petal length for the 150 flowers in the Iris data set. Petal width is broken into the categories *low*, *medium*, and *high*, which correspond to the intervals $[0, 0.75]$, $[0.75, 1.75]$, $[1.75, \infty)$, respectively. Also, petal length is broken into categories *low*, *medium*, and *high*, which correspond to the intervals $[0, 2.5]$, $[2.5, 5]$, $[5, \infty)$, respectively. Based on these categories of petal width and length, the following rules can be derived:

Petal width low and petal length low implies Setosa.

Petal width medium and petal length medium implies Versicolour.

Petal width high and petal length high implies Virginica.

While these rules do not classify all the flowers, they do a good (but not perfect) job of classifying most of the flowers. Note that flowers from the Setosa species are well separated from the Versicolour and Virginica species with respect to petal width and length, but the latter two species overlap somewhat with respect to these attributes.

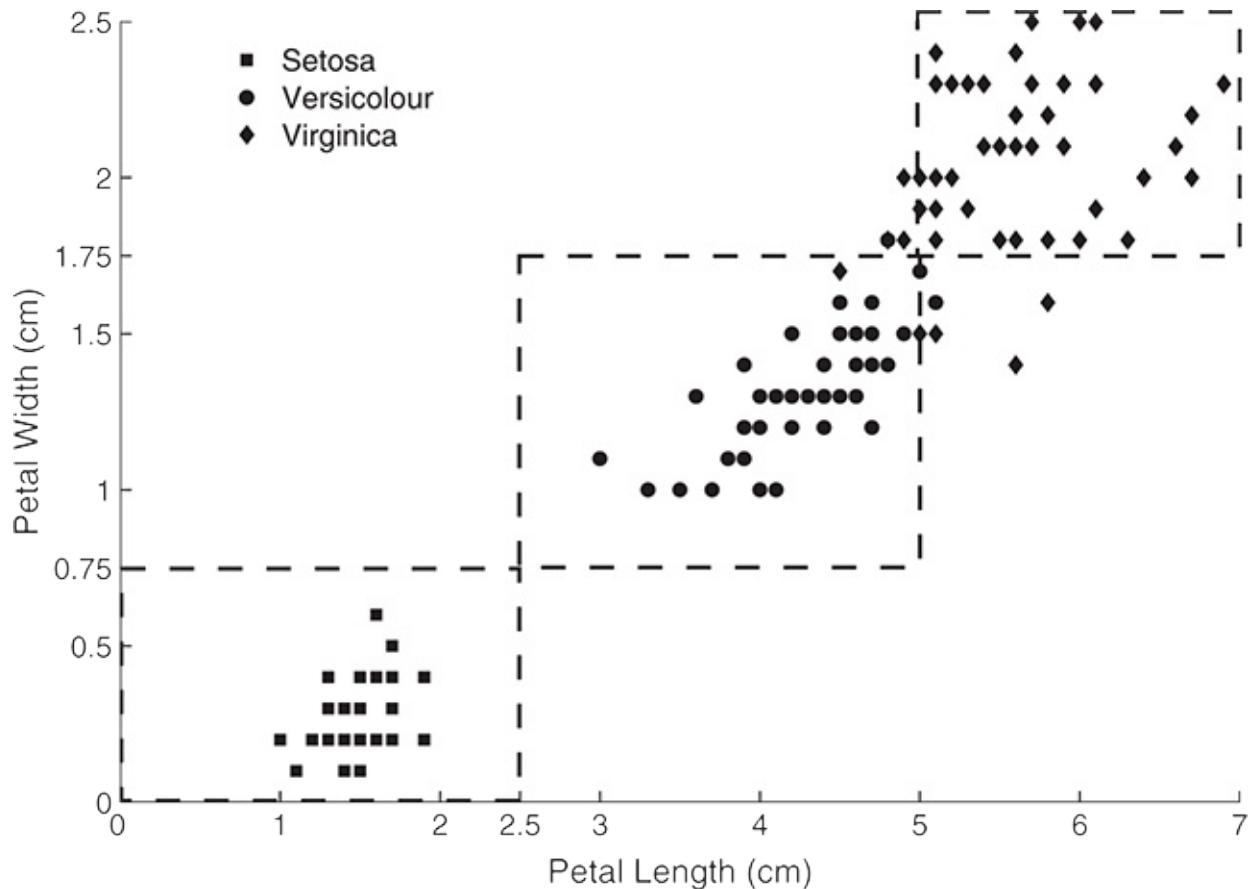


Figure 1.4.

Petal width versus petal length for 150 Iris flowers.

Association analysis is used to discover patterns that describe strongly associated features in the data. The discovered patterns are typically represented in the form of implication rules or feature subsets. Because of the exponential size of its search space, the goal of association analysis is to extract the most interesting patterns in an efficient manner. Useful applications of association analysis include finding groups of genes that have related functionality, identifying web pages that are accessed together, or understanding the relationships between different elements of Earth's climate system.

Example 1.2 (Market Basket Analysis).

The transactions shown in [Table 1.1](#) illustrate point-of-sale data collected at the checkout counters of a grocery store. Association analysis can be applied to find items that are frequently bought together by customers. For example, we may discover the rule {Diapers}→{Milk}, which suggests that customers who buy diapers also tend to buy milk. This type of rule can be used to identify potential cross-selling opportunities among related items.

Table 1.1. Market basket data.

Transaction ID	Items
1	{Bread, Butter, Diapers, Milk}
2	{Coffee, Sugar, Cookies, Salmon}
3	{Bread, Butter, Coffee, Diapers, Milk, Eggs}
4	{Bread, Butter, Salmon, Chicken}
5	{Eggs, Bread, Butter}
6	{Salmon, Diapers, Milk}
7	{Bread, Tea, Sugar, Eggs}
8	{Coffee, Sugar, Chicken, Eggs}
9	{Bread, Diapers, Milk, Salt}
10	{Tea, Eggs, Cookies, Diapers, Milk}

Cluster analysis seeks to find groups of closely related observations so that observations that belong to the same cluster are more similar to each other than observations that belong to other clusters. Clustering has been used to

group sets of related customers, find areas of the ocean that have a significant impact on the Earth's climate, and compress data.

Example 1.3 (Document Clustering).

The collection of news articles shown in [Table 1.2](#) can be grouped based on their respective topics. Each article is represented as a set of word-frequency pairs ($w : c$), where w is a word and c is the number of times the word appears in the article. There are two natural clusters in the data set. The first cluster consists of the first four articles, which correspond to news about the economy, while the second cluster contains the last four articles, which correspond to news about health care. A good clustering algorithm should be able to identify these two clusters based on the similarity between words that appear in the articles.

Table 1.2. Collection of news articles.

Article	Word-frequency pairs
1	dollar: 1, industry: 4, country: 2, loan: 3, deal: 2, government: 2
2	machinery: 2, labor: 3, market: 4, industry: 2, work: 3, country: 1
3	job: 5, inflation: 3, rise: 2, jobless: 2, market: 3, country: 2, index: 3
4	domestic: 3, forecast: 2, gain: 1, market: 2, sale: 3, price: 2
5	patient: 4, symptom: 2, drug: 3, health: 2, clinic: 2, doctor: 2
6	pharmaceutical: 2, company: 3, drug: 2, vaccine: 1, flu: 3
7	death: 2, cancer: 4, drug: 3, public: 4, health: 3, director: 2
8	medical: 2, cost: 3, increase: 2, patient: 2, health: 3, care: 1

Anomaly detection is the task of identifying observations whose characteristics are significantly different from the rest of the data. Such observations are known as **anomalies** or **outliers**. The goal of an anomaly detection algorithm is to discover the real anomalies and avoid falsely labeling normal objects as anomalous. In other words, a good anomaly detector must have a high detection rate and a low false alarm rate. Applications of anomaly detection include the detection of fraud, network intrusions, unusual patterns of disease, and ecosystem disturbances, such as droughts, floods, fires, hurricanes, etc.

Example 1.4 (Credit Card Fraud Detection).

A credit card company records the transactions made by every credit card holder, along with personal information such as credit limit, age, annual income, and address. Since the number of fraudulent cases is relatively small compared to the number of legitimate transactions, anomaly detection techniques can be applied to build a profile of legitimate transactions for the users. When a new transaction arrives, it is compared against the profile of the user. If the characteristics of the transaction are very different from the previously created profile, then the transaction is flagged as potentially fraudulent.

1.5 Scope and Organization of the Book

This book introduces the major principles and techniques used in data mining from an algorithmic perspective. A study of these principles and techniques is essential for developing a better understanding of how data mining technology can be applied to various kinds of data. This book also serves as a starting point for readers who are interested in doing research in this field.

We begin the technical discussion of this book with a chapter on data ([Chapter 2](#)), which discusses the basic types of data, data quality, preprocessing techniques, and measures of similarity and dissimilarity. Although this material can be covered quickly, it provides an essential foundation for data analysis. [Chapters 3](#) and [4](#) cover classification. [Chapter 3](#) provides a foundation by discussing decision tree classifiers and several issues that are important to all classification: overfitting, underfitting, model selection, and performance evaluation. Using this foundation, [Chapter 4](#) describes a number of other important classification techniques: rule-based systems, nearest neighbor classifiers, Bayesian classifiers, artificial neural networks, including deep learning, support vector machines, and ensemble classifiers, which are collections of classifiers. The multiclass and imbalanced class problems are also discussed. These topics can be covered independently.

Association analysis is explored in [Chapters 5](#) and [6](#). [Chapter 5](#) describes the basics of association analysis: frequent itemsets, association rules, and some of the algorithms used to generate them. Specific types of frequent itemsets—maximal, closed, and hyperclique—that are important for

data mining are also discussed, and the chapter concludes with a discussion of evaluation measures for association analysis. [Chapter 6](#) considers a variety of more advanced topics, including how association analysis can be applied to categorical and continuous data or to data that has a concept hierarchy. (A concept hierarchy is a hierarchical categorization of objects, e.g., store items→clothing→shoes→sneakers.) This chapter also describes how association analysis can be extended to find sequential patterns (patterns involving order), patterns in graphs, and negative relationships (if one item is present, then the other is not).

Cluster analysis is discussed in [Chapters 7](#) and [8](#). [Chapter 7](#) first describes the different types of clusters, and then presents three specific clustering techniques: K-means, agglomerative hierarchical clustering, and DBSCAN. This is followed by a discussion of techniques for validating the results of a clustering algorithm. Additional clustering concepts and techniques are explored in [Chapter 8](#), including fuzzy and probabilistic clustering, Self-Organizing Maps (SOM), graph-based clustering, spectral clustering, and density-based clustering. There is also a discussion of scalability issues and factors to consider when selecting a clustering algorithm.

[Chapter 9](#), is on anomaly detection. After some basic definitions, several different types of anomaly detection are considered: statistical, distance-based, density-based, clustering-based, reconstruction-based, one-class classification, and information theoretic. The last chapter, [Chapter 10](#), supplements the discussions in the other Chapters with a discussion of the statistical concepts important for avoiding spurious results, and then discusses those concepts in the context of data mining techniques studied in the previous chapters. These techniques include statistical hypothesis testing, p-values, the false discovery rate, and permutation testing. Appendices A through F give a brief review of important topics that are used in portions of

the book: linear algebra, dimensionality reduction, statistics, regression, optimization, and scaling up data mining techniques for big data.

The subject of data mining, while relatively young compared to statistics or machine learning, is already too large to cover in a single book. Selected references to topics that are only briefly covered, such as data quality, are provided in the Bibliographic Notes section of the appropriate chapter. References to topics not covered in this book, such as mining streaming data and privacy-preserving data mining are provided in the Bibliographic Notes of this chapter.

1.6 Bibliographic Notes

The topic of data mining has inspired many textbooks. Introductory textbooks include those by Dunham [16], Han et al. [29], Hand et al. [31], Roiger and Geatz [50], Zaki and Meira [61], and Aggarwal [2]. Data mining books with a stronger emphasis on business applications include the works by Berry and Linoff [5], Pyle [47], and Parr Rud [45]. Books with an emphasis on statistical learning include those by Cherkassky and Mulier [11], and Hastie et al. [32]. Similar books with an emphasis on machine learning or pattern recognition are those by Duda et al. [15], Kantardzic [34], Mitchell [43], Webb [57], and Witten and Frank [58]. There are also some more specialized books: Chakrabarti [9] (web mining), Fayyad et al. [20] (collection of early articles on data mining), Fayyad et al. [18] (visualization), Grossman et al. [25] (science and engineering), Kargupta and Chan [35] (distributed data mining), Wang et al. [56] (bioinformatics), and Zaki and Ho [60] (parallel data mining).

There are several conferences related to data mining. Some of the main conferences dedicated to this field include the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), the IEEE International Conference on Data Mining (ICDM), the SIAM International Conference on Data Mining (SDM), the European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), and the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD). Data mining papers can also be found in other major conferences such as the Conference and Workshop on Neural Information Processing Systems (NIPS), the International Conference on Machine Learning (ICML), the ACM SIGMOD/PODS conference, the International Conference on Very Large Data Bases (VLDB), the Conference on Information and Knowledge Management (CIKM), the International Conference on Data Engineering (ICDE), the

National Conference on Artificial Intelligence (AAAI), the IEEE International Conference on Big Data, and the IEEE International Conference on Data Science and Advanced Analytics (DSAA).

Journal publications on data mining include IEEE Transactions on Knowledge and Data Engineering, Data Mining and Knowledge Discovery, Knowledge and Information Systems, ACM Transactions on Knowledge Discovery from Data, Statistical Analysis and Data Mining, and Information Systems. There are various open-source data mining software available, including Weka [27] and Scikit-learn [46]. More recently, data mining software such as Apache Mahout and Apache Spark have been developed for large-scale problems on the distributed computing platform.

There have been a number of general articles on data mining that define the field or its relationship to other fields, particularly statistics. Fayyad et al. [19] describe data mining and how it fits into the total knowledge discovery process. Chen et al. [10] give a database perspective on data mining. Ramakrishnan and Grama [48] provide a general discussion of data mining and present several viewpoints. Hand [30] describes how data mining differs from statistics, as does Friedman [21]. Lambert [40] explores the use of statistics for large data sets and provides some comments on the respective roles of data mining and statistics. Glymour et al. [23] consider the lessons that statistics may have for data mining. Smyth et al. [53] describe how the evolution of data mining is being driven by new types of data and applications, such as those involving streams, graphs, and text. Han et al. [28] consider emerging applications in data mining and Smyth [52] describes some research challenges in data mining. Wu et al. [59] discuss how developments in data mining research can be turned into practical tools. Data mining standards are the subject of a paper by Grossman et al. [24]. Bradley [7] discusses how data mining algorithms can be scaled to large data sets.

The emergence of new data mining applications has produced new challenges that need to be addressed. For instance, concerns about privacy breaches as a result of data mining have escalated in recent years, particularly in application domains such as web commerce and health care. As a result, there is growing interest in developing data mining algorithms that maintain user privacy. Developing techniques for mining encrypted or randomized data is known as **privacy-preserving data mining**. Some general references in this area include papers by Agrawal and Srikant [3], Clifton et al. [12] and Kargupta et al. [36]. Vassilios et al. [55] provide a survey. Another area of concern is the bias in predictive models that may be used for some applications, e.g., screening job applicants or deciding prison parole [39]. Assessing whether such applications are producing biased results is made more difficult by the fact that the predictive models used for such applications are often black box models, i.e., models that are not interpretable in any straightforward way.

Data science, its constituent fields, and more generally, the new paradigm of knowledge discovery they represent [33], have great potential, some of which has been realized. However, it is important to emphasize that data science works mostly with observational data, i.e., data that was collected by various organizations as part of their normal operation. The consequence of this is that sampling biases are common and the determination of causal factors becomes more problematic. For this and a number of other reasons, it is often hard to interpret the predictive models built from this data [42, 49]. Thus, theory, experimentation and computational simulations will continue to be the methods of choice in many areas, especially those related to science.

More importantly, a purely data-driven approach often ignores the existing knowledge in a particular field. Such models may perform poorly, for example, predicting impossible outcomes or failing to generalize to new situations. However, if the model does work well, e.g., has high predictive accuracy, then

this approach may be sufficient for practical purposes in some fields. But in many areas, such as medicine and science, gaining insight into the underlying domain is often the goal. Some recent work attempts to address these issues in order to create theory-guided data science, which takes pre-existing domain knowledge into account [17, 37].

Recent years have witnessed a growing number of applications that rapidly generate continuous streams of data. Examples of stream data include network traffic, multimedia streams, and stock prices. Several issues must be considered when mining data streams, such as the limited amount of memory available, the need for online analysis, and the change of the data over time. Data mining for stream data has become an important area in data mining. Some selected publications are Domingos and Hulten [14] (classification), Giannella et al. [22] (association analysis), Guha et al. [26] (clustering), Kifer et al. [38] (change detection), Papadimitriou et al. [44] (time series), and Law et al. [41] (dimensionality reduction).

Another area of interest is recommender and collaborative filtering systems [1, 6, 8, 13, 54], which suggest movies, television shows, books, products, etc. that a person might like. In many cases, this problem, or at least a component of it, is treated as a prediction problem and thus, data mining techniques can be applied [4, 51].

Bibliography

- [1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.
- [2] C. Aggarwal. *Data mining: The Textbook*. Springer, 2009.
- [3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 439–450, Dallas, Texas, 2000. ACM Press.
- [4] X. Amatriain and J. M. Pujol. Data mining methods for recommender systems. In *Recommender Systems Handbook*, pages 227–262. Springer, 2015.
- [5] M. J. A. Berry and G. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. Wiley Computer Publishing, 2nd edition, 2004.
- [6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

- [7] P. S. Bradley, J. Gehrke, R. Ramakrishnan, and R. Srikant. Scaling mining algorithms to large databases. *Communications of the ACM*, 45(8):38–43, 2002.
- [8] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.
- [9] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Francisco, CA, 2003.
- [10] M.-S. Chen, J. Han, and P. S. Yu. Data Mining: An Overview from a Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.
- [11] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press, 2nd edition, 1998.
- [12] C. Clifton, M. Kantarcio glu, and J. Vaidya. Defining privacy for data mining. In *National Science Foundation Workshop on Next Generation Data Mining*, pages 126– 133, Baltimore, MD, November 2002.
- [13] C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. *Recommender systems handbook*, pages 107–144, 2011.
- [14] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 71–80,

Boston, Massachusetts, 2000. ACM Press.

- [15] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley ... Sons, Inc., New York, 2nd edition, 2001.
- [16] M. H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2006.
- [17] J. H. Faghmous, A. Banerjee, S. Shekhar, M. Steinbach, V. Kumar, A. R. Ganguly, and N. Samatova. Theory-guided data science for climate change. *Computer*, 47(11):74–78, 2014.
- [18] U. M. Fayyad, G. G. Grinstein, and A. Wierse, editors. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers, San Francisco, CA, September 2001.
- [19] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press, 1996.
- [20] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [21] J. H. Friedman. Data Mining and Statistics: What's the Connection? Unpublished. www-stat.stanford.edu/~jhf/ftp/dm-stat.ps, 1997.

- [22] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*, pages 191–212. AAAI/MIT, 2003.
- [23] C. Glymour, D. Madigan, D. Pregibon, and P. Smyth. Statistical Themes and Lessons for Data Mining. *Data Mining and Knowledge Discovery*, 1(1):11–28, 1997.
- [24] R. L. Grossman, M. F. Hornick, and G. Meyer. Data mining standards initiatives. *Communications of the ACM*, 45(8):59–61, 2002.
- [25] R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, and R. Namburu, editors. *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.
- [26] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, May/June 2003.
- [27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009.
- [28] J. Han, R. B. Altman, V. Kumar, H. Mannila, and D. Pregibon. Emerging scientific applications in data mining. *Communications of the ACM*, 45(8):54–58, 2002.

- [29] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 3rd edition, 2011.
- [30] D. J. Hand. Data Mining: Statistics and More? *The American Statistician*, 52(2): 112–118, 1998.
- [31] D. J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.
- [32] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, 2nd edition, 2009.
- [33] T. Hey, S. Tansley, K. M. Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.
- [34] M. Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms*. Wiley-IEEE Press, Piscataway, NJ, 2003.
- [35] H. Kargupta and P. K. Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press, September 2002.
- [36] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the Privacy Preserving Properties of Random Data Perturbation Techniques. In *Proc. of the 2003 IEEE Intl. Conf. on Data Mining*, pages 99–106, Melbourne, Florida, December 2003. IEEE Computer Society.

- [37] A. Karpatne, G. Atluri, J. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar. Theory-guided Data Science: A New Paradigm for Scientific Discovery from Data. *IEEE Transactions on Knowledge and Data Engineering*, 2017.
- [38] D. Kifer, S. Ben-David, and J. Gehrke. Detecting Change in Data Streams. In *Proc. of the 30th VLDB Conf.*, pages 180–191, Toronto, Canada, 2004. Morgan Kaufmann.
- [39] J. Kleinberg, J. Ludwig, and S. Mullainathan. A Guide to Solving Social Problems with Machine Learning. *Harvard Business Review*, December 2016.
- [40] D. Lambert. What Use is Statistics for Massive Data? In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 54–62, 2000.
- [41] M. H. C. Law, N. Zhang, and A. K. Jain. Nonlinear Manifold Learning for Data Streams. In *Proc. of the SIAM Intl. Conf. on Data Mining*, Lake Buena Vista, Florida, April 2004. SIAM.
- [42] Z. C. Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.
- [43] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.

- [44] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, unsupervised stream mining. *VLDB Journal*, 13(3):222–239, 2004.
- [45] O. Parr Rud. *Data Mining Cookbook: Modeling Data for Marketing, Risk and Customer Relationship Management*. John Wiley ... Sons, New York, NY, 2001.
- [46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [47] D. Pyle. *Business Modeling and Data Mining*. Morgan Kaufmann, San Francisco, CA, 2003.
- [48] N. Ramakrishnan and A. Grama. Data Mining: From Serendipity to Science—Guest Editors' Introduction. *IEEE Computer*, 32(8):34–37, 1999.
- [49] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.
- [50] R. Roiger and M. Geatz. *Data Mining: A Tutorial Based Primer*. Addison-Wesley, 2002.

- [51] J. Schafer. The Application of Data-Mining to Recommender Systems. *Encyclopedia of data warehousing and mining*, 1:44–48, 2009.
- [52] P. Smyth. Breaking out of the Black-Box: Research Challenges in Data Mining. In *Proc. of the 2001 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.
- [53] P. Smyth, D. Pregibon, and C. Faloutsos. Data-driven evolution of data mining algorithms. *Communications of the ACM*, 45(8):33–37, 2002.
- [54] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.
- [55] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.
- [56] J. T. L. Wang, M. J. Zaki, H. Toivonen, and D. E. Shasha, editors. *Data Mining in Bioinformatics*. Springer, September 2004.
- [57] A. R. Webb. *Statistical Pattern Recognition*. John Wiley ... Sons, 2nd edition, 2002.
- [58] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3rd edition, 2011.

- [59] X. Wu, P. S. Yu, and G. Piatetsky-Shapiro. Data Mining: How Research Meets Practical Development? *Knowledge and Information Systems*, 5(2):248–261, 2003.
- [60] M. J. Zaki and C.-T. Ho, editors. *Large-Scale Parallel Data Mining*. Springer, September 2002.
- [61] M. J. Zaki and W. Meira Jr. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, New York, 2014.

1.7 Exercises

1. Discuss whether or not each of the following activities is a data mining task.
 - a. Dividing the customers of a company according to their gender.
 - b. Dividing the customers of a company according to their profitability.
 - c. Computing the total sales of a company.
 - d. Sorting a student database based on student identification numbers.
 - e. Predicting the outcomes of tossing a (fair) pair of dice.
 - f. Predicting the future stock price of a company using historical records.
 - g. Monitoring the heart rate of a patient for abnormalities.
 - h. Monitoring seismic waves for earthquake activities.
 - i. Extracting the frequencies of a sound wave.
2. Suppose that you are employed as a data mining consultant for an Internet search engine company. Describe how data mining can help the company by giving specific examples of how techniques, such as clustering, classification, association rule mining, and anomaly detection can be applied.
3. For each of the following data sets, explain whether or not data privacy is an important issue.
 - a. Census data collected from 1900–1950.
 - b. IP addresses and visit times of web users who visit your website.
 - c. Images from Earth-orbiting satellites.

- d. Names and addresses of people from the telephone book.
- e. Names and email addresses collected from the Web.

2 Data

This chapter discusses several data-related issues that are important for successful data mining:

The Type of Data Data sets differ in a number of ways. For example, the attributes used to describe data objects can be of different types—quantitative or qualitative—and data sets often have special characteristics; e.g., some data sets contain time series or objects with explicit relationships to one another. Not surprisingly, the type of data determines which tools and techniques can be used to analyze the data. Indeed, new research in data mining is often driven by the need to accommodate new application areas and their new types of data.

The Quality of the Data Data is often far from perfect. While most data mining techniques can tolerate some level of imperfection in the data, a focus on understanding and improving data quality typically improves the quality of the resulting analysis. Data quality issues that often need to be addressed include the presence of noise and outliers; missing, inconsistent, or duplicate data; and data that is biased or, in some other way, unrepresentative of the phenomenon or population that the data is supposed to describe.

Preprocessing Steps to Make the Data More Suitable for Data Mining

Often, the raw data must be processed in order to make it suitable for

analysis. While one objective may be to improve data quality, other goals focus on modifying the data so that it better fits a specified data mining technique or tool. For example, a continuous attribute, e.g., length, sometimes needs to be transformed into an attribute with discrete categories, e.g., *short*, *medium*, or *long*, in order to apply a particular technique. As another example, the number of attributes in a data set is often reduced because many techniques are more effective when the data has a relatively small number of attributes.

Analyzing Data in Terms of Its Relationships One approach to data analysis is to find relationships among the data objects and then perform the remaining analysis using these relationships rather than the data objects themselves. For instance, we can compute the similarity or distance between pairs of objects and then perform the analysis—clustering, classification, or anomaly detection—based on these similarities or distances. There are many such similarity or distance measures, and the proper choice depends on the type of data and the particular application.

Example 2.1 (An Illustration of Data-Related Issues).

To further illustrate the importance of these issues, consider the following hypothetical situation. You receive an email from a medical researcher concerning a project that you are eager to work on.

Hi,

I've attached the data file that I mentioned in my previous email. Each line contains the information for a single patient and consists of five fields. We want to predict the last field using the other fields. I don't have time to provide any more information about the data since I'm going out of town for a couple of days, but hopefully that won't slow you down too much. And if you

don't mind, could we meet when I get back to discuss your preliminary results? I might invite a few other members of my team.

Thanks and see you in a couple of days.

Despite some misgivings, you proceed to analyze the data. The first few rows of the file are as follows:

012	232	33.5	0	10.7
020	121	16.9	2	210.1
027	165	24.0	0	427.6
:				

A brief look at the data reveals nothing strange. You put your doubts aside and start the analysis. There are only 1000 lines, a smaller data file than you had hoped for, but two days later, you feel that you have made some progress. You arrive for the meeting, and while waiting for others to arrive, you strike up a conversation with a statistician who is working on the project. When she learns that you have also been analyzing the data from the project, she asks if you would mind giving her a brief overview of your results.

Statistician: So, you got the data for all the patients?

Data Miner: Yes. I haven't had much time for analysis, but I do have a few interesting results.

Statistician: Amazing. There were so many data issues with this set of patients that I couldn't do much.

Data Miner: Oh? I didn't hear about any possible problems.

Statistician: Well, first there is field 5, the variable we want to predict.

It's common knowledge among people who analyze this type of data that results are better if you work with the log of the values, but I didn't discover this until later. Was it mentioned to you?

Data Miner: No.

Statistician: But surely you heard about what happened to field 4? It's supposed to be measured on a scale from 1 to 10, with 0 indicating a missing value, but because of a data entry error, all 10's were changed into 0's. Unfortunately, since some of the patients have missing values for this field, it's impossible to say whether a 0 in this field is a real 0 or a 10. Quite a few of the records have that problem.

Data Miner: Interesting. Were there any other problems?

Statistician: Yes, fields 2 and 3 are basically the same, but I assume that you probably noticed that.

Data Miner: Yes, but these fields were only weak predictors of field 5.

Statistician: Anyway, given all those problems, I'm surprised you were able to accomplish anything.

Data Miner: True, but my results are really quite good. Field 1 is a very strong predictor of field 5. I'm surprised that this wasn't noticed before.

Statistician: What? Field 1 is just an identification number.

Data Miner: Nonetheless, my results speak for themselves.

Statistician: Oh, no! I just remembered. We assigned ID numbers after we sorted the records based on field 5. There is a strong connection, but it's meaningless. Sorry.

Although this scenario represents an extreme situation, it emphasizes the importance of "knowing your data." To that end, this chapter will address each

of the four issues mentioned above, outlining some of the basic challenges and standard approaches.

2.1 Types of Data

A **data set** can often be viewed as a collection of **data objects**. Other names for a data object are *record*, *point*, *vector*, *pattern*, *event*, *case*, *sample*, *instance*, *observation*, or *entity*. In turn, data objects are described by a number of **attributes** that capture the characteristics of an object, such as the mass of a physical object or the time at which an event occurred. Other names for an attribute are *variable*, *characteristic*, *field*, *feature*, or *dimension*.

Example 2.2 (Student Information).

Often, a data set is a file, in which the objects are records (or rows) in the file and each field (or column) corresponds to an attribute. For example, [Table 2.1](#) shows a data set that consists of student information. Each row corresponds to a student and each column is an attribute that describes some aspect of a student, such as grade point average (GPA) or identification number (ID).

Table 2.1. A sample data set containing student information.

Student ID	Year	Grade Point Average (GPA)	...
	:		
1034262	Senior	3.24	...
1052663	Freshman	3.51	...
1082246	Sophomore	3.62	...

Although record-based data sets are common, either in flat files or relational database systems, there are other important types of data sets and systems for storing data. In [Section 2.1.2](#), we will discuss some of the types of data sets that are commonly encountered in data mining. However, we first consider attributes.

2.1.1 Attributes and Measurement

In this section, we consider the types of attributes used to describe data objects. We first define an attribute, then consider what we mean by the type of an attribute, and finally describe the types of attributes that are commonly encountered.

What Is an Attribute?

We start with a more detailed definition of an attribute.

Definition 2.1.

An **attribute** is a property or characteristic of an object that can vary, either from one object to another or from one time to another.

For example, eye color varies from person to person, while the temperature of an object varies over time. Note that eye color is a symbolic attribute with a

small number of possible values {*brown, black, blue, green, hazel, etc.*} , while temperature is a numerical attribute with a potentially unlimited number of values.

At the most basic level, attributes are not about numbers or symbols. However, to discuss and more precisely analyze the characteristics of objects, we assign numbers or symbols to them. To do this in a well-defined way, we need a measurement scale.

Definition 2.2.

A **measurement scale** is a rule (function) that associates a numerical or symbolic value with an attribute of an object.

Formally, the process of **measurement** is the application of a measurement scale to associate a value with a particular attribute of a specific object. While this may seem a bit abstract, we engage in the process of measurement all the time. For instance, we step on a bathroom scale to determine our weight, we classify someone as male or female, or we count the number of chairs in a room to see if there will be enough to seat all the people coming to a meeting. In all these cases, the “physical value” of an attribute of an object is mapped to a numerical or symbolic value.

With this background, we can discuss the type of an attribute, a concept that is important in determining if a particular data analysis technique is consistent with a specific type of attribute.

The Type of an Attribute

It is common to refer to the type of an attribute as the **type of a measurement scale**. It should be apparent from the previous discussion that an attribute can be described using different measurement scales and that the properties of an attribute need not be the same as the properties of the values used to measure it. In other words, the values used to represent an attribute can have properties that are not properties of the attribute itself, and vice versa. This is illustrated with two examples.

Example 2.3 (Employee Age and ID Number).

Two attributes that might be associated with an employee are *ID* and *age* (in years). Both of these attributes can be represented as integers. However, while it is reasonable to talk about the average age of an employee, it makes no sense to talk about the average employee ID. Indeed, the only aspect of employees that we want to capture with the ID attribute is that they are distinct. Consequently, the only valid operation for employee IDs is to test whether they are equal. There is no hint of this limitation, however, when integers are used to represent the employee ID attribute. For the age attribute, the properties of the integers used to represent age are very much the properties of the attribute. Even so, the correspondence is not complete because, for example, ages have a maximum, while integers do not.

Example 2.4 (Length of Line Segments).

Consider [Figure 2.1](#), which shows some objects—line segments—and how the length attribute of these objects can be mapped to numbers in two different ways. Each successive line segment, going from the top to the bottom, is formed by appending the topmost line segment to itself. Thus,

the second line segment from the top is formed by appending the topmost line segment to itself twice, the third line segment from the top is formed by appending the topmost line segment to itself three times, and so forth. In a very real (physical) sense, all the line segments are multiples of the first. This fact is captured by the measurements on the right side of the figure, but not by those on the left side. More specifically, the measurement scale on the left side captures only the ordering of the length attribute, while the scale on the right side captures both the ordering and additivity properties. Thus, an attribute can be measured in a way that does not capture all the properties of the attribute.

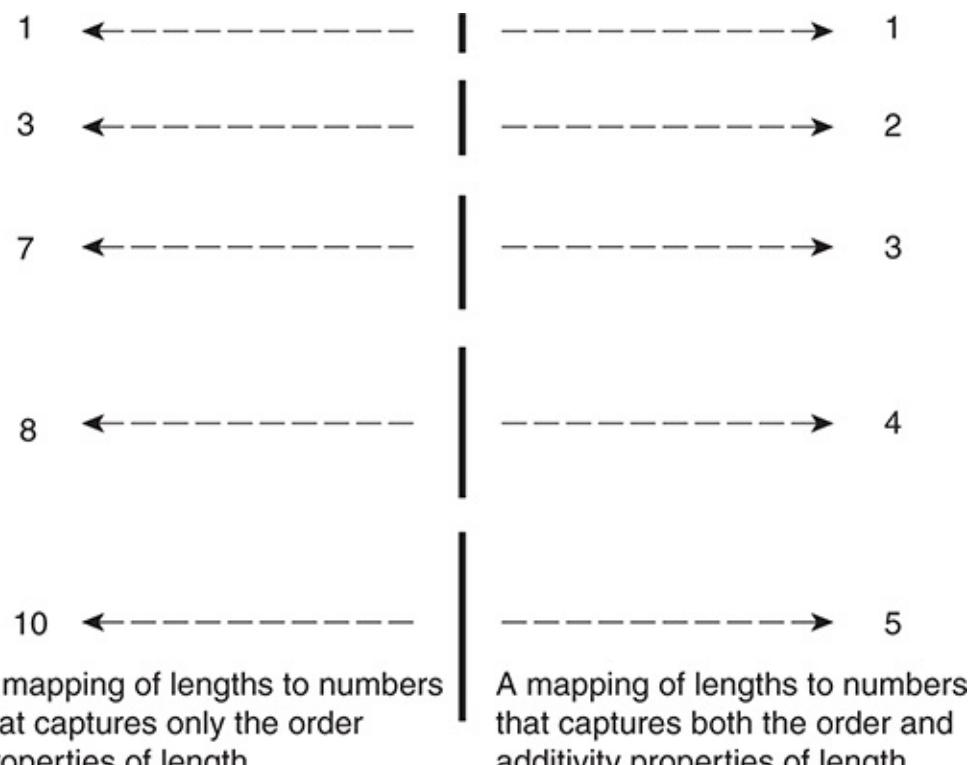


Figure 2.1.

The measurement of the length of line segments on two different scales of measurement.

Knowing the type of an attribute is important because it tells us which properties of the measured values are consistent with the underlying

properties of the attribute, and therefore, it allows us to avoid foolish actions, such as computing the average employee ID.

The Different Types of Attributes

A useful (and simple) way to specify the type of an attribute is to identify the properties of numbers that correspond to underlying properties of the attribute. For example, an attribute such as length has many of the properties of numbers. It makes sense to compare and order objects by length, as well as to talk about the differences and ratios of length. The following properties (operations) of numbers are typically used to describe attributes.

1. **Distinctness** = and \neq
2. **Order** $<$, \leq , $>$, and \geq
3. **Addition** $+$ and $-$
4. **Multiplication** \times and $/$

Given these properties, we can define four types of attributes: **nominal** , **ordinal**, **interval** , and **ratio**. [Table 2.2](#) gives the definitions of these types, along with information about the statistical operations that are valid for each type. Each attribute type possesses all of the properties and operations of the attribute types above it. Consequently, any property or operation that is valid for nominal, ordinal, and interval attributes is also valid for ratio attributes. In other words, the definition of the attribute types is cumulative. However, this does not mean that the statistical operations appropriate for one attribute type are appropriate for the attribute types above it.

Table 2.2. Different attribute types.

Attribute Type		Description	Examples	Operations
Categorical	Nominal	The values of a nominal attribute	zip codes,	mode,

(Qualitative)		are just different names; i.e., nominal values provide only enough information to distinguish one object from another. ($=$, \neq)	employee ID numbers, eye color, gender	entropy, contingency correlation, χ^2 test
	Ordinal	The values of an ordinal attribute provide enough information to order objects. ($<$, $>$)	hardness of minerals, <i>{good, better, best}</i> , grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
Numeric (Quantitative)	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. (+, -)	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviation, Pearson's correlation, <i>t</i> and <i>F</i> tests
	Ratio	For ratio variables, both differences and ratios are meaningful. (\times , $/$)	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current	geometric mean, harmonic mean, percent variation

Nominal and ordinal attributes are collectively referred to as **categorical** or **qualitative** attributes. As the name suggests, qualitative attributes, such as employee ID, lack most of the properties of numbers. Even if they are represented by numbers, i.e., integers, they should be treated more like symbols. The remaining two types of attributes, interval and ratio, are collectively referred to as **quantitative** or **numeric** attributes. Quantitative attributes are represented by numbers and have most of the properties of

numbers. Note that quantitative attributes can be integer-valued or continuous.

The types of attributes can also be described in terms of transformations that do not change the meaning of an attribute. Indeed, S. Smith Stevens, the psychologist who originally defined the types of attributes shown in [Table 2.2](#), defined them in terms of these **permissible transformations**. For example, the meaning of a length attribute is unchanged if it is measured in meters instead of feet.

The statistical operations that make sense for a particular type of attribute are those that will yield the same results when the attribute is transformed by using a transformation that preserves the attribute's meaning. To illustrate, the average length of a set of objects is different when measured in meters rather than in feet, but both averages represent the same length. [Table 2.3](#) shows the meaning-preserving transformations for the four attribute types of [Table 2.2](#).

Table 2.3. Transformations that define attribute levels.

Attribute Type		Transformation	Comment
Categorical (Qualitative)	Nominal	Any one-to-one mapping, e.g., a permutation of values	If all employee ID numbers are reassigned, it will not make any difference.
	Ordinal	An order-preserving change of values, i.e., $\text{new_value} = f(\text{old_value})$, where f is a monotonic function.	An attribute encompassing the notion of good, better, best can be represented equally well by the values $\{1, 2, 3\}$ or by $\{0.5, 1, 10\}$.
Numeric (Quantitative)	Interval	$\text{new_value} = a \times \text{old_value} + b$, a and b constants.	The Fahrenheit and Celsius temperature scales differ in the

		location of their zero value and the size of a degree (unit).
Ratio	$\text{new_value} = a \times \text{old_value}$	Length can be measured in meters or feet.

Example 2.5 (Temperature Scales).

Temperature provides a good illustration of some of the concepts that have been described. First, temperature can be either an interval or a ratio attribute, depending on its measurement scale. When measured on the Kelvin scale, a temperature of 2° is, in a physically meaningful way, twice that of a temperature of 1° . This is not true when temperature is measured on either the Celsius or Fahrenheit scales, because, physically, a temperature of 1° Fahrenheit (Celsius) is not much different than a temperature of 2° Fahrenheit (Celsius). The problem is that the zero points of the Fahrenheit and Celsius scales are, in a physical sense, arbitrary, and therefore, the ratio of two Celsius or Fahrenheit temperatures is not physically meaningful.

Describing Attributes by the Number of Values

An independent way of distinguishing between attributes is by the number of values they can take.

Discrete A discrete attribute has a finite or countably infinite set of values. Such attributes can be categorical, such as zip codes or ID numbers, or numeric, such as counts. Discrete attributes are often represented using integer variables. **Binary attributes** are a special case of discrete attributes and assume only two values, e.g., true/false, yes/no, male/female, or 0/1.

Binary attributes are often represented as Boolean variables, or as integer variables that only take the values 0 or 1.

Continuous A continuous attribute is one whose values are real numbers.

Examples include attributes such as temperature, height, or weight.

Continuous attributes are typically represented as floating-point variables.

Practically, real values can be measured and represented only with limited precision.

In theory, any of the measurement scale types—nominal, ordinal, interval, and ratio—could be combined with any of the types based on the number of attribute values—binary, discrete, and continuous. However, some combinations occur only infrequently or do not make much sense. For instance, it is difficult to think of a realistic data set that contains a continuous binary attribute. Typically, nominal and ordinal attributes are binary or discrete, while interval and ratio attributes are continuous. However, **count attributes**, which are discrete, are also ratio attributes.

Asymmetric Attributes

For asymmetric attributes, only presence—a non-zero attribute value—is regarded as important. Consider a data set in which each object is a student and each attribute records whether a student took a particular course at a university.

For a specific student, an attribute has a value of 1 if the student took the course associated with that attribute and a value of 0 otherwise.

Because students take only a small fraction of all available courses, most of the values in such a data set would be 0. Therefore, it is more meaningful and more efficient to focus on the non-zero values. To illustrate, if students are compared on the basis of the courses they don't take, then most students would seem very similar, at least if the number of courses is large. Binary attributes where only non-zero values are important are called **asymmetric**

binary attributes. This type of attribute is particularly important for association analysis, which is discussed in [Chapter 5](#). It is also possible to have discrete or continuous asymmetric features. For instance, if the number of credits associated with each course is recorded, then the resulting data set will consist of **asymmetric discrete or continuous attributes**.

General Comments on Levels of Measurement

As described in the rest of this chapter, there are many diverse types of data. The previous discussion of measurement scales, while useful, is not complete and has some limitations. We provide the following comments and guidance.

- **Distinctness, order, and meaningful intervals and ratios are only four properties of data—many others are possible.** For instance, some data is inherently cyclical, e.g., position on the surface of the Earth or time. As another example, consider set valued attributes, where each attribute value is a set of elements, e.g., the set of movies seen in the last year. Define one set of elements (movies) to be greater (larger) than a second set if the second set is a subset of the first. However, such a relationship defines only a partial order that does not match any of the attribute types just defined.
- **The numbers or symbols used to capture attribute values may not capture all the properties of the attributes or may suggest properties that are not there.** An illustration of this for integers was presented in [Example 2.3](#), i.e., averages of IDs and out of range ages.
- **Data is often transformed for the purpose of analysis**—see [Section 2.3.7](#). This often changes the distribution of the observed variable to a distribution that is easier to analyze, e.g., a Gaussian (normal) distribution. Often, such transformations only preserve the order of the original values, and other properties are lost. Nonetheless, if the desired outcome is a

statistical test of differences or a predictive model, such a transformation is justified.

- **The final evaluation of any data analysis, including operations on attributes, is whether the results make sense from a domain point of view.**

In summary, it can be challenging to determine which operations can be performed on a particular attribute or a collection of attributes without compromising the integrity of the analysis. Fortunately, established practice often serves as a reliable guide. Occasionally, however, standard practices are erroneous or have limitations.

2.1.2 Types of Data Sets

There are many types of data sets, and as the field of data mining develops and matures, a greater variety of data sets become available for analysis. In this section, we describe some of the most common types. For convenience, we have grouped the types of data sets into three groups: record data, graph-based data, and ordered data. These categories do not cover all possibilities and other groupings are certainly possible.

General Characteristics of Data Sets

Before providing details of specific kinds of data sets, we discuss three characteristics that apply to many data sets and have a significant impact on the data mining techniques that are used: dimensionality, distribution, and resolution.

Dimensionality

The dimensionality of a data set is the number of attributes that the objects in the data set possess. Analyzing data with a small number of dimensions tends to be qualitatively different from analyzing moderate or high-dimensional data. Indeed, the difficulties associated with the analysis of high-dimensional data are sometimes referred to as the **curse of dimensionality**. Because of this, an important motivation in preprocessing the data is **dimensionality reduction**. These issues are discussed in more depth later in this chapter and in Appendix B.

Distribution

The distribution of a data set is the frequency of occurrence of various values or sets of values for the attributes comprising data objects. Equivalently, the distribution of a data set can be considered as a description of the concentration of objects in various regions of the data space. Statisticians have enumerated many types of distributions, e.g., Gaussian (normal), and described their properties. (See Appendix C.) Although statistical approaches for describing distributions can yield powerful analysis techniques, many data sets have distributions that are not well captured by standard statistical distributions.

As a result, many data mining algorithms do not assume a particular statistical distribution for the data they analyze. However, some general aspects of distributions often have a strong impact. For example, suppose a categorical attribute is used as a class variable, where one of the categories occurs 95% of the time, while the other categories together occur only 5% of the time. This **skewness** in the distribution can make classification difficult as discussed in Section 4.11. (Skewness has other impacts on data analysis that are not discussed here.)

A special case of skewed data is **sparsity**. For sparse binary, count or continuous data, most attributes of an object have values of 0. In many cases, fewer than 1% of the values are non-zero. In practical terms, sparsity is an advantage because usually only the non-zero values need to be stored and manipulated. This results in significant savings with respect to computation time and storage. Indeed, some data mining algorithms, such as the association rule mining algorithms described in [Chapter 5](#), work well only for sparse data. Finally, note that often the attributes in sparse data sets are asymmetric attributes.

Resolution

It is frequently possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions. For instance, the surface of the Earth seems very uneven at a resolution of a few meters, but is relatively smooth at a resolution of tens of kilometers. The patterns in the data also depend on the level of resolution. If the resolution is too fine, a pattern may not be visible or may be buried in noise; if the resolution is too coarse, the pattern can disappear. For example, variations in atmospheric pressure on a scale of hours reflect the movement of storms and other weather systems. On a scale of months, such phenomena are not detectable.

Record Data

Much data mining work assumes that the data set is a collection of records (data objects), each of which consists of a fixed set of data fields (attributes). See [Figure 2.2\(a\)](#). For the most basic form of record data, there is no explicit relationship among records or data fields, and every record (object) has the same set of attributes. Record data is usually stored either in **flat** files or in relational databases. Relational databases are certainly more than a

collection of records, but data mining often does not use any of the additional information available in a relational database. Rather, the database serves as a convenient place to find records. Different types of record data are described below and are illustrated in **Figure 2.2**.

Tid	Refund	Marital Status	Taxable Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

(a) Record data.

TID	ITEMS
1	Bread, Soda, Milk
2	Beer, Bread
3	Beer, Soda, Diapers, Milk
4	Beer, Bread, Diapers, Milk
5	Soda, Diapers, Milk

(b) Transaction data.

Projection of x Load	Projection of y Load	Distance	Load	Thickness
10.23	5.27	15.22	27	1.2
12.65	6.25	16.22	22	1.1
13.54	7.23	17.34	23	1.2
14.27	8.43	18.45	25	0.9

(c) Data matrix.

	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

(d) Document-term matrix.

Figure 2.2.
Different variations of record data.

Transaction or Market Basket Data

Transaction data is a special type of record data, where each record (transaction) involves a set of items. Consider a grocery store. The set of products purchased by a customer during one shopping trip constitutes a transaction, while the individual products that were purchased are the items. This type of data is called **market basket data** because the items in each record are the products in a person's "market basket." Transaction data is a collection of sets of items, but it can be viewed as a set of records whose fields are asymmetric attributes. Most often, the attributes are binary, indicating whether an item was purchased, but more generally, the attributes can be discrete or continuous, such as the number of items purchased or the amount spent on those items. [Figure 2.2\(b\)](#) shows a sample transaction data set. Each row represents the purchases of a particular customer at a particular time.

The Data Matrix

If all the data objects in a collection of data have the same fixed set of numeric attributes, then the data objects can be thought of as points (vectors) in a multidimensional space, where each dimension represents a distinct attribute describing the object. A set of such data objects can be interpreted as an m by n matrix, where there are m rows, one for each object, and n columns, one for each attribute. (A representation that has data objects as columns and attributes as rows is also fine.) This matrix is called a **data matrix** or a **pattern matrix**. A data matrix is a variation of record data, but because it consists of numeric attributes, standard matrix operation can be applied to transform and manipulate the data. Therefore, the data matrix is the standard data format for most statistical data. [Figure 2.2\(c\)](#) shows a sample data matrix.

The Sparse Data Matrix

A sparse data matrix is a special case of a data matrix where the attributes are of the same type and are asymmetric; i.e., only non-zero values are important. Transaction data is an example of a sparse data matrix that has only 0–1 entries. Another common example is document data. In particular, if the order of the terms (words) in a document is ignored—the “bag of words” approach—then a document can be represented as a term vector, where each term is a component (attribute) of the vector and the value of each component is the number of times the corresponding term occurs in the document. This representation of a collection of documents is often called a **document-term matrix**. [Figure 2.2\(d\)](#) shows a sample document-term matrix. The documents are the rows of this matrix, while the terms are the columns. In practice, only the non-zero entries of sparse data matrices are stored.

Graph-Based Data

A graph can sometimes be a convenient and powerful representation for data. We consider two specific cases: (1) the graph captures relationships among data objects and (2) the data objects themselves are represented as graphs.

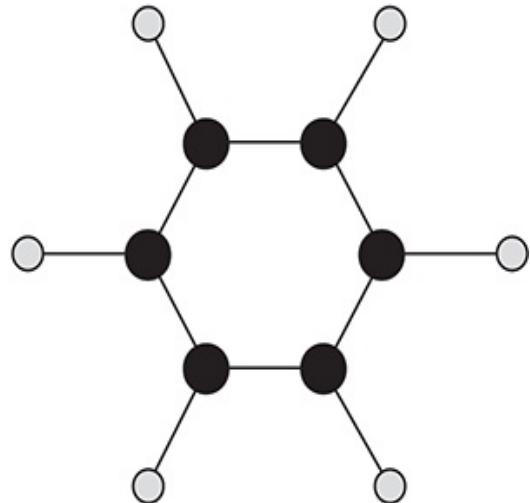
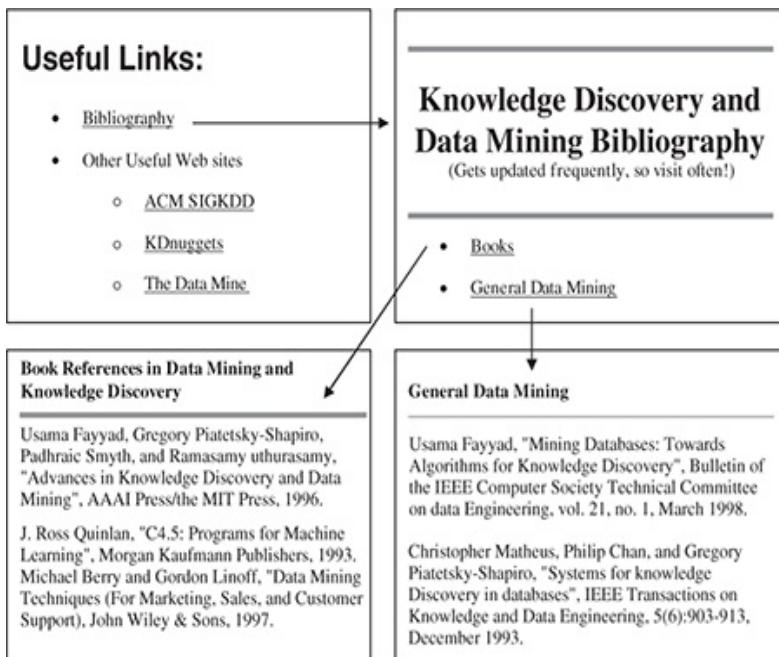
Data with Relationships among Objects

The relationships among objects frequently convey important information. In such cases, the data is often represented as a graph. In particular, the data objects are mapped to nodes of the graph, while the relationships among objects are captured by the links between objects and link properties, such as direction and weight. Consider web pages on the World Wide Web, which contain both text and links to other pages. In order to process search queries, web search engines collect and process web pages to extract their contents. It is well-known, however, that the links to and from each page provide a great deal of information about the relevance of a web page to a query, and thus, must also be taken into consideration. [Figure 2.3\(a\)](#) shows a set of linked

web pages. Another important example of such graph data are the social networks, where data objects are people and the relationships among them are their interactions via social media.

Data with Objects That Are Graphs

If objects have structure, that is, the objects contain subobjects that have relationships, then such objects are frequently represented as graphs. For example, the structure of chemical compounds can be represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds. [**Figure 2.3\(b\)**](#) shows a ball-and-stick diagram of the chemical compound benzene, which contains atoms of carbon (black) and hydrogen (gray). A graph representation makes it possible to determine which substructures occur frequently in a set of compounds and to ascertain whether the presence of any of these substructures is associated with the presence or absence of certain chemical properties, such as melting point or heat of formation. Frequent graph mining, which is a branch of data mining that analyzes such data, is considered in Section 6.5.



(a) Linked web pages.

(b) Benzene molecule.

Figure 2.3.

Different variations of graph data.

Ordered Data

For some types of data, the attributes have relationships that involve order in time or space. Different types of ordered data are described next and are shown in [Figure 2.4](#).

Sequential Transaction Data

Sequential transaction data can be thought of as an extension of transaction data, where each transaction has a time associated with it. Consider a retail transaction data set that also stores the time at which the transaction took place. This time information makes it possible to find patterns such as "candy sales peak before Halloween." A time can also be associated with each attribute. For example, each record could be the purchase history of a

customer, with a listing of items purchased at different times. Using this information, it is possible to find patterns such as “people who buy DVD players tend to buy DVDs in the period immediately following the purchase.”

Figure 2.4(a) shows an example of sequential transaction data. There are five different times— t_1 , t_2 , t_3 , t_4 , and t_5 ; three different customers—C1, C2, and C3; and five different items—A, B, C, D, and E. In the top table, each row corresponds to the items purchased at a particular time by each customer. For instance, at time t_3 , customer C2 purchased items A and D. In the bottom table, the same information is displayed, but each row corresponds to a particular customer. Each row contains information about each transaction involving the customer, where a transaction is considered to be a set of items and the time at which those items were purchased. For example, customer C3 bought items A and C at time t_2 .

Time Series Data

Time series data is a special type of ordered data where each record is a **time series**, i.e., a series of measurements taken over time. For example, a financial data set might contain objects that are time series of the daily prices of various stocks. As another example, consider **Figure 2.4(c)**, which shows a time series of the average monthly temperature for Minneapolis during the years 1982 to 1994. When working with temporal data, such as time series, it is important to consider **temporal autocorrelation**; i.e., if two measurements are close in time, then the values of those measurements are often very similar.

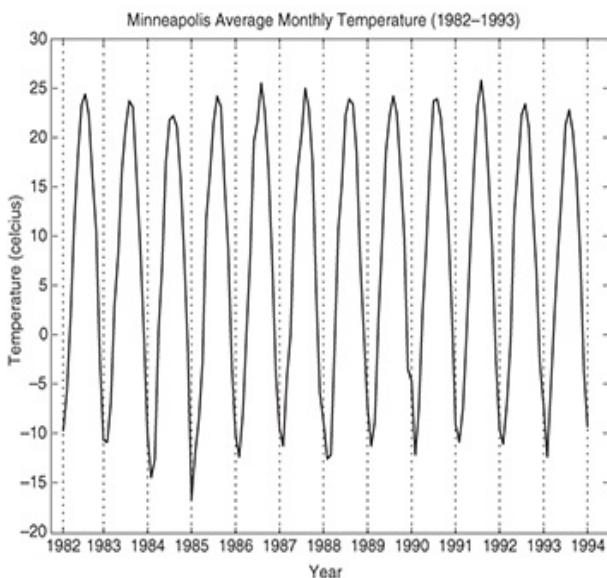
Time	Customer	Items Purchased
t1	C1	A, B
t2	C3	A, C
t2	C1	C, D
t3	C2	A, D
t4	C2	E
t5	C1	A, E

Customer	Time and Items Purchased
C1	(t1: A,B) (t2:C,D) (t5:A,E)
C2	(t3: A, D) (t4: E)
C3	(t2: A, C)

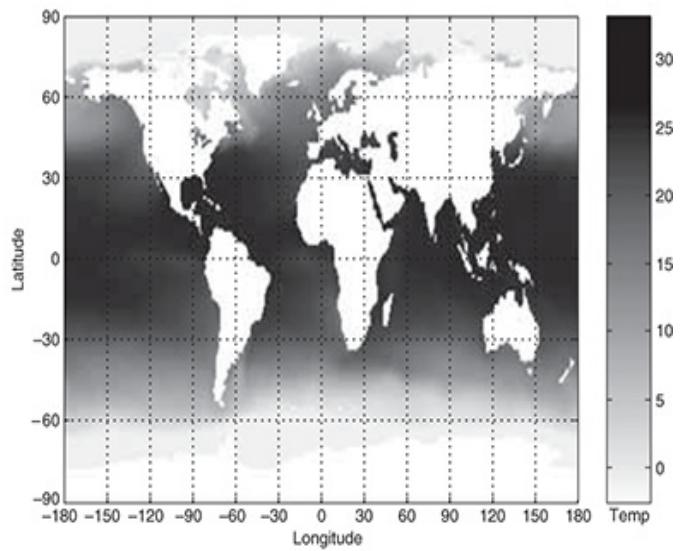
(a) Sequential transaction data.

GGTTCCGCCTTCAGCCCCGCGC
 CGCAGGGCCCCGCCCGCGCCGTC
 GAGAAGGGCCCAGCCTGGCGGGCG
 GGGGGAGGCAGGGCCGCCCCGAGC
 CCAACCGAGTCCGACCAGGTGCC
 CCCTCTGCTCGGCCTAGACCTGA
 GCTCATTAGGCAGCAGCGAACAG
 GCCAAGTAGAACACACGCGAAGCGC
 TGGGCTGCCTGCTGCGACCAGGG

(b) Genomic sequence data.



(c) Temperature time series.



(d) Spatial temperature data.

Figure 2.4.

Different variations of ordered data.

Sequence Data

Sequence data consists of a data set that is a sequence of individual entities, such as a sequence of words or letters. It is quite similar to sequential data, except that there are no time stamps; instead, there are positions in an ordered sequence. For example, the genetic information of plants and animals can be represented in the form of sequences of nucleotides that are known as genes. Many of the problems associated with genetic sequence data involve predicting similarities in the structure and function of genes from similarities in nucleotide sequences. [Figure 2.4\(b\)](#) shows a section of the human genetic code expressed using the four nucleotides from which all DNA is constructed: A, T, G, and C.

Spatial and Spatio-Temporal Data

Some objects have spatial attributes, such as positions or areas, in addition to other types of attributes. An example of spatial data is weather data (precipitation, temperature, pressure) that is collected for a variety of geographical locations. Often such measurements are collected over time, and thus, the data consists of time series at various locations. In that case, we refer to the data as spatio-temporal data. Although analysis can be conducted separately for each specific time or location, a more complete analysis of spatio-temporal data requires consideration of both the spatial and temporal aspects of the data.

An important aspect of spatial data is **spatial autocorrelation**; i.e., objects that are physically close tend to be similar in other ways as well. Thus, two points on the Earth that are close to each other usually have similar values for temperature and rainfall. Note that spatial autocorrelation is analogous to temporal autocorrelation.

Important examples of spatial and spatio-temporal data are the science and engineering data sets that are the result of measurements or model output

taken at regularly or irregularly distributed points on a two- or three-dimensional grid or mesh. For instance, Earth science data sets record the temperature or pressure measured at points (grid cells) on latitude–longitude spherical grids of various resolutions, e.g., 1° by 1° . See [Figure 2.4\(d\)](#). As another example, in the simulation of the flow of a gas, the speed and direction of flow at various instants in time can be recorded for each grid point in the simulation. A different type of spatio-temporal data arises from tracking the trajectories of objects, e.g., vehicles, in time and space.

Handling Non-Record Data

Most data mining algorithms are designed for record data or its variations, such as transaction data and data matrices. Record-oriented techniques can be applied to non-record data by extracting features from data objects and using these features to create a record corresponding to each object. Consider the chemical structure data that was described earlier. Given a set of common substructures, each compound can be represented as a record with binary attributes that indicate whether a compound contains a specific substructure. Such a representation is actually a transaction data set, where the transactions are the compounds and the items are the substructures.

In some cases, it is easy to represent the data in a record format, but this type of representation does not capture all the information in the data. Consider spatio-temporal data consisting of a time series from each point on a spatial grid. This data is often stored in a data matrix, where each row represents a location and each column represents a particular point in time. However, such a representation does not explicitly capture the time relationships that are present among attributes and the spatial relationships that exist among objects. This does not mean that such a representation is inappropriate, but rather that these relationships must be taken into consideration during the analysis. For example, it would not be a good idea to use a data mining

technique that ignores the temporal autocorrelation of the attributes or the spatial autocorrelation of the data objects, i.e., the locations on the spatial grid.

2.2 Data Quality

Data mining algorithms are often applied to data that was collected for another purpose, or for future, but unspecified applications. For that reason, data mining cannot usually take advantage of the significant benefits of “addressing quality issues at the source.” In contrast, much of statistics deals with the design of experiments or surveys that achieve a prespecified level of data quality. Because preventing data quality problems is typically not an option, data mining focuses on (1) the detection and correction of data quality problems and (2) the use of algorithms that can tolerate poor data quality. The first step, detection and correction, is often called **data cleaning**.

The following sections discuss specific aspects of data quality. The focus is on measurement and data collection issues, although some application-related issues are also discussed.

2.2.1 Measurement and Data Collection Issues

It is unrealistic to expect that data will be perfect. There may be problems due to human error, limitations of measuring devices, or flaws in the data collection process. Values or even entire data objects can be missing. In other cases, there can be spurious or duplicate objects; i.e., multiple data objects that all correspond to a single “real” object. For example, there might be two different records for a person who has recently lived at two different addresses. Even if

all the data is present and “looks fine,” there may be inconsistencies—a person has a height of 2 meters, but weighs only 2 kilograms.

In the next few sections, we focus on aspects of data quality that are related to data measurement and collection. We begin with a definition of measurement and data collection errors and then consider a variety of problems that involve measurement error: noise, artifacts, bias, precision, and accuracy. We conclude by discussing data quality issues that involve both measurement and data collection problems: outliers, missing and inconsistent values, and duplicate data.

Measurement and Data Collection Errors

The term **measurement error** refers to any problem resulting from the measurement process. A common problem is that the value recorded differs from the true value to some extent. For continuous attributes, the numerical difference of the measured and true value is called the **error**. The term **data collection error** refers to errors such as omitting data objects or attribute values, or inappropriately including a data object. For example, a study of animals of a certain species might include animals of a related species that are similar in appearance to the species of interest. Both measurement errors and data collection errors can be either systematic or random.

We will only consider general types of errors. Within particular domains, certain types of data errors are commonplace, and well-developed techniques often exist for detecting and/or correcting these errors. For example, keyboard errors are common when data is entered manually, and as a result, many data entry programs have techniques for detecting and, with human intervention, correcting such errors.

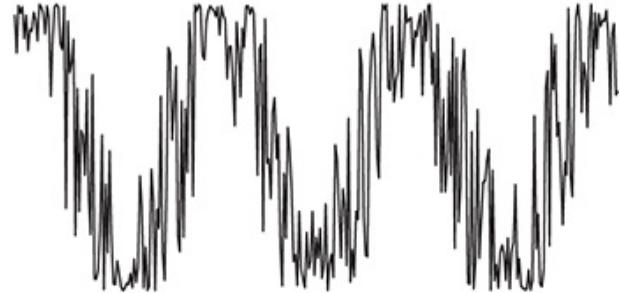
Noise and Artifacts

Noise is the random component of a measurement error. It typically involves the distortion of a value or the addition of spurious objects. [Figure 2.5](#) shows a time series before and after it has been disrupted by random noise. If a bit more noise were added to the time series, its shape would be lost.

[Figure 2.6](#) shows a set of data points before and after some noise points (indicated by '+'s) have been added. Notice that some of the noise points are intermixed with the non-noise points.



(a) Time series.



(b) Time series with noise.

Figure 2.5.
Noise in a time series context.



(a) Three groups of points.

(b) With noise points (+) added.

Figure 2.6.

Noise in a spatial context.

The term noise is often used in connection with data that has a spatial or temporal component. In such cases, techniques from signal or image processing can frequently be used to reduce noise and thus, help to discover patterns (signals) that might be “lost in the noise.” Nonetheless, the elimination of noise is frequently difficult, and much work in data mining focuses on devising **robust algorithms** that produce acceptable results even when noise is present.

Data errors can be the result of a more deterministic phenomenon, such as a streak in the same place on a set of photographs. Such deterministic distortions of the data are often referred to as **artifacts**.

Precision, Bias, and Accuracy

In statistics and experimental science, the quality of the measurement process and the resulting data are measured by precision and bias. We provide the

standard definitions, followed by a brief discussion. For the following definitions, we assume that we make repeated measurements of the same underlying quantity.

Definition 2.3 (Precision).

The closeness of repeated measurements (of the same quantity) to one another.

Definition 2.4 (Bias).

A systematic variation of measurements from the quantity being measured.

Precision is often measured by the standard deviation of a set of values, while bias is measured by taking the difference between the mean of the set of values and the known value of the quantity being measured. Bias can be determined only for objects whose measured quantity is known by means external to the current situation. Suppose that we have a standard laboratory weight with a mass of 1g and want to assess the precision and bias of our new laboratory scale. We weigh the mass five times, and obtain the following five values:{ 1.015, 0.990, 1.013, 1.001, 0.986}. The mean of these values is

1.001, and hence, the bias is 0.001. The precision, as measured by the standard deviation, is 0.013.

It is common to use the more general term, **accuracy**, to refer to the degree of measurement error in data.

Definition 2.5 (Accuracy)

The closeness of measurements to the true value of the quantity being measured.

Accuracy depends on precision and bias, but there is no specific formula for accuracy in terms of these two quantities.

One important aspect of accuracy is the use of **significant digits**. The goal is to use only as many digits to represent the result of a measurement or calculation as are justified by the precision of the data. For example, if the length of an object is measured with a meter stick whose smallest markings are millimeters, then we should record the length of data only to the nearest millimeter. The precision of such a measurement would be $\pm 0.5\text{mm}$. We do not review the details of working with significant digits because most readers will have encountered them in previous courses and they are covered in considerable depth in science, engineering, and statistics textbooks.

Issues such as significant digits, precision, bias, and accuracy are sometimes overlooked, but they are important for data mining as well as statistics and science. Many times, data sets do not come with information about the

precision of the data, and furthermore, the programs used for analysis return results without any such information. Nonetheless, without some understanding of the accuracy of the data and the results, an analyst runs the risk of committing serious data analysis blunders.

Outliers

Outliers are either (1) data objects that, in some sense, have characteristics that are different from most of the other data objects in the data set, or (2) values of an attribute that are unusual with respect to the typical values for that attribute. Alternatively, they can be referred to as **anomalous** objects or values. There is considerable leeway in the definition of an outlier, and many different definitions have been proposed by the statistics and data mining communities. Furthermore, it is important to distinguish between the notions of noise and outliers. Unlike noise, outliers can be legitimate data objects or values that we are interested in detecting. For instance, in fraud and network intrusion detection, the goal is to find unusual objects or events from among a large number of normal ones. [Chapter 9](#) discusses anomaly detection in more detail.

Missing Values

It is not unusual for an object to be missing one or more attribute values. In some cases, the information was not collected; e.g., some people decline to give their age or weight. In other cases, some attributes are not applicable to all objects; e.g., often, forms have conditional parts that are filled out only when a person answers a previous question in a certain way, but for simplicity, all fields are stored. Regardless, missing values should be taken into account during the data analysis.

There are several strategies (and variations on these strategies) for dealing with missing data, each of which is appropriate in certain circumstances. These strategies are listed next, along with an indication of their advantages and disadvantages.

Eliminate Data Objects or Attributes

A simple and effective strategy is to eliminate objects with missing values. However, even a partially specified data object contains some information, and if many objects have missing values, then a reliable analysis can be difficult or impossible. Nonetheless, if a data set has only a few objects that have missing values, then it may be expedient to omit them. A related strategy is to eliminate attributes that have missing values. This should be done with caution, however, because the eliminated attributes may be the ones that are critical to the analysis.

Estimate Missing Values

Sometimes missing data can be reliably estimated. For example, consider a time series that changes in a reasonably smooth fashion, but has a few, widely scattered missing values. In such cases, the missing values can be estimated (interpolated) by using the remaining values. As another example, consider a data set that has many similar data points. In this situation, the attribute values of the points closest to the point with the missing value are often used to estimate the missing value. If the attribute is continuous, then the average attribute value of the nearest neighbors is used; if the attribute is categorical, then the most commonly occurring attribute value can be taken. For a concrete illustration, consider precipitation measurements that are recorded by ground stations. For areas not containing a ground station, the precipitation can be estimated using values observed at nearby ground stations.

Ignore the Missing Value during Analysis

Many data mining approaches can be modified to ignore missing values. For example, suppose that objects are being clustered and the similarity between pairs of data objects needs to be calculated. If one or both objects of a pair have missing values for some attributes, then the similarity can be calculated by using only the attributes that do not have missing values. It is true that the similarity will only be approximate, but unless the total number of attributes is small or the number of missing values is high, this degree of inaccuracy may not matter much. Likewise, many classification schemes can be modified to work with missing values.

Inconsistent Values

Data can contain inconsistent values. Consider an address field, where both a zip code and city are listed, but the specified zip code area is not contained in that city. It is possible that the individual entering this information transposed two digits, or perhaps a digit was misread when the information was scanned from a handwritten form. Regardless of the cause of the inconsistent values, it is important to detect and, if possible, correct such problems.

Some types of inconsistencies are easy to detect. For instance, a person's height should not be negative. In other cases, it can be necessary to consult an external source of information. For example, when an insurance company processes claims for reimbursement, it checks the names and addresses on the reimbursement forms against a database of its customers.

Once an inconsistency has been detected, it is sometimes possible to correct the data. A product code may have "check" digits, or it may be possible to double-check a product code against a list of known product codes, and then

correct the code if it is incorrect, but close to a known code. The correction of an inconsistency requires additional or redundant information.

Example 2.6 (Inconsistent Sea Surface Temperature).

This example illustrates an inconsistency in actual time series data that measures the sea surface temperature (SST) at various points on the ocean. SST data was originally collected using ocean-based measurements from ships or buoys, but more recently, satellites have been used to gather the data. To create a long-term data set, both sources of data must be used. However, because the data comes from different sources, the two parts of the data are subtly different. This discrepancy is visually displayed in [Figure 2.7](#), which shows the correlation of SST values between pairs of years. If a pair of years has a positive correlation, then the location corresponding to the pair of years is colored white; otherwise it is colored black. (Seasonal variations were removed from the data since, otherwise, all the years would be highly correlated.) There is a distinct change in behavior where the data has been put together in 1983. Years within each of the two groups, 1958–1982 and 1983–1999, tend to have a positive correlation with one another, but a negative correlation with years in the other group. This does not mean that this data should not be used, only that the analyst should consider the potential impact of such discrepancies on the data mining analysis.

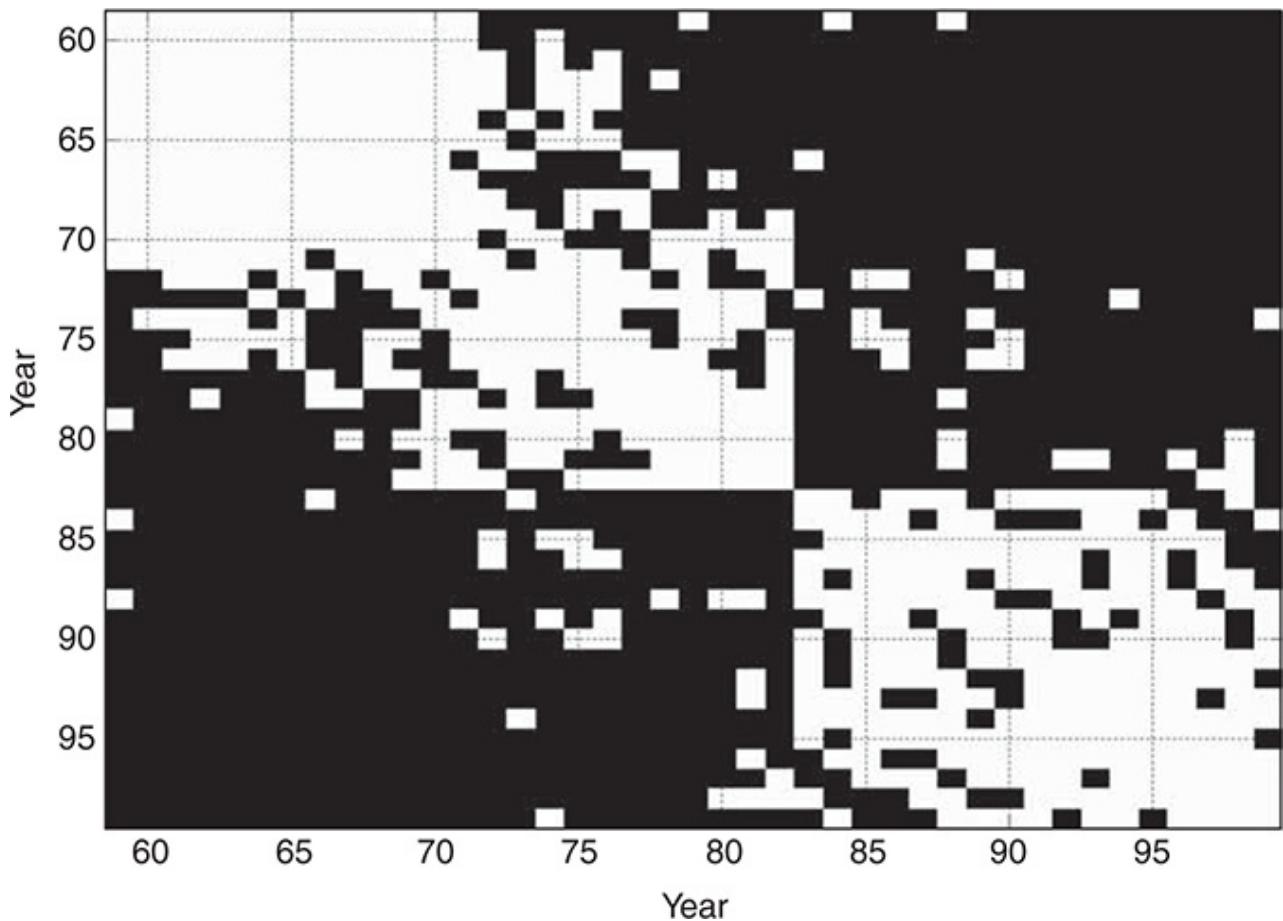


Figure 2.7.

Correlation of SST data between pairs of years. White areas indicate positive correlation. Black areas indicate negative correlation.

Duplicate Data

A data set can include data objects that are duplicates, or almost duplicates, of one another. Many people receive duplicate mailings because they appear in a database multiple times under slightly different names. To detect and eliminate such duplicates, two main issues must be addressed. First, if there are two objects that actually represent a single object, then one or more values of corresponding attributes are usually different, and these inconsistent values must be resolved. Second, care needs to be taken to avoid accidentally combining data objects that are similar, but not duplicates, such

as two distinct people with identical names. The term **deduplication** is often used to refer to the process of dealing with these issues.

In some cases, two or more objects are identical with respect to the attributes measured by the database, but they still represent different objects. Here, the duplicates are legitimate, but can still cause problems for some algorithms if the possibility of identical objects is not specifically accounted for in their design. An example of this is given in [Exercise 13](#) on page [108](#).

2.2.2 Issues Related to Applications

Data quality issues can also be considered from an application viewpoint as expressed by the statement “data is of high quality if it is suitable for its intended use.” This approach to data quality has proven quite useful, particularly in business and industry. A similar viewpoint is also present in statistics and the experimental sciences, with their emphasis on the careful design of experiments to collect the data relevant to a specific hypothesis. As with quality issues at the measurement and data collection level, many issues are specific to particular applications and fields. Again, we consider only a few of the general issues.

Timeliness

Some data starts to age as soon as it has been collected. In particular, if the data provides a snapshot of some ongoing phenomenon or process, such as the purchasing behavior of customers or web browsing patterns, then this snapshot represents reality for only a limited time. If the data is out of date, then so are the models and patterns that are based on it.

Relevance

The available data must contain the information necessary for the application. Consider the task of building a model that predicts the accident rate for drivers. If information about the age and gender of the driver is omitted, then it is likely that the model will have limited accuracy unless this information is indirectly available through other attributes.

Making sure that the objects in a data set are relevant is also challenging. A common problem is **sampling bias**, which occurs when a sample does not contain different types of objects in proportion to their actual occurrence in the population. For example, survey data describes only those who respond to the survey. (Other aspects of sampling are discussed further in [Section 2.3.2](#).) Because the results of a data analysis can reflect only the data that is present, sampling bias will typically lead to erroneous results when applied to the broader population.

Knowledge about the Data

Ideally, data sets are accompanied by documentation that describes different aspects of the data; the quality of this documentation can either aid or hinder the subsequent analysis. For example, if the documentation identifies several attributes as being strongly related, these attributes are likely to provide highly redundant information, and we usually decide to keep just one. (Consider sales tax and purchase price.) If the documentation is poor, however, and fails to tell us, for example, that the missing values for a particular field are indicated with a -9999, then our analysis of the data may be faulty. Other important characteristics are the precision of the data, the type of features (nominal, ordinal, interval, ratio), the scale of measurement (e.g., meters or feet for length), and the origin of the data.

2.3 Data Preprocessing

In this section, we consider which preprocessing steps should be applied to make the data more suitable for data mining. Data preprocessing is a broad area and consists of a number of different strategies and techniques that are interrelated in complex ways. We will present some of the most important ideas and approaches, and try to point out the interrelationships among them. Specifically, we will discuss the following topics:

- Aggregation
- Sampling
- Dimensionality reduction
- Feature subset selection
- Feature creation
- Discretization and binarization
- Variable transformation

Roughly speaking, these topics fall into two categories: selecting data objects and attributes for the analysis or for creating/changing the attributes. In both cases, the goal is to improve the data mining analysis with respect to time, cost, and quality. Details are provided in the following sections.

A quick note about terminology: In the following, we sometimes use synonyms for attribute, such as feature or variable, in order to follow common usage.

2.3.1 Aggregation

Sometimes “less is more,” and this is the case with **aggregation**, the combining of two or more objects into a single object. Consider a data set consisting of transactions (data objects) recording the daily sales of products in various store locations (Minneapolis, Chicago, Paris, ...) for different days over the course of a year. See [Table 2.4](#). One way to aggregate transactions for this data set is to replace all the transactions of a single store with a single storewide transaction. This reduces the hundreds or thousands of transactions that occur daily at a specific store to a single daily transaction, and the number of data objects per day is reduced to the number of stores.

Table 2.4. Data set containing information about customer purchases.

Transaction ID	Item	Store Location	Date	Price	...
:	:	:	:	:	
101123	Watch	Chicago	09/06/04	\$25.99	...
101123	Battery	Chicago	09/06/04	\$5.99	...
101124	Shoes	Minneapolis	09/06/04	\$75.00	...

An obvious issue is how an aggregate transaction is created; i.e., how the values of each attribute are combined across all the records corresponding to a particular location to create the aggregate transaction that represents the sales of a single store or date. Quantitative attributes, such as *price*, are typically aggregated by taking a sum or an average. A qualitative attribute, such as *item*, can either be omitted or summarized in terms of a higher level category, e.g., televisions versus electronics.

The data in [Table 2.4](#) can also be viewed as a multidimensional array, where each attribute is a dimension. From this viewpoint, aggregation is the process of eliminating attributes, such as the type of item, or reducing the

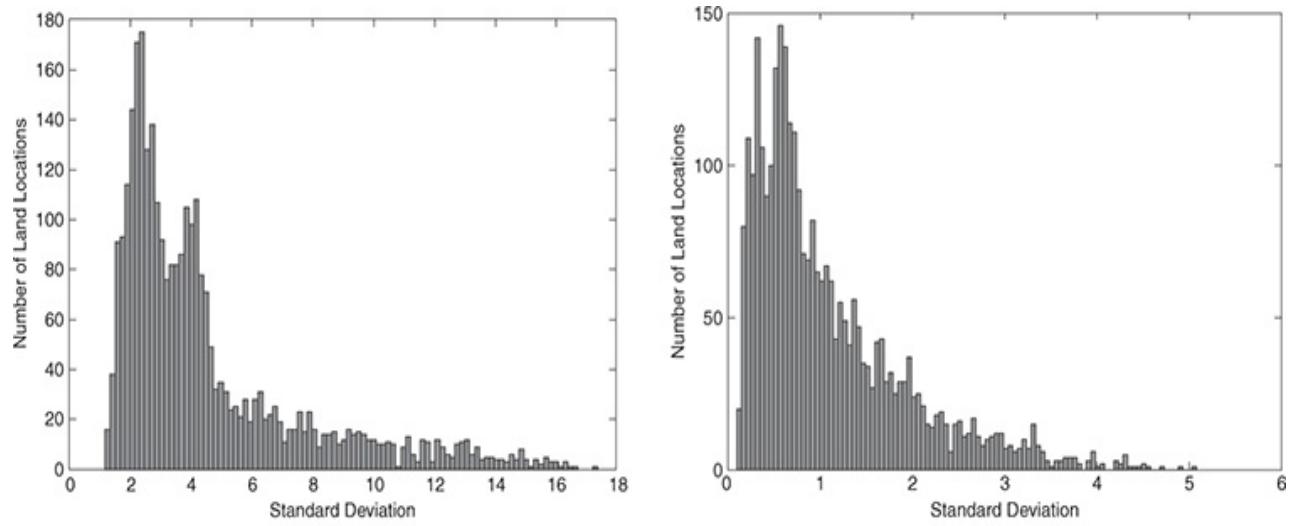
number of values for a particular attribute; e.g., reducing the possible values for *date* from 365 days to 12 months. This type of aggregation is commonly used in Online Analytical Processing (OLAP). References to OLAP are given in the bibliographic Notes.

There are several motivations for aggregation. First, the smaller data sets resulting from data reduction require less memory and processing time, and hence, aggregation often enables the use of more expensive data mining algorithms. Second, aggregation can act as a change of scope or scale by providing a high-level view of the data instead of a low-level view. In the previous example, aggregating over store locations and months gives us a monthly, per store view of the data instead of a daily, per item view. Finally, the behavior of groups of objects or attributes is often more stable than that of individual objects or attributes. This statement reflects the statistical fact that aggregate quantities, such as averages or totals, have less variability than the individual values being aggregated. For totals, the actual amount of variation is larger than that of individual objects (on average), but the percentage of the variation is smaller, while for means, the actual amount of variation is less than that of individual objects (on average). A disadvantage of aggregation is the potential loss of interesting details. In the store example, aggregating over months loses information about which day of the week has the highest sales.

Example 2.7 (Australian Precipitation).

This example is based on precipitation in Australia from the period 1982–1993. **Figure 2.8(a)** shows a histogram for the standard deviation of average monthly precipitation for 3,030 0.5° by 0.5° grid cells in Australia, while **Figure 2.8(b)** shows a histogram for the standard deviation of the average yearly precipitation for the same locations. The average yearly precipitation has less variability than the average monthly precipitation. All

precipitation measurements (and their standard deviations) are in centimeters.



(a) Histogram of standard deviation of average monthly precipitation

(b) Histogram of standard deviation of average yearly precipitation

Figure 2.8.

Histograms of standard deviation for monthly and yearly precipitation in Australia for the period 1982–1993.

2.3.2 Sampling

Sampling is a commonly used approach for selecting a subset of the data objects to be analyzed. In statistics, it has long been used for both the preliminary investigation of the data and the final data analysis. Sampling can also be very useful in data mining. However, the motivations for sampling in statistics and data mining are often different. Statisticians use sampling because obtaining the entire set of data of interest is too expensive or time consuming, while data miners usually sample because it is too computationally expensive in terms of the memory or time required to process

all the data. In some cases, using a sampling algorithm can reduce the data size to the point where a better, but more computationally expensive algorithm can be used.

The key principle for effective sampling is the following: Using a sample will work almost as well as using the entire data set if the sample is representative. In turn, **a sample is representative** if it has approximately the same property (of interest) as the original set of data. If the mean (average) of the data objects is the property of interest, then a sample is representative if it has a mean that is close to that of the original data. Because sampling is a statistical process, the representativeness of any particular sample will vary, and the best that we can do is choose a sampling scheme that guarantees a high probability of getting a representative sample. As discussed next, this involves choosing the appropriate sample size and sampling technique.

Sampling Approaches

There are many sampling techniques, but only a few of the most basic ones and their variations will be covered here. The simplest type of sampling is **simple random sampling**. For this type of sampling, there is an equal probability of selecting any particular object. There are two variations on random sampling (and other sampling techniques as well): (1) **sampling without replacement** —as each object is selected, it is removed from the set of all objects that together constitute the **population**, and (2) **sampling with replacement** —objects are not removed from the population as they are selected for the sample. In sampling with replacement, the same object can be picked more than once. The samples produced by the two methods are not much different when samples are relatively small compared to the data set size, but sampling with replacement is simpler to analyze because the probability of selecting any object remains constant during the sampling process.

When the population consists of different types of objects, with widely different numbers of objects, simple random sampling can fail to adequately represent those types of objects that are less frequent. This can cause problems when the analysis requires proper representation of all object types. For example, when building classification models for rare classes, it is critical that the rare classes be adequately represented in the sample. Hence, a sampling scheme that can accommodate differing frequencies for the object types of interest is needed. **Stratified sampling**, which starts with prespecified groups of objects, is such an approach. In the simplest version, equal numbers of objects are drawn from each group even though the groups are of different sizes. In another variation, the number of objects drawn from each group is proportional to the size of that group.

Example 2.8 (Sampling and Loss of Information).

Once a sampling technique has been selected, it is still necessary to choose the sample size. Larger sample sizes increase the probability that a sample will be representative, but they also eliminate much of the advantage of sampling. Conversely, with smaller sample sizes, patterns can be missed or erroneous patterns can be detected. [Figure 2.9\(a\)](#) shows a data set that contains 8000 two-dimensional points, while [Figures 2.9\(b\)](#) and [2.9\(c\)](#) show samples from this data set of size 2000 and 500, respectively. Although most of the structure of this data set is present in the sample of 2000 points, much of the structure is missing in the sample of 500 points.

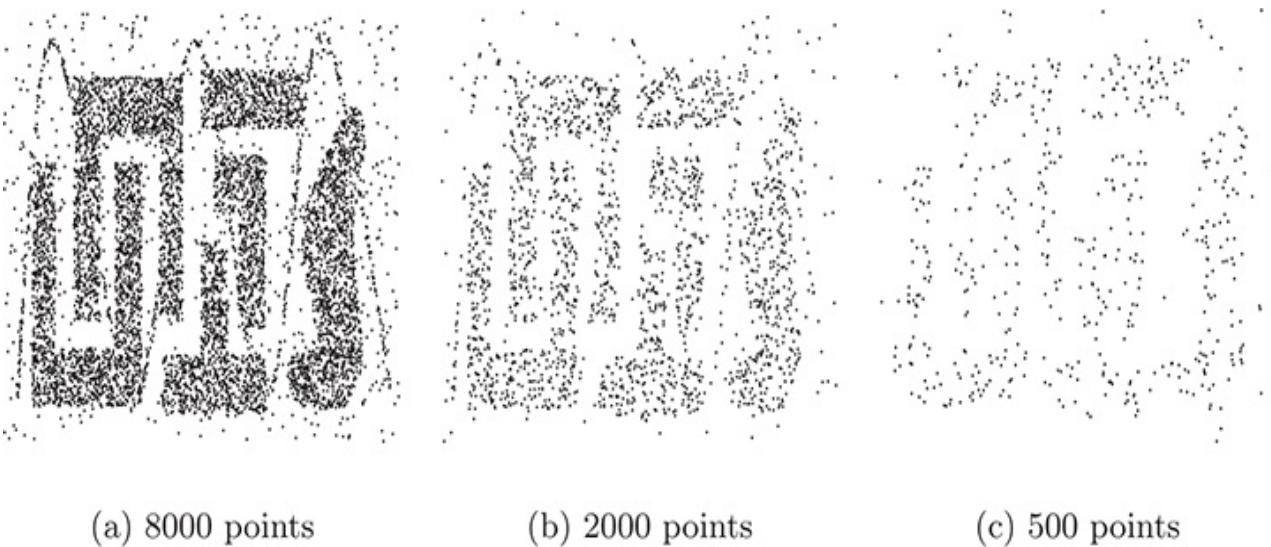
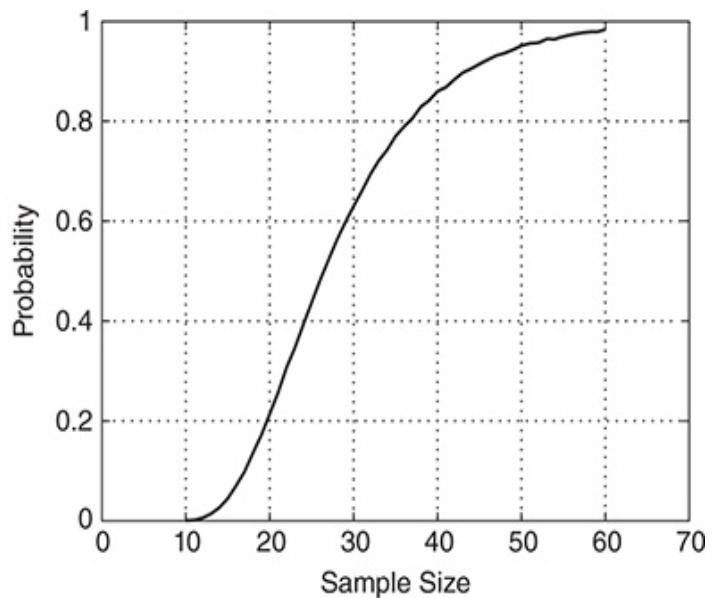
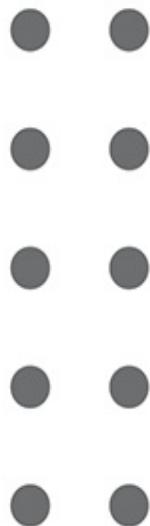


Figure 2.9.
Example of the loss of structure with sampling.

Example 2.9 (Determining the Proper Sample Size).

To illustrate that determining the proper sample size requires a methodical approach, consider the following task.

Given a set of data consisting of a small number of almost equalsized groups, find at least one representative point for each of the groups. Assume that the objects in each group are highly similar to each other, but not very similar to objects in different groups. **Figure 2.10(a)** shows an idealized set of clusters (groups) from which these points might be drawn.



(a) Ten groups of points.

(b) Probability a sample contains points from each of 10 groups.

Figure 2.10.

Finding representative points from 10 groups.

This problem can be efficiently solved using sampling. One approach is to take a small sample of data points, compute the pairwise similarities between points, and then form groups of points that are highly similar. The desired set of representative points is then obtained by taking one point from each of these groups. To follow this approach, however, we need to determine a sample size that would guarantee, with a high probability, the desired outcome; that is, that at least one point will be obtained from each cluster. **Figure 2.10(b)** shows the probability of getting one object from each of the 10 groups as the sample size runs from 10 to 60. Interestingly, with a sample size of 20, there is little chance (20%) of getting a sample that includes all 10 clusters. Even with a sample size of 30, there is still a moderate chance (almost 40%) of getting a sample that doesn't contain objects from all 10 clusters. This issue is further explored in the context of clustering by **Exercise 4** on page 603.

Progressive Sampling

The proper sample size can be difficult to determine, so **adaptive** or **progressive sampling** schemes are sometimes used. These approaches start with a small sample, and then increase the sample size until a sample of sufficient size has been obtained. While this technique eliminates the need to determine the correct sample size initially, it requires that there be a way to evaluate the sample to judge if it is large enough.

Suppose, for instance, that progressive sampling is used to learn a predictive model. Although the accuracy of predictive models increases as the sample size increases, at some point the increase in accuracy levels off. We want to stop increasing the sample size at this leveling-off point. By keeping track of the change in accuracy of the model as we take progressively larger samples, and by taking other samples close to the size of the current one, we can get an estimate of how close we are to this leveling-off point, and thus, stop sampling.

2.3.3 Dimensionality Reduction

Data sets can have a large number of features. Consider a set of documents, where each document is represented by a vector whose components are the frequencies with which each word occurs in the document. In such cases, there are typically thousands or tens of thousands of attributes (components), one for each word in the vocabulary. As another example, consider a set of time series consisting of the daily closing price of various stocks over a period of 30 years. In this case, the attributes, which are the prices on specific days, again number in the thousands.

There are a variety of benefits to dimensionality reduction. A key benefit is that many data mining algorithms work better if the dimensionality—the number of attributes in the data—is lower. This is partly because dimensionality reduction can eliminate irrelevant features and reduce noise and partly because of the curse of dimensionality, which is explained below. Another benefit is that a reduction of dimensionality can lead to a more understandable model because the model usually involves fewer attributes. Also, dimensionality reduction may allow the data to be more easily visualized. Even if dimensionality reduction doesn't reduce the data to two or three dimensions, data is often visualized by looking at pairs or triplets of attributes, and the number of such combinations is greatly reduced. Finally, the amount of time and memory required by the data mining algorithm is reduced with a reduction in dimensionality.

The term dimensionality reduction is often reserved for those techniques that reduce the dimensionality of a data set by creating new attributes that are a combination of the old attributes. The reduction of dimensionality by selecting attributes that are a subset of the old is known as feature subset selection or feature selection. It will be discussed in [Section 2.3.4](#).

In the remainder of this section, we briefly introduce two important topics: the curse of dimensionality and dimensionality reduction techniques based on linear algebra approaches such as principal components analysis (PCA). More details on dimensionality reduction can be found in Appendix B.

The Curse of Dimensionality

The curse of dimensionality refers to the phenomenon that many types of data analysis become significantly harder as the dimensionality of the data increases. Specifically, as dimensionality increases, the data becomes increasingly sparse in the space that it occupies. Thus, the data objects we

observe are quite possibly not a representative sample of all possible objects. For classification, this can mean that there are not enough data objects to allow the creation of a model that reliably assigns a class to all possible objects. For clustering, the differences in density and in the distances between points, which are critical for clustering, become less meaningful. (This is discussed further in Sections 8.1.2, 8.4.6, and 8.4.8.) As a result, many clustering and classification algorithms (and other data analysis algorithms) have trouble with high-dimensional data leading to reduced classification accuracy and poor quality clusters.

Linear Algebra Techniques for Dimensionality Reduction

Some of the most common approaches for dimensionality reduction, particularly for continuous data, use techniques from linear algebra to project the data from a high-dimensional space into a lower-dimensional space.

Principal Components Analysis (PCA) is a linear algebra technique for continuous attributes that finds new attributes (principal components) that (1) are linear combinations of the original attributes, (2) are **orthogonal** (perpendicular) to each other, and (3) capture the maximum amount of variation in the data. For example, the first two principal components capture as much of the variation in the data as is possible with two orthogonal attributes that are linear combinations of the original attributes. **Singular Value Decomposition (SVD)** is a linear algebra technique that is related to PCA and is also commonly used for dimensionality reduction. For additional details, see Appendices A and B.

2.3.4 Feature Subset Selection

Another way to reduce the dimensionality is to use only a subset of the features. While it might seem that such an approach would lose information, this is not the case if redundant and irrelevant features are present.

Redundant features duplicate much or all of the information contained in one or more other attributes. For example, the purchase price of a product and the amount of sales tax paid contain much of the same information. **Irrelevant features** contain almost no useful information for the data mining task at hand. For instance, students' ID numbers are irrelevant to the task of predicting students' grade point averages. Redundant and irrelevant features can reduce classification accuracy and the quality of the clusters that are found.

While some irrelevant and redundant attributes can be eliminated immediately by using common sense or domain knowledge, selecting the best subset of features frequently requires a systematic approach. The ideal approach to feature selection is to try all possible subsets of features as input to the data mining algorithm of interest, and then take the subset that produces the best results. This method has the advantage of reflecting the objective and bias of the data mining algorithm that will eventually be used. Unfortunately, since the number of subsets involving n attributes is 2^n , such an approach is impractical in most situations and alternative strategies are needed. There are three standard approaches to feature selection: embedded, filter, and wrapper.

Embedded approaches

Feature selection occurs naturally as part of the data mining algorithm. Specifically, during the operation of the data mining algorithm, the algorithm itself decides which attributes to use and which to ignore. Algorithms for building decision tree classifiers, which are discussed in [Chapter 3](#), often operate in this manner.

Filter approaches

Features are selected before the data mining algorithm is run, using some approach that is independent of the data mining task. For example, we might select sets of attributes whose pairwise correlation is as low as possible so that the attributes are non-redundant.

Wrapper approaches

These methods use the target data mining algorithm as a black box to find the best subset of attributes, in a way similar to that of the ideal algorithm described above, but typically without enumerating all possible subsets.

Because the embedded approaches are algorithm-specific, only the filter and wrapper approaches will be discussed further here.

An Architecture for Feature Subset Selection

It is possible to encompass both the filter and wrapper approaches within a common architecture. The feature selection process is viewed as consisting of four parts: a measure for evaluating a subset, a search strategy that controls the generation of a new subset of features, a stopping criterion, and a validation procedure. Filter methods and wrapper methods differ only in the way in which they evaluate a subset of features. For a wrapper method, subset evaluation uses the target data mining algorithm, while for a filter approach, the evaluation technique is distinct from the target data mining algorithm. The following discussion provides some details of this approach, which is summarized in [Figure 2.11](#).

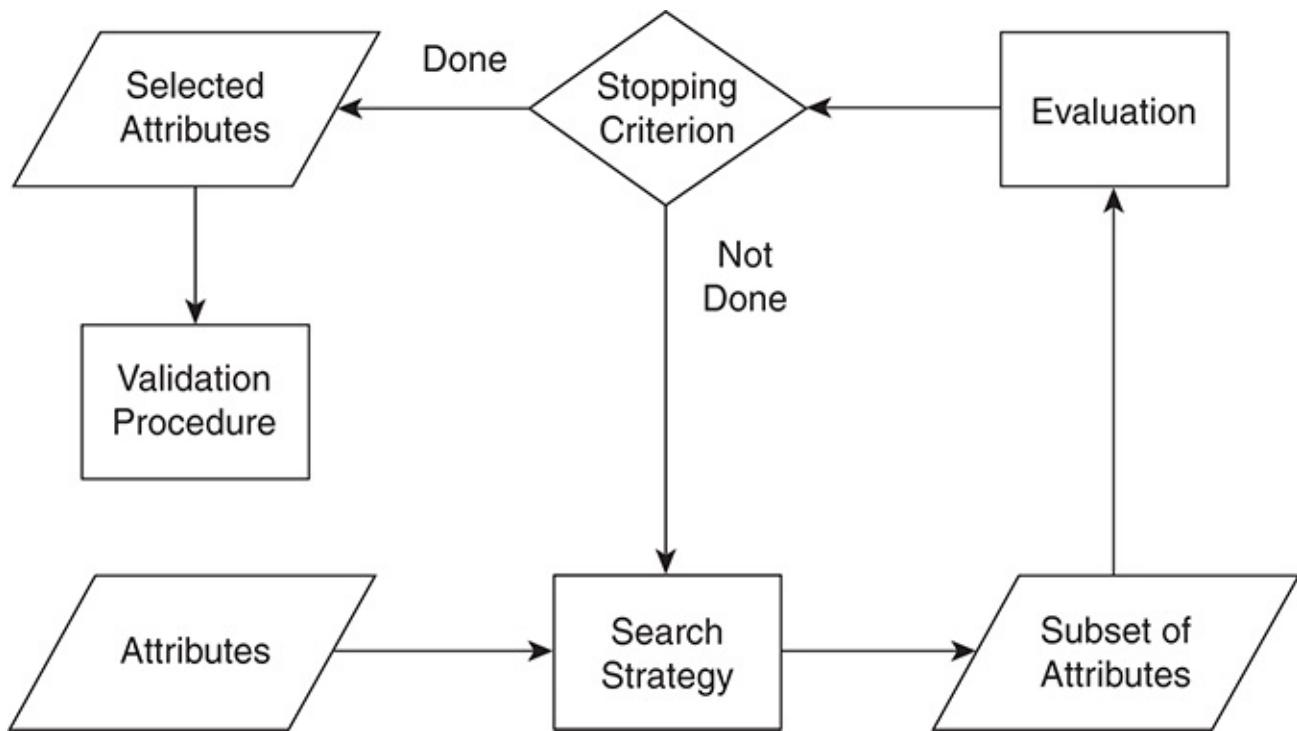


Figure 2.11.

Flowchart of a feature subset selection process.

Conceptually, feature subset selection is a search over all possible subsets of features. Many different types of search strategies can be used, but the search strategy should be computationally inexpensive and should find optimal or near optimal sets of features. It is usually not possible to satisfy both requirements, and thus, trade-offs are necessary.

An integral part of the search is an evaluation step to judge how the current subset of features compares to others that have been considered. This requires an evaluation measure that attempts to determine the goodness of a subset of attributes with respect to a particular data mining task, such as classification or clustering. For the filter approach, such measures attempt to predict how well the actual data mining algorithm will perform on a given set of attributes. For the wrapper approach, where evaluation consists of actually running the target data mining algorithm, the subset evaluation function is simply the criterion normally used to measure the result of the data mining.

Because the number of subsets can be enormous and it is impractical to examine them all, some sort of stopping criterion is necessary. This strategy is usually based on one or more conditions involving the following: the number of iterations, whether the value of the subset evaluation measure is optimal or exceeds a certain threshold, whether a subset of a certain size has been obtained, and whether any improvement can be achieved by the options available to the search strategy.

Finally, once a subset of features has been selected, the results of the target data mining algorithm on the selected subset should be validated. A straightforward validation approach is to run the algorithm with the full set of features and compare the full results to results obtained using the subset of features. Hopefully, the subset of features will produce results that are better than or almost as good as those produced when using all features. Another validation approach is to use a number of different feature selection algorithms to obtain subsets of features and then compare the results of running the data mining algorithm on each subset.

Feature Weighting

Feature weighting is an alternative to keeping or eliminating features. More important features are assigned a higher weight, while less important features are given a lower weight. These weights are sometimes assigned based on domain knowledge about the relative importance of features. Alternatively, they can sometimes be determined automatically. For example, some classification schemes, such as support vector machines ([Chapter 4](#)), produce classification models in which each feature is given a weight. Features with larger weights play a more important role in the model. The normalization of objects that takes place when computing the cosine similarity ([Section 2.4.5](#)) can also be regarded as a type of feature weighting.

2.3.5 Feature Creation

It is frequently possible to create, from the original attributes, a new set of attributes that captures the important information in a data set much more effectively. Furthermore, the number of new attributes can be smaller than the number of original attributes, allowing us to reap all the previously described benefits of dimensionality reduction. Two related methodologies for creating new attributes are described next: feature extraction and mapping the data to a new space.

Feature Extraction

The creation of a new set of features from the original raw data is known as **feature extraction**. Consider a set of photographs, where each photograph is to be classified according to whether it contains a human face. The raw data is a set of pixels, and as such, is not suitable for many types of classification algorithms. However, if the data is processed to provide higher-level features, such as the presence or absence of certain types of edges and areas that are highly correlated with the presence of human faces, then a much broader set of classification techniques can be applied to this problem.

Unfortunately, in the sense in which it is most commonly used, feature extraction is highly domain-specific. For a particular field, such as image processing, various features and the techniques to extract them have been developed over a period of time, and often these techniques have limited applicability to other fields. Consequently, whenever data mining is applied to a relatively new area, a key task is the development of new features and feature extraction methods.

Although feature extraction is often complicated, [Example 2.10](#) illustrates that it can be relatively straightforward.

Example 2.10 (Density).

Consider a data set consisting of information about historical artifacts, which, along with other information, contains the volume and mass of each artifact. For simplicity, assume that these artifacts are made of a small number of materials (wood, clay, bronze, gold) and that we want to classify the artifacts with respect to the material of which they are made. In this case, a density feature constructed from the mass and volume features, i.e., $\text{density} = \text{mass}/\text{volume}$, would most directly yield an accurate classification. Although there have been some attempts to automatically perform such simple feature extraction by exploring basic mathematical combinations of existing attributes, the most common approach is to construct features using domain expertise.

Mapping the Data to a New Space

A totally different view of the data can reveal important and interesting features. Consider, for example, time series data, which often contains periodic patterns. If there is only a single periodic pattern and not much noise, then the pattern is easily detected. If, on the other hand, there are a number of periodic patterns and a significant amount of noise, then these patterns are hard to detect. Such patterns can, nonetheless, often be detected by applying a **Fourier transform** to the time series in order to change to a representation in which frequency information is explicit. In [Example 2.11](#), it will not be necessary to know the details of the Fourier transform. It is enough to know that, for each time series, the Fourier transform produces a new data object whose attributes are related to frequencies.

Example 2.11 (Fourier Analysis).

The time series presented in [Figure 2.12\(b\)](#) is the sum of three other time series, two of which are shown in [Figure 2.12\(a\)](#) and have frequencies of 7 and 17 cycles per second, respectively. The third time series is random noise. [Figure 2.12\(c\)](#) shows the power spectrum that can be computed after applying a Fourier transform to the original time series. (Informally, the power spectrum is proportional to the square of each frequency attribute.) In spite of the noise, there are two peaks that correspond to the periods of the two original, non-noisy time series. Again, the main point is that better features can reveal important aspects of the data.

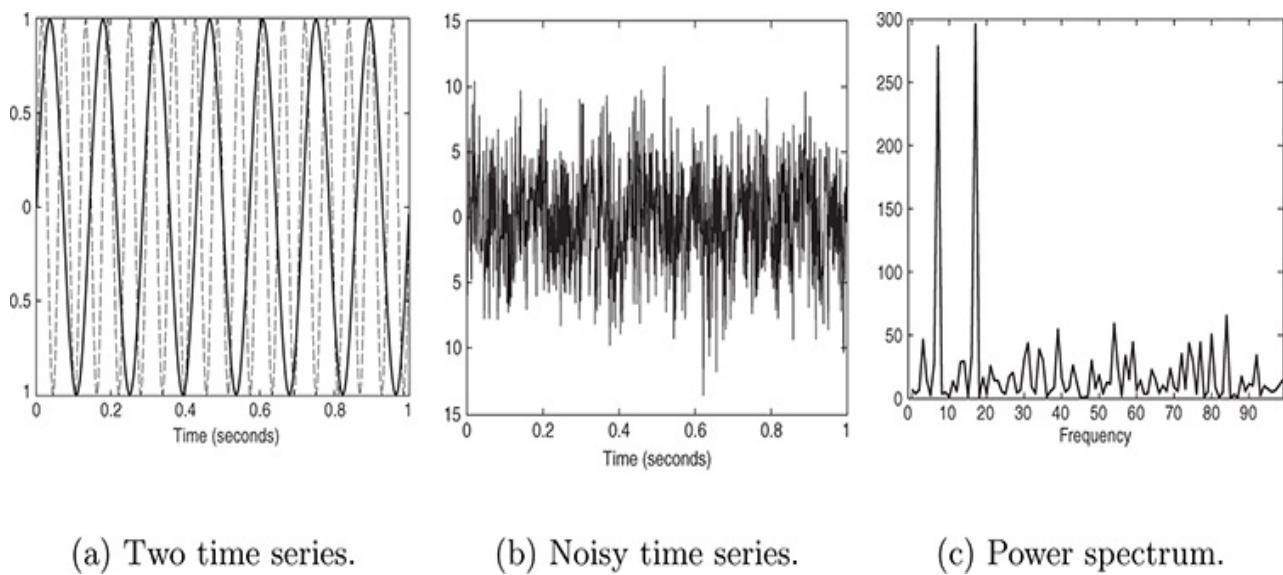


Figure 2.12.

Application of the Fourier transform to identify the underlying frequencies in time series data.

Many other sorts of transformations are also possible. Besides the Fourier transform, the **wavelet transform** has also proven very useful for time series and other types of data.

2.3.6 Discretization and Binarization

Some data mining algorithms, especially certain classification algorithms, require that the data be in the form of categorical attributes. Algorithms that find association patterns require that the data be in the form of binary attributes. Thus, it is often necessary to transform a continuous attribute into a categorical attribute (**discretization**), and both continuous and discrete attributes may need to be transformed into one or more binary attributes (**binarization**). Additionally, if a categorical attribute has a large number of values (categories), or some values occur infrequently, then it can be beneficial for certain data mining tasks to reduce the number of categories by combining some of the values.

As with feature selection, the best discretization or binarization approach is the one that “produces the best result for the data mining algorithm that will be used to analyze the data.” It is typically not practical to apply such a criterion directly. Consequently, discretization or binarization is performed in a way that satisfies a criterion that is thought to have a relationship to good performance for the data mining task being considered. In general, the best discretization depends on the algorithm being used, as well as the other attributes being considered. Typically, however, the discretization of each attribute is considered in isolation.

Binarization

A simple technique to binarize a categorical attribute is the following: If there are m categorical values, then uniquely assign each original value to an integer in the interval $[0, m-1]$. If the attribute is ordinal, then order must be maintained by the assignment. (Note that even if the attribute is originally represented using integers, this process is necessary if the integers are not in

the interval $[0, m-1]$.) Next, convert each of these m integers to a binary number. Since $n=[\log_2(m)]$ binary digits are required to represent these integers, represent these binary numbers using n binary attributes. To illustrate, a categorical variable with 5 values $\{\text{awful}, \text{poor}, \text{OK}, \text{good}, \text{great}\}$ would require three binary variables x_1 , x_2 , and x_3 . The conversion is shown in [Table 2.5](#).

Table 2.5. Conversion of a categorical attribute to three binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3
<i>awful</i>	0	0	0	0
<i>poor</i>	1	0	0	1
<i>OK</i>	2	0	1	0
<i>good</i>	3	0	1	1
<i>great</i>	4	1	0	0

Such a transformation can cause complications, such as creating unintended relationships among the transformed attributes. For example, in [Table 2.5](#), attributes x_2 and x_3 are correlated because information about the *good* value is encoded using both attributes. Furthermore, association analysis requires asymmetric binary attributes, where only the presence of the attribute (value =1) is important. For association problems, it is therefore necessary to introduce one asymmetric binary attribute for each categorical value, as shown in [Table 2.6](#). If the number of resulting attributes is too large, then the techniques described in the following sections can be used to reduce the number of categorical values before binarization.

Table 2.6. Conversion of a categorical attribute to five asymmetric binary

attributes.

Categorical Value	Integer Value	x_1	x_2	x_3	x_4	x_5
<i>awful</i>	0	1	0	0	0	0
<i>poor</i>	1	0	1	0	0	0
<i>OK</i>	2	0	0	1	0	0
<i>good</i>	3	0	0	0	1	0
<i>great</i>	4	0	0	0	0	1

Likewise, for association problems, it can be necessary to replace a single binary attribute with two asymmetric binary attributes. Consider a binary attribute that records a person's gender, male or female. For traditional association rule algorithms, this information needs to be transformed into two asymmetric binary attributes, one that is a 1 only when the person is male and one that is a 1 only when the person is female. (For asymmetric binary attributes, the information representation is somewhat inefficient in that two bits of storage are required to represent each bit of information.)

Discretization of Continuous Attributes

Discretization is typically applied to attributes that are used in classification or association analysis. Transformation of a continuous attribute to a categorical attribute involves two subtasks: deciding how many categories, n , to have and determining how to map the values of the continuous attribute to these categories. In the first step, after the values of the continuous attribute are sorted, they are then divided into n intervals by specifying $n-1$ **split points**. In the second, rather trivial step, all the values in one interval are mapped to the same categorical value. Therefore, the problem of discretization is one of

deciding how many split points to choose and where to place them. The result can be represented either as a set of intervals $\{(x_0, x_1], (x_1, x_2], \dots, (x_{n-1}, x_n]\}$, where x_0 and x_n can be $+\infty$ or $-\infty$, respectively, or equivalently, as a series of inequalities $x_0 < x \leq x_1, \dots, x_{n-1} < x < x_n$.

Unsupervised Discretization

A basic distinction between discretization methods for classification is whether class information is used (supervised) or not (unsupervised). If class information is not used, then relatively simple approaches are common. For instance, the **equal width** approach divides the range of the attribute into a user-specified number of intervals each having the same width. Such an approach can be badly affected by outliers, and for that reason, an **equal frequency (equal depth)** approach, which tries to put the same number of objects into each interval, is often preferred. As another example of unsupervised discretization, a clustering method, such as K-means (see [Chapter 7](#)), can also be used. Finally, visually inspecting the data can sometimes be an effective approach.

Example 2.12 (Discretization Techniques).

This example demonstrates how these approaches work on an actual data set. [Figure 2.13\(a\)](#) shows data points belonging to four different groups, along with two outliers—the large dots on either end. The techniques of the previous paragraph were applied to discretize the x values of these data points into four categorical values. (Points in the data set have a random y component to make it easy to see how many points are in each group.) Visually inspecting the data works quite well, but is not automatic, and thus, we focus on the other three approaches. The split points produced by the techniques equal width, equal frequency, and K-means are shown in

Figures 2.13(b) □, 2.13(c) □, and 2.13(d) □, respectively. The split points are represented as dashed lines.

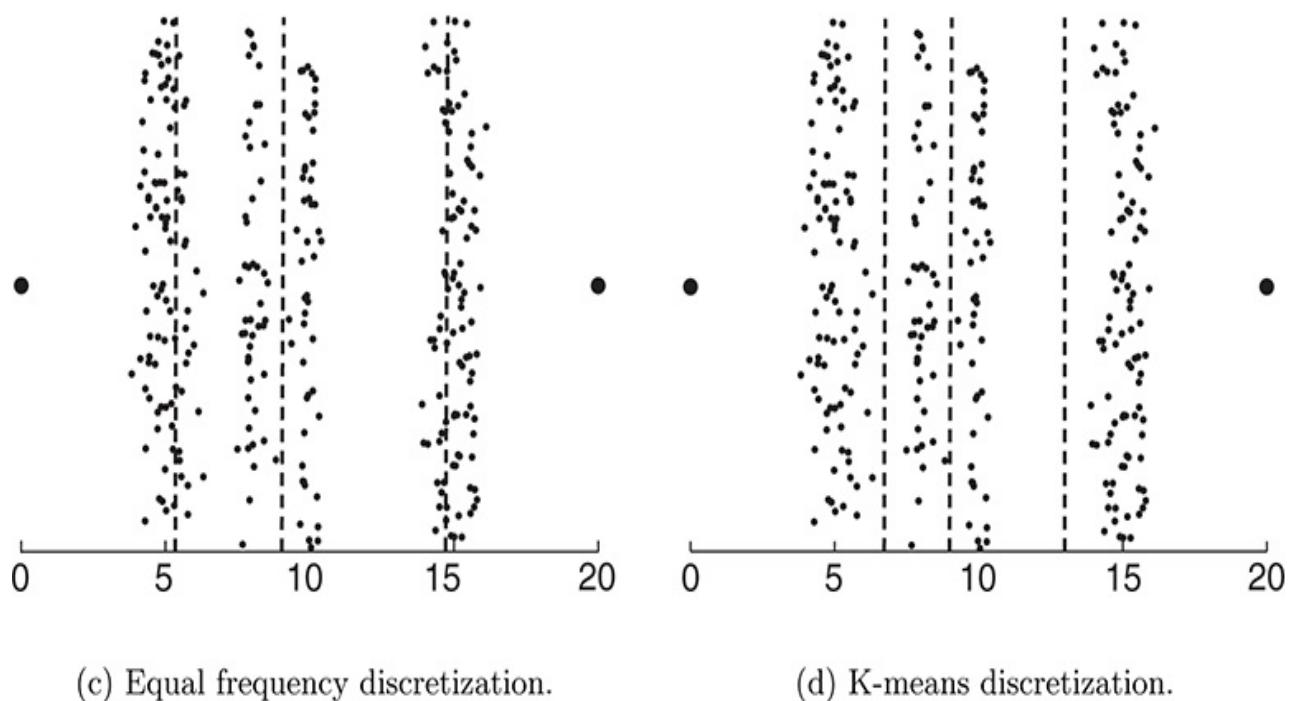
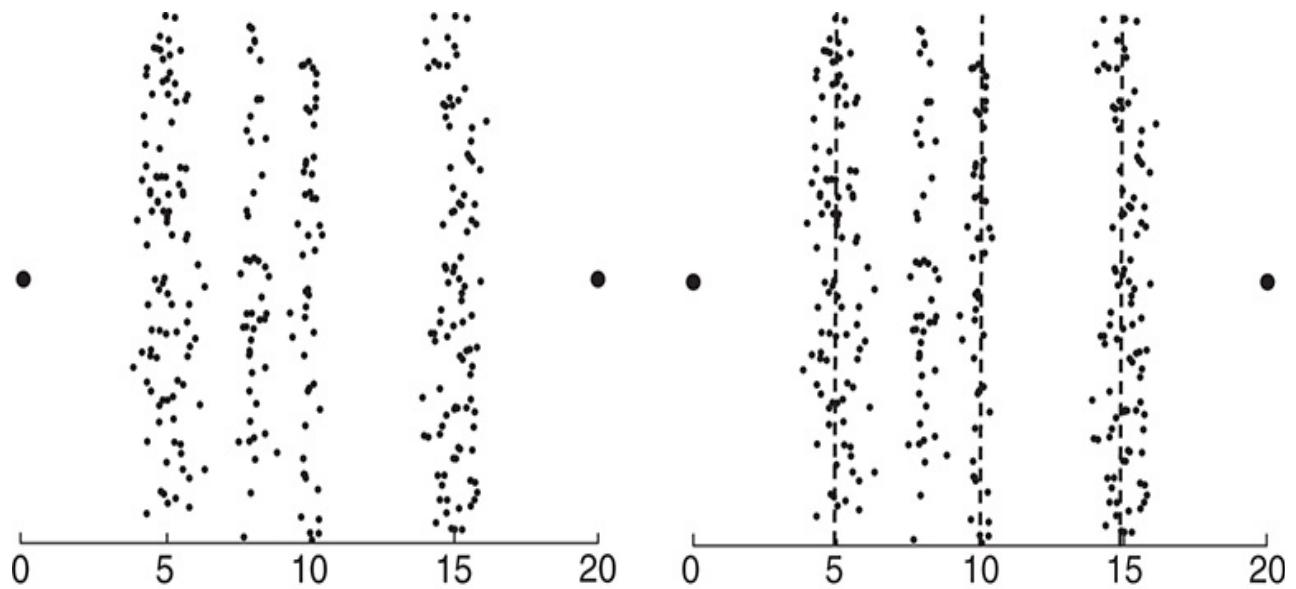


Figure 2.13.
Different discretization techniques.

In this particular example, if we measure the performance of a discretization technique by the extent to which different objects that clump together have the same categorical value, then K-means performs best, followed by equal frequency, and finally, equal width. More generally, the best discretization will depend on the application and often involves domain-specific discretization. For example, the discretization of people into low income, middle income, and high income is based on economic factors.

Supervised Discretization

If classification is our application and class labels are known for some data objects, then discretization approaches that use class labels often produce better classification. This should not be surprising, since an interval constructed with no knowledge of class labels often contains a mixture of class labels. A conceptually simple approach is to place the splits in a way that maximizes the purity of the intervals, i.e., the extent to which an interval contains a single class label. In practice, however, such an approach requires potentially arbitrary decisions about the purity of an interval and the minimum size of an interval.

To overcome such concerns, some statistically based approaches start with each attribute value in a separate interval and create larger intervals by merging adjacent intervals that are similar according to a statistical test. An alternative to this bottom-up approach is a top-down approach that starts by bisecting the initial values so that the resulting two intervals give minimum entropy. This technique only needs to consider each value as a possible split point, because it is assumed that intervals contain ordered sets of values. The splitting process is then repeated with another interval, typically choosing the interval with the worst (highest) entropy, until a user-specified number of intervals is reached, or a stopping criterion is satisfied.

Entropy-based approaches are one of the most promising approaches to discretization, whether bottom-up or top-down. First, it is necessary to define **entropy**. Let k be the number of different class labels, m_i be the number of values in the i^{th} interval of a partition, and m_{ij} be the number of values of class j in interval i . Then the entropy e_i of the i^{th} interval is given by the equation

$$e_i = -\sum_{j=1}^k p_{ij} \log_2 p_{ij},$$

where $p_{ij} = m_{ij}/m_i$ is the probability (fraction of values) of class j in the i^{th} interval. The total entropy, e , of the partition is the weighted average of the individual interval entropies, i.e.,

$$e = \sum_{i=1}^n w_i e_i,$$

where m is the number of values, $w_i = m_i/m$ is the fraction of values in the i^{th} interval, and n is the number of intervals. Intuitively, the entropy of an interval is a measure of the purity of an interval. If an interval contains only values of one class (is perfectly pure), then the entropy is 0 and it contributes nothing to the overall entropy. If the classes of values in an interval occur equally often (the interval is as impure as possible), then the entropy is a maximum.

Example 2.13 (Discretization of Two Attributes).

The top-down method based on entropy was used to independently discretize both the x and y attributes of the two-dimensional data shown in [Figure 2.14](#). In the first discretization, shown in [Figure 2.14\(a\)](#), the x and y attributes were both split into three intervals. (The dashed lines indicate the split points.) In the second discretization, shown in [Figure 2.14\(b\)](#), the x and y attributes were both split into five intervals.

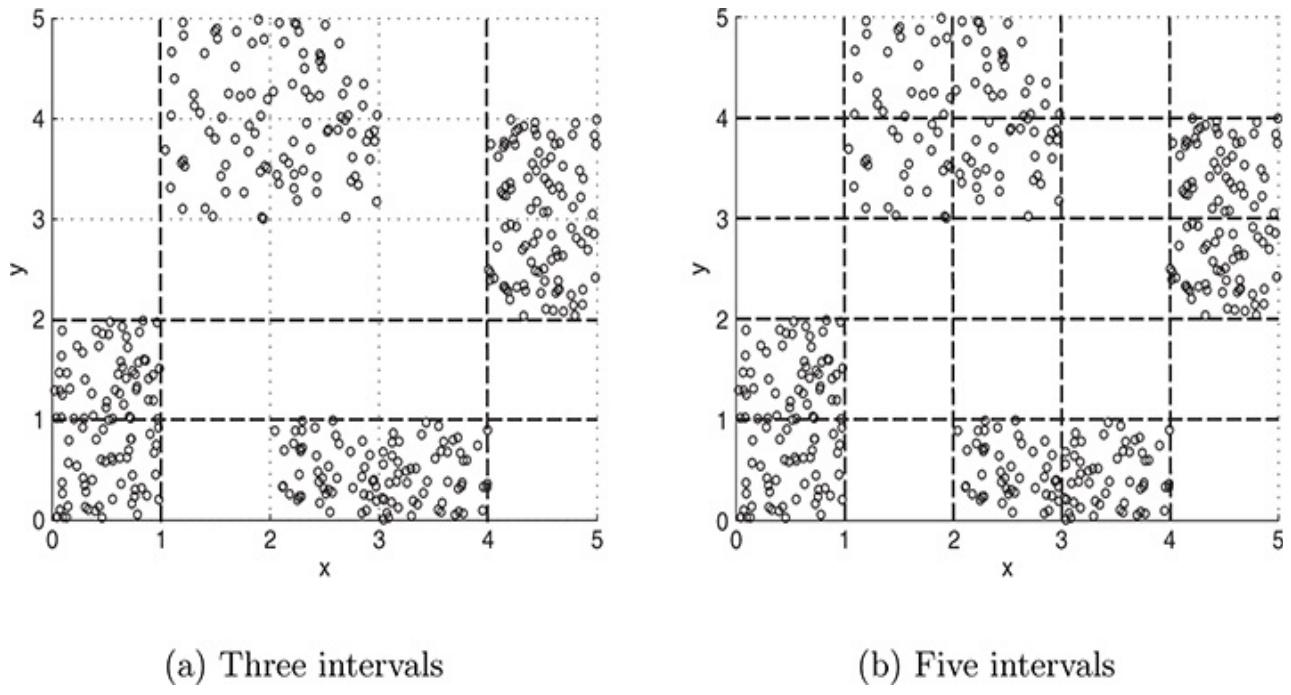


Figure 2.14.
Discretizing x and y attributes for four groups (classes) of points.

This simple example illustrates two aspects of discretization. First, in two dimensions, the classes of points are well separated, but in one dimension, this is not so. In general, discretizing each attribute separately often guarantees suboptimal results. Second, five intervals work better than three, but six intervals do not improve the discretization much, at least in terms of entropy. (Entropy values and results for six intervals are not shown.) Consequently, it is desirable to have a stopping criterion that automatically finds the right number of partitions.

Categorical Attributes with Too Many Values

Categorical attributes can sometimes have too many values. If the categorical attribute is an ordinal attribute, then techniques similar to those for continuous attributes can be used to reduce the number of categories. If the categorical attribute is nominal, however, then other approaches are needed. Consider a

university that has a large number of departments. Consequently, a *department name* attribute might have dozens of different values. In this situation, we could use our knowledge of the relationships among different departments to combine departments into larger groups, such as *engineering*, *social sciences*, or *biological sciences*. If domain knowledge does not serve as a useful guide or such an approach results in poor classification performance, then it is necessary to use a more empirical approach, such as grouping values together only if such a grouping results in improved classification accuracy or achieves some other data mining objective.

2.3.7 Variable Transformation

A **variable transformation** refers to a transformation that is applied to all the values of a variable. (We use the term variable instead of attribute to adhere to common usage, although we will also refer to attribute transformation on occasion.) In other words, for each object, the transformation is applied to the value of the variable for that object. For example, if only the magnitude of a variable is important, then the values of the variable can be transformed by taking the absolute value. In the following section, we discuss two important types of variable transformations: simple functional transformations and normalization.

Simple Functions

For this type of variable transformation, a simple mathematical function is applied to each value individually. If x is a variable, then examples of such transformations include x^k , $\log x$, e^x , x , $1/x$, $\sin x$, or $|x|$. In statistics, variable transformations, especially \sqrt{x} , $\log x$, and $1/x$, are often used to transform data that does not have a Gaussian (normal) distribution into data that does. While

this can be important, other reasons often take precedence in data mining. Suppose the variable of interest is the number of data bytes in a session, and the number of bytes ranges from 1 to 1 billion. This is a huge range, and it can be advantageous to compress it by using a \log_{10} transformation. In this case, sessions that transferred 108 and 109 bytes would be more similar to each other than sessions that transferred 10 and 1000 bytes ($9-8=1$ versus $3-1=3$). For some applications, such as network intrusion detection, this may be what is desired, since the first two sessions most likely represent transfers of large files, while the latter two sessions could be two quite distinct types of sessions.

Variable transformations should be applied with caution because they change the nature of the data. While this is what is desired, there can be problems if the nature of the transformation is not fully appreciated. For instance, the transformation $1/x$ reduces the magnitude of values that are 1 or larger, but increases the magnitude of values between 0 and 1. To illustrate, the values $\{1, 2, 3\}$ go to $\{1, 12, 13\}$, but the values $\{1, 12, 13\}$ go to $\{1, 2, 3\}$. Thus, for all sets of values, the transformation $1/x$ reverses the order. To help clarify the effect of a transformation, it is important to ask questions such as the following: What is the desired property of the transformed attribute? Does the order need to be maintained? Does the transformation apply to all values, especially negative values and 0? What is the effect of the transformation on the values between 0 and 1? [Exercise 17](#) on [page 109](#) explores other aspects of variable transformation.

Normalization or Standardization

The goal of standardization or normalization is to make an entire set of values have a particular property. A traditional example is that of “standardizing a variable” in statistics. If \bar{x} is the mean (average) of the attribute values and s_x is their standard deviation, then the transformation $x' = (x - \bar{x})/s_x$ creates a new

variable that has a mean of 0 and a standard deviation of 1. If different variables are to be used together, e.g., for clustering, then such a transformation is often necessary to avoid having a variable with large values dominate the results of the analysis. To illustrate, consider comparing people based on two variables: age and income. For any two people, the difference in income will likely be much higher in absolute terms (hundreds or thousands of dollars) than the difference in age (less than 150). If the differences in the range of values of age and income are not taken into account, then the comparison between people will be dominated by differences in income. In particular, if the similarity or dissimilarity of two people is calculated using the similarity or dissimilarity measures defined later in this chapter, then in many cases, such as that of Euclidean distance, the income values will dominate the calculation.

The mean and standard deviation are strongly affected by outliers, so the above transformation is often modified. First, the mean is replaced by the **median**, i.e., the middle value. Second, the standard deviation is replaced by the **absolute standard deviation**. Specifically, if x is a variable, then the absolute standard deviation of x is given by $\sigma_A = \sqrt{\sum_{i=1}^m |x_i - \mu|}$, where x_i is the i th value of the variable, m is the number of objects, and μ is either the mean or median. Other approaches for computing estimates of the location (center) and spread of a set of values in the presence of outliers are described in statistics books. These more robust measures can also be used to define a standardization transformation.

2.4 Measures of Similarity and Dissimilarity

Similarity and dissimilarity are important because they are used by a number of data mining techniques, such as clustering, nearest neighbor classification, and anomaly detection. In many cases, the initial data set is not needed once these similarities or dissimilarities have been computed. Such approaches can be viewed as transforming the data to a similarity (dissimilarity) space and then performing the analysis. Indeed, **kernel methods** are a powerful realization of this idea. These methods are introduced in [Section 2.4.7](#) and are discussed more fully in the context of classification in Section 4.9.4.

We begin with a discussion of the basics: high-level definitions of similarity and dissimilarity, and a discussion of how they are related. For convenience, the term **proximity** is used to refer to either similarity or dissimilarity. Since the proximity between two objects is a function of the proximity between the corresponding attributes of the two objects, we first describe how to measure the proximity between objects having only one attribute.

We then consider proximity measures for objects with multiple attributes. This includes measures such as the Jaccard and cosine similarity measures, which are useful for sparse data, such as documents, as well as correlation and Euclidean distance, which are useful for non-sparse (dense) data, such as time series or multi-dimensional points. We also consider mutual information, which can be applied to many types of data and is good for detecting nonlinear relationships. In this discussion, we restrict ourselves to objects with relatively homogeneous attribute types, typically binary or continuous.

Next, we consider several important issues concerning proximity measures. This includes how to compute proximity between objects when they have heterogeneous types of attributes, and approaches to account for differences of scale and correlation among variables when computing distance between numerical objects. The section concludes with a brief discussion of how to select the right proximity measure.

Although this section focuses on the computation of proximity between data objects, proximity can also be computed between attributes. For example, for the document-term matrix of [Figure 2.2\(d\)](#), the cosine measure can be used to compute similarity between a pair of documents or a pair of terms (words). Knowing that two variables are strongly related can, for example, be helpful for eliminating redundancy. In particular, the correlation and mutual information measures discussed later are often used for that purpose.

2.4.1 Basics

Definitions

Informally, the **similarity** between two objects is a numerical measure of the degree to which the two objects are alike. Consequently, similarities are *higher* for pairs of objects that are more alike. Similarities are usually non-negative and are often between 0 (no similarity) and 1 (complete similarity).

The **dissimilarity** between two objects is a numerical measure of the degree to which the two objects are different. Dissimilarities are *lower* for more similar pairs of objects. Frequently, the term **distance** is used as a synonym for dissimilarity, although, as we shall see, distance often refers to a special class

of dissimilarities. Dissimilarities sometimes fall in the interval [0, 1], but it is also common for them to range from 0 to ∞ .

Transformations

Transformations are often applied to convert a similarity to a dissimilarity, or vice versa, or to transform a proximity measure to fall within a particular range, such as [0,1]. For instance, we may have similarities that range from 1 to 10, but the particular algorithm or software package that we want to use may be designed to work only with dissimilarities, or it may work only with similarities in the interval [0,1]. We discuss these issues here because we will employ such transformations later in our discussion of proximity. In addition, these issues are relatively independent of the details of specific proximity measures.

Frequently, proximity measures, especially similarities, are defined or transformed to have values in the interval [0,1]. Informally, the motivation for this is to use a scale in which a proximity value indicates the fraction of similarity (or dissimilarity) between two objects. Such a transformation is often relatively straightforward. For example, if the similarities between objects range from 1 (not at all similar) to 10 (completely similar), we can make them fall within the range [0, 1] by using the transformation $s'=(s-1)/9$, where s and s' are the original and new similarity values, respectively. In the more general case, the transformation of similarities to the interval [0, 1] is given by the expression $s'=(s-\min_s)/(\max_s-\min_s)$, where \max_s and \min_s are the maximum and minimum similarity values, respectively. Likewise, dissimilarity measures with a finite range can be mapped to the interval [0,1] by using the formula $d'=(d-\min_d)/(\max_d-\min_d)$. This is an example of a linear transformation, which preserves the relative distances between points. In other words, if points, x_1 and x_2 , are twice as far apart as points, x_3 and x_4 , the same will be true after a linear transformation.

However, there can be complications in mapping proximity measures to the interval $[0, 1]$ using a linear transformation. If, for example, the proximity measure originally takes values in the interval $[0, \infty]$, then \max_d is not defined and a nonlinear transformation is needed. Values will not have the same relationship to one another on the new scale. Consider the transformation $d=d/(1+d)$ for a dissimilarity measure that ranges from 0 to ∞ . The dissimilarities 0, 0.5, 2, 10, 100, and 1000 will be transformed into the new dissimilarities 0, 0.33, 0.67, 0.90, 0.99, and 0.999, respectively. Larger values on the original dissimilarity scale are compressed into the range of values near 1, but whether this is desirable depends on the application.

Note that mapping proximity measures to the interval $[0, 1]$ can also change the meaning of the proximity measure. For example, correlation, which is discussed later, is a measure of similarity that takes values in the interval $[-1, 1]$. Mapping these values to the interval $[0, 1]$ by taking the absolute value loses information about the sign, which can be important in some applications. See [Exercise 22](#) on [page 111](#).

Transforming similarities to dissimilarities and vice versa is also relatively straightforward, although we again face the issues of preserving meaning and changing a linear scale into a nonlinear scale. If the similarity (or dissimilarity) falls in the interval $[0, 1]$, then the dissimilarity can be defined as $d=1-s$ ($s=1-d$). Another simple approach is to define similarity as the negative of the dissimilarity (or vice versa). To illustrate, the dissimilarities 0, 1, 10, and 100 can be transformed into the similarities 0, -1, -10, and -100, respectively.

The similarities resulting from the negation transformation are not restricted to the range $[0, 1]$, but if that is desired, then transformations such as $s=1d+1$, $s=e-d$, or $s=1-d-\min_d$ can be used. For the transformation $s=1d+1$, the dissimilarities 0, 1, 10, 100 are transformed into 1,

0.5, 0.09, 0.01, respectively. For $s=e-d$, they become 1.00, 0.37, 0.00, 0.00, respectively, while for $s=1-d-\min_d$ they become 1.00, 0.99, 0.90, 0.00, respectively. In this discussion, we have focused on converting dissimilarities to similarities. Conversion in the opposite direction is considered in [Exercise 23](#) on [page 111](#).

In general, any monotonic decreasing function can be used to convert dissimilarities to similarities, or vice versa. Of course, other factors also must be considered when transforming similarities to dissimilarities, or vice versa, or when transforming the values of a proximity measure to a new scale. We have mentioned issues related to preserving meaning, distortion of scale, and requirements of data analysis tools, but this list is certainly not exhaustive.

2.4.2 Similarity and Dissimilarity between Simple Attributes

The proximity of objects with a number of attributes is typically defined by combining the proximities of individual attributes, and thus, we first discuss proximity between objects having a single attribute. Consider objects described by one nominal attribute. What would it mean for two such objects to be similar? Because nominal attributes convey only information about the distinctness of objects, all we can say is that two objects either have the same value or they do not. Hence, in this case similarity is traditionally defined as 1 if attribute values match, and as 0 otherwise. A dissimilarity would be defined in the opposite way: 0 if the attribute values match, and 1 if they do not.

For objects with a single ordinal attribute, the situation is more complicated because information about order should be taken into account. Consider an

attribute that measures the quality of a product, e.g., a candy bar, on the scale $\{\text{poor}, \text{fair}, \text{OK}, \text{good}, \text{wonderful}\}$. It would seem reasonable that a product, P1, which is rated *wonderful*, would be closer to a product P2, which is rated *good*, than it would be to a product P3, which is rated *OK*. To make this observation quantitative, the values of the ordinal attribute are often mapped to successive integers, beginning at 0 or 1, e.g., $\{\text{poor}=0, \text{fair}=1, \text{OK}=2, \text{good}=3, \text{wonderful}=4\}$. Then, $d(P1, P2)=3-2=1$ or, if we want the dissimilarity to fall between 0 and 1, $d(P1, P2)=3-24=0.25$. A similarity for ordinal attributes can then be defined as $s=1-d$.

This definition of similarity (dissimilarity) for an ordinal attribute should make the reader a bit uneasy since this assumes equal intervals between successive values of the attribute, and this is not necessarily so. Otherwise, we would have an interval or ratio attribute. Is the difference between the values *fair* and *good* really the same as that between the values *OK* and *wonderful*? Probably not, but in practice, our options are limited, and in the absence of more information, this is the standard approach for defining proximity between ordinal attributes.

For interval or ratio attributes, the natural measure of dissimilarity between two objects is the absolute difference of their values. For example, we might compare our current weight and our weight a year ago by saying “I am ten pounds heavier.” In cases such as these, the dissimilarities typically range from 0 to ∞ , rather than from 0 to 1. The similarity of interval or ratio attributes is typically expressed by transforming a dissimilarity into a similarity, as previously described.

Table 2.7  summarizes this discussion. In this table, x and y are two objects that have one attribute of the indicated type. Also, $d(x, y)$ and $s(x, y)$ are the dissimilarity and similarity between x and y , respectively. Other approaches are possible; these are the most common ones.

Table 2.7. Similarity and dissimilarity for simple attributes

Attribute Type	Dissimilarity	Similarity
Nominal	$d=\{ 0 \text{if } x=y, 1 \text{if } x \neq y \}$	$s=\{ 1 \text{if } x=y, 0 \text{if } x \neq y \}$
Ordinal	$d= x-y /(n-1)$ (values mapped to integers 0 to $n-1$, where n is the number of values)	$s=1-d$
Interval or Ratio	$d= x-y $	$s=-d, s=1+d, s=e-d, s=1-d-\min_d, s=\max_d - d - \min_d$

The following two sections consider more complicated measures of proximity between objects that involve multiple attributes: (1) dissimilarities between data objects and (2) similarities between data objects. This division allows us to more naturally display the underlying motivations for employing various proximity measures. We emphasize, however, that similarities can be transformed into dissimilarities and vice versa using the approaches described earlier.

2.4.3 Dissimilarities between Data Objects

In this section, we discuss various kinds of dissimilarities. We begin with a discussion of distances, which are dissimilarities with certain properties, and then provide examples of more general kinds of dissimilarities.

Distances

We first present some examples, and then offer a more formal description of distances in terms of the properties common to all distances. The **Euclidean distance**, d , between two points, \mathbf{x} and \mathbf{y} , in one-, two-, three-, or higher-dimensional space, is given by the following familiar formula:

$$d(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^n (x_k - y_k)^2, \quad (2.1)$$

where n is the number of dimensions and x_k and y_k are, respectively, the k th attributes (components) of \mathbf{x} and \mathbf{y} . We illustrate this formula with [Figure 2.15](#) and [Tables 2.8](#) and [2.9](#), which show a set of points, the x and y coordinates of these points, and the **distance matrix** containing the pairwise distances of these points.

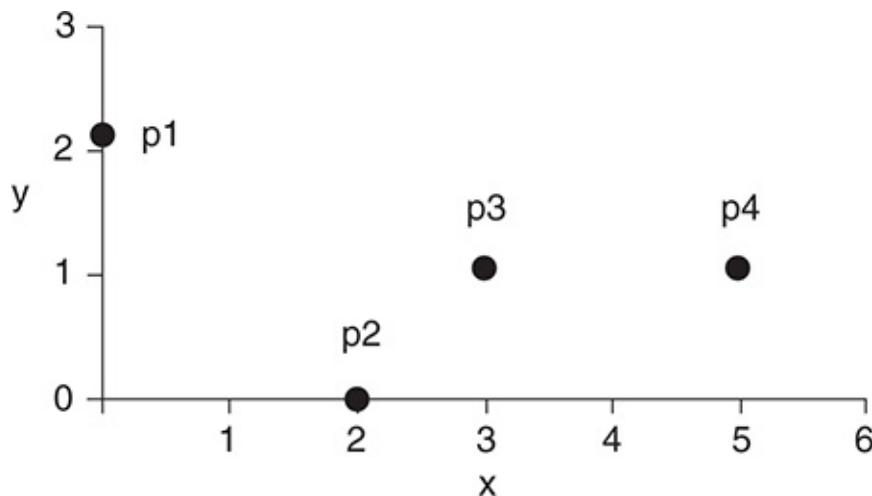


Figure 2.15.
Four two-dimensional points.

The Euclidean distance measure given in [Equation 2.1](#) is generalized by the **Minkowski** distance metric shown in [Equation 2.2](#),

$$d(\mathbf{x}, \mathbf{y}) = (\sum_{k=1}^n |x_k - y_k|^r)^{1/r}, \quad (2.2)$$

where r is a parameter. The following are the three most common examples of Minkowski distances.

- $r=1$. City block (Manhattan, taxicab, L1 norm) distance. A common example is the **Hamming distance**, which is the number of bits that is different between two objects that have only binary attributes, i.e., between two binary vectors.
 - $r=2$. Euclidean distance (L2 norm).
 - $r=\infty$. Supremum (L_{max} or L _{∞} norm) distance. This is the maximum difference between any attribute of the objects. More formally, the L _{∞} distance is defined by [Equation 2.3](#)
- $$d(x, y) = \lim_{r \rightarrow \infty} (\sum_{k=1}^n |x_k - y_k|^r)^{1/r}. \quad (2.3)$$

The r parameter should not be confused with the number of dimensions (attributes) n . The Euclidean, Manhattan, and supremum distances are defined for all values of n : 1, 2, 3, ..., and specify different ways of combining the differences in each dimension (attribute) into an overall distance.

[Tables 2.10](#) and [2.11](#), respectively, give the proximity matrices for the L1 and L _{∞} distances using data from [Table 2.8](#). Notice that all these distance matrices are symmetric; i.e., the ijth entry is the same as the jith entry. In [Table 2.9](#), for instance, the fourth row of the first column and the fourth column of the first row both contain the value 5.1.

Table 2.8. x and y coordinates of four points.

point	x coordinate	y coordinate
p1	0	2
p2	2	0
p3	3	1

p4	5	1
----	---	---

Table 2.9. Euclidean distance matrix for Table 2.8.

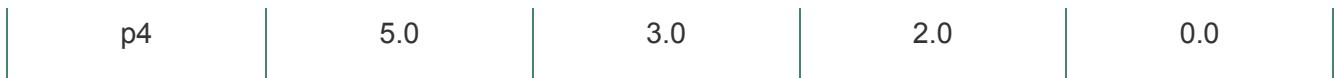
	p1	p2	p3	p4
p1	0.0	2.8	3.2	5.1
p2	2.8	0.0	1.4	3.2
p3	3.2	1.4	0.0	2.0
p4	5.1	3.2	2.0	0.0

Table 2.10. L1 distance matrix for Table 2.8.

L1	p1	p2	p3	p4
p1	0.0	4.0	4.0	6.0
p2	4.0	0.0	2.0	4.0
p3	4.0	2.0	0.0	2.0
p4	6.0	4.0	2.0	0.0

Table 2.11. L ∞ distance matrix for Table 2.8.

L ∞	p1	p2	p3	p4
p1	0.0	2.0	3.0	5.0
p2	2.0	0.0	1.0	3.0
p3	3.0	1.0	0.0	2.0



Distances, such as the Euclidean distance, have some well-known properties. If $d(x, y)$ is the distance between two points, x and y , then the following properties hold.

1. Positivity

- a. $d(x, y) \geq 0$ for all x and y ,
- b. $d(x, y) = 0$ only if $x = y$.

2. Symmetry $d(x, y) = d(y, x)$ for all x and y .

3. Triangle Inequality $d(x, z) \leq d(x, y) + d(y, z)$ for all points x , y , and z .

Measures that satisfy all three properties are known as **metrics**. Some people use the term distance only for dissimilarity measures that satisfy these properties, but that practice is often violated. The three properties described here are useful, as well as mathematically pleasing. Also, if the triangle inequality holds, then this property can be used to increase the efficiency of techniques (including clustering) that depend on distances possessing this property. (See [Exercise 25](#).) Nonetheless, many dissimilarities do not satisfy one or more of the metric properties. [Example 2.14](#) illustrates such a measure.

Example 2.14 (Non-metric Dissimilarities: Set Differences).

This example is based on the notion of the difference of two sets, as defined in set theory. Given two sets A and B , $A - B$ is the set of elements of A that are not in

B. For example, if $A=\{1, 2, 3, 4\}$ and $B=\{2, 3, 4\}$, then $A-B=\{1\}$ and $B-A=\emptyset$, the empty set. We can define the distance d between two sets A and B as $d(A, B)=\text{size}(A-B)$, where *size* is a function returning the number of elements in a set. This distance measure, which is an integer value greater than or equal to 0, does not satisfy the second part of the positivity property, the symmetry property, or the triangle inequality. However, these properties can be made to hold if the dissimilarity measure is modified as follows: $d(A, B)=\text{size}(A-B)+\text{size}(B-A)$. See [Exercise 21](#) on [page 110](#).

2.4.4 Similarities between Data Objects

For similarities, the triangle inequality (or the analogous property) typically does not hold, but symmetry and positivity typically do. To be explicit, if $s(x, y)$ is the similarity between points x and y , then the typical properties of similarities are the following:

1. $s(x, y)=1$ only if $x=y$. ($0 \leq s \leq 1$)
2. $s(x, y)=s(y, x)$ for all x and y . (Symmetry)

There is no general analog of the triangle inequality for similarity measures. It is sometimes possible, however, to show that a similarity measure can easily be converted to a metric distance. The cosine and Jaccard similarity measures, which are discussed shortly, are two examples. Also, for specific similarity measures, it is possible to derive mathematical bounds on the similarity between two objects that are similar in spirit to the triangle inequality.

Example 2.15 (A Non-symmetric Similarity

Measure).

Consider an experiment in which people are asked to classify a small set of characters as they flash on a screen. The **confusion matrix** for this experiment records how often each character is classified as itself, and how often each is classified as another character. Using the confusion matrix, we can define a similarity measure between a character x and a character y as the number of times that x is misclassified as y , but note that this measure is not symmetric. For example, suppose that “0” appeared 200 times and was classified as a “0” 160 times, but as an “o” 40 times. Likewise, suppose that “o” appeared 200 times and was classified as an “o” 170 times, but as “0” only 30 times. Then, $s(0,o)=40$, but $s(o, 0)=30$. In such situations, the similarity measure can be made symmetric by setting $s'=(x,y)=s'(y,x)=(s(x,y)+s(y,x))/2$, where s indicates the new similarity measure.

2.4.5 Examples of Proximity Measures

This section provides specific examples of some similarity and dissimilarity measures.

Similarity Measures for Binary Data

Similarity measures between objects that contain only binary attributes are called **similarity coefficients**, and typically have values between 0 and 1. A value of 1 indicates that the two objects are completely similar, while a value of 0 indicates that the objects are not at all similar. There are many rationales for why one coefficient is better than another in specific instances.

Let x and y be two objects that consist of n binary attributes. The comparison of two such objects, i.e., two binary vectors, leads to the following four quantities (frequencies):

f_{00} =the number of attributes where x is 0 and y is 0
 f_{01} = the number of attributes where x is 0 and y is 1
 f_{10} = the number of attributes where x is 1 and y is 0
 f_{11} = the number of attributes where x is 1 and y is 1

Simple Matching Coefficient

One commonly used similarity coefficient is the **simple matching coefficient (SMC)**, which is defined as

$$SMC = \frac{\text{number of matching attribute values}}{\text{number of attributes}} = f_{11} + f_{00} / (f_{11} + f_{00} + f_{10} + f_{01})$$

This measure counts both presences and absences equally. Consequently, the SMC could be used to find students who had answered questions similarly on a test that consisted only of true/false questions.

Jaccard Coefficient

Suppose that x and y are data objects that represent two rows (two transactions) of a transaction matrix (see [Section 2.1.2](#)). If each asymmetric binary attribute corresponds to an item in a store, then a 1 indicates that the item was purchased, while a 0 indicates that the product was not purchased. Because the number of products not purchased by any customer far outnumbers the number of products that were purchased, a similarity measure such as SMC would say that all transactions are very similar. As a result, the Jaccard coefficient is frequently used to handle objects consisting of asymmetric binary attributes. The **Jaccard coefficient**, which is often symbolized by j , is given by the following equation:

$$J = \frac{\text{number of matching presences}}{\text{number of attributes not involved in 00}} = \frac{f_{11}}{f_{11} + f_{00} + f_{10} + f_{01}}$$

Example 2.16 (The SMC and Jaccard Similarity Coefficients).

To illustrate the difference between these two similarity measures, we calculate SMC and J for the following two binary vectors.

$$x = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0) \quad y = (0, 0, 0, 0, 0, 0, 1, 0, 0, 1)$$

$f_{01}=2$ the number of attributes where x was 0 and y was 1 $f_{10}=1$ the number

$$SMC=f_{11}+f_{00}f_{01}+f_{10}+f_{11}+f_{00}=0+72+1+0+7=0.7$$

$$J=f_{11}f_{01}+f_{10}+f_{11}=02+1+0=0$$

Cosine Similarity

Documents are often represented as vectors, where each component (attribute) represents the frequency with which a particular term (word) occurs in the document. Even though documents have thousands or tens of thousands of attributes (terms), each document is sparse since it has relatively few nonzero attributes. Thus, as with transaction data, similarity should not depend on the number of shared 0 values because any two documents are likely to “not contain” many of the same words, and therefore, if 0–0 matches are counted, most documents will be highly similar to most other documents. Therefore, a similarity measure for documents needs to ignore 0–0 matches like the Jaccard measure, but also must be able to handle non-binary vectors. The **cosine similarity**, defined next, is one of the most common measures of document similarity. If x and y are two document vectors, then

$$\cos(x, y) = \langle x, y \rangle / \|x\| \|y\| = x'y / \|x\| \|y\|, \quad (2.6)$$

where ' indicates vector or matrix transpose and $\langle x, y \rangle$ indicates the inner product of the two vectors,

$$\langle x, y \rangle = \sum_{k=1}^n x_k y_k = x'y, \quad (2.7)$$

and $\|x\|$ is the length of vector x , $\|x\| = \sqrt{\sum_{k=1}^n x_k^2} = \sqrt{\langle x, x \rangle} = \sqrt{x'x}$.

The inner product of two vectors works well for asymmetric attributes since it depends only on components that are non-zero in both vectors. Hence, the similarity between two documents depends only upon the words that appear in both of them.

Example 2.17 (Cosine Similarity between Two Document Vectors).

This example calculates the cosine similarity for the following two data objects, which might represent document vectors:

$$x = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0) \quad y = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$$

$$\begin{aligned} \langle x, y \rangle &= 3 \times 1 + 2 \times 0 + 0 \times 0 + 5 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 2 \times 1 + 0 \times 0 + 2 \times 0 = 5 \\ \|x\| &= \sqrt{3^2 + 2^2 + 0^2 + 5^2 + 0^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2} = \sqrt{34} \\ \|y\| &= \sqrt{1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 2^2} = \sqrt{6} \end{aligned}$$

As indicated by [Figure 2.16](#), cosine similarity really is a measure of the (cosine of the) angle between x and y . Thus, if the cosine similarity is 1, the angle between x and y is 0° , and x and y are the same except for length. If the cosine similarity is 0, then the angle between x and y is 90° , and they do not share any terms (words).

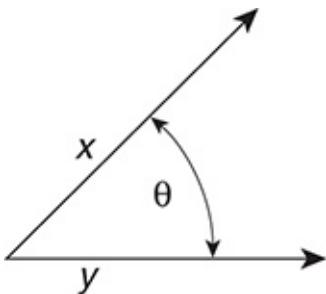


Figure 2.16.

Geometric illustration of the cosine measure.

Equation 2.6  also can be written as **Equation 2.8** .

$$\cos(x, y) = \langle x/\|x\|, y/\|y\| \rangle = \langle x', y' \rangle, \quad (2.8)$$

where $x' = x/\|x\|$ and $y' = y/\|y\|$. Dividing \mathbf{x} and \mathbf{y} by their lengths normalizes them to have a length of 1. This means that cosine similarity does not take the *length* of the two data objects into account when computing similarity. (Euclidean distance might be a better choice when length is important.) For vectors with a length of 1, the cosine measure can be calculated by taking a simple inner product. Consequently, when many cosine similarities between objects are being computed, normalizing the objects to have unit length can reduce the time required.

Extended Jaccard Coefficient (Tanimoto Coefficient)

The extended Jaccard coefficient can be used for document data and that reduces to the Jaccard coefficient in the case of binary attributes. This coefficient, which we shall represent as EJ , is defined by the following equation:

$$EJ(x, y) = \langle x, y \rangle / \|x\|^2 + \|y\|^2 - \langle x, y \rangle = x'y / \|x\|^2 + \|y\|^2 - x'y. \quad (2.9)$$

Correlation

Correlation is frequently used to measure the linear relationship between two sets of values that are observed together. Thus, correlation can measure the relationship between two variables (height and weight) or between two objects (a pair of temperature time series). Correlation is used much more frequently to measure the similarity between attributes since the values in two data objects come from different attributes, which can have very different attribute types and scales. There are many types of correlation, and indeed correlation is sometimes used in a general sense to mean the relationship between two sets of values that are observed together. In this discussion, we will focus on a measure appropriate for numerical values.

Specifically, **Pearson's correlation** between two sets of numerical values, i.e., two vectors, \mathbf{x} and \mathbf{y} , is defined by the following equation:

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \frac{\text{covariance}(\mathbf{x}, \mathbf{y})}{\text{standard_deviation}(\mathbf{x}) \times \text{standard_deviation}(\mathbf{y})} = \frac{s_{xy}}{s_x s_y}$$

where we use the following standard statistical notation and definitions:

$$\text{covariance}(\mathbf{x}, \mathbf{y}) = s_{xy} = \frac{1}{n-1} \sum_{k=1}^n (x_k - \bar{x})(y_k - \bar{y}) \quad (2.11)$$

$$\text{standard_deviation}(\mathbf{x}) = s_x = \sqrt{\frac{1}{n-1} \sum_{k=1}^n (x_k - \bar{x})^2}$$

$$\text{standard_deviation}(\mathbf{y}) = s_y = \sqrt{\frac{1}{n-1} \sum_{k=1}^n (y_k - \bar{y})^2}$$

$$\bar{x} = \frac{1}{n} \sum_{k=1}^n x_k \text{ is the mean of } \mathbf{x}$$

$$\bar{y} = \frac{1}{n} \sum_{k=1}^n y_k \text{ is the mean of } \mathbf{y}$$

Example 2.18 (Perfect Correlation).

Correlation is always in the range -1 to 1 . A correlation of 1 (-1) means that x and y have a perfect positive (negative) linear relationship; that is, $xk=ayk+b$, where a and b are constants. The following two vectors x and y illustrate cases where the correlation is -1 and $+1$, respectively. In the first case, the means of x and y were chosen to be 0 , for simplicity.

$$x=(-3, 6, 0, 3, -6) \quad y=(1, -2, 0, -1, 2) \quad \text{corr}(x, y)=-1 \quad xk=-3yk$$

$$x=(3, 6, 0, 3, 6) \quad y=(1, 2, 0, 1, 2) \quad \text{corr}(x, y)=1 \quad xk=3yk$$

Example 2.19 (Nonlinear Relationships).

If the correlation is 0 , then there is no linear relationship between the two sets of values. However, nonlinear relationships can still exist. In the following example, $yk=xk^2$, but their correlation is 0 .

$$x=(-3, -2, -1, 0, 1, 2, 3) \quad y=(9, 4, 1, 0, 1, 4, 9)$$

Example 2.20 (Visualizing Correlation).

It is also easy to judge the correlation between two vectors x and y by plotting pairs of corresponding values of x and y in a scatter plot. [Figure 2.17](#) shows a number of these scatter plots when x and y consist of a set of 30 pairs of values that are randomly generated (with a normal distribution) so that the correlation of x and y ranges from -1 to 1 . Each circle in a plot represents one of the 30 pairs of x and y values; its x coordinate is the value of that pair for x , while its y coordinate is the value of the same pair for y .

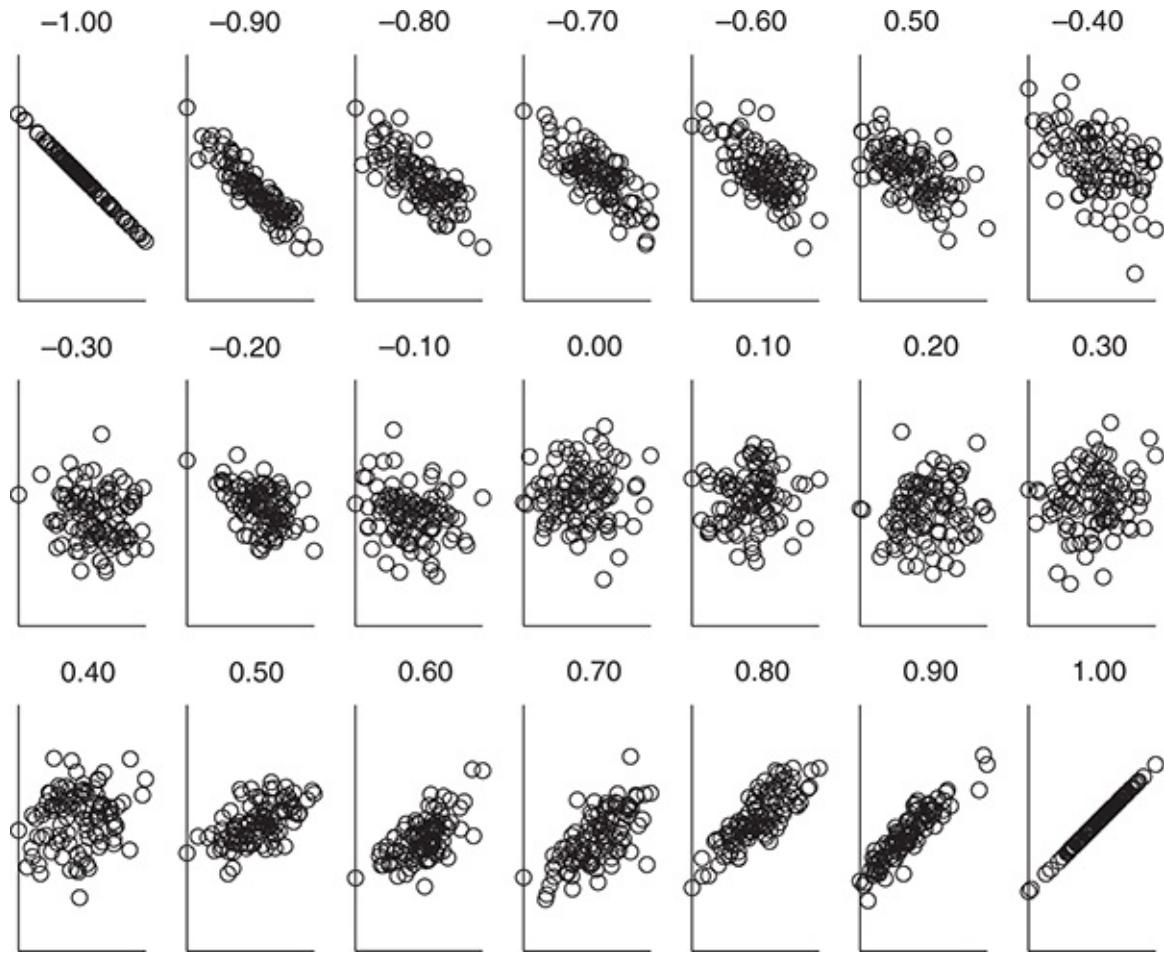


Figure 2.17.

Scatter plots illustrating correlations from -1 to 1 .

If we transform \mathbf{x} and \mathbf{y} by subtracting off their means and then normalizing them so that their lengths are 1 , then their correlation can be calculated by taking the dot product. Let us refer to these transformed vectors of \mathbf{x} and \mathbf{y} as \mathbf{x}' and \mathbf{y}' , respectively. (Notice that this transformation is not the same as the standardization used in other contexts, where we subtract the means and divide by the standard deviations, as discussed in [Section 2.3.7](#).) This transformation highlights an interesting relationship between the correlation measure and the cosine measure. Specifically, the correlation between \mathbf{x} and \mathbf{y} is identical to the cosine between \mathbf{x}' and \mathbf{y}' . However, the cosine between \mathbf{x} and \mathbf{y} is not the same as the cosine between \mathbf{x}' and \mathbf{y}' , even though they both have the same correlation measure. In general, the correlation between two

vectors is equal to the cosine measure only in the special case when the means of the two vectors are 0.

Differences Among Measures For Continuous Attributes

In this section, we illustrate the difference among the three proximity measures for continuous attributes that we have just defined: cosine, correlation, and Minkowski distance. Specifically, we consider two types of data transformations that are commonly used, namely, scaling (multiplication) by a constant factor and translation (addition) by a constant value. A proximity measure is considered to be invariant to a data transformation if its value remains unchanged even after performing the transformation. [Table 2.12](#) compares the behavior of cosine, correlation, and Minkowski distance measures regarding their invariance to scaling and translation operations. It can be seen that while correlation is invariant to both scaling and translation, cosine is only invariant to scaling but not to translation. Minkowski distance measures, on the other hand, are sensitive to both scaling and translation and are thus invariant to neither.

Table 2.12. Properties of cosine, correlation, and Minkowski distance measures.

Property	Cosine	Correlation	Minkowski Distance
Invariant to scaling (multiplication)	Yes	Yes	No
Invariant to translation (addition)	No	Yes	No

Let us consider an example to demonstrate the significance of these differences among different proximity measures.

Example 2.21 (Comparing proximity measures).

Consider the following two vectors \mathbf{x} and \mathbf{y} with seven numeric attributes.

$$\mathbf{x} = (1, 2, 4, 3, 0, 0, 0) \quad \mathbf{y} = (1, 2, 3, 4, 0, 0, 0)$$

It can be seen that both \mathbf{x} and \mathbf{y} have 4 non-zero values, and the values in the two vectors are mostly the same, except for the third and the fourth components. The cosine, correlation, and Euclidean distance between the two vectors can be computed as follows.

$$\begin{aligned} \cos(\mathbf{x}, \mathbf{y}) &= 2930 \times 30 = 0.9667 \\ \text{correlation}(\mathbf{x}, \mathbf{y}) &= 2.35711.5811 \times 1.5811 = 0.9429E \\ \|\mathbf{x} - \mathbf{y}\| &= 1.4142 \end{aligned}$$

Not surprisingly, \mathbf{x} and \mathbf{y} have a cosine and correlation measure close to 1, while the Euclidean distance between them is small, indicating that they are quite similar. Now let us consider the vector \mathbf{ys} , which is a scaled version of \mathbf{y} (multiplied by a constant factor of 2), and the vector \mathbf{yt} , which is constructed by translating \mathbf{y} by 5 units as follows.

$$\mathbf{ys} = 2 \times \mathbf{y} = (2, 4, 6, 8, 0, 0, 0)$$

$$\mathbf{yt} = \mathbf{y} + 5 = (6, 7, 8, 9, 5, 5, 5)$$

We are interested in finding whether \mathbf{ys} and \mathbf{yt} show the same proximity with \mathbf{x} as shown by the original vector \mathbf{y} . [Table 2.13](#) shows the different measures of proximity computed for the pairs (\mathbf{x}, \mathbf{y}) , $(\mathbf{x}, \mathbf{ys})$, and $(\mathbf{x}, \mathbf{yt})$. It can be seen that the value of correlation between \mathbf{x} and \mathbf{y} remains unchanged even after replacing \mathbf{y} with \mathbf{ys} or \mathbf{yt} . However, the value of cosine remains equal to 0.9667 when computed for (\mathbf{x}, \mathbf{y}) and $(\mathbf{x}, \mathbf{ys})$, but significantly reduces to 0.7940 when computed for $(\mathbf{x}, \mathbf{yt})$. This highlights

the fact that cosine is invariant to the scaling operation but not to the translation operation, in contrast with the correlation measure. The Euclidean distance, on the other hand, shows different values for all three pairs of vectors, as it is sensitive to both scaling and translation.

Table 2.13. Similarity between (x, y) , (x, ys) , and (x, yt) .

Measure	(x, y)	(x, ys)	(x, yt)
Cosine	0.9667	0.9667	0.7940
Correlation	0.9429	0.9429	0.9429
Euclidean Distance	1.4142	5.8310	14.2127

We can observe from this example that different proximity measures behave differently when scaling or translation operations are applied on the data. The choice of the right proximity measure thus depends on the desired notion of similarity between data objects that is meaningful for a given application. For example, if x and y represented the frequencies of different words in a document-term matrix, it would be meaningful to use a proximity measure that remains unchanged when y is replaced by ys , because ys is just a scaled version of y with the same distribution of words occurring in the document. However, yt is different from y , since it contains a large number of words with non-zero frequencies that do not occur in y . Because cosine is invariant to scaling but not to translation, it will be an ideal choice of proximity measure for this application.

Consider a different scenario in which x represents a location's temperature measured on the Celsius scale for seven days. Let y , ys , and yt be the temperatures measured on those days at a different location, but using three different measurement scales. Note that different units of

temperature have different offsets (e.g. Celsius and Kelvin) and different scaling factors (e.g. Celsius and Fahrenheit). It is thus desirable to use a proximity measure that captures the proximity between temperature values without being affected by the measurement scale. Correlation would then be the ideal choice of proximity measure for this application, as it is invariant to both scaling and translation.

As another example, consider a scenario where \mathbf{x} represents the amount of precipitation (in cm) measured at seven locations. Let \mathbf{y} , \mathbf{y}_s , and \mathbf{y}_t be estimates of the precipitation at these locations, which are predicted using three different models. Ideally, we would like to choose a model that accurately reconstructs the measurements in \mathbf{x} without making any error. It is evident that \mathbf{y} provides a good approximation of the values in \mathbf{x} , whereas \mathbf{y}_s and \mathbf{y}_t provide poor estimates of precipitation, even though they do capture the trend in precipitation across locations. Hence, we need to choose a proximity measure that penalizes any difference in the model estimates from the actual observations, and is sensitive to both the scaling and translation operations. The Euclidean distance satisfies this property and thus would be the right choice of proximity measure for this application. Indeed, the Euclidean distance is commonly used in computing the accuracy of models, which will be discussed later in

[Chapter 3](#) ▾.

2.4.6 Mutual Information

Like correlation, mutual information is used as a measure of similarity between two sets of paired values that is sometimes used as an alternative to correlation, particularly when a nonlinear relationship is suspected between the pairs of values. This measure comes from information theory, which is the

study of how to formally define and quantify information. Indeed, mutual information is a measure of how much information one set of values provides about another, given that the values come in pairs, e.g., height and weight. If the two sets of values are independent, i.e., the value of one tells us nothing about the other, then their mutual information is 0. On the other hand, if the two sets of values are completely dependent, i.e., knowing the value of one tells us the value of the other and vice-versa, then they have maximum mutual information. Mutual information does not have a maximum value, but we will define a normalized version of it that ranges between 0 and 1.

To define mutual information, we consider two sets of values, X and Y , which occur in pairs (X, Y). We need to measure the average information in a single set of values, i.e., either in X or in Y , and in the pairs of their values. This is commonly measured by entropy. More specifically, assume X and Y are discrete, that is, X can take m distinct values, u_1, u_2, \dots, u_m and Y can take n distinct values, v_1, v_2, \dots, v_n . Then their individual and joint entropy can be defined in terms of the probabilities of each value and pair of values as follows:

$$H(X) = -\sum_{j=1}^m P(X=u_j) \log_2 P(X=u_j) \quad (2.12)$$

$$H(Y) = -\sum_{k=1}^n P(Y=v_k) \log_2 P(Y=v_k) \quad (2.13)$$

$$H(X, Y) = -\sum_{j=1}^m \sum_{k=1}^n P(X=u_j, Y=v_k) \log_2 P(X=u_j, Y=v_k) \quad (2.14)$$

where if the probability of a value or combination of values is 0, then $0 \log_2(0)$ is conventionally taken to be 0.

The mutual information of X and Y can now be defined straightforwardly:

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \quad (2.15)$$

Note that $H(X, Y)$ is symmetric, i.e., $H(X, Y)=H(Y, X)$, and thus mutual information is also symmetric, i.e., $I(X, Y)=I(Y, X)$.

Practically, X and Y are either the values in two attributes or two rows of the same data set. In [Example 2.22](#), we will represent those values as two vectors \mathbf{x} and \mathbf{y} and calculate the probability of each value or pair of values from the frequency with which values or pairs of values occur in \mathbf{x} , \mathbf{y} and (x_i, y_i) , where x_i is the i th component of \mathbf{x} and y_i is the i th component of \mathbf{y} . Let us illustrate using a previous example.

Example 2.22 (Evaluating Nonlinear Relationships with Mutual Information).

Recall [Example 2.19](#) where $y_k = x_k^2$, but their correlation was 0.

$$x = (-3, -2, -1, 0, 1, 2, 3) \quad y = (9, 4, 1, 0, 1, 4, 9)$$

From Figure 2.22 □, $I(x, y) = H(x) + H(y) - H(x, y) = 1.9502$. Although a variety of approaches to normalize mutual information are possible—see Bibliographic Notes—for this example, we will apply one that divides the mutual information by $\log_2(\min(m, n))$ and produces a result between 0 and 1. This yields a value of $1.9502/\log_2(4) = 0.9751$. Thus, we can see that x and y are strongly related. They are not perfectly related because given a value of y there is, except for $y=0$, some ambiguity about the value of x . Notice that for $y=-x$, the normalized mutual information would be 1.

Figure 2.18.

Computation of mutual information.

Table 2.14. Entropy for x

x_j	$P(x=x_j)$	$-P(x=x_j)\log_2 P(x=x_j)$
-------	------------	----------------------------

-3	1/7	0.4011
-2	1/7	0.4011
-1	1/7	0.4011
0	1/7	0.4011
1	1/7	0.4011
2	1/7	0.4011
3	1/7	0.4011
$H(x)$		2.8074

Table 2.15. Entropy for y

yk	P(y=yk)	$-P(y=yk)\log_2 P(y=yk)$
9	2/7	0.5164
4	2/7	0.5164
1	2/7	0.5164
0	1/7	0.4011
$H(y)$		1.9502

Table 2.16. Joint entropy for x and y

xj	yk	$P(x=x_j, y=y_k)$	$-P(x=x_j, y=y_k)\log_2 P(x=x_j, y=y_k)$
-3	9	1/7	0.4011
-2	4	1/7	0.4011

-1	1	1/7	0.4011
0	0	1/7	0.4011
1	1	1/7	0.4011
2	4	1/7	0.4011
3	9	1/7	0.4011
$H(\mathbf{x}, \mathbf{y})$			2.8074

2.4.7 Kernel Functions*

It is easy to understand how similarity and distance might be useful in an application such as clustering, which tries to group similar objects together. What is much less obvious is that many other data analysis tasks, including predictive modeling and dimensionality reduction, can be expressed in terms of pairwise “proximities” of data objects. More specifically, many data analysis problems can be mathematically formulated to take as input, a **kernel matrix**, \mathbf{K} , which can be considered a type of proximity matrix. Thus, an initial preprocessing step is used to convert the input data into a kernel matrix, which is the input to the data analysis algorithm.

More formally, if a data set has m data objects, then \mathbf{K} is an m by m matrix. If x_i and x_j are the i th and j th data objects, respectively, then k_{ij} , the ij th entry of \mathbf{K} , is computed by a **kernel function**:

$$k_{ij} = \kappa(x_i, x_j) \quad (2.16)$$

As we will see in the material that follows, the use of a kernel matrix allows both wider applicability of an algorithm to various kinds of data and an ability to model nonlinear relationships with algorithms that are designed only for detecting linear relationships.

Kernels make an algorithm data independent

If an algorithm uses a kernel matrix, then it can be used with any type of data for which a kernel function can be designed. This is illustrated by Algorithm 2.1. Although only some data analysis algorithms can be modified to use a kernel matrix as input, this approach is extremely powerful because it allows such an algorithm to be used with almost any type of data for which an appropriate kernel function can be defined. Thus, a classification algorithm can be used, for example, with record data, string data, or graph data. If an algorithm can be reformulated to use a kernel matrix, then its applicability to different types of data increases dramatically. As we will see in later chapters, many clustering, classification, and anomaly detection algorithms work only with similarities or distances, and thus, can be easily modified to work with kernels.

Algorithm 2.1 Basic kernel algorithm.

1. Read in the m data objects in the data set.
2. Compute the kernel matrix, \mathbf{K} by applying the kernel function, κ , to each pair of data objects.
3. Run the data analysis algorithm with \mathbf{K} as input.
4. Return the analysis result, e.g., predicted class or cluster labels.

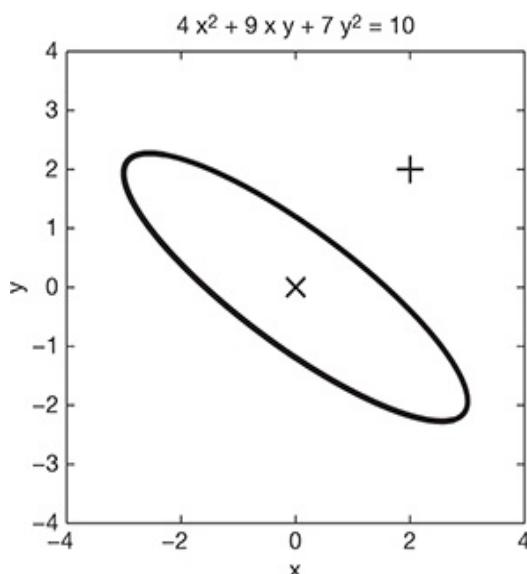
Mapping data into a higher dimensional data space can

allow modeling of nonlinear relationships

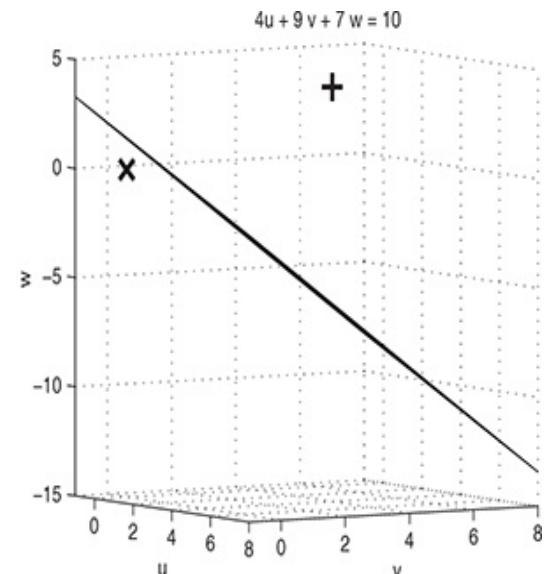
There is yet another, equally important, aspect of kernel based data algorithms—their ability to model nonlinear relationships with algorithms that model only linear relationships. Typically, this works by first transforming (mapping) the data from a lower dimensional data space to a higher dimensional space.

Example 2.23 (Mapping Data to a Higher Dimensional Space).

Consider the relationship between two variables x and y given by the following equation, which defines an ellipse in two dimensions ([Figure 2.19\(a\)](#)):



(a) Ellipse and two points in 2 dimensions.



(b) Data mapped to 3 dimensions.

Figure 2.19.

Mapping data to a higher dimensional space: two to three dimensions.

$$4x^2 + 9xy + 7y^2 = 10 \quad (2.17)$$

We can map our two dimensional data to three dimensions by creating three new variables, u , v , and w , which are defined as follows:

$$w=x^2, u=xy, v=y^2$$

As a result, we can now express [Equation 2.17](#) as a linear one. This equation describes a plane in three dimensions. Points on the ellipse will lie on that plane, while points inside and outside the ellipse will lie on opposite sides of the plane. See [Figure 2.19\(b\)](#). The viewpoint of this 3D plot is along the surface of the separating plane so that the plane appears as a line.

$$4u + 9v + 7w = 10 \quad (2.18)$$

The Kernel Trick

The approach illustrated above shows the value in mapping data to higher dimensional space, an operation that is integral to kernel-based methods. Conceptually, we first define a function ϕ that maps data points x and y to data points $\phi(x)$ and $\phi(y)$ in a higher dimensional space such that the inner product $\langle x, y \rangle$ gives the desired measure of proximity of x and y . It may seem that we have potentially sacrificed a great deal by using such an approach, because we can greatly expand the size of our data, increase the computational complexity of our analysis, and encounter problems with the curse of dimensionality by computing similarity in a high-dimensional space. However, this is not the case since these problems can be avoided by defining a kernel function κ that can compute the same similarity value, but with the data points in the original space, i.e., $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle$. This is known as the *kernel trick*. Despite the name, the kernel trick has a very solid

mathematical foundation and is a remarkably powerful approach for data analysis.

Not every function of a pair of data objects satisfies the properties needed for a kernel function, but it has been possible to design many useful kernels for a wide variety of data types. For example, three common kernel functions are the polynomial, Gaussian (radial basis function (RBF)), and sigmoid kernels. If x and y are two data objects, specifically, two data vectors, then these two kernel functions can be expressed as follows, respectively:

$$\kappa(x, y) = (x'y + c)^d \quad (2.19)$$

$$\kappa(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2) \quad (2.20)$$

$$\kappa(x, y) = \tanh(\alpha x'y + c) \quad (2.21)$$

where α and $c \geq 0$ are constants, d is an integer parameter that gives the polynomial degree, $\|x - y\|$ is the length of the vector $x - y$ and $\sigma > 0$ is a parameter that governs the “spread” of a Gaussian.

Example 2.24 (The Polynomial Kernel).

Note that the kernel functions presented in the previous section are computing the same similarity value as would be computed if we actually mapped the data to a higher dimensional space and then computed an inner product there. For example, for the polynomial kernel of degree 2, let φ be the function that maps a two-dimensional data vector $x = (x_1, x_2)$ to the higher dimensional space. Specifically, let

$$\varphi(x) = (x_1^2, x_2^2, 2x_1x_2, 2cx_1, 2cx_2, c). \quad (2.22)$$

For the higher dimensional space, let the proximity be defined as the inner product of $\varphi(x)$ and $\varphi(y)$, i.e., $\langle \varphi(x), \varphi(y) \rangle$. Then, as previously mentioned, it can be shown that

$$\kappa(x, y) = \langle \varphi(x), \varphi(y) \rangle \quad (2.23)$$

where κ is defined by [Equation 2.19](#) above. Specifically, if $x=(x_1, x_2)$ and $y=(y_1, y_2)$, then

$$\kappa(x, y) = \langle x, y \rangle = x'y = (x_1y_1, x_2y_2, 2x_1x_2y_1y_2, 2cx_1y_1, 2cx_2y_2, c^2) \quad (2.24)$$

More generally, the kernel trick depends on defining κ and φ so that [Equation 2.23](#) holds. This has been done for a wide variety of kernels.

This discussion of kernel-based approaches was intended only to provide a brief introduction to this topic and has omitted many details. A fuller discussion of the kernel-based approach is provided in Section 4.9.4, which discusses these issues in the context of nonlinear support vector machines for classification. More general references for the kernel based analysis can be found in the Bibliographic Notes of this chapter.

2.4.8 Bregman Divergence*

This section provides a brief description of Bregman divergences, which are a family of proximity functions that share some common properties. As a result, it is possible to construct general data mining algorithms, such as clustering algorithms, that work with any Bregman divergence. A concrete example is the K-means clustering algorithm (Section 7.2). Note that this section requires knowledge of vector calculus.

Bregman divergences are loss or distortion functions. To understand the idea of a loss function, consider the following. Let \mathbf{x} and \mathbf{y} be two points, where \mathbf{y} is regarded as the original point and \mathbf{x} is some distortion or approximation of it. For example, \mathbf{x} may be a point that was generated by adding random noise to \mathbf{y} . The goal is to measure the resulting distortion or loss that results if \mathbf{y} is approximated by \mathbf{x} . Of course, the more similar \mathbf{x} and \mathbf{y} are, the smaller the loss or distortion. Thus, Bregman divergences can be used as dissimilarity functions.

More formally, we have the following definition.

Definition 2.6 (Bregman divergence)

Given a strictly convex function ϕ (with a few modest restrictions that are generally satisfied), the Bregman divergence (loss function) $D(\mathbf{x}, \mathbf{y})$ generated by that function is given by the following equation:

$$D(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) - \langle \nabla\phi(\mathbf{y}), (\mathbf{x}-\mathbf{y}) \rangle \quad (2.25)$$

where $\nabla\phi(\mathbf{y})$ is the gradient of ϕ evaluated at \mathbf{y} , $\mathbf{x}-\mathbf{y}$, is the vector difference between \mathbf{x} and \mathbf{y} , and $\langle \nabla\phi(\mathbf{y}), (\mathbf{x}-\mathbf{y}) \rangle$ is the inner product between $\nabla\phi(\mathbf{y})$ and $(\mathbf{x}-\mathbf{y})$. For points in Euclidean space, the inner product is just the dot product.

$D(\mathbf{x}, \mathbf{y})$ can be written as $D(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - L(\mathbf{x})$, where $L(\mathbf{x}) = \phi(\mathbf{y}) + \langle \nabla\phi(\mathbf{y}), (\mathbf{x}-\mathbf{y}) \rangle$ and represents the equation of a plane that is tangent to the function ϕ at \mathbf{y} .

Using calculus terminology, $L(\mathbf{x})$ is the linearization of ϕ around the point \mathbf{y} , and the Bregman divergence is just the difference between a function and a linear approximation to that function. Different Bregman divergences are obtained by using different choices for ϕ .

Example 2.25.

We provide a concrete example using squared Euclidean distance, but restrict ourselves to one dimension to simplify the mathematics. Let x and y be real numbers and $\phi(t)$ be the real-valued function, $\phi(t)=t^2$. In that case, the gradient reduces to the derivative, and the dot product reduces to multiplication. Specifically, [Equation 2.25](#) becomes [Equation 2.26](#).

$$D(x,y) = x^2 - y^2 - 2y(x-y) = (x-y)^2 \quad (2.26)$$

The graph for this example, with $y=1$, is shown in [Figure 2.20](#). The Bregman divergence is shown for two values of x : $x=2$ and $x=3$.

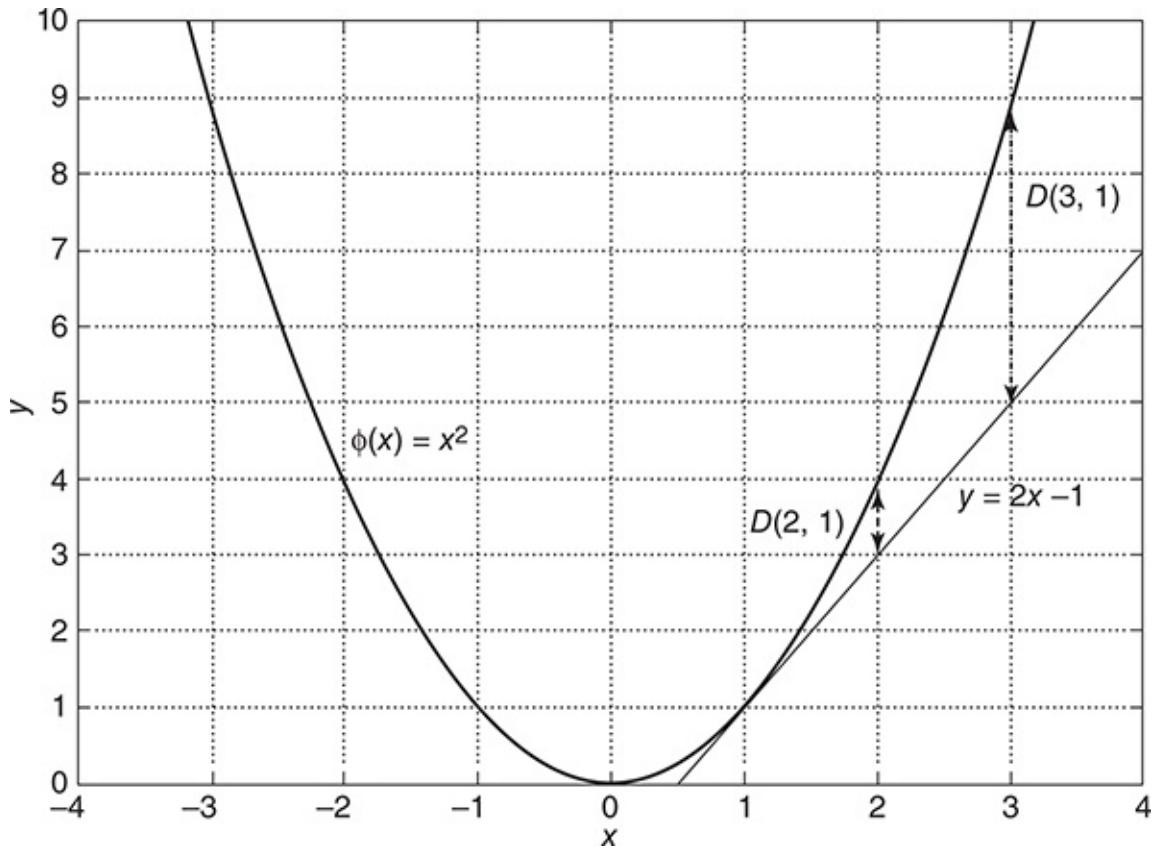


Figure 2.20.

Illustration of Bregman divergence.

2.4.9 Issues in Proximity Calculation

This section discusses several important issues related to proximity measures: (1) how to handle the case in which attributes have different scales and/or are correlated, (2) how to calculate proximity between objects that are composed of different types of attributes, e.g., quantitative and qualitative, (3) and how to handle proximity calculations when attributes have different weights; i.e., when not all attributes contribute equally to the proximity of objects.

Standardization and Correlation for Distance

Measures

An important issue with distance measures is how to handle the situation when attributes do not have the same range of values. (This situation is often described by saying that “the variables have different scales.”) In a previous example, Euclidean distance was used to measure the distance between people based on two attributes: age and income. Unless these two attributes are standardized, the distance between two people will be dominated by income.

A related issue is how to compute distance when there is correlation between some of the attributes, perhaps in addition to differences in the ranges of values. A generalization of Euclidean distance, the **Mahalanobis distance**, is useful when attributes are correlated, have different ranges of values (different variances), and the distribution of the data is approximately Gaussian (normal). Correlated variables have a large impact on standard distance measures since a change in any of the correlated variables is reflected in a change in all the correlated variables. Specifically, the Mahalanobis distance between two objects (vectors) \mathbf{x} and \mathbf{y} is defined as

$$\text{Mahalanobis}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})' \Sigma^{-1} (\mathbf{x} - \mathbf{y}), \quad (2.27)$$

where Σ^{-1} is the inverse of the covariance matrix of the data. Note that the covariance matrix Σ is the matrix whose ij th entry is the covariance of the i th and j th attributes as defined by [Equation 2.11](#).

Example 2.26.

In [Figure 2.21](#), there are 1000 points, whose x and y attributes have a correlation of 0.6. The distance between the two large points at the opposite ends of the long axis of the ellipse is 14.7 in terms of Euclidean

distance, but only 6 with respect to Mahalanobis distance. This is because the Mahalanobis distance gives less emphasis to the direction of largest variance. In practice, computing the Mahalanobis distance is expensive, but can be worthwhile for data whose attributes are correlated. If the attributes are relatively uncorrelated, but have different ranges, then standardizing the variables is sufficient.

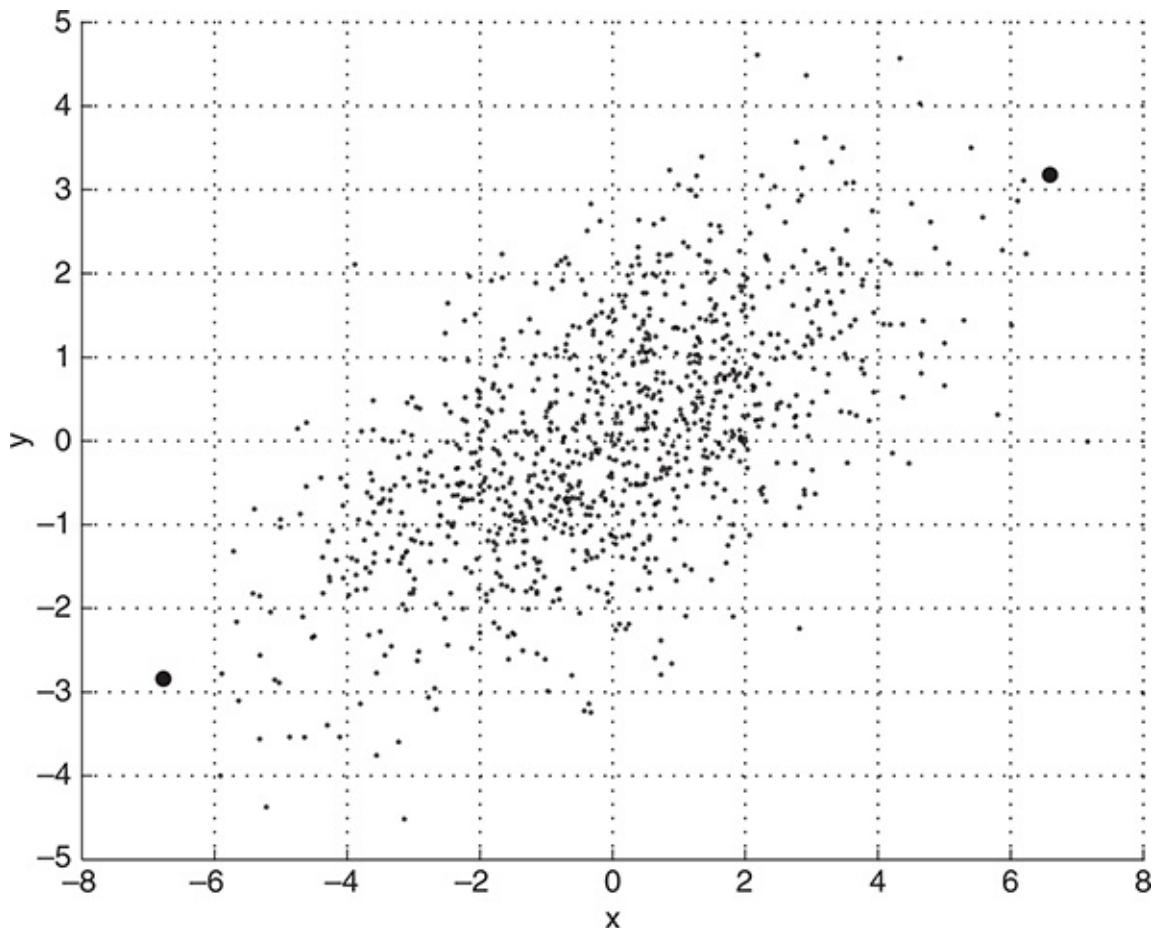


Figure 2.21.

Set of two-dimensional points. The Mahalanobis distance between the two points represented by large dots is 6; their Euclidean distance is 14.7.

Combining Similarities for Heterogeneous Attributes

The previous definitions of similarity were based on approaches that assumed all the attributes were of the same type. A general approach is needed when the attributes are of different types. One straightforward approach is to compute the similarity between each attribute separately using [Table 2.7](#), and then combine these similarities using a method that results in a similarity between 0 and 1. One possible approach is to define the overall similarity as the average of all the individual attribute similarities. Unfortunately, this approach does not work well if some of the attributes are asymmetric attributes. For example, if all the attributes are asymmetric binary attributes, then the similarity measure suggested previously reduces to the simple matching coefficient, a measure that is not appropriate for asymmetric binary attributes. The easiest way to fix this problem is to omit asymmetric attributes from the similarity calculation when their values are 0 for both of the objects whose similarity is being computed. A similar approach also works well for handling missing values.

In summary, Algorithm 2.2 is effective for computing an overall similarity between two objects, x and y , with different types of attributes. This procedure can be easily modified to work with dissimilarities.

Algorithm 2.2 Similarities of heterogeneous objects.

1: For the k th attribute, compute a similarity, $s_k(x, y)$, in the range [0, 1].

2: Define an indicator variable, δ_k , for the k th attribute as follows:

$\delta_k = \{$

0 if the k th attribute is an asymmetric attribute and both objects have :

3: Compute the overall similarity between the two objects using the following formula:

$$(2.28) \text{similarity } (x, y) = \sum_{k=1}^n w_k \delta_k s_k(x, y) / \sum_{k=1}^n w_k$$

Using Weights

In much of the previous discussion, all attributes were treated equally when computing proximity. This is not desirable when some attributes are more important to the definition of proximity than others. To address these situations, the formulas for proximity can be modified by weighting the contribution of each attribute.

With attribute weights, w_k , (2.28) becomes

$$\text{similarity } (x, y) = \sum_{k=1}^n w_k \delta_k s_k(x, y) / \sum_{k=1}^n w_k. \quad (2.29)$$

The definition of the Minkowski distance can also be modified as follows:

$$d(x, y) = (\sum_{k=1}^n w_k |x_k - y_k|^r)^{1/r}. \quad (2.30)$$

2.4.10 Selecting the Right Proximity Measure

A few general observations may be helpful. First, the type of proximity measure should fit the type of data. For many types of dense, continuous data, metric distance measures such as Euclidean distance are often used. Proximity between continuous attributes is most often expressed in terms of

differences, and distance measures provide a well-defined way of combining these differences into an overall proximity measure. Although attributes can have different scales and be of differing importance, these issues can often be dealt with as described earlier, such as normalization and weighting of attributes.

For sparse data, which often consists of asymmetric attributes, we typically employ similarity measures that ignore 0–0 matches. Conceptually, this reflects the fact that, for a pair of complex objects, similarity depends on the number of characteristics they both share, rather than the number of characteristics they both lack. The cosine, Jaccard, and extended Jaccard measures are appropriate for such data.

There are other characteristics of data vectors that often need to be considered. Invariance to scaling (multiplication) and to translation (addition) were previously discussed with respect to Euclidean distance and the cosine and correlation measures. The practical implications of such considerations are that, for example, cosine is more suitable for sparse document data where only scaling is important, while correlation works better for time series, where both scaling and translation are important. Euclidean distance or other types of Minkowski distance are most appropriate when two data vectors are to match as closely as possible across all components (features).

In some cases, transformation or normalization of the data is needed to obtain a proper similarity measure. For instance, time series can have trends or periodic patterns that significantly impact similarity. Also, a proper computation of similarity often requires that time lags be taken into account. Finally, two time series may be similar only over specific periods of time. For example, there is a strong relationship between temperature and the use of natural gas, but only during the heating season.

Practical consideration can also be important. Sometimes, one or more proximity measures are already in use in a particular field, and thus, others will have answered the question of which proximity measures should be used. Other times, the software package or clustering algorithm being used can drastically limit the choices. If efficiency is a concern, then we may want to choose a proximity measure that has a property, such as the triangle inequality, that can be used to reduce the number of proximity calculations. (See [Exercise 25](#).)

However, if common practice or practical restrictions do not dictate a choice, then the proper choice of a proximity measure can be a time-consuming task that requires careful consideration of both domain knowledge and the purpose for which the measure is being used. A number of different similarity measures may need to be evaluated to see which ones produce results that make the most sense.

2.5 Bibliographic Notes

It is essential to understand the nature of the data that is being analyzed, and at a fundamental level, this is the subject of measurement theory. In particular, one of the initial motivations for defining types of attributes was to be precise about which statistical operations were valid for what sorts of data. We have presented the view of measurement theory that was initially described in a classic paper by S. S. Stevens [112]. ([Tables 2.2](#) and [2.3](#) are derived from those presented by Stevens [113].) While this is the most common view and is reasonably easy to understand and apply, there is, of course, much more to measurement theory. An authoritative discussion can be found in a three-volume series on the foundations of measurement theory [88, 94, 114]. Also of interest is a wide-ranging article by Hand [77], which discusses measurement theory and statistics, and is accompanied by comments from other researchers in the field. Numerous critiques and extensions of the approach of Stevens have been made [66, 97, 117]. Finally, many books and articles describe measurement issues for particular areas of science and engineering.

Data quality is a broad subject that spans every discipline that uses data. Discussions of precision, bias, accuracy, and significant figures can be found in many introductory science, engineering, and statistics textbooks. The view of data quality as “fitness for use” is explained in more detail in the book by Redman [103]. Those interested in data quality may also be interested in MIT’s Information Quality (MITIQ) Program [95, 118]. However, the knowledge needed to deal with specific data quality issues in a particular domain is often best obtained by investigating the data quality practices of researchers in that field.

Aggregation is a less well-defined subject than many other preprocessing tasks. However, aggregation is one of the main techniques used by the database area of Online Analytical Processing (OLAP) [68, 76, 102]. There has also been relevant work in the area of symbolic data analysis (Bock and Diday [64]). One of the goals in this area is to summarize traditional record data in terms of symbolic data objects whose attributes are more complex than traditional attributes. Specifically, these attributes can have values that are sets of values (categories), intervals, or sets of values with weights (histograms). Another goal of symbolic data analysis is to be able to perform clustering, classification, and other kinds of data analysis on data that consists of symbolic data objects.

Sampling is a subject that has been well studied in statistics and related fields. Many introductory statistics books, such as the one by Lindgren [90], have some discussion about sampling, and entire books are devoted to the subject, such as the classic text by Cochran [67]. A survey of sampling for data mining is provided by Gu and Liu [74], while a survey of sampling for databases is provided by Olken and Rotem [98]. There are a number of other data mining and database-related sampling references that may be of interest, including papers by Palmer and Faloutsos [100], Provost et al. [101], Toivonen [115], and Zaki et al. [119].

In statistics, the traditional techniques that have been used for dimensionality reduction are multidimensional scaling (MDS) (Borg and Groenen [65], Kruskal and Uslaner [89]) and principal component analysis (PCA) (Jolliffe [80]), which is similar to singular value decomposition (SVD) (Demmel [70]). Dimensionality reduction is discussed in more detail in Appendix B.

Discretization is a topic that has been extensively investigated in data mining. Some classification algorithms work only with categorical data, and association analysis requires binary data, and thus, there is a significant

motivation to investigate how to best binarize or discretize continuous attributes. For association analysis, we refer the reader to work by Srikant and Agrawal [111], while some useful references for discretization in the area of classification include work by Dougherty et al. [71], Elomaa and Rousu [72], Fayyad and Irani [73], and Hussain et al. [78].

Feature selection is another topic well investigated in data mining. A broad coverage of this topic is provided in a survey by Molina et al. [96] and two books by Liu and Motada [91, 92]. Other useful papers include those by Blum and Langley [63], Kohavi and John [87], and Liu et al. [93].

It is difficult to provide references for the subject of feature transformations because practices vary from one discipline to another. Many statistics books have a discussion of transformations, but typically the discussion is restricted to a particular purpose, such as ensuring the normality of a variable or making sure that variables have equal variance. We offer two references: Osborne [99] and Tukey [116].

While we have covered some of the most commonly used distance and similarity measures, there are hundreds of such measures and more are being created all the time. As with so many other topics in this chapter, many of these measures are specific to particular fields, e.g., in the area of time series see papers by Kalpakis et al. [81] and Keogh and Pazzani [83]. Clustering books provide the best general discussions. In particular, see the books by Anderberg [62], Jain and Dubes [79], Kaufman and Rousseeuw [82], and Sneath and Sokal [109].

Information-based measures of similarity have become more popular lately despite the computational difficulties and expense of calculating them. A good introduction to information theory is provided by Cover and Thomas [69]. Computing the mutual information for continuous variables can be

straightforward if they follow a well-known distribution, such as Gaussian. However, this is often not the case, and many techniques have been developed. As one example, the article by Khan, et al. [85] compares various methods in the context of comparing short time series. See also the information and mutual information packages for R and Matlab. Mutual information has been the subject of considerable recent attention due to paper by Reshef, et al. [104, 105] that introduced an alternative measure, albeit one based on mutual information, which was claimed to have superior properties. Although this approach had some early support, e.g., [110], others have pointed out various limitations [75, 86, 108].

Two popular books on the topic of kernel methods are [106] and [107]. The latter also has a website with links to kernel-related materials [84]. In addition, many current data mining, machine learning, and statistical learning textbooks have some material about kernel methods. Further references for kernel methods in the context of support vector machine classifiers are provided in the bibliographic Notes of Section 4.9.4.

Bibliography

- [62] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, December 1973.
- [63] A. Blum and P. Langley. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, 97(1–2):245–271, 1997.
- [64] H. H. Bock and E. Diday. *Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data (Studies in Classification, Data Analysis, and Knowledge Organization)*. Springer-Verlag Telos, January 2000.
- [65] I. Borg and P. Groenen. *Modern Multidimensional Scaling—Theory and Applications*. Springer-Verlag, February 1997.
- [66] N. R. Chrisman. Rethinking levels of measurement for cartography. *Cartography and Geographic Information Systems*, 25(4):231–242, 1998.
- [67] W. G. Cochran. *Sampling Techniques*. John Wiley & Sons, 3rd edition, July 1977.
- [68] E. F. Codd, S. B. Codd, and C. T. Smalley. Providing OLAP (On-line Analytical Processing) to User- Analysts: An IT Mandate. White Paper, E.F. Codd and Associates, 1993.

- [69] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.
- [70] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial & Applied Mathematics, September 1997.
- [71] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Proc. of the 12th Intl. Conf. on Machine Learning*, pages 194–202, 1995.
- [72] T. Elomaa and J. Rousu. General and Efficient Multisplitting of Numerical Attributes. *Machine Learning*, 36(3):201–244, 1999.
- [73] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuousvalued attributes for classification learning. In *Proc. 13th Int. Joint Conf. on Artificial Intelligence*, pages 1022–1027. Morgan Kaufman, 1993.
- [74] F. H. Gaohua Gu and H. Liu. Sampling and Its Application in Data Mining: A Survey. Technical Report TRA6/00, National University of Singapore, Singapore, 2000.
- [75] M. Gorfine, R. Heller, and Y. Heller. Comment on Detecting novel associations in large data sets. *Unpublished (available at <http://emotion.technion.ac.il/gorfinm/files/science6.pdf> on 11 Nov. 2012)*, 2012.

- [76] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Journal Data Mining and Knowledge Discovery*, 1(1): 29–53, 1997.
- [77] D. J. Hand. Statistics and the Theory of Measurement. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 159(3):445–492, 1996.
- [78] F. Hussain, H. Liu, C. L. Tan, and M. Dash. TRC6/99: Discretization: an enabling technique. Technical report, National University of Singapore, Singapore, 1999.
- [79] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series. Prentice Hall, March 1988.
- [80] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2nd edition, October 2002.
- [81] K. Kalpakis, D. Gada, and V. Puttagunta. Distance Measures for Effective Clustering of ARIMA Time-Series. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 273–280. IEEE Computer Society, 2001.
- [82] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. John Wiley and Sons, New York, November 1990.

- [83] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *KDD*, pages 285–289, 2000.
- [84] Kernel Methods for Pattern Analysis Website. <http://www.kernel-methods.net/>, 2014.
- [85] S. Khan, S. Bandyopadhyay, A. R. Ganguly, S. Saigal, D. J. Erickson III, V. Protopopescu, and G. Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Physical Review E*, 76(2):026209, 2007.
- [86] J. B. Kinney and G. S. Atwal. Equitability, mutual information, and the maximal information coefficient. *Proceedings of the National Academy of Sciences*, 2014.
- [87] R. Kohavi and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.
- [88] D. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurements: Volume 1: Additive and polynomial representations*. Academic Press, New York, 1971.
- [89] J. B. Kruskal and E. M. Uslaner. *Multidimensional Scaling*. Sage Publications, August 1978.
- [90] B. W. Lindgren. *Statistical Theory*. CRC Press, January 1993.

- [91] H. Liu and H. Motoda, editors. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer International Series in Engineering and Computer Science, 453. Kluwer Academic Publishers, July 1998.
- [92] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer International Series in Engineering and Computer Science, 454. Kluwer Academic Publishers, July 1998.
- [93] H. Liu, H. Motoda, and L. Yu. Feature Extraction, Selection, and Construction. In N. Ye, editor, *The Handbook of Data Mining*, pages 22–41. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 2003.
- [94] R. D. Luce, D. Krantz, P. Suppes, and A. Tversky. *Foundations of Measurements: Volume 3: Representation, Axiomatization, and Invariance*. Academic Press, New York, 1990.
- [95] MIT Information Quality (MITIQ) Program. <http://mitiq.mit.edu/>, 2014.
- [96] L. C. Molina, L. Belanche, and A. Nebot. Feature Selection Algorithms: A Survey and Experimental Evaluation. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, 2002.
- [97] F. Mosteller and J. W. Tukey. *Data analysis and regression: a second course in statistics*. Addison-Wesley, 1977.

- [98] F. Olken and D. Rotem. Random Sampling from Databases—A Survey. *Statistics & Computing*, 5(1):25–42, March 1995.
- [99] J. Osborne. Notes on the Use of Data Transformations. *Practical Assessment, Research & Evaluation*, 28(6), 2002.
- [100] C. R. Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. *ACM SIGMOD Record*, 29(2):82–92, 2000.
- [101] F. J. Provost, D. Jensen, and T. Oates. Efficient Progressive Sampling. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 23–32, 1999.
- [102] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, August 2002.
- [103] T. C. Redman. *Data Quality: The Field Guide*. Digital Press, January 2001.
- [104] D. Reshef, Y. Reshef, M. Mitzenmacher, and P. Sabeti. Equitability analysis of the maximal information coefficient, with comparisons. *arXiv preprint arXiv:1301.6314*, 2013.
- [105] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C.

Sabeti. Detecting novel associations in large data sets. *science*, 334(6062):1518–1524, 2011.

[106] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[107] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[108] N. Simon and R. Tibshirani. Comment on "Detecting Novel Associations In Large Data Sets" by Reshef Et Al, Science Dec 16, 2011. *arXiv preprint arXiv:1401.7645*, 2014.

[109] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy*. Freeman, San Francisco, 1971.

[110] T. Speed. A correlation for the 21st century. *Science*, 334(6062):1502–1503, 2011.

[111] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of 1996 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 1–12, Montreal, Quebec, Canada, August 1996.

[112] S. S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, June 1946.

- [113] S. S. Stevens. Measurement. In G. M. Maranell, editor, *Scaling: A Sourcebook for Behavioral Scientists*, pages 22–41. Aldine Publishing Co., Chicago, 1974.
- [114] P. Suppes, D. Krantz, R. D. Luce, and A. Tversky. *Foundations of Measurements: Volume 2: Geometrical, Threshold, and Probabilistic Representations*. Academic Press, New York, 1989.
- [115] H. Toivonen. Sampling Large Databases for Association Rules. In *VLDB96*, pages 134–145. Morgan Kaufman, September 1996.
- [116] J. W. Tukey. On the Comparative Anatomy of Transformations. *Annals of Mathematical Statistics*, 28(3):602–632, September 1957.
- [117] P. F. Velleman and L. Wilkinson. Nominal, ordinal, interval, and ratio typologies are misleading. *The American Statistician*, 47(1):65–72, 1993.
- [118] R. Y. Wang, M. Ziad, Y. W. Lee, and Y. R. Wang. *Data Quality*. The Kluwer International Series on Advances in Database Systems, Volume 23. Kluwer Academic Publishers, January 2001.
- [119] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of Sampling for Data Mining of Association Rules. Technical Report TR617, Rensselaer Polytechnic Institute, 1996.

2.6 Exercises

1. In the initial example of [Chapter 2](#), the statistician says, “Yes, fields 2 and 3 are basically the same.” Can you tell from the three lines of sample data that are shown why she says that?
2. Classify the following attributes as binary, discrete, or continuous. Also classify them as qualitative (nominal or ordinal) or quantitative (interval or ratio). Some cases may have more than one interpretation, so briefly indicate your reasoning if you think there may be some ambiguity.

Example: Age in years. **Answer:** Discrete, quantitative, ratio

- a. Time in terms of AM or PM.
- b. Brightness as measured by a light meter.
- c. Brightness as measured by people’s judgments.
- d. Angles as measured in degrees between 0 and 360.
- e. Bronze, Silver, and Gold medals as awarded at the Olympics.
- f. Height above sea level.
- g. Number of patients in a hospital.
- h. ISBN numbers for books. (Look up the format on the Web.)
- i. Ability to pass light in terms of the following values: opaque, translucent, transparent.
- j. Military rank.
- k. Distance from the center of campus.

- I. Density of a substance in grams per cubic centimeter.
 - m. Coat check number. (When you attend an event, you can often give your coat to someone who, in turn, gives you a number that you can use to claim your coat when you leave.)
3. You are approached by the marketing director of a local company, who believes that he has devised a foolproof way to measure customer satisfaction. He explains his scheme as follows: “It’s so simple that I can’t believe that no one has thought of it before. I just keep track of the number of customer complaints for each product. I read in a data mining book that counts are ratio attributes, and so, my measure of product satisfaction must be a ratio attribute. But when I rated the products based on my new customer satisfaction measure and showed them to my boss, he told me that I had overlooked the obvious, and that my measure was worthless. I think that he was just mad because our bestselling product had the worst satisfaction since it had the most complaints. Could you help me set him straight?”
- a. Who is right, the marketing director or his boss? If you answered, his boss, what would you do to fix the measure of satisfaction?
 - b. What can you say about the attribute type of the original product satisfaction attribute?
4. A few months later, you are again approached by the same marketing director as in [Exercise 3](#). This time, he has devised a better approach to measure the extent to which a customer prefers one product over other similar products. He explains, “When we develop new products, we typically create several variations and evaluate which one customers prefer. Our standard procedure is to give our test subjects all of the product variations at one time and then ask them to rank the product variations in order of preference. However, our test subjects are very indecisive, especially when there are

more than two products. As a result, testing takes forever. I suggested that we perform the comparisons in pairs and then use these comparisons to get the rankings. Thus, if we have three product variations, we have the customers compare variations 1 and 2, then 2 and 3, and finally 3 and 1. Our testing time with my new procedure is a third of what it was for the old procedure, but the employees conducting the tests complain that they cannot come up with a consistent ranking from the results. And my boss wants the latest product evaluations, yesterday. I should also mention that he was the person who came up with the old product evaluation approach. Can you help me?"

- a. Is the marketing director in trouble? Will his approach work for generating an ordinal ranking of the product variations in terms of customer preference? Explain.
 - b. Is there a way to fix the marketing director's approach? More generally, what can you say about trying to create an ordinal measurement scale based on pairwise comparisons?
 - c. For the original product evaluation scheme, the overall rankings of each product variation are found by computing its average over all test subjects. Comment on whether you think that this is a reasonable approach. What other approaches might you take?
5. Can you think of a situation in which identification numbers would be useful for prediction?
6. An educational psychologist wants to use association analysis to analyze test results. The test consists of 100 questions with four possible answers each.
- a. How would you convert this data into a form suitable for association analysis?

b. In particular, what type of attributes would you have and how many of them are there?

7. Which of the following quantities is likely to show more temporal autocorrelation: daily rainfall or daily temperature? Why?

8. Discuss why a document-term matrix is an example of a data set that has asymmetric discrete or asymmetric continuous features.

9. Many sciences rely on observation instead of (or in addition to) designed experiments. Compare the data quality issues involved in observational science with those of experimental science and data mining.

10. Discuss the difference between the precision of a measurement and the terms single and double precision, as they are used in computer science, typically to represent floating-point numbers that require 32 and 64 bits, respectively.

11. Give at least two advantages to working with data stored in text files instead of in a binary format.

12. Distinguish between noise and outliers. Be sure to consider the following questions.

a. Is noise ever interesting or desirable? Outliers?

b. Can noise objects be outliers?

c. Are noise objects always outliers?

d. Are outliers always noise objects?

e. Can noise make a typical value into an unusual one, or vice versa?

Algorithm 2.3 Algorithm for finding k -

nearest neighbors.

- 1 : **for** $i=1$ to *number of data objects* **do**
- 2 : Find the distances of the i th object to all other objects.
- 3: Sort these distances in decreasing order.
(Keep track of which object is associated with each distance.)
- 4: **return** the objects associated with the first k distances of the sorted list
- 5: **end for**

13. Consider the problem of finding the K -nearest neighbors of a data object. A programmer designs Algorithm 2.3 for this task.
 - a. Describe the potential problems with this algorithm if there are duplicate objects in the data set. Assume the distance function will return a distance of 0 only for objects that are the same.
 - b. How would you fix this problem?
14. The following attributes are measured for members of a herd of Asian elephants: *weight*, *height*, *tusk length*, *trunk length*, and *ear area*. Based on these measurements, what sort of proximity measure from [Section 2.4](#) would you use to compare or group these elephants? Justify your answer and explain any special circumstances.
15. You are given a set of m objects that is divided into k groups, where the i^{th} group is of size m_i . If the goal is to obtain a sample of size $n < m$, what is the difference between the following two sampling schemes? (Assume sampling with replacement.)

- a. We randomly select $n \times m_i/m$ elements from each group.
- b. We randomly select n elements from the data set, without regard for the group to which an object belongs.

16. Consider a document-term matrix, where tf_{ij} is the frequency of the i th word (term) in the j th document and m is the number of documents. Consider the variable transformation that is defined by

$$tf'_{ij} = tf_{ij} \times \log m/d_{fi}, \quad (2.31)$$

where d_{fi} is the number of documents in which the i th term appears, which is known as the **document frequency** of the term. This transformation is known as the **inverse document frequency** transformation.

- a. What is the effect of this transformation if a term occurs in one document? In every document?
- b. What might be the purpose of this transformation?

17. Assume that we apply a square root transformation to a ratio attribute x to obtain the new attribute x^* . As part of your analysis, you identify an interval (a, b) in which x^* has a linear relationship to another attribute y .

- a. What is the corresponding interval (A, B) in terms of x ?
- b. Give an equation that relates y to x .

18. This exercise compares and contrasts some similarity and distance measures.

- a. For binary data, the L1 distance corresponds to the Hamming distance; that is, the number of bits that are different between two binary vectors. The Jaccard similarity is a measure of the similarity between two binary

vectors. Compute the Hamming distance and the Jaccard similarity between the following two binary vectors.

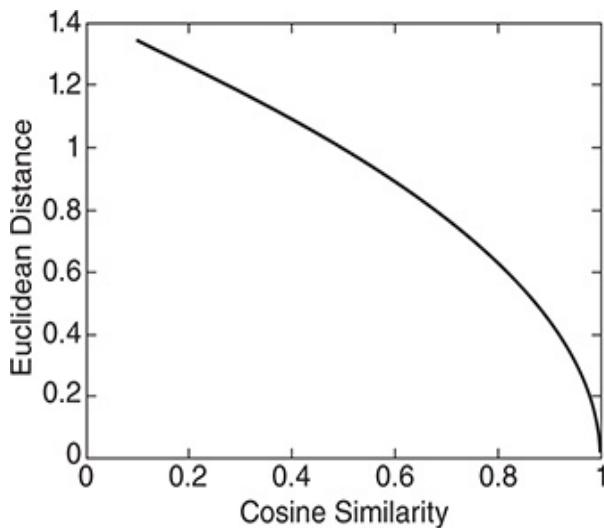
$$x=0101010001 \quad y=0100011000$$

- b. Which approach, Jaccard or Hamming distance, is more similar to the Simple Matching Coefficient, and which approach is more similar to the cosine measure? Explain. (Note: The Hamming measure is a distance, while the other three measures are similarities, but don't let this confuse you.)
 - c. Suppose that you are comparing how similar two organisms of different species are in terms of the number of genes they share. Describe which measure, Hamming or Jaccard, you think would be more appropriate for comparing the genetic makeup of two organisms. Explain. (Assume that each animal is represented as a binary vector, where each attribute is 1 if a particular gene is present in the organism and 0 otherwise.)
 - d. If you wanted to compare the genetic makeup of two organisms of the same species, e.g., two human beings, would you use the Hamming distance, the Jaccard coefficient, or a different measure of similarity or distance? Explain. (Note that two human beings share >99.9% of the same genes.)
19. For the following vectors, \mathbf{x} and \mathbf{y} , calculate the indicated similarity or distance measures.
- a. $\mathbf{x}=(1, 1, 1, 1)$, $\mathbf{y}=(2, 2, 2, 2)$ cosine, correlation, Euclidean
 - b. $\mathbf{x}=(0, 1, 0, 1)$, $\mathbf{y}=(1, 0, 1, 0)$ cosine, correlation, Euclidean, Jaccard
 - c. $\mathbf{x}=(0, -1, 0, 1)$, $\mathbf{y}=(1, 0, -1, 0)$ cosine, correlation, Euclidean
 - d. $\mathbf{x}=(1, 1, 0, 1, 0, 1)$, $\mathbf{y}=(1, 1, 1, 0, 0, 1)$ cosine, correlation, Jaccard

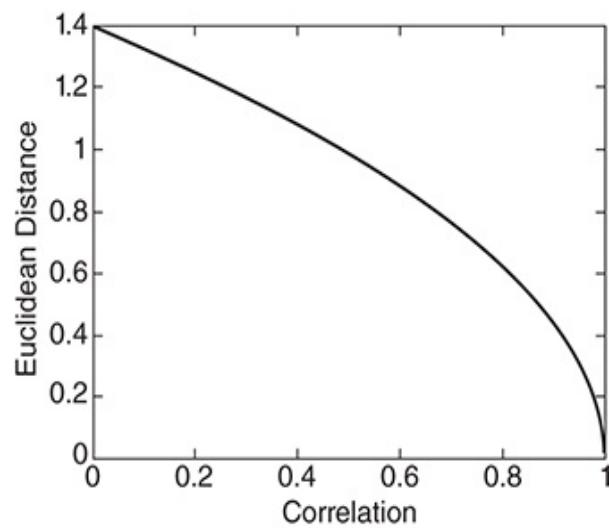
e. $x=(2, -1, 0, 2, 0, -3)$, $y=(-1, 1, -1, 0, 0, -1)$ cosine, correlation

20. Here, we further explore the cosine and correlation measures.

- What is the range of values possible for the cosine measure?
- If two objects have a cosine measure of 1, are they identical? Explain.
- What is the relationship of the cosine measure to correlation, if any?
(Hint: Look at statistical measures such as mean and standard deviation in cases where cosine and correlation are the same and different.)
- Figure 2.22(a)** shows the relationship of the cosine measure to Euclidean distance for 100,000 randomly generated points that have been normalized to have an L2 length of 1. What general observation can you make about the relationship between Euclidean distance and cosine similarity when vectors have an L2 norm of 1?



(a) Relationship between Euclidean distance and the cosine measure.



(b) Relationship between Euclidean distance and correlation.

Figure 2.22.

Graphs for **Exercise 20**.

- e. **Figure 2.22(b)** shows the relationship of correlation to Euclidean distance for 100,000 randomly generated points that have been standardized to have a mean of 0 and a standard deviation of 1. What general observation can you make about the relationship between Euclidean distance and correlation when the vectors have been standardized to have a mean of 0 and a standard deviation of 1?
- f. Derive the mathematical relationship between cosine similarity and Euclidean distance when each data object has an L_2 length of 1.
- g. Derive the mathematical relationship between correlation and Euclidean distance when each data point has been been standardized by subtracting its mean and dividing by its standard deviation.

21. Show that the set difference metric given by

$$d(A, B) = \text{size}(A - B) + \text{size}(B - A) \quad (2.32)$$

satisfies the metric axioms given on page [77](#). A and B are sets and $A - B$ is the set difference.

22. Discuss how you might map correlation values from the interval $[-1, 1]$ to the interval $[0, 1]$. Note that the type of transformation that you use might depend on the application that you have in mind. Thus, consider two applications: clustering time series and predicting the behavior of one time series given another.

23. Given a similarity measure with values in the interval $[0, 1]$, describe two ways to transform this similarity value into a dissimilarity value in the interval $[0, \infty]$.

24. Proximity is typically defined between a pair of objects.

- a. Define two ways in which you might define the proximity among a group of objects.
 - b. How might you define the distance between two sets of points in Euclidean space?
 - c. How might you define the proximity between two sets of data objects? (Make no assumption about the data objects, except that a proximity measure is defined between any pair of objects.)
25. You are given a set of points s in Euclidean space, as well as the distance of each point in s to a point x . (It does not matter if $x \in s$.)
- a. If the goal is to find all points within a specified distance ϵ of point y , $y \neq x$, explain how you could use the triangle inequality and the already calculated distances to x to potentially reduce the number of distance calculations necessary? Hint: The triangle inequality, $d(x, z) \leq d(x, y) + d(y, z)$, can be rewritten as $d(x, y) \geq d(x, z) - d(y, z)$.
 - b. In general, how would the distance between x and y affect the number of distance calculations?
 - c. Suppose that you can find a small subset of points S' , from the original data set, such that every point in the data set is within a specified distance ϵ of at least one of the points in S' , and that you also have the pairwise distance matrix for S' . Describe a technique that uses this information to compute, with a minimum of distance calculations, the set of all points within a distance of β of a specified point from the data set.
26. Show that 1 minus the Jaccard similarity is a distance measure between two data objects, x and y , that satisfies the metric axioms given on page 77. Specifically, $d(x, y) = 1 - J(x, y)$.

27. Show that the distance measure defined as the angle between two data vectors, \mathbf{x} and \mathbf{y} , satisfies the metric axioms given on page 77 . Specifically, $d(\mathbf{x}, \mathbf{y}) = \arccos(\cos(\mathbf{x}, \mathbf{y}))$.
28. Explain why computing the proximity between two attributes is often simpler than computing the similarity between two objects.

3 Classification: Basic Concepts and Techniques

Humans have an innate ability to classify things into categories, e.g., mundane tasks such as filtering spam email messages or more specialized tasks such as recognizing celestial objects in telescope images (see [Figure 3.1](#)). While manual classification often suffices for small and simple data sets with only a few attributes, larger and more complex data sets require an automated solution.



(a) A spiral galaxy.



(b) An elliptical galaxy.

Figure 3.1.

Classification of galaxies from telescope images taken from the NASA website.

This chapter introduces the basic concepts of classification and describes some of its key issues such as model overfitting, model selection, and model evaluation. While these topics are illustrated using a classification technique known as decision tree induction, most of the discussion in this chapter is also applicable to other classification techniques, many of which are covered in [Chapter 4](#).

3.1 Basic Concepts

Figure 3.2 illustrates the general idea behind classification. The data for a classification task consists of a collection of instances (records). Each such instance is characterized by the tuple (\underline{x}, y) , where \underline{x} is the set of attribute values that describe the instance and y is the class label of the instance. The attribute set \underline{x} can contain attributes of any type, while the class label y must be categorical.

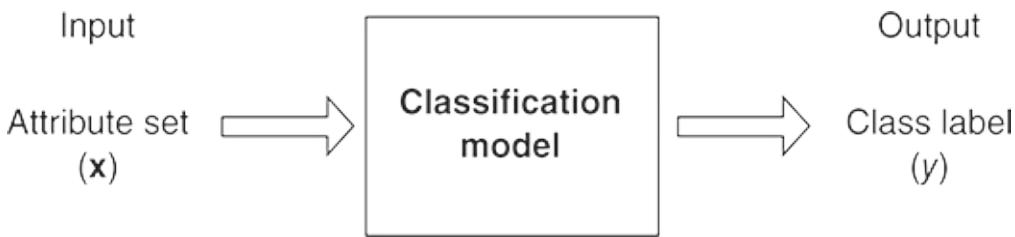


Figure 3.2.

A schematic illustration of a classification task.

A **classification model** is an abstract representation of the relationship between the attribute set and the class label. As will be seen in the next two chapters, the model can be represented in many ways, e.g., as a tree, a probability table, or simply, a vector of real-valued parameters. More formally, we can express it mathematically as a target function f that takes as input the attribute set \underline{x} and produces an output corresponding to the predicted class label. The model is said to classify an instance (\underline{x}, y) correctly if $f(\underline{x})=y$.

Table 3.1 shows examples of attribute sets and class labels for various classification tasks. Spam filtering and tumor identification are examples of binary classification problems, in which each data instance can be categorized into one of two classes. If the number of classes is larger than 2, as in the

galaxy classification example, then it is called a multiclass classification problem.

Table 3.1. Examples of classification tasks.

Task	Attribute set	Class label
Spam filtering	Features extracted from email message header and content	spam or non-spam
Tumor identification	Features extracted from magnetic resonance imaging (MRI) scans	malignant or benign
Galaxy classification	Features extracted from telescope images	elliptical, spiral, or irregular-shaped

We illustrate the basic concepts of classification in this chapter with the following two examples.

3.1. Example Vertebrate Classification

Table 3.2 shows a sample data set for classifying vertebrates into mammals, reptiles, birds, fishes, and amphibians. The attribute set includes characteristics of the vertebrate such as its body temperature, skin cover, and ability to fly. The data set can also be used for a binary classification task such as mammal classification, by grouping the reptiles, birds, fishes, and amphibians into a single category called non-mammals.

Table 3.2. A sample data for the vertebrate classification problem.

Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal

	blooded							
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo	cold-blooded	scales	no	no	no	yes	no	reptile
dragon								
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard	cold-blooded	scales	yes	yes	no	no	no	fish
shark								
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

3.2. Example Loan Borrower Classification

Consider the problem of predicting whether a loan borrower will repay the loan or default on the loan payments. The data set used to build the

classification model is shown in [Table 3.3](#). The attribute set includes personal information of the borrower such as marital status and annual income, while the class label indicates whether the borrower had defaulted on the loan payments.

Table 3.3. A sample data for the loan borrower classification problem.

ID	Home Owner	Marital Status	Annual Income	Defaulted?
1	Yes	Single	125000	No
2	No	Married	100000	No
3	No	Single	70000	No
4	Yes	Married	120000	No
5	No	Divorced	95000	Yes
6	No	Single	60000	No
7	Yes	Divorced	220000	No
8	No	Single	85000	Yes
9	No	Married	75000	No
10	No	Single	90000	Yes

A classification model serves two important roles in data mining. First, it is used as a **predictive model** to classify previously unlabeled instances. A good classification model must provide accurate predictions with a fast response time. Second, it serves as a **descriptive model** to identify the characteristics that distinguish instances from different classes. This is particularly useful for critical applications, such as medical diagnosis, where it

is insufficient to have a model that makes a prediction without justifying how it reaches such a decision.

For example, a classification model induced from the vertebrate data set shown in [Table 3.2](#) can be used to predict the class label of the following vertebrate:

Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

In addition, it can be used as a descriptive model to help determine characteristics that define a vertebrate as a mammal, a reptile, a bird, a fish, or an amphibian. For example, the model may identify mammals as warm-blooded vertebrates that give birth to their young.

There are several points worth noting regarding the previous example. First, although all the attributes shown in [Table 3.2](#) are qualitative, there are no restrictions on the type of attributes that can be used as predictor variables. The class label, on the other hand, must be of nominal type. This distinguishes classification from other predictive modeling tasks such as regression, where the predicted value is often quantitative. More information about regression can be found in Appendix D.

Another point worth noting is that not all attributes may be relevant to the classification task. For example, the average length or weight of a vertebrate may not be useful for classifying mammals, as these attributes can show same value for both mammals and non-mammals. Such an attribute is typically discarded during preprocessing. The remaining attributes might not be able to distinguish the classes by themselves, and thus, must be used in

concert with other attributes. For instance, the Body Temperature attribute is insufficient to distinguish mammals from other vertebrates. When it is used together with Gives Birth, the classification of mammals improves significantly. However, when additional attributes, such as Skin Cover are included, the model becomes overly specific and no longer covers all mammals. Finding the optimal combination of attributes that best discriminates instances from different classes is the key challenge in building classification models.

3.2 General Framework for Classification

Classification is the task of assigning labels to unlabeled data instances and a **classifier** is used to perform such a task. A classifier is typically described in terms of a model as illustrated in the previous section. The model is created using a given a set of instances, known as the **training set**, which contains attribute values as well as class labels for each instance. The systematic approach for learning a classification model given a training set is known as a **learning algorithm**. The process of using a learning algorithm to build a classification model from the training data is known as **induction**. This process is also often described as “learning a model” or “building a model.” This process of applying a classification model on unseen test instances to predict their class labels is known as **deduction**. Thus, the process of classification involves two steps: applying a learning algorithm to training data to learn a model, and then applying the model to assign labels to unlabeled instances. [Figure 3.3](#) illustrates the general framework for classification.

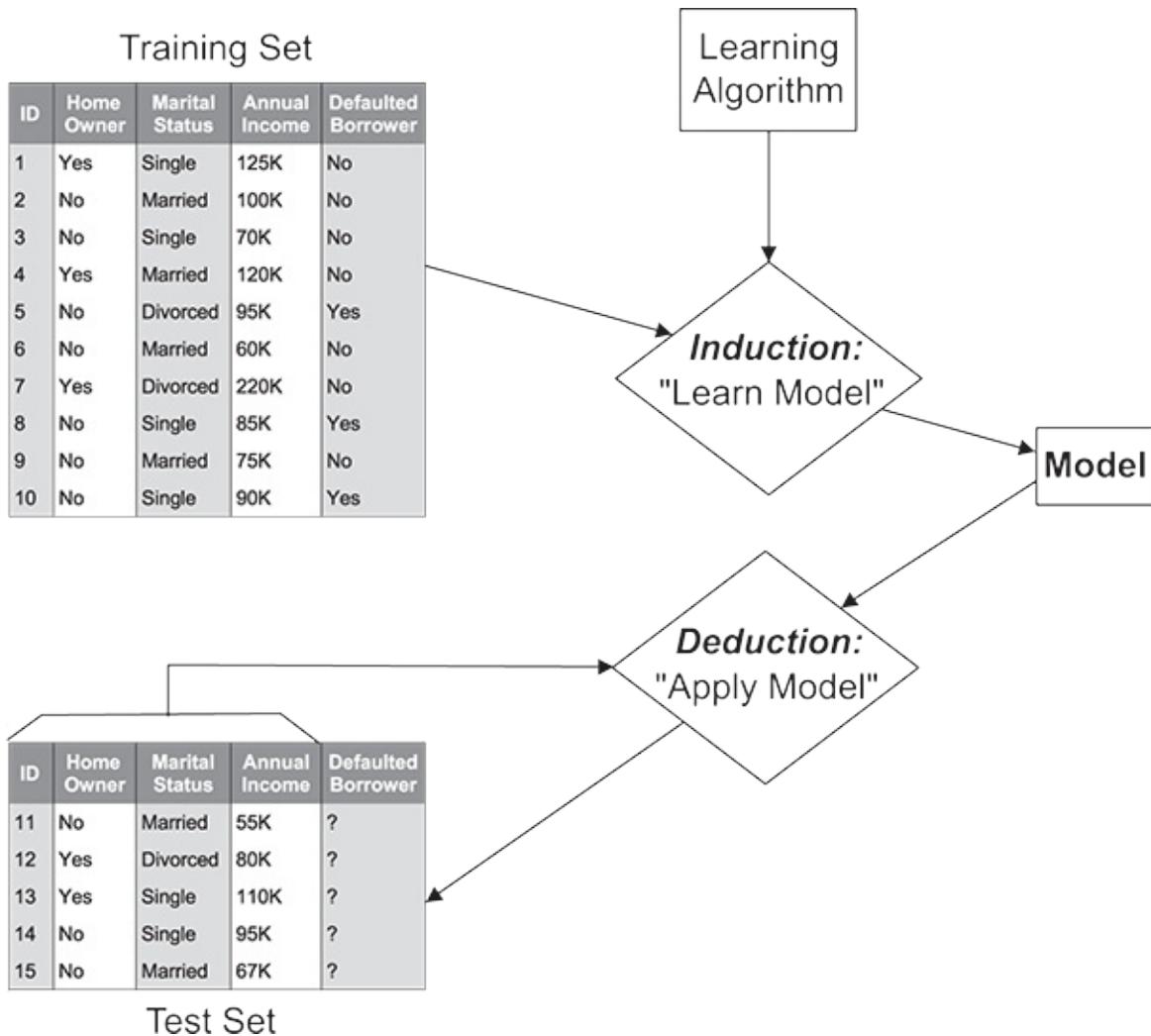


Figure 3.3.

General framework for building a classification model.

A **classification technique** refers to a general approach to classification, e.g., the decision tree technique that we will study in this chapter. This classification technique like most others, consists of a family of related models and a number of algorithms for learning these models. In [Chapter 4](#), we will study additional classification techniques, including neural networks and support vector machines.

A couple notes on terminology. First, the terms “classifier” and “model” are often taken to be synonymous. If a classification technique builds a single,

global model, then this is fine. However, while every model defines a classifier, not every classifier is defined by a single model. Some classifiers, such as k -nearest neighbor classifiers, do not build an explicit model ([Section 4.3](#)), while other classifiers, such as ensemble classifiers, combine the output of a collection of models ([Section 4.10](#)). Second, the term “classifier” is often used in a more general sense to refer to a classification technique. Thus, for example, “decision tree classifier” can refer to the decision tree classification technique or a specific classifier built using that technique. Fortunately, the meaning of “classifier” is usually clear from the context.

In the general framework shown in [Figure 3.3](#), the induction and deduction steps should be performed separately. In fact, as will be discussed later in [Section 3.6](#), the training and test sets should be independent of each other to ensure that the induced model can accurately predict the class labels of instances it has never encountered before. Models that deliver such predictive insights are said to have good **generalization performance**. The performance of a model (classifier) can be evaluated by comparing the predicted labels against the true labels of instances. This information can be summarized in a table called a **confusion matrix**. [Table 3.4](#) depicts the confusion matrix for a binary classification problem. Each entry f_{ij} denotes the number of instances from class i predicted to be of class j . For example, f_{01} is the number of instances from class 0 incorrectly predicted as class 1. The number of correct predictions made by the model is $(f_{11} + f_{00})$ and the number of incorrect predictions is $(f_{10} + f_{01})$.

Table 3.4. Confusion matrix for a binary classification problem.

		Predicted Class	
		Class=1	Class=0
Actual Class	Class=1	f_{11}	f_{10}
	Class=0	f_{01}	f_{00}

	Class=0	f01	f00
--	---------	-----	-----

Although a confusion matrix provides the information needed to determine how well a classification model performs, summarizing this information into a single number makes it more convenient to compare the relative performance of different models. This can be done using an **evaluation metric** such as **accuracy**, which is computed in the following way:

Accuracy =

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \quad (3.1)$$

For binary classification problems, the accuracy of a model is given by

$$\text{Accuracy} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (3.2)$$

Error rate is another related metric, which is defined as follows for binary classification problems:

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (3.3)$$

The learning algorithms of most classification techniques are designed to learn models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set. We will revisit the topic of model evaluation in [Section 3.6](#).

3.3 Decision Tree Classifier

This section introduces a simple classification technique known as the **decision tree** classifier. To illustrate how a decision tree works, consider the classification problem of distinguishing mammals from non-mammals using the vertebrate data set shown in [Table 3.2](#). Suppose a new species is discovered by scientists. How can we tell whether it is a mammal or a non-mammal? One approach is to pose a series of questions about the characteristics of the species. The first question we may ask is whether the species is cold- or warm-blooded. If it is cold-blooded, then it is definitely not a mammal. Otherwise, it is either a bird or a mammal. In the latter case, we need to ask a follow-up question: Do the females of the species give birth to their young? Those that do give birth are definitely mammals, while those that do not are likely to be non-mammals (with the exception of egg-laying mammals such as the platypus and spiny anteater).

The previous example illustrates how we can solve a classification problem by asking a series of carefully crafted questions about the attributes of the test instance. Each time we receive an answer, we could ask a follow-up question until we can conclusively decide on its class label. The series of questions and their possible answers can be organized into a hierarchical structure called a decision tree. [Figure 3.4](#) shows an example of the decision tree for the mammal classification problem. The tree has three types of nodes:

- A **root node**, with no incoming links and zero or more outgoing links.
- **Internal nodes**, each of which has exactly one incoming link and two or more outgoing links.
- **Leaf or terminal nodes**, each of which has exactly one incoming link and no outgoing links.

Every leaf node in the decision tree is associated with a class label. The **non-terminal** nodes, which include the root and internal nodes, contain **attribute test conditions** that are typically defined using a single attribute. Each possible outcome of the attribute test condition is associated with exactly one child of this node. For example, the root node of the tree shown in [Figure 3.4](#) uses the attribute `Body Temperature` to define an attribute test condition that has two outcomes, warm and cold, resulting in two child nodes.

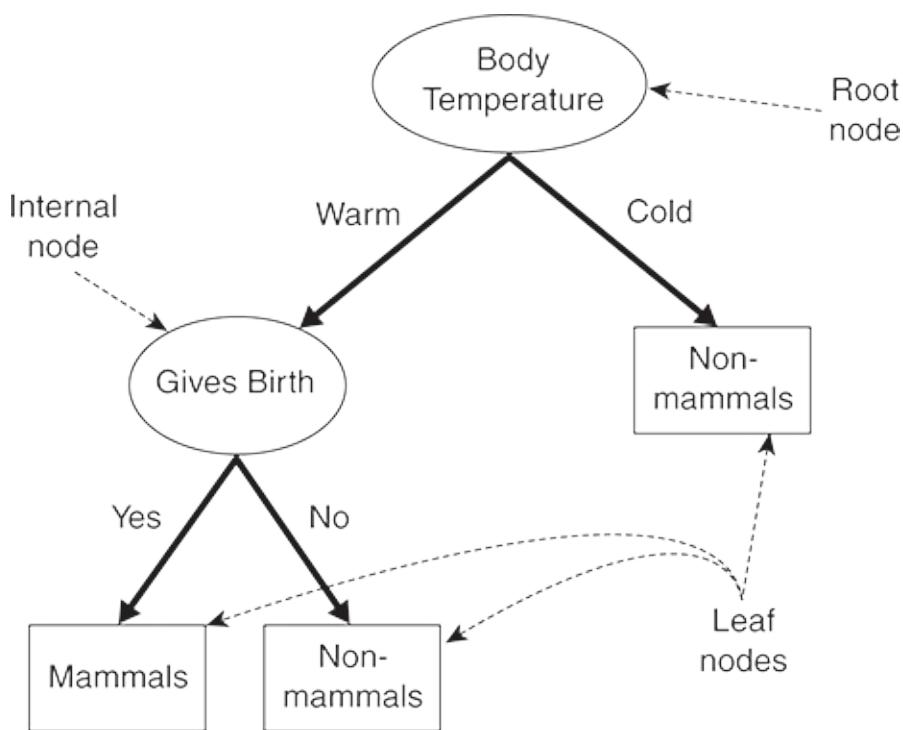


Figure 3.4.

A decision tree for the mammal classification problem.

Given a decision tree, classifying a test instance is straightforward. Starting from the root node, we apply its attribute test condition and follow the appropriate branch based on the outcome of the test. This will lead us either to another internal node, for which a new attribute test condition is applied, or to a leaf node. Once a leaf node is reached, we assign the class label associated with the node to the test instance. As an illustration, [Figure 3.5](#)

traces the path used to predict the class label of a flamingo. The path terminates at a leaf node labeled as `Non-mammals`.

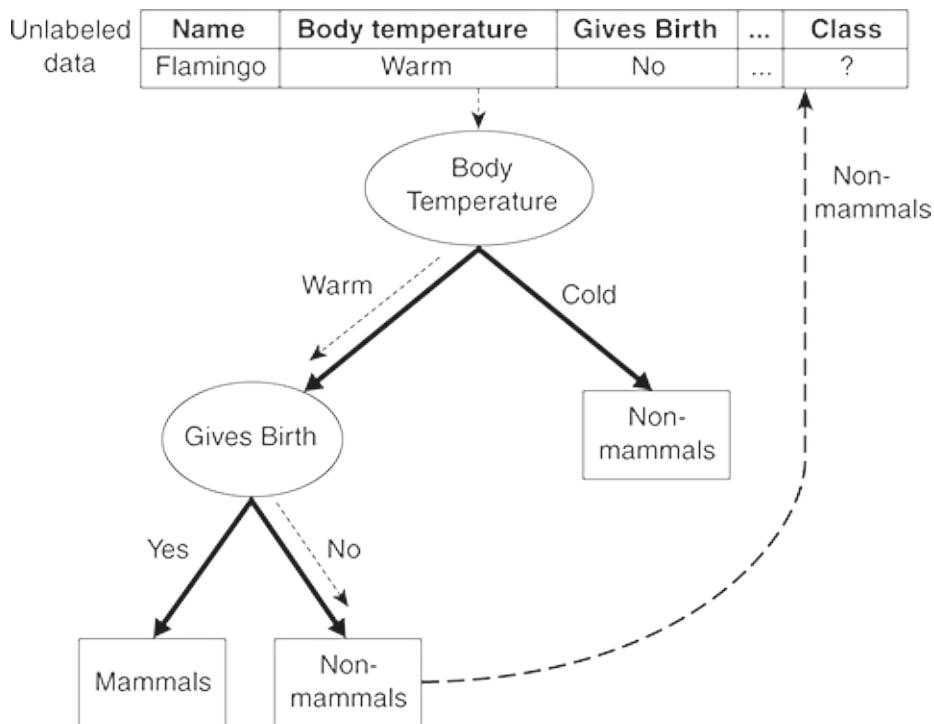


Figure 3.5.

Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the `Non-mammals` class.

3.3.1 A Basic Algorithm to Build a Decision Tree

Many possible decision trees that can be constructed from a particular data set. While some trees are better than others, finding an optimal one is computationally expensive due to the exponential size of the search space. Efficient algorithms have been developed to induce a reasonably accurate,

albeit suboptimal, decision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy to grow the decision tree in a top-down fashion by making a series of locally optimal decisions about which attribute to use when partitioning the training data. One of the earliest method is **Hunt's algorithm**, which is the basis for many current implementations of decision tree classifiers, including ID3, C4.5, and CART. This subsection presents Hunt's algorithm and describes some of the design issues that must be considered when building a decision tree.

Hunt's Algorithm

In Hunt's algorithm, a decision tree is grown in a recursive fashion. The tree initially contains a single root node that is associated with all the training instances. If a node is associated with instances from more than one class, it is expanded using an attribute test condition that is determined using a **splitting criterion**. A child leaf node is created for each outcome of the attribute test condition and the instances associated with the parent node are distributed to the children based on the test outcomes. This node expansion step can then be recursively applied to each child node, as long as it has labels of more than one class. If all the instances associated with a leaf node have identical class labels, then the node is not expanded any further. Each leaf node is assigned a class label that occurs most frequently in the training instances associated with the node.

To illustrate how the algorithm works, consider the training set shown in [Table 3.3](#) for the loan borrower classification problem. Suppose we apply Hunt's algorithm to fit the training data. The tree initially contains only a single leaf node as shown in [Figure 3.6\(a\)](#). This node is labeled as Defaulted = No, since the majority of the borrowers did not default on their loan payments. The training error of this tree is 30% as three out of the ten training instances have

the class label Defaulted = Yes. The leaf node can therefore be further expanded because it contains training instances from more than one class.

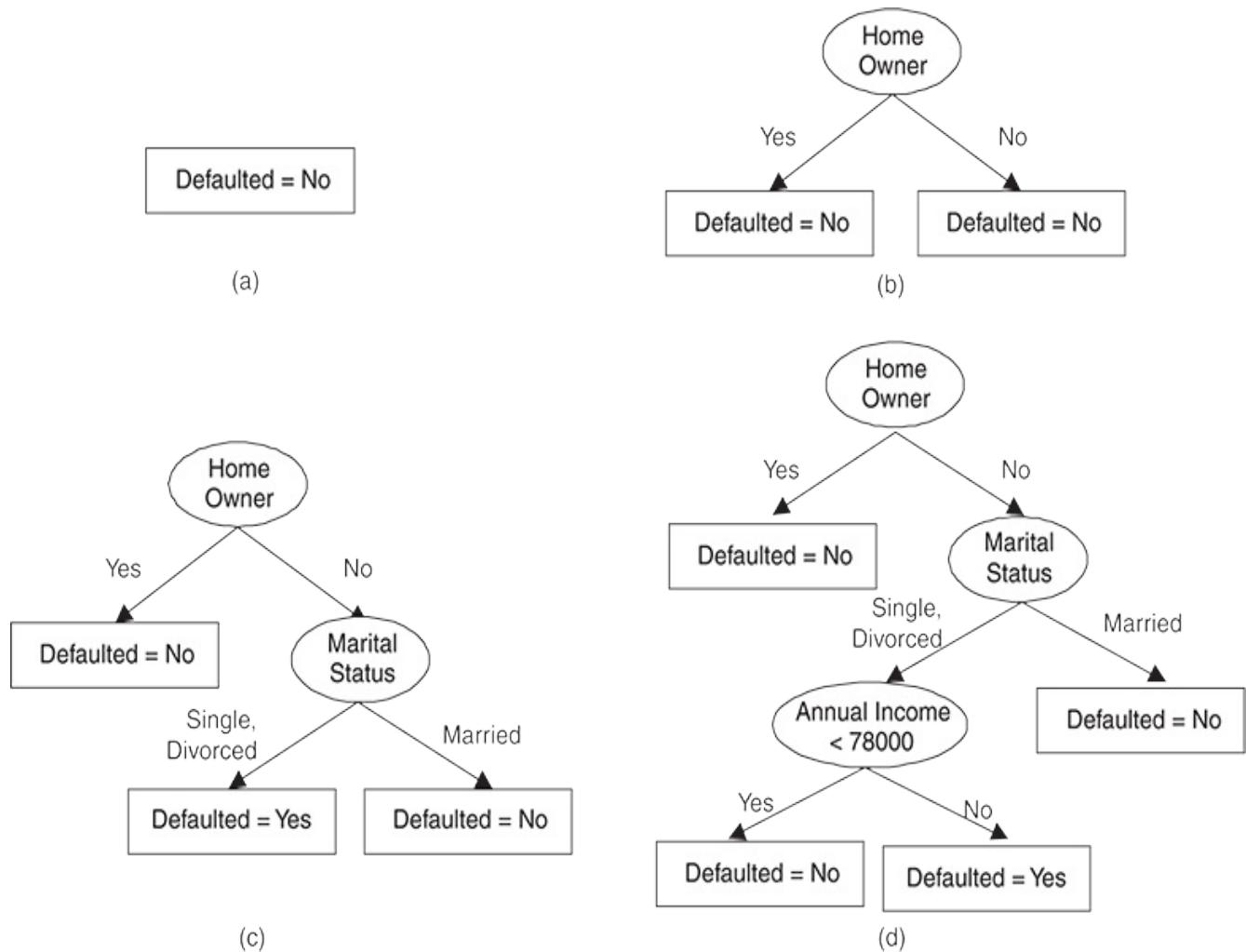


Figure 3.6.
Hunt's algorithm for building decision trees.

Let Home Owner be the attribute chosen to split the training instances. The justification for choosing this attribute as the attribute test condition will be discussed later. The resulting binary split on the Home Owner attribute is shown in [Figure 3.6\(b\)](#). All the training instances for which Home Owner = Yes are propagated to the left child of the root node and the rest are propagated to the right child. Hunt's algorithm is then recursively applied to each child. The left child becomes a leaf node labeled Defaulted = No, since

all instances associated with this node have identical class label

Defaulted = No. The right child has instances from each class label. Hence, we split it further. The resulting subtrees after recursively expanding the right child are shown in [Figures 3.6\(c\)](#) and [\(d\)](#).

Hunt's algorithm, as described above, makes some simplifying assumptions that are often not true in practice. In the following, we describe these assumptions and briefly discuss some of the possible ways for handling them.

1. Some of the child nodes created in Hunt's algorithm can be empty if none of the training instances have the particular attribute values. One way to handle this is by declaring each of them as a leaf node with a class label that occurs most frequently among the training instances associated with their parent nodes.
2. If all training instances associated with a node have identical attribute values but different class labels, it is not possible to expand this node any further. One way to handle this case is to declare it a leaf node and assign it the class label that occurs most frequently in the training instances associated with this node.

Design Issues of Decision Tree Induction

Hunt's algorithm is a generic procedure for growing decision trees in a greedy fashion. To implement the algorithm, there are two key design issues that must be addressed.

1. **What is the splitting criterion?** At each recursive step, an attribute must be selected to partition the training instances associated with a node into smaller subsets associated with its child nodes. The splitting criterion determines which attribute is chosen as the test condition and

how the training instances should be distributed to the child nodes. This will be discussed in [Sections 3.3.2](#) and [3.3.3](#).

2. **What is the stopping criterion?** The basic algorithm stops expanding a node only when all the training instances associated with the node have the same class labels or have identical attribute values. Although these conditions are sufficient, there are reasons to stop expanding a node much earlier even if the leaf node contains training instances from more than one class. This process is called early termination and the condition used to determine when a node should be stopped from expanding is called a stopping criterion. The advantages of early termination are discussed in [Section 3.4](#).

3.3.2 Methods for Expressing Attribute Test Conditions

Decision tree induction algorithms must provide a method for expressing an attribute test condition and its corresponding outcomes for different attribute types.

Binary Attributes

The test condition for a binary attribute generates two potential outcomes, as shown in [Figure 3.7](#).

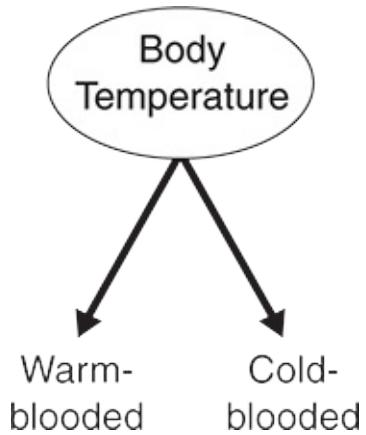


Figure 3.7.

Attribute test condition for a binary attribute.

Nominal Attributes

Since a nominal attribute can have many values, its attribute test condition can be expressed in two ways, as a multiway split or a binary split as shown in [Figure 3.8](#). For a multiway split ([Figure 3.8\(a\)](#)), the number of outcomes depends on the number of distinct values for the corresponding attribute. For example, if an attribute such as marital status has three distinct values—single, married, or divorced—its test condition will produce a three-way split. It is also possible to create a binary split by partitioning all values taken by the nominal attribute into two groups. For example, some decision tree algorithms, such as CART, produce only binary splits by considering all $2^k - 1$ ways of creating a binary partition of k attribute values. [Figure 3.8\(b\)](#) illustrates three different ways of grouping the attribute values for marital status into two subsets.

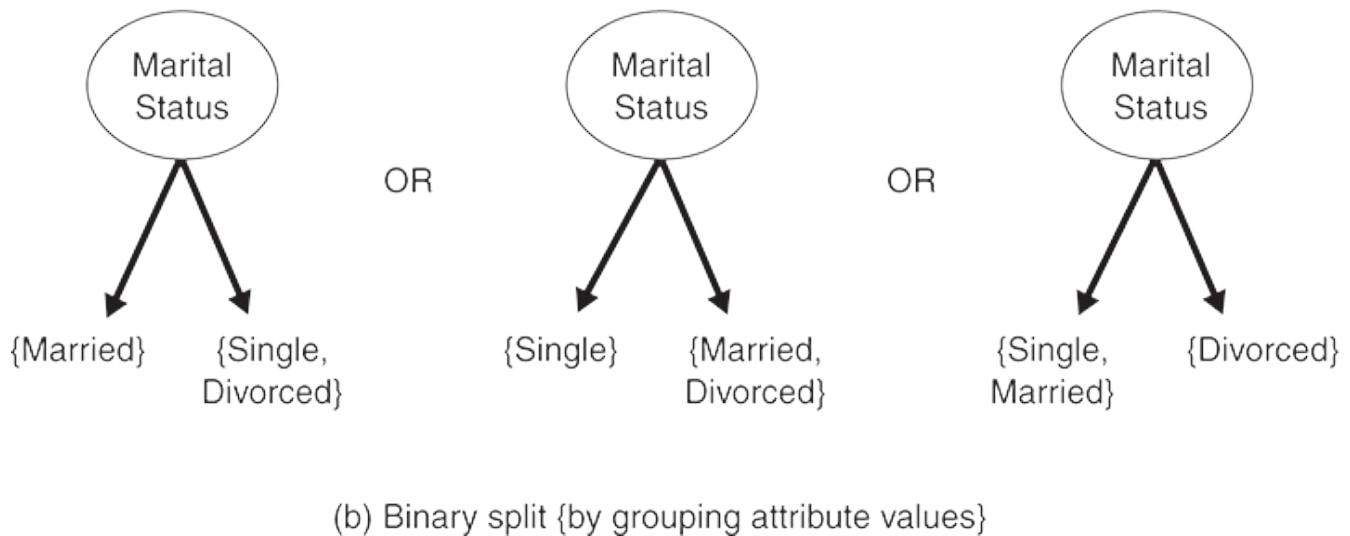
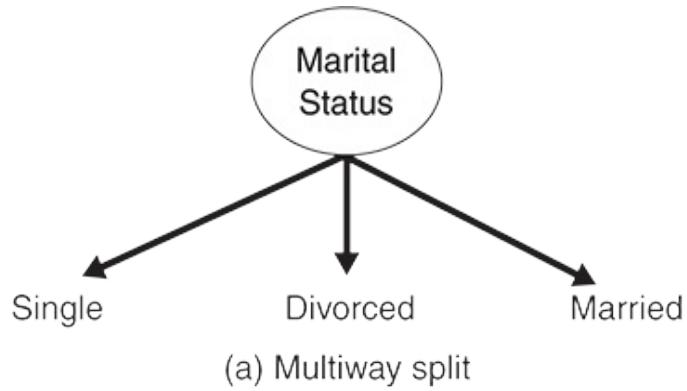


Figure 3.8.

Attribute test conditions for nominal attributes.

Ordinal Attributes

Ordinal attributes can also produce binary or multi-way splits. Ordinal attribute values can be grouped as long as the grouping does not violate the order property of the attribute values. [Figure 3.9\(a\)](#) illustrates various ways of splitting training records based on the Shirt Size attribute. The groupings shown in [Figures 3.9\(a\)](#) and [\(b\)](#) preserve the order among the attribute values, whereas the grouping shown in [Figure 3.9\(c\)](#) violates this property because it combines the attribute values Small and Large into the same partition while Medium and Extra Large are combined into another partition.

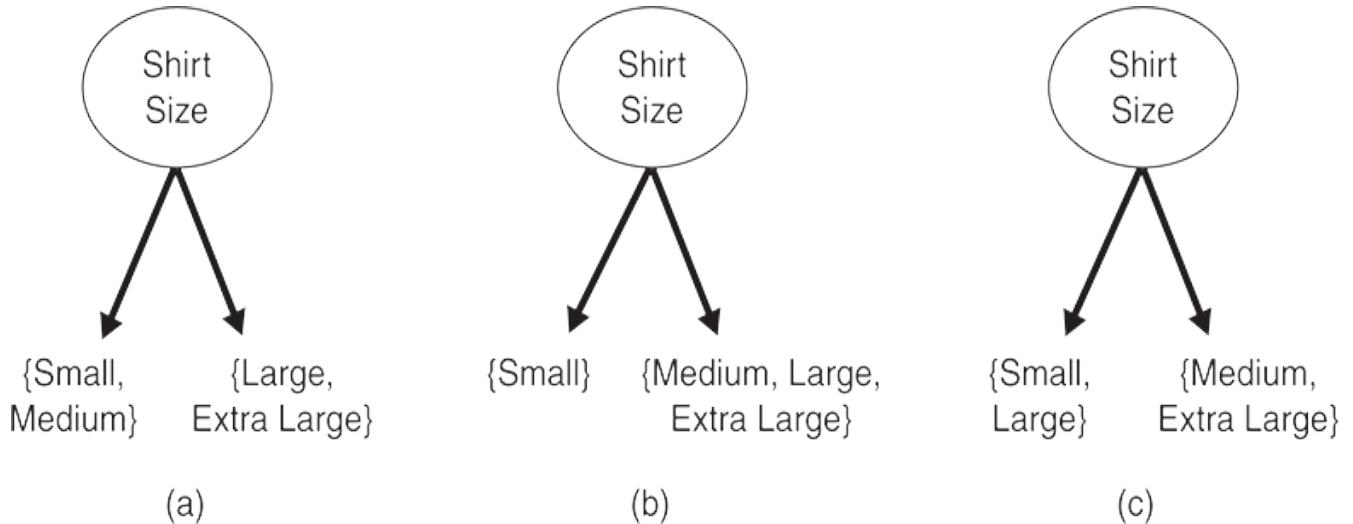


Figure 3.9.

Different ways of grouping ordinal attribute values.

Continuous Attributes

For continuous attributes, the attribute test condition can be expressed as a comparison test (e.g., $A < v$) producing a binary split, or as a range query of the form $v_i \leq A < v_{i+1}$, for $i=1, \dots, k$, producing a multiway split. The difference between these approaches is shown in [Figure 3.10](#). For the binary split, any possible value v between the minimum and maximum attribute values in the training data can be used for constructing the comparison test $A < v$. However, it is sufficient to only consider distinct attribute values in the training set as candidate split positions. For the multiway split, any possible collection of attribute value ranges can be used, as long as they are mutually exclusive and cover the entire range of attribute values between the minimum and maximum values observed in the training set. One approach for constructing multiway splits is to apply the discretization strategies described in [Section 2.3.6](#) on [page 63](#). After discretization, a new ordinal value is assigned to each discretized interval, and the attribute test condition is then defined using this newly constructed ordinal attribute.

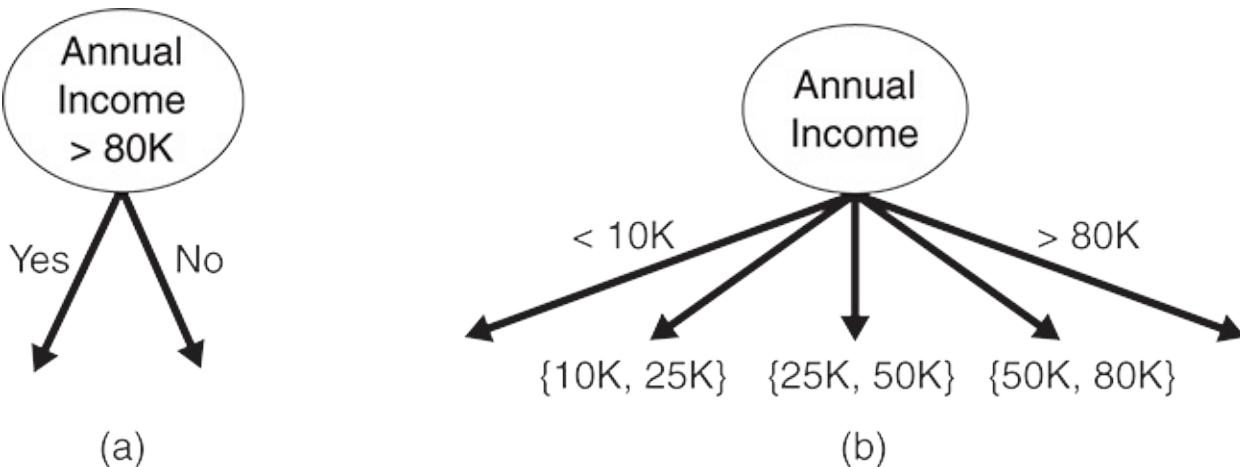


Figure 3.10.

Test condition for continuous attributes.

3.3.3 Measures for Selecting an Attribute Test Condition

There are many measures that can be used to determine the goodness of an attribute test condition. These measures try to give preference to attribute test conditions that partition the training instances into *purer* subsets in the child nodes, which mostly have the same class labels. Having purer nodes is useful since a node that has all of its training instances from the same class does not need to be expanded further. In contrast, an impure node containing training instances from multiple classes is likely to require several levels of node expansions, thereby increasing the depth of the tree considerably. Larger trees are less desirable as they are more susceptible to model overfitting, a condition that may degrade the classification performance on unseen instances, as will be discussed in [Section 3.4](#). They are also difficult to interpret and incur more training and test time as compared to smaller trees.

In the following, we present different ways of measuring the impurity of a node and the collective impurity of its child nodes, both of which will be used to identify the best attribute test condition for a node.

Impurity Measure for a Single Node

The impurity of a node measures how dissimilar the class labels are for the data instances belonging to a common node. Following are examples of measures that can be used to evaluate the impurity of a node t :

$$\text{Entropy} = -\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t), \quad (3.4)$$

$$\text{Gini index} = 1 - \sum_{i=0}^{c-1} p_i(t)^2, \quad (3.5)$$

$$\text{Classification error} = 1 - \max_i [p_i(t)], \quad (3.6)$$

where $p_i(t)$ is the relative frequency of training instances that belong to class i at node t , c is the total number of classes, and $0 \log_2 0 = 0$ in entropy calculations. All three measures give a zero impurity value if a node contains instances from a single class and maximum impurity if the node has equal proportion of instances from multiple classes.

Figure 3.11 compares the relative magnitude of the impurity measures when applied to binary classification problems. Since there are only two classes, $p_0(t) + p_1(t) = 1$. The horizontal axis p refers to the fraction of instances that belong to one of the two classes. Observe that all three measures attain their maximum value when the class distribution is uniform (i.e., $p_0(t) + p_1(t) = 0.5$) and minimum value when all the instances belong to a single class (i.e., either $p_0(t)$ or $p_1(t)$ equals to 1). The following examples illustrate how the values of the impurity measures vary as we alter the class distribution.

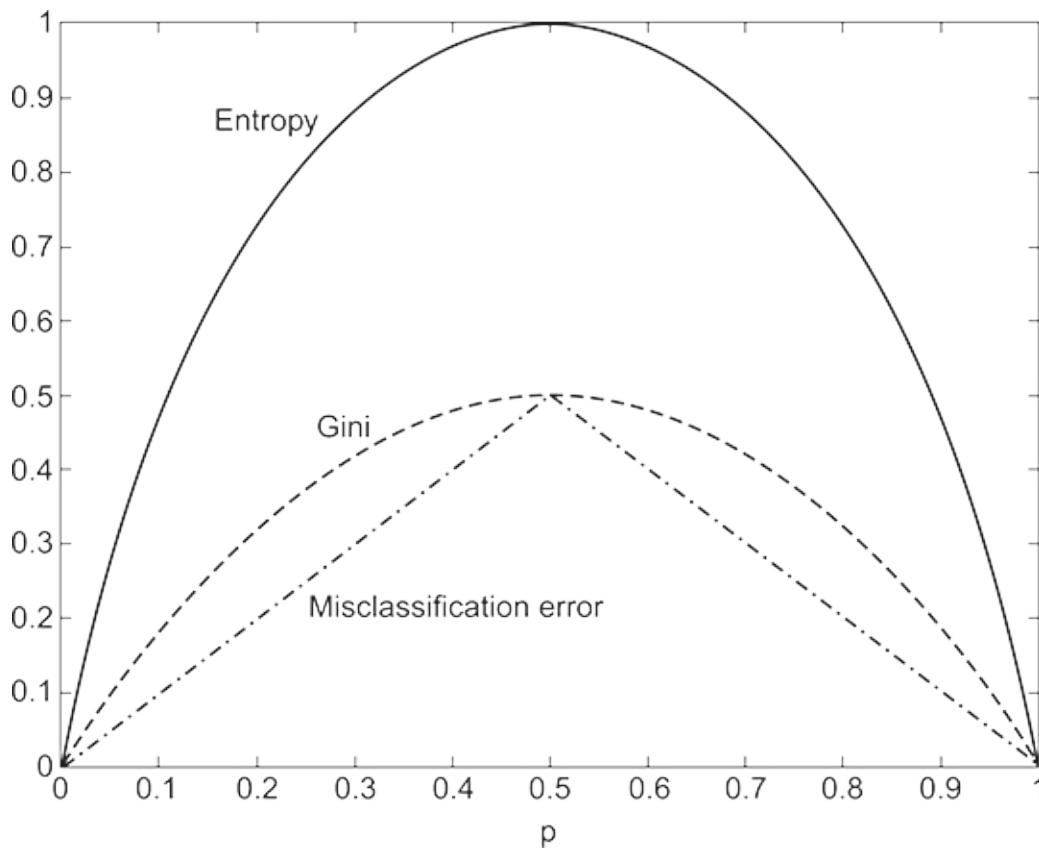


Figure 3.11.

Comparison among the impurity measures for binary classification problems.

Node N1	Count	$\text{Gini} = 1 - (0/6)2 - (6/6)2 = 0$
Class=0	0	$\text{Entropy} = -(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0$
Class=1	6	$\text{Error} = 1 - \max[0/6, 6/6] = 0$
Node N2	Count	$\text{Gini} = 1 - (1/6)2 - (5/6)2 = 0.278$
Class=0	1	$\text{Entropy} = -(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650$
Class=1	5	$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$
Node N3	Count	$\text{Gini} = 1 - (3/6)2 - (3/6)2 = 0.5$
Class=0	3	$\text{Entropy} = -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1$

Class=1

3

Error=1–max[6/6, 3/6]=0.5

Based on these calculations, node N1 has the lowest impurity value, followed by N2 and N3. This example, along with [Figure 3.11](#), shows the consistency among the impurity measures, i.e., if a node N1 has lower entropy than node N2, then the Gini index and error rate of N1 will also be lower than that of N2. Despite their agreement, the attribute chosen as splitting criterion by the impurity measures can still be different (see Exercise 6 on page 187).

Collective Impurity of Child Nodes

Consider an attribute test condition that splits a node containing N training instances into k children, $\{v_1, v_2, \dots, v_k\}$, where every child node represents a partition of the data resulting from one of the k outcomes of the attribute test condition. Let $N(v_j)$ be the number of training instances associated with a child node v_j , whose impurity value is $I(v_j)$. Since a training instance in the parent node reaches node v_j for a fraction of $N(v_j)/N$ times, the collective impurity of the child nodes can be computed by taking a weighted sum of the impurities of the child nodes, as follows:

$$I(\text{children}) = \sum_{j=1}^k N(v_j) I(v_j), \quad (3.7)$$

3.3. Example Weighted Entropy

Consider the candidate attribute test condition shown in [Figures 3.12\(a\)](#) and [\(b\)](#) for the loan borrower classification problem. Splitting on the Home Owner attribute will generate two child nodes

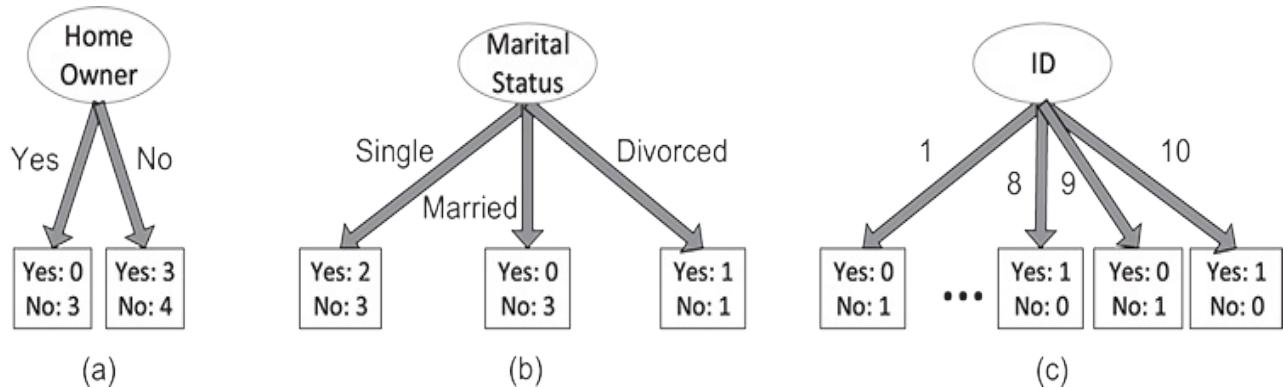


Figure 3.12.

Examples of candidate attribute test conditions.

whose weighted entropy can be calculated as follows:

$$I(\text{Home Owner=yes}) = 0.3 \log 2 + 0.3 - 0.33 \log 2 + 0.33 = 0.1 \\ I(\text{Home Owner=no}) = -0.37 \log 2 + 0.7 - 0.47 \log 2 + 0.47 = 0.985 \\ I(\text{Home Owner}) = 0.310 \times 0 + 0.710 \times 0.985 = 0.690$$

Splitting on Marital Status, on the other hand, leads to three child nodes with a weighted entropy given by

$$I(\text{Marital Status=Single}) = -0.25 \log 2 + 0.25 - 0.35 \log 2 + 0.35 = 0.971 \\ I(\text{Marital Status=Married}) = -0.3 \log 2 + 0.3 - 0.33 \log 2 + 0.33 = 0.1 \\ I(\text{Marital Status=Divorced}) = -0.12 \log 2 + 0.12 - 0.12 \log 2 + 0.12 = 1.000 \\ I(\text{Marital Status}) = 0.510 \times 0.971 + 0.310 \times 0 + 0.210 \times 1 = 0.4855$$

Thus, Marital Status has a lower weighted entropy than Home Owner.

Identifying the best attribute test condition

To determine the goodness of an attribute test condition, we need to compare the degree of impurity of the parent node (before splitting) with the weighted degree of impurity of the child nodes (after splitting). The larger their

difference, the better the test condition. This difference, Δ , also termed as the **gain** in purity of an attribute test condition, can be defined as follows:

$$\Delta = I(\text{parent}) - I(\text{children}), \quad (3.8)$$

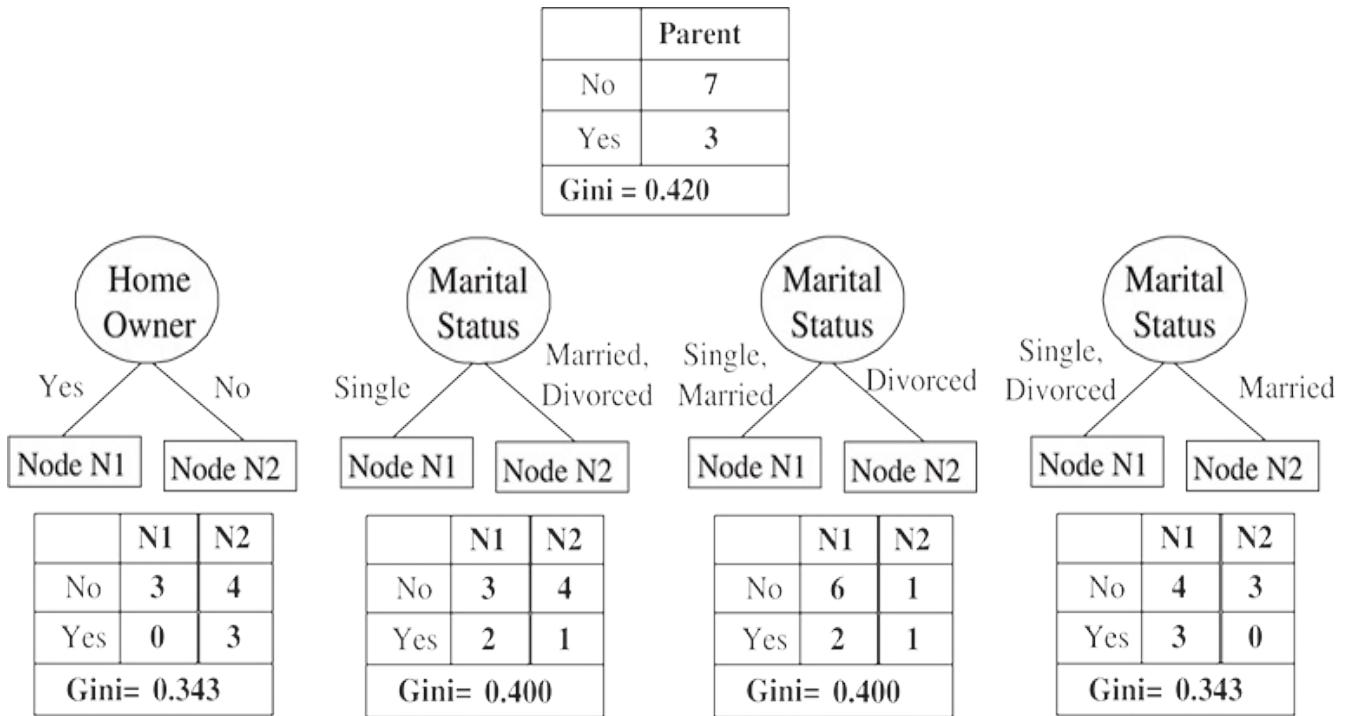


Figure 3.13.

Splitting criteria for the loan borrower classification problem using Gini index.

where $I(\text{parent})$ is the impurity of a node before splitting and $I(\text{children})$ is the weighted impurity measure after splitting. It can be shown that the gain is non-negative since $I(\text{parent}) \geq I(\text{children})$ for any reasonable measure such as those presented above. The higher the gain, the purer are the classes in the child nodes relative to the parent node. The splitting criterion in the decision tree learning algorithm selects the attribute test condition that shows the maximum gain. Note that maximizing the gain at a given node is equivalent to minimizing the weighted impurity measure of its children since $I(\text{parent})$ is the same for all candidate attribute test conditions. Finally, when entropy is used

as the impurity measure, the difference in entropy is commonly known as **information gain**, Δinfo .

In the following, we present illustrative approaches for identifying the best attribute test condition given qualitative or quantitative attributes.

Splitting of Qualitative Attributes

Consider the first two candidate splits shown in [Figure 3.12](#) involving qualitative attributes `Home Owner` and `Marital Status`. The initial class distribution at the parent node is (0.3, 0.7), since there are 3 instances of class `Yes` and 7 instances of class `No` in the training data. Thus,

$$I(\text{parent}) = -310\log_2 3/10 - 710\log_2 7/10 = 0.881$$

The information gains for Home Owner and Marital Status are each given by

$$\Delta\text{info}(\text{Home Owner}) = 0.881 - 0.690 = 0.191 \quad \Delta\text{info}(\text{Marital Status}) = 0.881 - 0.686 = 0.$$

The information gain for Marital Status is thus higher due to its lower weighted entropy, which will thus be considered for splitting.

Binary Splitting of Qualitative Attributes

Consider building a decision tree using only binary splits and the Gini index as the impurity measure. [Figure 3.13](#) shows examples of four candidate splitting criteria for the `Home Owner` and `Marital Status` attributes. Since there are 3 borrowers in the training set who defaulted and 7 others who repaid their loan (see Table in [Figure 3.13](#)), the Gini index of the parent node before splitting is

$$1 - (3/10)2 - (7/10)2 = 0.420.$$

If `Home Owner` is chosen as the splitting attribute, the Gini index for the child nodes N1 and N2 are 0 and 0.490, respectively. The weighted average Gini index for the children is

$$(3/10) \times 0 + (7/10) \times 0.490 = 0.343,$$

where the weights represent the proportion of training instances assigned to each child. The gain using `Home Owner` as splitting attribute is $0.420 - 0.343 = 0.077$. Similarly, we can apply a binary split on the `Marital Status` attribute. However, since `Marital Status` is a nominal attribute with three outcomes, there are three possible ways to group the attribute values into a binary split. The weighted average Gini index of the children for each candidate binary split is shown in [Figure 3.13](#). Based on these results, `Home Owner` and the last binary split using `Marital Status` are clearly the best candidates, since they both produce the lowest weighted average Gini index. Binary splits can also be used for ordinal attributes, if the binary partitioning of the attribute values does not violate the ordering property of the values.

Binary Splitting of Quantitative Attributes

Consider the problem of identifying the best binary split $\text{Annual Income} \leq \tau$ for the preceding loan approval classification problem. As discussed previously, even though τ can take any value between the minimum and maximum values of annual income in the training set, it is sufficient to only consider the annual income values observed in the training set as candidate split positions. For each candidate τ , the training set is scanned once to count the number of borrowers with annual income less than or greater than τ along with their class proportions. We can then compute the Gini index at each candidate split

position and choose the τ that produces the lowest value. Computing the Gini index at each candidate split position requires $O(N)$ operations, where N is the number of training instances. Since there are at most N possible candidates, the overall complexity of this brute-force method is $O(N^2)$. It is possible to reduce the complexity of this problem to $O(N \log N)$ by using a method described as follows (see illustration in [Figure 3.14](#)). In this method, we first sort the training instances based on their annual income, a one-time cost that requires $O(N \log N)$ operations. The candidate split positions are given by the midpoints between every two adjacent sorted values: \$55,000, \$65,000, \$72,500, and so on. For the first candidate, since none of the instances has an annual income less than or equal to \$55,000, the Gini index for the child node with Annual Income $< \$55,000$ is equal to zero. In contrast, there are 3 training instances of class Yes and 7 instances of class No with annual income greater than \$55,000. The Gini index for this node is 0.420. The weighted average Gini index for the first candidate split position, $\tau = \$55,000$, is equal to $0 \times 0 + 1 \times 0.420 = 0.420$.

		Annual Income (in '000s)											
		60	70	75	85	90	95	100	120	125	172.5	220	
		55	65	72.5	80	87.5	92.5	97.5	110	122.5	172.5	230	
		<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Sorted Values	Yes	0	3	0	3	0	3	1	2	2	1	3	0
	No	0	7	1	6	2	5	3	4	3	4	4	3
	Gini	0.420	0.400	0.375	0.343	0.417	0.400	0.300	0.343	0.375	0.400	0.420	

Figure 3.14.

Splitting continuous attributes.

For the next candidate, $\tau = \$65,000$, the class distribution of its child nodes can be obtained with a simple update of the distribution for the previous candidate. This is because, as τ increases from \$55,000 to \$65,000, there is only one

training instance affected by the change. By examining the class label of the affected training instance, the new class distribution is obtained. For example, as τ increases to \$65,000, there is only one borrower in the training set, with an annual income of \$60,000, affected by this change. Since the class label for the borrower is `No`, the count for class `No` increases from 0 to 1 (for Annual Income \leq \$65,000) and decreases from 7 to 6 (for Annual Income $>$ \$65,000), as shown in [Figure 3.14](#). The distribution for the `Yes` class remains unaffected. The updated Gini index for this candidate split position is 0.400.

This procedure is repeated until the Gini index for all candidates are found. The best split position corresponds to the one that produces the lowest Gini index, which occurs at $\tau=\$97,500$. Since the Gini index at each candidate split position can be computed in $O(1)$ time, the complexity of finding the best split position is $O(N)$ once all the values are kept sorted, a one-time operation that takes $O(N \log N)$ time. The overall complexity of this method is thus $O(N \log N)$, which is much smaller than the $O(N^2)$ time taken by the brute-force method. The amount of computation can be further reduced by considering only candidate split positions located between two adjacent sorted instances with different class labels. For example, we do not need to consider candidate split positions located between \$60,000 and \$75,000 because all three instances with annual income in this range (\$60,000, \$70,000, and \$75,000) have the same class labels. Choosing a split position within this range only increases the degree of impurity, compared to a split position located outside this range. Therefore, the candidate split positions at $\tau=\$65,000$ and $\tau=\$72,500$ can be ignored. Similarly, we do not need to consider the candidate split positions at \$87,500, \$92,500, \$110,000, \$122,500, and \$172,500 because they are located between two adjacent instances with the same labels. This strategy reduces the number of candidate split positions to consider from 9 to 2 (excluding the two boundary cases $\tau=\$55,000$ and $\tau=\$230,000$).

Gain Ratio

One potential limitation of impurity measures such as entropy and Gini index is that they tend to favor qualitative attributes with large number of distinct values. [Figure 3.12](#) shows three candidate attributes for partitioning the data set given in [Table 3.3](#). As previously mentioned, the attribute `Marital Status` is a better choice than the attribute `Home Owner`, because it provides a larger information gain. However, if we compare them against `Customer ID`, the latter produces the purest partitions with the maximum information gain, since the weighted entropy and Gini index is equal to zero for its children. Yet, `Customer ID` is not a good attribute for splitting because it has a unique value for each instance. Even though a test condition involving `Customer ID` will accurately classify every instance in the training data, we cannot use such a test condition on new test instances with `Customer ID` values that haven't been seen before during training. This example suggests having a low impurity value alone is insufficient to find a good attribute test condition for a node. As we will see later in [Section 3.4](#), having more number of child nodes can make a decision tree more complex and consequently more susceptible to overfitting. Hence, the number of children produced by the splitting attribute should also be taken into consideration while deciding the best attribute test condition.

There are two ways to overcome this problem. One way is to generate only binary decision trees, thus avoiding the difficulty of handling attributes with varying number of partitions. This strategy is employed by decision tree classifiers such as CART. Another way is to modify the splitting criterion to take into account the number of partitions produced by the attribute. For example, in the C4.5 decision tree algorithm, a measure known as **gain ratio** is used to compensate for attributes that produce a large number of child nodes. This measure is computed as follows:

$$\text{Gain ratio} = \Delta \text{InfoSplit} \quad \text{Info} = \text{Entropy}(\text{Parent}) - \sum_{i=1}^k N(v_i) N \text{Entropy}(v_i) \quad (3.9)$$

$$- \sum_{i=1}^k N(v_i) N \log_2 N(v_i) N$$

where $N(v_i)$ is the number of instances assigned to node v_i and k is the total number of splits. The split information measures the entropy of splitting a node into its child nodes and evaluates if the split results in a larger number of equally-sized child nodes or not. For example, if every partition has the same number of instances, then $\forall i: N(v_i)/N = 1/k$ and the split information would be equal to $\log_2 k$. Thus, if an attribute produces a large number of splits, its split information is also large, which in turn, reduces the gain ratio.

3.4. Example Gain Ratio

Consider the data set given in Exercise 2 on page 185. We want to select the best attribute test condition among the following three attributes:

`Gender`, `Car Type`, and `Customer ID`. The entropy before splitting is

$$\text{Entropy}(\text{parent}) = -1020 \log_2 1020 - 1020 \log_2 1020 = 1.$$

If `Gender` is used as attribute test condition:

$$\begin{aligned} \text{Entropy}(\text{children}) &= 1020[-610 \log_2 610 - 410 \log_2 410] \\ &\times 2 = 0.971 \end{aligned}$$

$$\text{Gain Ratio} = 1 - 0.971 - 1020 \log_2 1020 - 1020 \log_2 1020 = 0.0291 = 0.02$$

If `Car Type` is used as attribute test condition:

$$\begin{aligned} \text{Entropy}(\text{children}) &= 420[-14 \log_2 14 - 34 \log_2 34] \\ &+ 820 \times 0 + 820[-18 \log_2 18 - 78 \log_2 78] \\ &= 0.380 \end{aligned}$$

$$\text{Gain Ratio} = 1 - 0.380 - 420 \log_2 420 - 820 \log_2 820 - 820 \log_2 820 = 0.6201$$

Finally, if `Customer ID` is used as attribute test condition:

$$\begin{aligned} \text{Entropy(children)} &= 120[-11\log_2 11 - 01\log_2 01] \\ &\times 20 = 0 \\ \text{Gain Ratio} &= 1 - 0 - 120\log_2 120 \times 20 = 14.32 = 0.23 \end{aligned}$$

Thus, even though `Customer ID` has the highest information gain, its gain ratio is lower than `Car Type` since it produces a larger number of splits.

3.3.4 Algorithm for Decision Tree Induction

Algorithm 3.1 presents a pseudocode for decision tree induction algorithm. The input to this algorithm is a set of training instances E along with the attribute set F . The algorithm works by recursively selecting the best attribute to split the data (Step 7) and expanding the nodes of the tree (Steps 11 and 12) until the stopping criterion is met (Step 1). The details of this algorithm are explained below.

1. The `createNode()` function extends the decision tree by creating a new node. A node in the decision tree either has a test condition, denoted as $node.test\ cond$, or a class label, denoted as $node.label$.
2. The `find best split()` function determines the attribute test condition for partitioning the training instances associated with a node. The splitting attribute chosen depends on the impurity measure used. The popular measures include entropy and the Gini index.
3. The `classify()` function determines the class label to be assigned to a leaf node. For each leaf node t , let $p(i|t)$ denote the fraction of training instances from class i associated with the node t . The label assigned to

the leaf node is typically the one that occurs most frequently in the training instances that are associated with this node.

Algorithm 3.1 A skeleton decision tree induction algorithm.

```

TreeGrowth (E, F)

1: if stopping cond(E, F) = true then
2:   leaf = createNode().
3:   leaf.label = Classify(E).
4:   return leaf.
5: else
6:   root = createNode().
7:   root.test cond = find best split(E, F).
8:   let V = {v|v is a possible outcome of root.test cond}.
9:   for each v ∈ V do
10:    Ev = {e | root.test cond(e) = v and e ∈ E}.
11:    child = TreeGrowth(Ev, F).
12:    add child as descendent of root and label the edge
        (root → child) as v.
13: end for
14: end if
15: return root.

```

$$\text{leaf.label} = \arg\max_i p(i|t), \quad (3.10)$$

where the $\arg\max$ operator returns the class i that maximizes $p(i|t)$. Besides providing the information needed to determine the class label

of a leaf node, $p(i|t)$ can also be used as a rough estimate of the probability that an instance assigned to the leaf node t belongs to class i . [Sections 4.11.2](#) and [4.11.4](#) in the next chapter describe how such probability estimates can be used to determine the performance of a decision tree under different cost functions.

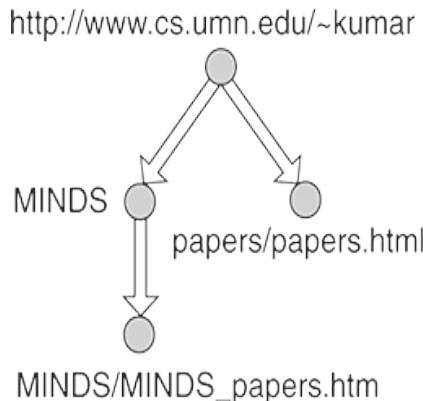
4. The `stopping cond()` function is used to terminate the tree-growing process by checking whether all the instances have identical class label or attribute values. Since decision tree classifiers employ a top-down, recursive partitioning approach for building a model, the number of training instances associated with a node decreases as the depth of the tree increases. As a result, a leaf node may contain too few training instances to make a statistically significant decision about its class label. This is known as the **data fragmentation** problem. One way to avoid this problem is to disallow splitting of a node when the number of instances associated with the node fall below a certain threshold. A more systematic way to control the size of a decision tree (number of leaf nodes) will be discussed in [Section 3.5.4](#).

3.3.5 Example Application: Web Robot Detection

Consider the task of distinguishing the access patterns of web robots from those generated by human users. A web robot (also known as a web crawler) is a software program that automatically retrieves files from one or more websites by following the hyperlinks extracted from an initial set of seed URLs. These programs have been deployed for various purposes, from gathering web pages on behalf of search engines to more malicious activities such as spamming and committing click frauds in online advertisements.

Session	IP Address	Timestamp	Request Method	Requested Web Page	Protocol	Status	Number of Bytes	Referrer	User Agent
1	160.11.11.11	08/Aug/2004 10:15:21	GET	http://www.cs.umn.edu/~kumar	HTTP/1.1	200	6424		Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:34	GET	http://www.cs.umn.edu/~kumar/MINDS	HTTP/1.1	200	41378	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:41	GET	http://www.cs.umn.edu/~kumar/MINDS/MINDS_papers.htm	HTTP/1.1	200	1018516	http://www.cs.umn.edu/~kumar/MINDS	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:16:11	GET	http://www.cs.umn.edu/~kumar/papers/papers.html	HTTP/1.1	200	7463	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
2	35.9.2.2	08/Aug/2004 10:16:15	GET	http://www.cs.umn.edu/~steinbac	HTTP/1.0	200	3149		Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040616

(a) Example of a Web server log.



(b) Graph of a Web session.

Attribute Name	Description
totalPages	Total number of pages retrieved in a Web session
ImagePages	Total number of image pages retrieved in a Web session
TotalTime	Total amount of time spent by Web site visitor
RepeatedAccess	The same page requested more than once in a Web session
ErrorRequest	Errors in requesting for Web pages
GET	Percentage of requests made using GET method
POST	Percentage of requests made using POST method
HEAD	Percentage of requests made using HEAD method
Breadth	Breadth of Web traversal
Depth	Depth of Web traversal
MultiIP	Session with multiple IP addresses
MultiAgent	Session with multiple user agents

(c) Derived attributes for Web robot detection.

Figure 3.15.

Input data for web robot detection.

The web robot detection problem can be cast as a binary classification task. The input data for the classification task is a web server log, a sample of which is shown in [Figure 3.15\(a\)](#). Each line in the log file corresponds to a request made by a client (i.e., a human user or a web robot) to the web server. The fields recorded in the web log include the client's IP address, timestamp of the request, URL of the requested file, size of the file, and **user agent**, which is a field that contains identifying information about the client.

For human users, the user agent field specifies the type of web browser or mobile device used to fetch the files, whereas for web robots, it should technically contain the name of the crawler program. However, web robots may conceal their true identities by declaring their user agent fields to be identical to known browsers. Therefore, user agent is not a reliable field to detect web robots.

The first step toward building a classification model is to precisely define a data instance and associated attributes. A simple approach is to consider each log entry as a data instance and use the appropriate fields in the log file as its attribute set. This approach, however, is inadequate for several reasons. First, many of the attributes are nominal-valued and have a wide range of domain values. For example, the number of unique client IP addresses, URLs, and referrers in a log file can be very large. These attributes are undesirable for building a decision tree because their split information is extremely high (see [Equation \(3.9\)](#)). In addition, it might not be possible to classify test instances containing IP addresses, URLs, or referrers that are not present in the training data. Finally, by considering each log entry as a separate data instance, we disregard the sequence of web pages retrieved by the client—a critical piece of information that can help distinguish web robot accesses from those of a human user.

A better alternative is to consider each web session as a data instance. A web session is a sequence of requests made by a client during a given visit to the website. Each web session can be modeled as a directed graph, in which the nodes correspond to web pages and the edges correspond to hyperlinks connecting one web page to another. [Figure 3.15\(b\)](#) shows a graphical representation of the first web session given in the log file. Every web session can be characterized using some meaningful attributes about the graph that contain discriminatory information. [Figure 3.15\(c\)](#) shows some of the attributes extracted from the graph, including the depth and breadth of its

corresponding tree rooted at the entry point to the website. For example, the depth and breadth of the tree shown in [Figure 3.15\(b\)](#) are both equal to two.

The derived attributes shown in [Figure 3.15\(c\)](#) are more informative than the original attributes given in the log file because they characterize the behavior of the client at the website. Using this approach, a data set containing 2916 instances was created, with equal numbers of sessions due to web robots (class 1) and human users (class 0). 10% of the data were reserved for training while the remaining 90% were used for testing. The induced decision tree is shown in [Figure 3.16](#), which has an error rate equal to 3.8% on the training set and 5.3% on the test set. In addition to its low error rate, the tree also reveals some interesting properties that can help discriminate web robots from human users:

1. Accesses by web robots tend to be broad but shallow, whereas accesses by human users tend to be more focused (narrow but deep).
2. Web robots seldom retrieve the image pages associated with a web page.
3. Sessions due to web robots tend to be long and contain a large number of requested pages.
4. Web robots are more likely to make repeated requests for the same web page than human users since the web pages retrieved by human users are often cached by the browser.

3.3.6 Characteristics of Decision Tree Classifiers

The following is a summary of the important characteristics of decision tree induction algorithms.

1. **Applicability:** Decision trees are a nonparametric approach for building classification models. This approach does not require any prior assumption about the probability distribution governing the class and attributes of the data, and thus, is applicable to a wide variety of data sets. It is also applicable to both categorical and continuous data without requiring the attributes to be transformed into a common representation via binarization, normalization, or standardization. Unlike some binary classifiers described in [Chapter 4](#), it can also deal with multiclass problems without the need to decompose them into multiple binary classification tasks. Another appealing feature of decision tree classifiers is that the induced trees, especially the shorter ones, are relatively easy to interpret. The accuracies of the trees are also quite comparable to other classification techniques for many simple data sets.
2. **Expressiveness:** A decision tree provides a universal representation for discrete-valued functions. In other words, it can encode any function of discrete-valued attributes. This is because every discrete-valued function can be represented as an assignment table, where every unique combination of discrete attributes is assigned a class label. Since every combination of attributes can be represented as a leaf in the decision tree, we can always find a decision tree whose label assignments at the leaf nodes matches with the assignment table of the original function. Decision trees can also help in providing compact representations of functions when some of the unique combinations of attributes can be represented by the same leaf node. For example, [Figure 3.17](#) shows the assignment table of the Boolean function $(A \wedge B) \vee (C \wedge D)$ involving four binary attributes, resulting in a total of $2^4 = 16$ possible assignments. The tree shown in [Figure 3.17](#) shows

a compressed encoding of this assignment table. Instead of requiring a fully-grown tree with 16 leaf nodes, it is possible to encode the function using a simpler tree with only 7 leaf nodes. Nevertheless, not all decision trees for discrete-valued attributes can be simplified. One notable example is the parity function, whose value is 1 when there is an even number of true values among its Boolean attributes, and 0 otherwise. Accurate modeling of such a function requires a full decision tree with 2^d nodes, where d is the number of Boolean attributes (see Exercise 1 on page 185).

Decision Tree:

```
depth = 1:  
| breadth> 7 : class 1  
| breadth<= 7:  
| | breadth <= 3:  
| | | ImagePages> 0.375: class 0  
| | | ImagePages<= 0.375:  
| | | | totalPages<= 6: class 1  
| | | | totalPages> 6:  
| | | | | breadth <= 1: class 1  
| | | | | breadth > 1: class 0  
| | | width > 3:  
| | | | MultilP = 0:  
| | | | | ImagePages<= 0.1333: class 1  
| | | | | ImagePages> 0.1333:  
| | | | | breadth <= 6: class 0  
| | | | | breadth > 6: class 1  
| | | | MultilP = 1:  
| | | | | TotalTime <= 361: class 0  
| | | | | TotalTime > 361: class 1  
depth> 1:  
| MultiAgent = 0:  
| | depth > 2: class 0  
| | depth < 2:  
| | | MultilP = 1: class 0  
| | | MultilP = 0:  
| | | | breadth <= 6: class 0  
| | | | breadth > 6:  
| | | | | RepeatedAccess <= 0.322: class 0  
| | | | | RepeatedAccess > 0.322: class 1  
| MultiAgent = 1:  
| | totalPages <= 81: class 0  
| | totalPages > 81: class 1
```

Figure 3.16.

Decision tree model for web robot detection.

A	B	C	D	class
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

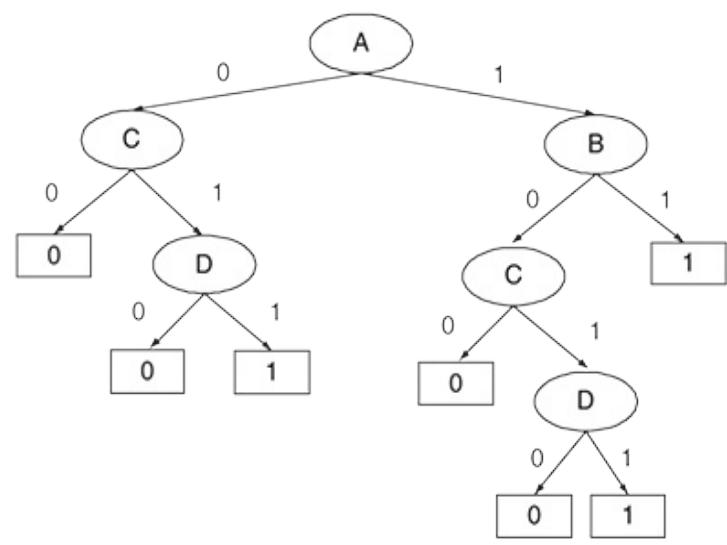


Figure 3.17.

Decision tree for the Boolean function $(A \wedge B) \vee (C \wedge D)$.

3. **Computational Efficiency:** Since the number of possible decision trees can be very large, many decision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space. For example, the algorithm presented in [Section 3.3.4](#) uses a greedy, top-down, recursive partitioning strategy for growing a decision tree. For many data sets, such techniques quickly construct a reasonably good decision tree even when the training set size is very large. Furthermore, once a decision tree has been built, classifying a test record is extremely fast, with a worst-case complexity of $O(w)$, where w is the maximum depth of the tree.
4. **Handling Missing Values:** A decision tree classifier can handle missing attribute values in a number of ways, both in the training and the test sets. When there are missing values in the test set, the classifier must decide which branch to follow if the value of a splitting

node attribute is missing for a given test instance. One approach, known as the **probabilistic split method**, which is employed by the C4.5 decision tree classifier, distributes the data instance to every child of the splitting node according to the probability that the missing attribute has a particular value. In contrast, the CART algorithm uses the **surrogate split method**, where the instance whose splitting attribute value is missing is assigned to one of the child nodes based on the value of another non-missing surrogate attribute whose splits most resemble the partitions made by the missing attribute. Another approach, known as the **separate class method** is used by the CHAID algorithm, where the missing value is treated as a separate categorical value distinct from other values of the splitting attribute. [Figure 3.18](#) shows an example of the three different ways for handling missing values in a decision tree classifier. Other strategies for dealing with missing values are based on data preprocessing, where the instance with missing value is either imputed with the mode (for categorical attribute) or mean (for continuous attribute) value or discarded before the classifier is trained.

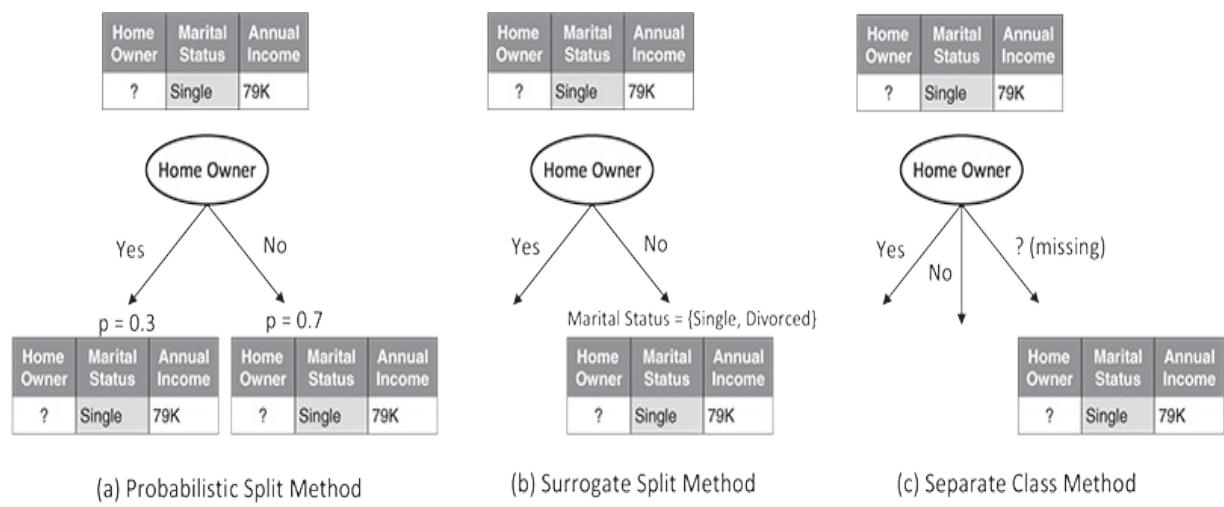


Figure 3.18.

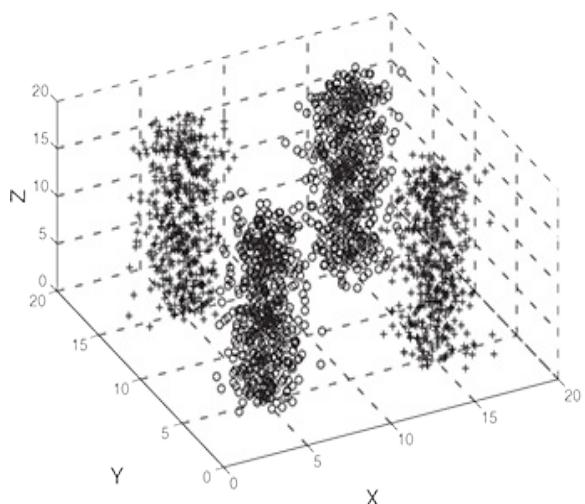
Methods for handling missing attribute values in decision tree classifier.

During training, if an attribute v has missing values in some of the training instances associated with a node, we need a way to measure the gain in purity if v is used for splitting. One simple way is to exclude instances with missing values of v in the counting of instances associated with every child node, generated for every possible outcome of v . Further, if v is chosen as the attribute test condition at a node, training instances with missing values of v can be propagated to the child nodes using any of the methods described above for handling missing values in test instances.

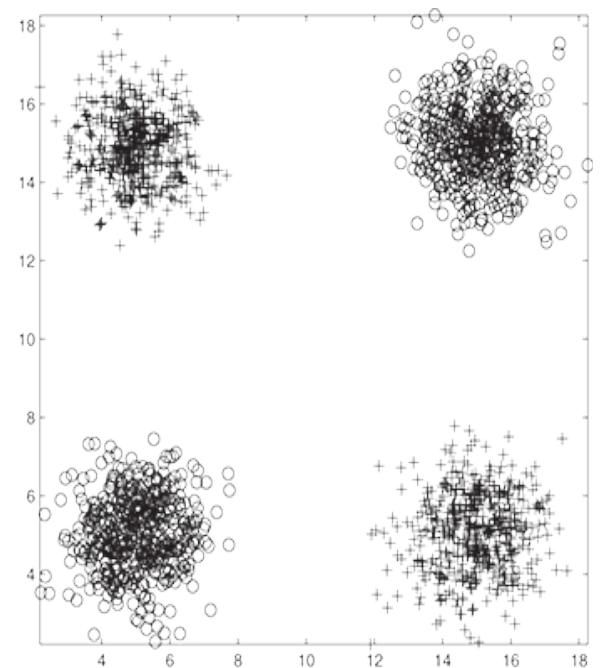
5. **Handling Interactions among Attributes:** Attributes are considered interacting if they are able to distinguish between classes when used together, but individually they provide little or no information. Due to the greedy nature of the splitting criteria in decision trees, such attributes could be passed over in favor of other attributes that are not as useful. This could result in more complex decision trees than necessary. Hence, decision trees can perform poorly when there are interactions among attributes.

To illustrate this point, consider the three-dimensional data shown in [Figure 3.19\(a\)](#), which contains 2000 data points from one of two classes, denoted as + and \circ in the diagram. [Figure 3.19\(b\)](#) shows the distribution of the two classes in the two-dimensional space involving attributes X and Y , which is a noisy version of the XOR Boolean function. We can see that even though the two classes are well-separated in this two-dimensional space, neither of the two attributes contain sufficient information to distinguish between the two classes when used alone. For example, the entropies of the following attribute test conditions: $X \leq 10$ and $Y \leq 10$, are close to 1, indicating that neither X nor Y provide any reduction in the impurity measure when used individually. X and Y thus represent a case of interaction among attributes. The data set also contains a third attribute, Z , in which both classes are distributed uniformly, as shown in [Figures 3.19\(c\)](#) and

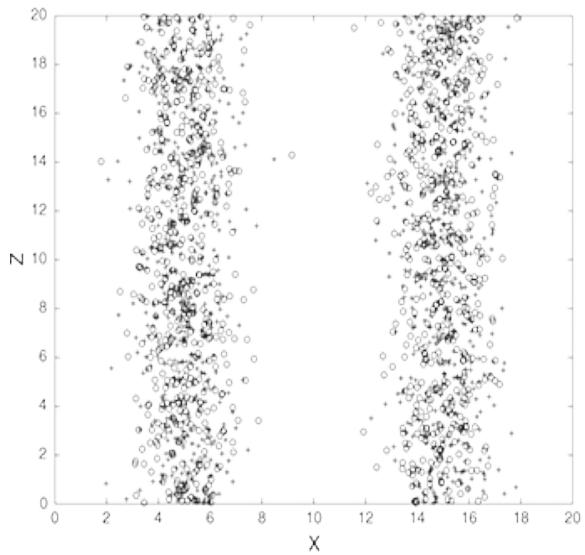
[3.19\(d\)](#), and hence, the entropy of any split involving Z is close to 1. As a result, Z is as likely to be chosen for splitting as the interacting but useful attributes, X and Y . For further illustration of this issue, readers are referred to [Example 3.7](#) in [Section 3.4.1](#) and Exercise 7 at the end of this chapter.



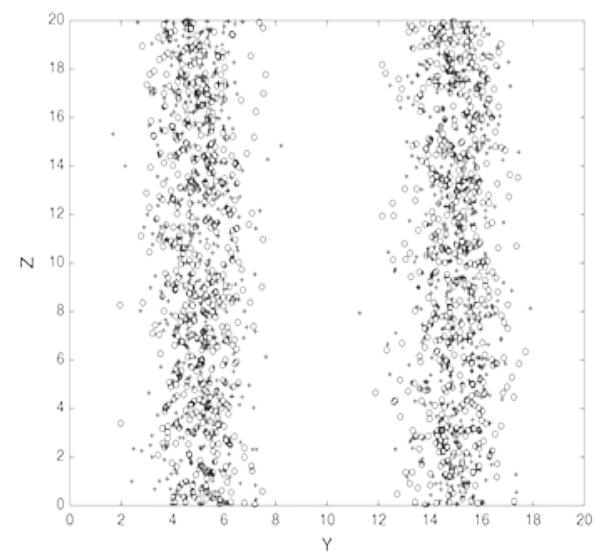
(a) Three-dimensional data with attributes X , Y , and Z .



(b) X and Y .



(c) X and Z .



(d) Y and Z .

Figure 3.19.

Example of a XOR data involving X and Y , along with an irrelevant attribute Z .

6. **Handling Irrelevant Attributes:** An attribute is irrelevant if it is not useful for the classification task. Since irrelevant attributes are poorly associated with the target class labels, they will provide little or no gain in purity and thus will be passed over by other more relevant features. Hence, the presence of a small number of irrelevant attributes will not impact the decision tree construction process. However, not all attributes that provide little to no gain are irrelevant (see [Figure 3.19](#)). Hence, if the classification problem is complex (e.g., involving interactions among attributes) and there are a large number of irrelevant attributes, then some of these attributes may be accidentally chosen during the tree-growing process, since they may provide a better gain than a relevant attribute just by random chance. Feature selection techniques can help to improve the accuracy of decision trees by eliminating the irrelevant attributes during preprocessing. We will investigate the issue of too many irrelevant attributes in [Section 3.4.1](#).
7. **Handling Redundant Attributes:** An attribute is redundant if it is strongly correlated with another attribute in the data. Since redundant attributes show similar gains in purity if they are selected for splitting, only one of them will be selected as an attribute test condition in the decision tree algorithm. Decision trees can thus handle the presence of redundant attributes.
8. **Using Rectilinear Splits:** The test conditions described so far in this chapter involve using only a single attribute at a time. As a consequence, the tree-growing procedure can be viewed as the process of partitioning the attribute space into disjoint regions until each region contains records of the same class. The border between two neighboring regions of different classes is known as a **decision boundary**. [Figure 3.20](#) shows the decision tree as well as the decision boundary for a binary classification problem. Since the test condition involves only a single attribute, the decision boundaries are

rectilinear; i.e., parallel to the coordinate axes. This limits the expressiveness of decision trees in representing decision boundaries of data sets with continuous attributes. [Figure 3.21](#) shows a two-dimensional data set involving binary classes that cannot be perfectly classified by a decision tree whose attribute test conditions are defined based on single attributes. The binary classes in the data set are generated from two skewed Gaussian distributions, centered at (8,8) and (12,12), respectively. The true decision boundary is represented by the diagonal dashed line, whereas the rectilinear decision boundary produced by the decision tree classifier is shown by the thick solid line. In contrast, an **oblique decision tree** may overcome this limitation by allowing the test condition to be specified using more than one attribute. For example, the binary classification data shown in [Figure 3.21](#) can be easily represented by an oblique decision tree with a single root node with test condition

$$x+y<20.$$

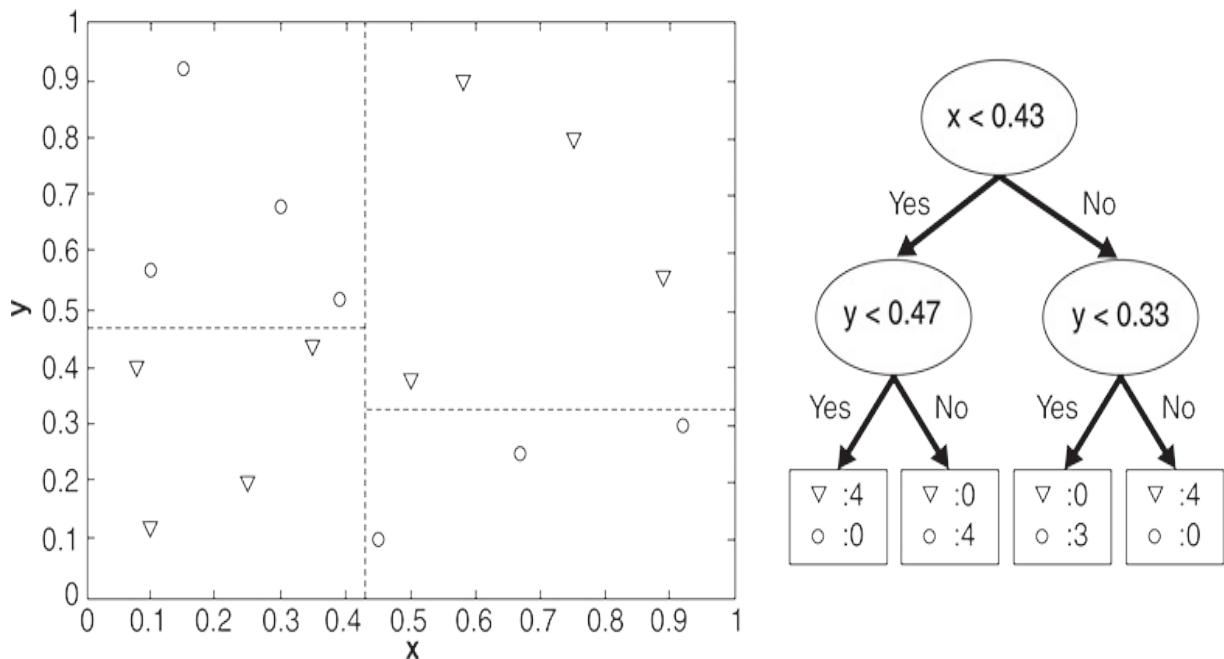


Figure 3.20.

Example of a decision tree and its decision boundaries for a two-dimensional data set.

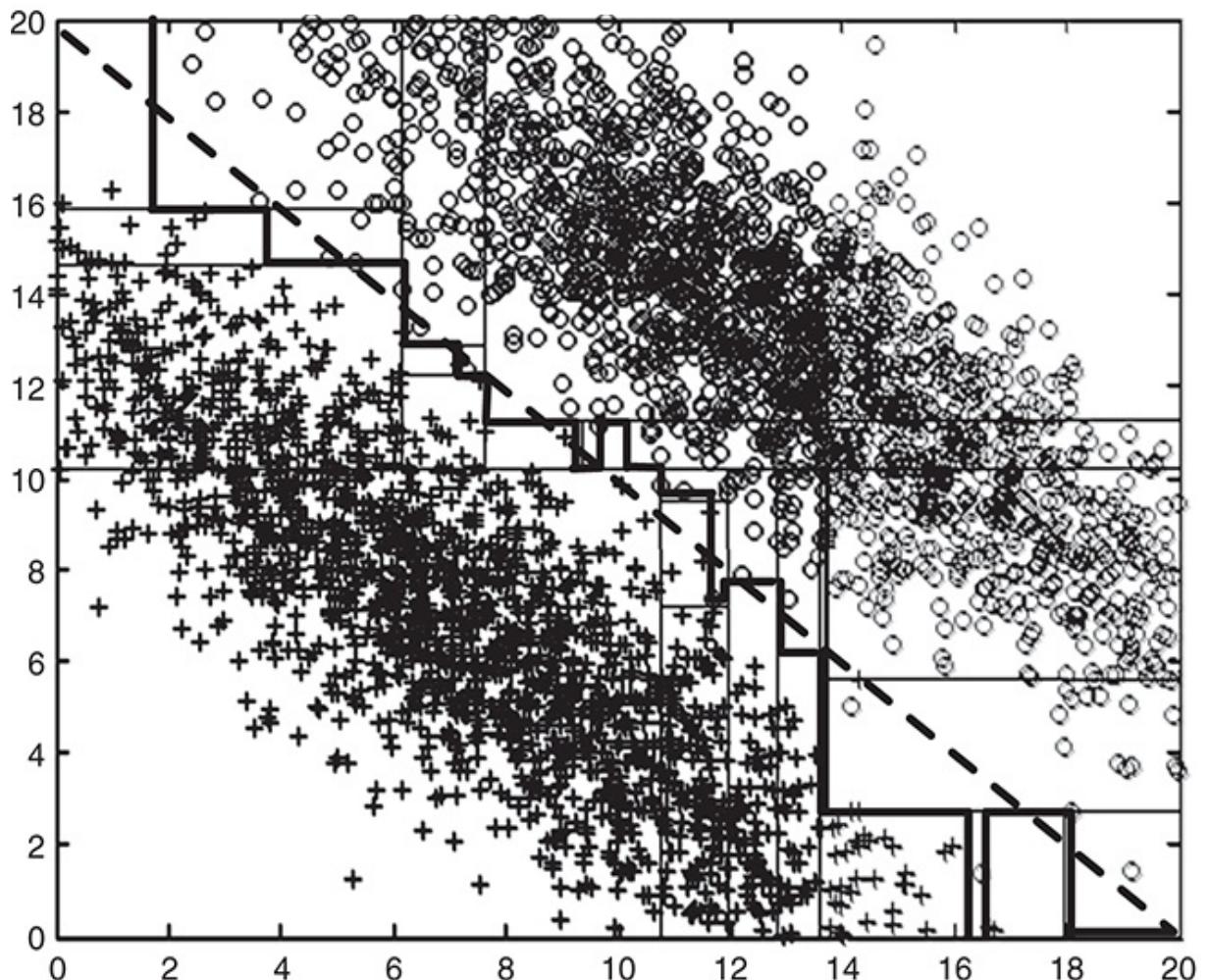


Figure 3.21.

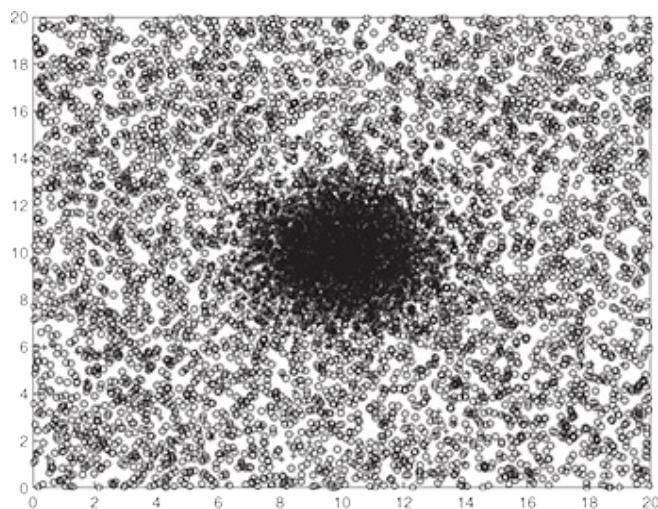
Example of data set that cannot be partitioned optimally using a decision tree with single attribute test conditions. The true decision boundary is shown by the dashed line.

Although an oblique decision tree is more expressive and can produce more compact trees, finding the optimal test condition is computationally more expensive.

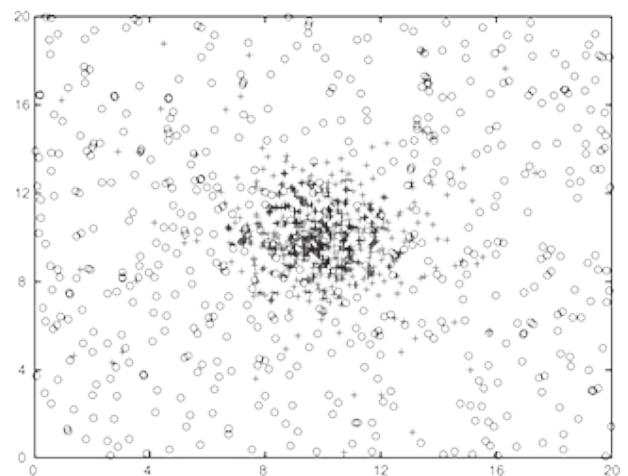
9. **Choice of Impurity Measure:** It should be noted that the choice of impurity measure often has little effect on the performance of decision tree classifiers since many of the impurity measures are quite consistent with each other, as shown in [Figure 3.11](#) on [page 129](#). Instead, the strategy used to prune the tree has a greater impact on the final tree than the choice of impurity measure.

3.4 Model Overfitting

Methods presented so far try to learn classification models that show the lowest error on the training set. However, as we will show in the following example, even if a model fits well over the training data, it can still show poor generalization performance, a phenomenon known as model overfitting.



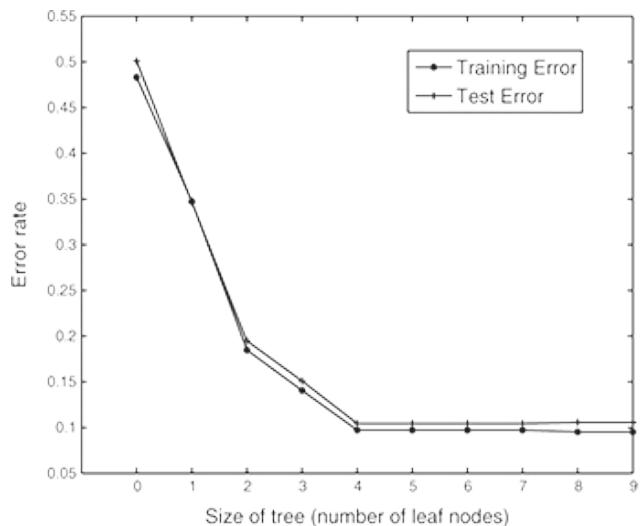
(a) Example of a 2-D data.



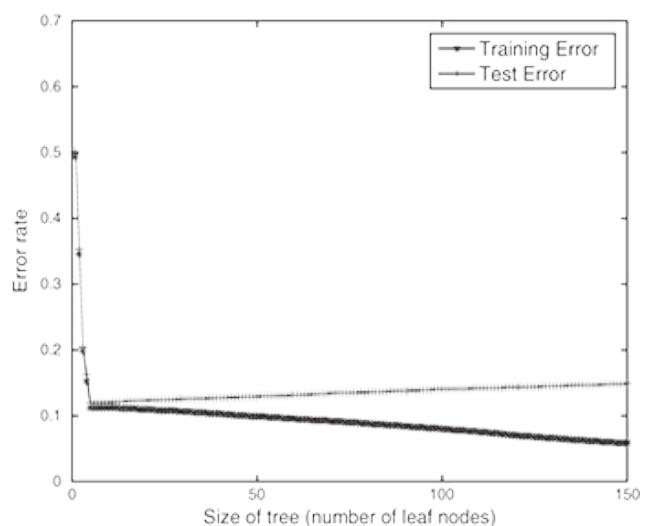
(b) Training set using 10% data.

Figure 3.22.

Examples of training and test sets of a two-dimensional classification problem.



(a) Varying tree size from 1 to 8.



(b) Varying tree size from 1 to 150.

Figure 3.23.

Effect of varying tree size (number of leaf nodes) on training and test errors.

3.5. Example Overfitting and Underfitting of Decision Trees

Consider the two-dimensional data set shown in [Figure 3.22\(a\)](#). The data set contains instances that belong to two separate classes, represented as + and \circ , respectively, where each class has 5400 instances. All instances belonging to the \circ class were generated from a uniform distribution. For the + class, 5000 instances were generated from a Gaussian distribution centered at (10,10) with unit variance, while the remaining 400 instances were sampled from the same uniform distribution as the \circ class. We can see from [Figure 3.22\(a\)](#) that the + class can be largely distinguished from the \circ class by drawing a circle of appropriate size centered at (10,10). To learn a classifier using this two-dimensional data set, we randomly sampled 10% of the data for training and used the remaining 90% for testing. The training set, shown in [Figure 3.22\(b\)](#), looks quite representative of the overall data. We used Gini index as the

impurity measure to construct decision trees of increasing sizes (number of leaf nodes), by recursively expanding a node into child nodes till every leaf node was pure, as described in [Section 3.3.4](#).

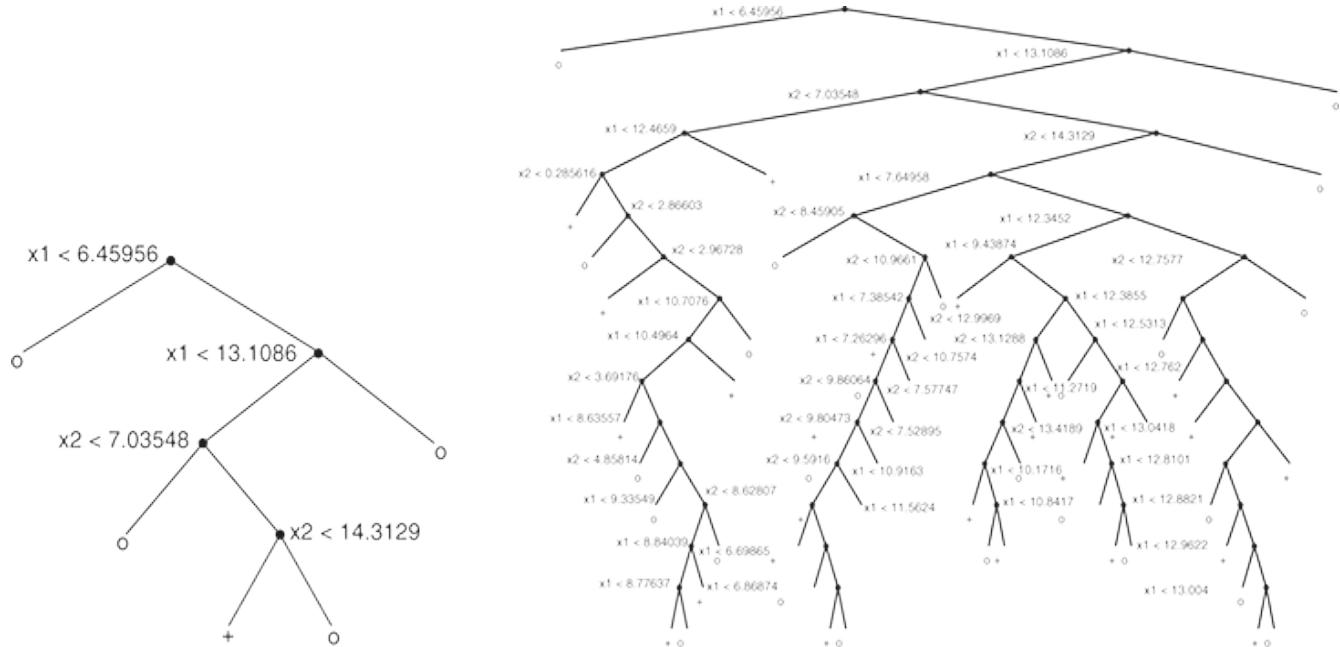
[Figure 3.23\(a\)](#) shows changes in the training and test error rates as the size of the tree varies from 1 to 8. Both error rates are initially large when the tree has only one or two leaf nodes. This situation is known as **model underfitting**. Underfitting occurs when the learned decision tree is too simplistic, and thus, incapable of fully representing the true relationship between the attributes and the class labels. As we increase the tree size from 1 to 8, we can observe two effects. First, both the error rates decrease since larger trees are able to represent more complex decision boundaries. Second, the training and test error rates are quite close to each other, which indicates that the performance on the training set is fairly representative of the generalization performance. As we further increase the size of the tree from 8 to 150, the training error continues to steadily decrease till it eventually reaches zero, as shown in [Figure 3.23\(b\)](#). However, in a striking contrast, the test error rate ceases to decrease any further beyond a certain tree size, and then it begins to increase. The training error rate thus grossly under-estimates the test error rate once the tree becomes too large. Further, the gap between the training and test error rates keeps on widening as we increase the tree size. This behavior, which may seem counter-intuitive at first, can be attributed to the phenomena of **model overfitting**.

3.4.1 Reasons for Model Overfitting

Model overfitting is the phenomena where, in the pursuit of minimizing the training error rate, an overly complex model is selected that captures specific

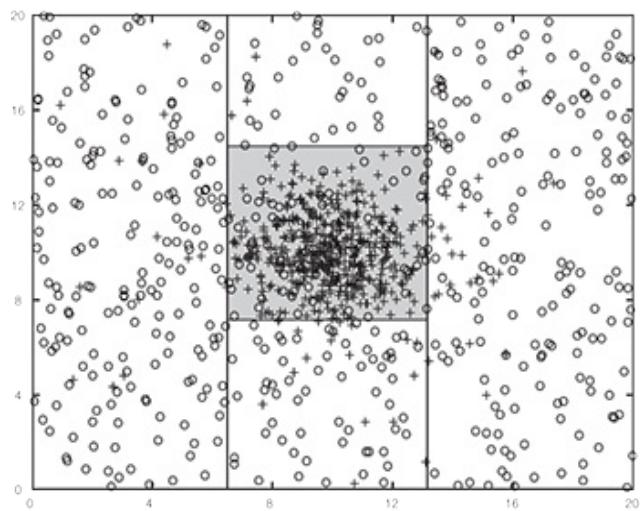
patterns in the training data but fails to learn the *true* nature of relationships between attributes and class labels in the overall data. To illustrate this,

Figure 3.24 shows decision trees and their corresponding decision boundaries (shaded rectangles represent regions assigned to the + class) for two trees of sizes 5 and 50. We can see that the decision tree of size 5 appears quite simple and its decision boundaries provide a reasonable approximation to the ideal decision boundary, which in this case corresponds to a circle centered around the Gaussian distribution at (10, 10). Although its training and test error rates are non-zero, they are very close to each other, which indicates that the patterns learned in the training set should generalize well over the test set. On the other hand, the decision tree of size 50 appears much more complex than the tree of size 5, with complicated decision boundaries. For example, some of its shaded rectangles (assigned the + class) attempt to cover narrow regions in the input space that contain only one or two + training instances. Note that the prevalence of + instances in such regions is highly specific to the training set, as these regions are mostly dominated by - instances in the overall data. Hence, in an attempt to perfectly fit the training data, the decision tree of size 50 starts fine tuning itself to specific patterns in the training data, leading to poor performance on an independently chosen test set.

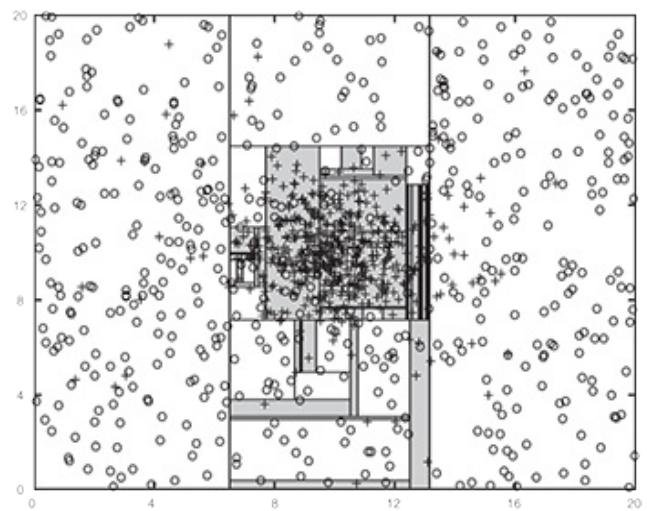


(a) Decision tree with 5 leaf nodes.

(b) Decision tree with 50 leaf nodes.



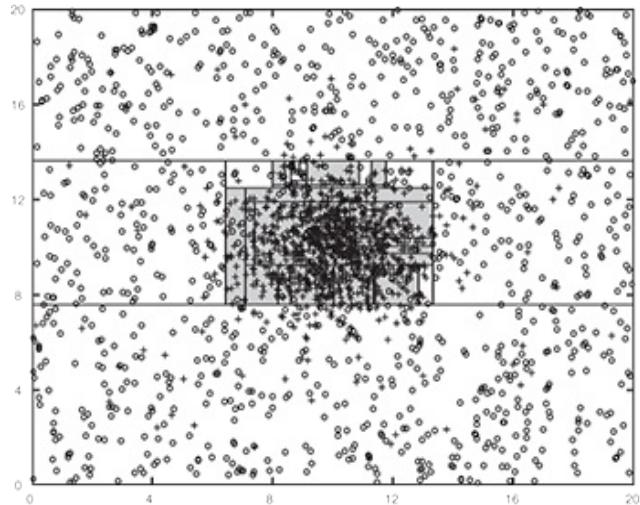
(c) Decision boundary for tree with 5 leaf nodes.



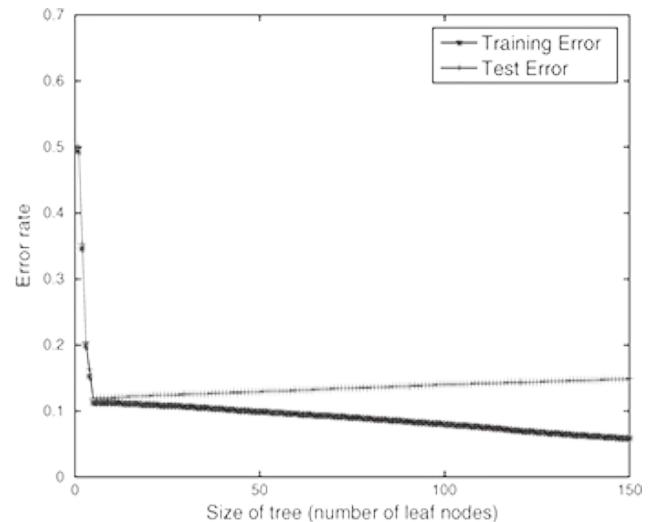
(d) Decision boundary for tree with 50 leaf nodes.

Figure 3.24.

Decision trees with different model complexities.



(a) Decision boundary for tree with 50 leaf nodes using 20% data for training.



(b) Training and test error rates using 20% data for training.

Figure 3.25.

Performance of decision trees using 20% data for training (twice the original training size).

There are a number of factors that influence model overfitting. In the following, we provide brief descriptions of two of the major factors: limited training size and high model complexity. Though they are not exhaustive, the interplay between them can help explain most of the common model overfitting phenomena in real-world applications.

Limited Training Size

Note that a training set consisting of a finite number of instances can only provide a limited representation of the overall data. Hence, it is possible that the patterns learned from a training set do not fully represent the true patterns in the overall data, leading to model overfitting. In general, as we increase the size of a training set (number of training instances), the patterns learned from the training set start resembling the true patterns in the overall data. Hence,

the effect of overfitting can be reduced by increasing the training size, as illustrated in the following example.

3.6 Example Effect of Training Size

Suppose that we use twice the number of training instances than what we had used in the experiments conducted in [Example 3.5](#). Specifically, we use 20% data for training and use the remainder for testing. [Figure 3.25\(b\)](#) shows the training and test error rates as the size of the tree is varied from 1 to 150. There are two major differences in the trends shown in this figure and those shown in [Figure 3.23\(b\)](#) (using only 10% of the data for training). First, even though the training error rate decreases with increasing tree size in both figures, its rate of decrease is much smaller when we use twice the training size. Second, for a given tree size, the gap between the training and test error rates is much smaller when we use twice the training size. These differences suggest that the patterns learned using 20% of data for training are more generalizable than those learned using 10% of data for training.

[Figure 3.25\(a\)](#) shows the decision boundaries for the tree of size 50, learned using 20% of data for training. In contrast to the tree of the same size learned using 10% data for training (see [Figure 3.24\(d\)](#)), we can see that the decision tree is not capturing specific patterns of noisy + instances in the training set. Instead, the high model complexity of 50 leaf nodes is being effectively used to learn the boundaries of the + instances centered at (10, 10).

High Model Complexity

Generally, a more complex model has a better ability to represent complex patterns in the data. For example, decision trees with larger number of leaf

nodes can represent more complex decision boundaries than decision trees with fewer leaf nodes. However, an overly complex model also has a tendency to learn specific patterns in the training set that do not generalize well over unseen instances. Models with high complexity should thus be judiciously used to avoid overfitting.

One measure of model complexity is the number of “parameters” that need to be inferred from the training set. For example, in the case of decision tree induction, the attribute test conditions at internal nodes correspond to the parameters of the model that need to be inferred from the training set. A decision tree with larger number of attribute test conditions (and consequently more leaf nodes) thus involves more “parameters” and hence is more complex.

Given a class of models with a certain number of parameters, a learning algorithm attempts to select the best combination of parameter values that maximizes an evaluation metric (e.g., accuracy) over the training set. If the number of parameter value combinations (and hence the complexity) is large, the learning algorithm has to select the best combination from a large number of possibilities, using a limited training set. In such cases, there is a high chance for the learning algorithm to pick a *spurious* combination of parameters that maximizes the evaluation metric just by random chance. This is similar to the **multiple comparisons problem** (also referred as multiple testing problem) in statistics.

As an illustration of the multiple comparisons problem, consider the task of predicting whether the stock market will rise or fall in the next ten trading days. If a stock analyst simply makes random guesses, the probability that her prediction is correct on any trading day is 0.5. However, the probability that she will predict correctly at least nine out of ten times is

$$(109)+(1010)210=0.0107,$$

which is extremely low.

Suppose we are interested in choosing an investment advisor from a pool of 200 stock analysts. Our strategy is to select the analyst who makes the most number of correct predictions in the next ten trading days. The flaw in this strategy is that even if all the analysts make their predictions in a random fashion, the probability that at least one of them makes at least nine correct predictions is

$$1-(1-0.0107)200=0.8847,$$

which is very high. Although each analyst has a low probability of predicting at least nine times correctly, considered together, we have a high probability of finding at least one analyst who can do so. However, there is no guarantee in the future that such an analyst will continue to make accurate predictions by random guessing.

How does the multiple comparisons problem relate to model overfitting? In the context of learning a classification model, each combination of parameter values corresponds to an analyst, while the number of training instances corresponds to the number of days. Analogous to the task of selecting the best analyst who makes the most accurate predictions on consecutive days, the task of a learning algorithm is to select the best combination of parameters that results in the highest accuracy on the training set. If the number of parameter combinations is large but the training size is small, it is highly likely for the learning algorithm to choose a spurious parameter combination that provides high training accuracy just by random chance. In the following example, we illustrate the phenomena of overfitting due to multiple comparisons in the context of decision tree induction.

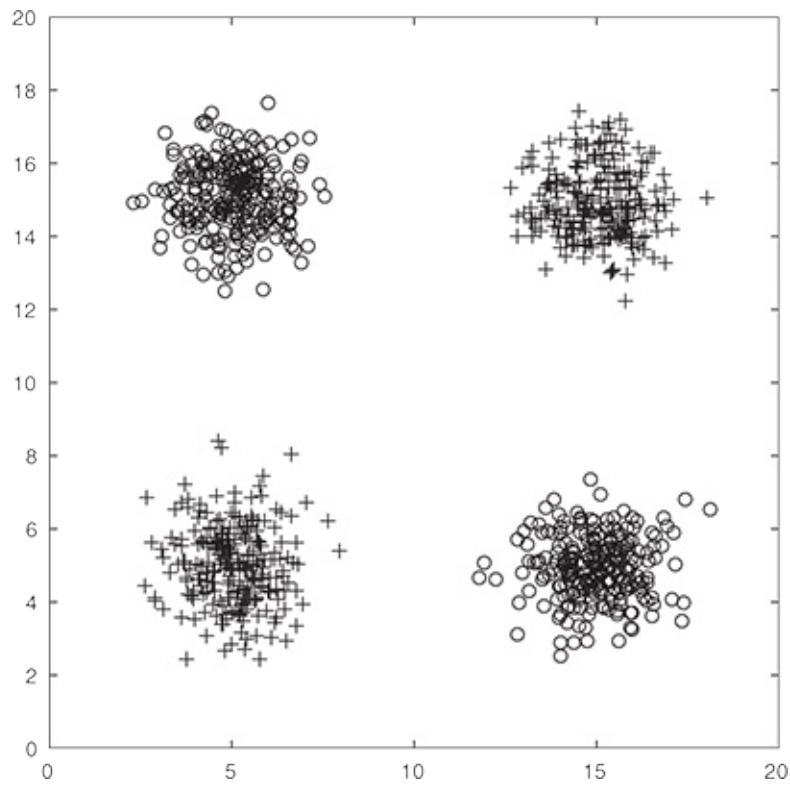
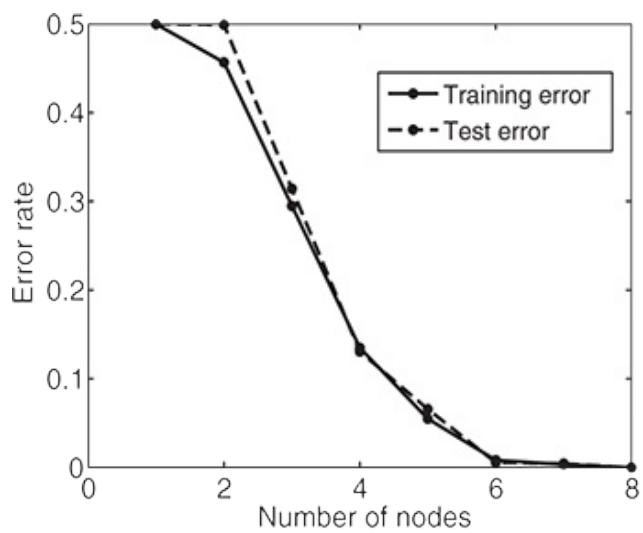
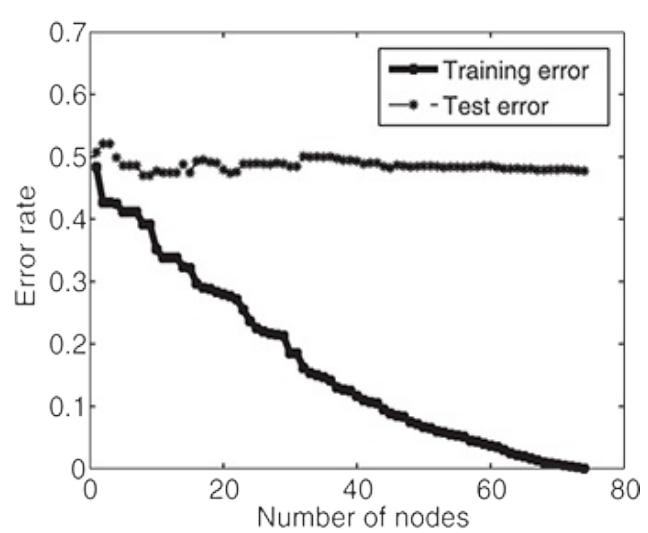


Figure 3.26.
Example of a two-dimensional (X-Y) data set.



(a) Using X and Y attributes only.



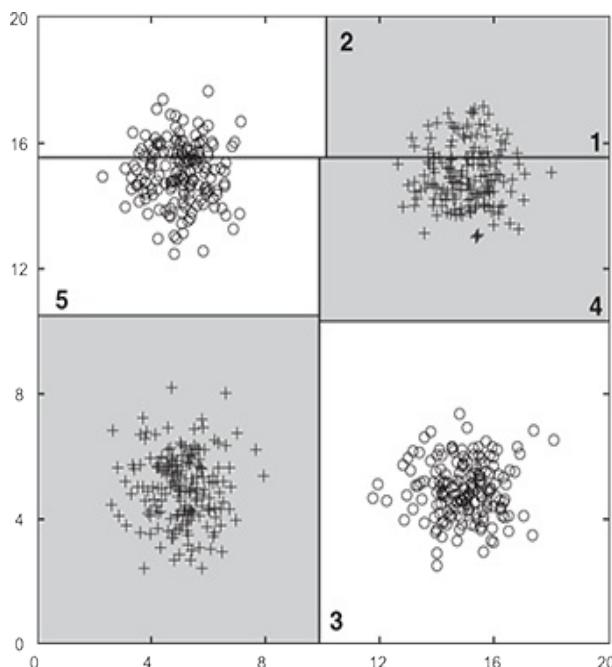
(b) After adding 100 irrelevant attributes.

Figure 3.27.

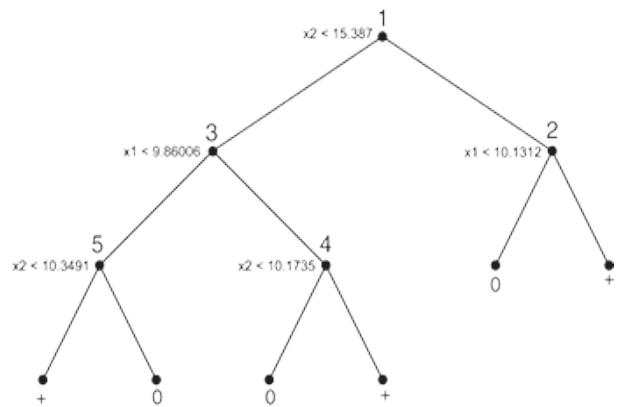
Training and test error rates illustrating the effect of multiple comparisons problem on model overfitting.

3.7. Example Multiple Comparisons and Overfitting

Consider the two-dimensional data set shown in [Figure 3.26](#) containing 500 + and 500 \circ instances, which is similar to the data shown in [Figure 3.19](#). In this data set, the distributions of both classes are well-separated in the two-dimensional (XY) attribute space, but none of the two attributes (X or Y) are sufficiently informative to be used alone for separating the two classes. Hence, splitting the data set based on any value of an X or Y attribute will provide close to zero reduction in an impurity measure. However, if X and Y attributes are used together in the splitting criterion (e.g., splitting X at 10 *and* Y at 10), the two classes can be effectively separated.



(a) Decision boundary for tree with 6 leaf nodes.



(b) Decision tree with 6 leaf nodes.

Figure 3.28.

Decision tree with 6 leaf nodes using X and Y as attributes. Splits have been numbered from 1 to 5 in order of other occurrence in the tree.

Figure 3.27(a) shows the training and test error rates for learning decision trees of varying sizes, when 30% of the data is used for training and the remainder of the data for testing. We can see that the two classes can be separated using a small number of leaf nodes. **Figure 3.28** shows the decision boundaries for the tree with six leaf nodes, where the splits have been numbered according to their order of appearance in the tree. Note that even though splits 1 and 3 provide trivial gains, their consequent splits (2, 4, and 5) provide large gains, resulting in effective discrimination of the two classes.

Assume we add 100 irrelevant attributes to the two-dimensional X-Y data. Learning a decision tree from this resultant data will be challenging because the number of candidate attributes to choose for splitting at every internal node will increase from two to 102. With such a large number of candidate attribute test conditions to choose from, it is quite likely that spurious attribute test conditions will be selected at internal nodes because of the multiple comparisons problem. **Figure 3.27(b)** shows the training and test error rates after adding 100 irrelevant attributes to the training set. We can see that the test error rate remains close to 0.5 even after using 50 leaf nodes, while the training error rate keeps on declining and eventually becomes 0.

3.5 Model Selection

There are many possible classification models with varying levels of model complexity that can be used to capture patterns in the training data. Among these possibilities, we want to select the model that shows lowest generalization error rate. The process of selecting a model with the right level of complexity, which is expected to generalize well over unseen test instances, is known as **model selection**. As described in the previous section, the training error rate cannot be reliably used as the sole criterion for model selection. In the following, we present three generic approaches to estimate the generalization performance of a model that can be used for model selection. We conclude this section by presenting specific strategies for using these approaches in the context of decision tree induction.

3.5.1 Using a Validation Set

Note that we can always estimate the generalization error rate of a model by using “out-of-sample” estimates, i.e. by evaluating the model on a separate **validation set** that is not used for training the model. The error rate on the validation set, termed as the validation error rate, is a better indicator of generalization performance than the training error rate, since the validation set has not been used for training the model. The validation error rate can be used for model selection as follows.

Given a training set $D.\text{train}$, we can partition $D.\text{train}$ into two smaller subsets, $D.\text{tr}$ and $D.\text{val}$, such that $D.\text{tr}$ is used for training while $D.\text{val}$ is used as the validation set. For example, two-thirds of $D.\text{train}$ can be reserved as $D.\text{tr}$ for

training, while the remaining one-third is used as $D.val$ for computing validation error rate. For any choice of classification model m that is trained on $D.tr$, we can estimate its validation error rate on $D.val$, $\text{errval}(m)$. The model that shows the lowest value of $\text{errval}(m)$ can then be selected as the preferred choice of model.

The use of validation set provides a generic approach for model selection. However, one limitation of this approach is that it is sensitive to the sizes of $D.tr$ and $D.val$, obtained by partitioning $D.train$. If the size of $D.tr$ is too small, it may result in the learning of a poor classification model with sub-standard performance, since a smaller training set will be less representative of the overall data. On the other hand, if the size of $D.val$ is too small, the validation error rate might not be reliable for selecting models, as it would be computed over a small number of instances.

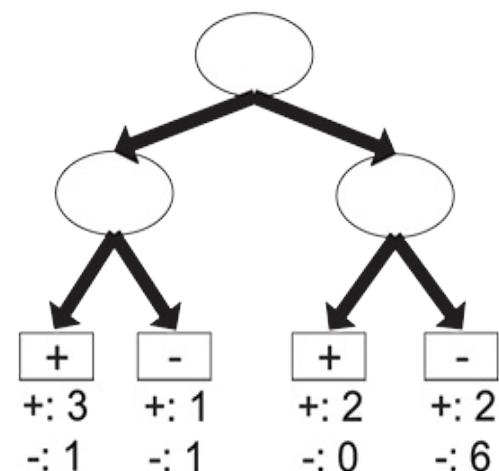
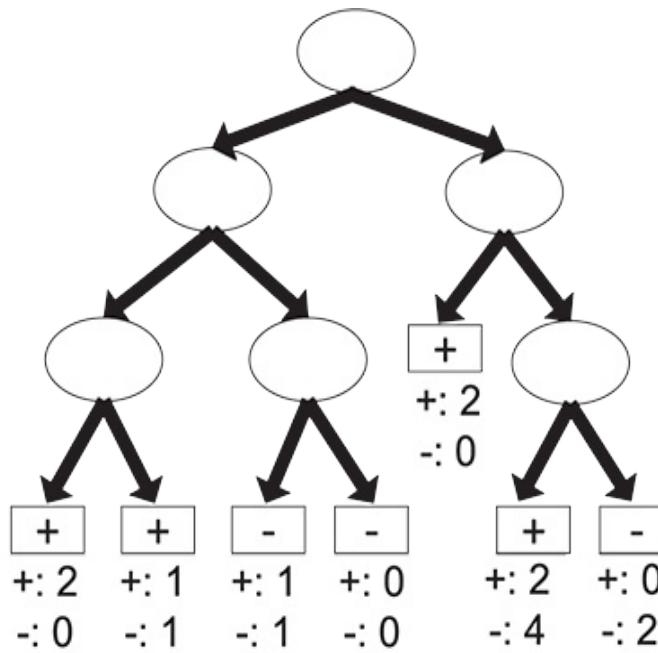


Figure 3.29.

Class distribution of validation data for the two decision trees shown in [Figure 3.30](#).

3.8. Example Validation Error

In the following example, we illustrate one possible approach for using a validation set in decision tree induction. [Figure 3.29](#) shows the predicted labels at the leaf nodes of the decision trees generated in [Figure 3.30](#). The counts given beneath the leaf nodes represent the proportion of data instances in the validation set that reach each of the nodes. Based on the predicted labels of the nodes, the validation error rate for the left tree is $\text{errval}(\text{TL})=6/16=0.375$, while the validation error rate for the right tree is $\text{errval}(\text{TR})=4/16=0.25$. Based on their validation error rates, the right tree is preferred over the left one.

3.5.2 Incorporating Model Complexity

Since the chance for model overfitting increases as the model becomes more complex, a model selection approach should not only consider the training error rate but also the model complexity. This strategy is inspired by a well-known principle known as **Occam's razor** or the **principle of parsimony**, which suggests that given two models with the same errors, the simpler model is preferred over the more complex model. A generic approach to account for model complexity while estimating generalization performance is formally described as follows.

Given a training set $D.\text{train}$, let us consider learning a classification model m that belongs to a certain class of models, M . For example, if M represents the set of all possible decision trees, then m can correspond to a specific decision

tree learned from the training set. We are interested in estimating the generalization error rate of m , $\text{gen.error}(m)$. As discussed previously, the training error rate of m , $\text{train.error}(m, D.\text{train})$, can under-estimate $\text{gen.error}(m)$ when the model complexity is high. Hence, we represent $\text{gen.error}(m)$ as a function of not just the training error rate but also the model complexity of M , $\text{complexity}(M)$, as follows:

$$\text{gen.error}(m) = \text{train.error}(m, D.\text{train}) + \alpha \times \text{complexity}(M), \quad (3.11)$$

where α is a hyper-parameter that strikes a balance between minimizing training error and reducing model complexity. A higher value of α gives more emphasis to the model complexity in the estimation of generalization performance. To choose the right value of α , we can make use of the validation set in a similar way as described in [3.5.1](#). For example, we can iterate through a range of values of α and for every possible value, we can learn a model on a subset of the training set, $D.\text{tr}$, and compute its validation error rate on a separate subset, $D.\text{val}$. We can then select the value of α that provides the lowest validation error rate.

[Equation 3.11](#) provides one possible approach for incorporating model complexity into the estimate of generalization performance. This approach is at the heart of a number of techniques for estimating generalization performance, such as the structural risk minimization principle, the Akaike's Information Criterion (AIC), and the Bayesian Information Criterion (BIC). The structural risk minimization principle serves as the building block for learning support vector machines, which will be discussed later in [Chapter 4](#). For more details on AIC and BIC, see the Bibliographic Notes.

In the following, we present two different approaches for estimating the complexity of a model, $\text{complexity}(M)$. While the former is specific to decision trees, the latter is more generic and can be used with any class of models.

Estimating the Complexity of Decision Trees

In the context of decision trees, the complexity of a decision tree can be estimated as the ratio of the number of leaf nodes to the number of training instances. Let k be the number of leaf nodes and N_{train} be the number of training instances. The complexity of a decision tree can then be described as k/N_{train} . This reflects the intuition that for a larger training size, we can learn a decision tree with larger number of leaf nodes without it becoming overly complex. The generalization error rate of a decision tree T can then be computed using [Equation 3.11](#) as follows:

$$\text{errgen}(T) = \text{err}(T) + \Omega \times k N_{\text{train}},$$

where $\text{err}(T)$ is the training error of the decision tree and Ω is a hyper-parameter that makes a trade-off between reducing training error and minimizing model complexity, similar to the use of α in [Equation 3.11](#). Ω can be viewed as the relative cost of adding a leaf node relative to incurring a training error. In the literature on decision tree induction, the above approach for estimating generalization error rate is also termed as the **pessimistic error estimate**. It is called pessimistic as it assumes the generalization error rate to be worse than the training error rate (by adding a penalty term for model complexity). On the other hand, simply using the training error rate as an estimate of the generalization error rate is called the **optimistic error estimate** or the **resubstitution estimate**.

3.9. Example Generalization Error Estimates

Consider the two binary decision trees, TL and TR, shown in [Figure 3.30](#). Both trees are generated from the same training data and TL is generated by expanding three leaf nodes of TR. The counts shown in the leaf nodes of the trees represent the class distribution of the training

instances. If each leaf node is labeled according to the majority class of training instances that reach the node, the training error rate for the left tree will be $\text{err}(TL)=4/24=0.167$, while the training error rate for the right tree will be $\text{err}(TR)=6/24=0.25$. Based on their training error rates alone, TL would prefer over TR, even though TL is more complex (contains larger number of leaf nodes) than TR.

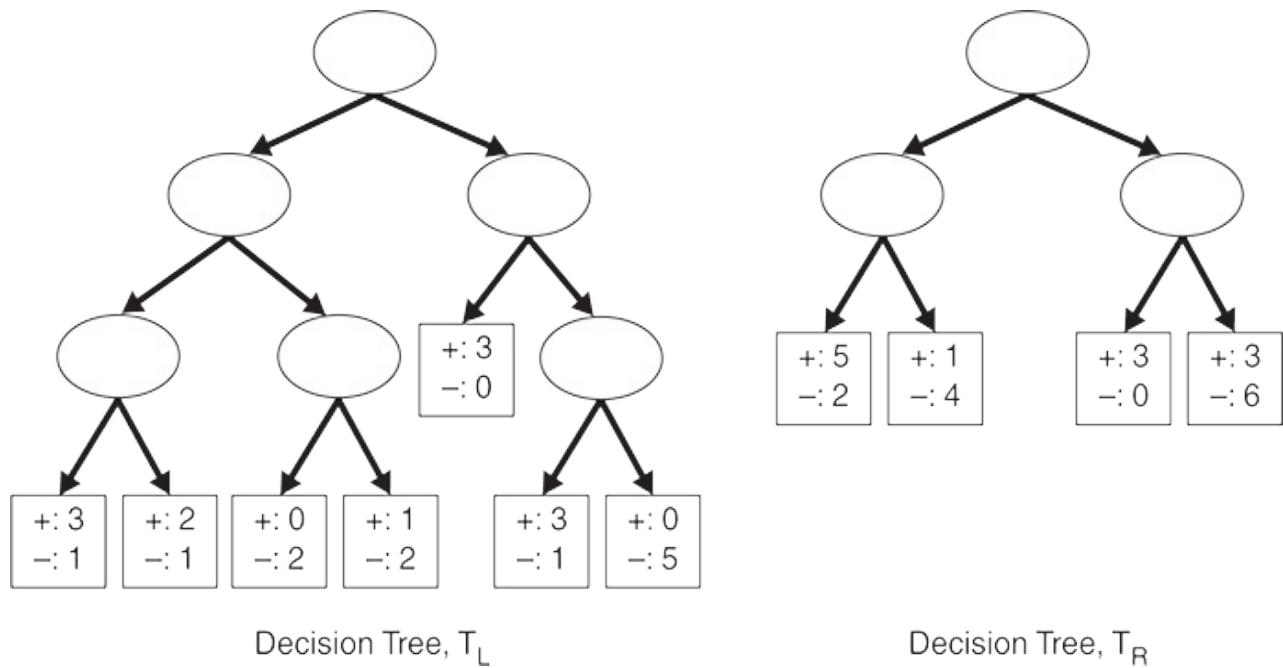


Figure 3.30.

Example of two decision trees generated from the same training data.

Now, assume that the cost associated with each leaf node is $\Omega=0.5$. Then, the generalization error estimate for TL will be

$$\text{errgen}(TL)=424+0.5\times724=7.524=0.3125$$

and the generalization error estimate for TR will be

$$\text{errgen}(TR)=624+0.5\times424=824=0.3333.$$

Since TL has a lower generalization error rate, it will still be preferred over TR. Note that $\Omega=0.5$ implies that a node should always be expanded into its two child nodes if it improves the prediction of at least one training instance, since expanding a node is less costly than misclassifying a training instance. On the other hand, if $\Omega=1$, then the generalization error rate for TL is $\text{errgen(TL)}=11/24=0.458$ and for TR is $\text{errgen(TR)}=10/24=0.417$. In this case, TR will be preferred over TL because it has a lower generalization error rate. This example illustrates that different choices of Ω can change our preference of decision trees based on their generalization error estimates. However, for a given choice of Ω , the pessimistic error estimate provides an approach for modeling the generalization performance on unseen test instances. The value of Ω can be selected with the help of a validation set.

Minimum Description Length Principle

Another way to incorporate model complexity is based on an information-theoretic approach known as the minimum description length or MDL principle. To illustrate this approach, consider the example shown in [Figure 3.31](#) . In this example, both person A and person B are given a set of instances with known attribute values x . Assume person A knows the class label y for every instance, while person B has no such information. A would like to share the class information with B by sending a message containing the labels. The message would contain $\Theta(N)$ bits of information, where N is the number of instances.

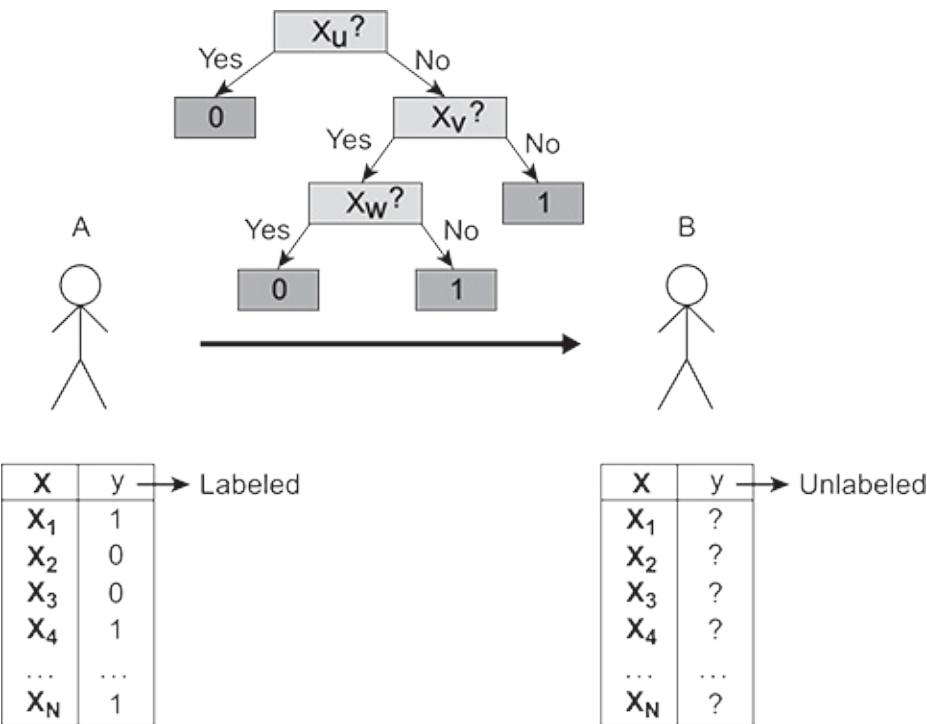


Figure 3.31.

An illustration of the minimum description length principle.

Alternatively, instead of sending the class labels explicitly, A can build a classification model from the instances and transmit it to B. B can then apply the model to determine the class labels of the instances. If the model is 100% accurate, then the cost for transmission is equal to the number of bits required to encode the model. Otherwise, A must also transmit information about which instances are misclassified by the model so that B can reproduce the same class labels. Thus, the overall transmission cost, which is equal to the total description length of the message, is

$$\text{Cost(model, data)} = \text{Cost(data|model)} + \alpha \times \text{Cost(model)}, \quad (3.12)$$

where the first term on the right-hand side is the number of bits needed to encode the misclassified instances, while the second term is the number of bits required to encode the model. There is also a hyper-parameter α that trades-off the relative costs of the misclassified instances and the model.

Notice the familiarity between this equation and the generic equation for generalization error rate presented in [Equation 3.11](#). A good model must have a total description length less than the number of bits required to encode the entire sequence of class labels. Furthermore, given two competing models, the model with lower total description length is preferred. An example showing how to compute the total description length of a decision tree is given in Exercise 10 on page 189.

3.5.3 Estimating Statistical Bounds

Instead of using [Equation 3.11](#) to estimate the generalization error rate of a model, an alternative way is to apply a statistical correction to the training error rate of the model that is indicative of its model complexity. This can be done if the probability distribution of training error is available or can be assumed. For example, the number of errors committed by a leaf node in a decision tree can be assumed to follow a binomial distribution. We can thus compute an upper bound limit to the observed training error rate that can be used for model selection, as illustrated in the following example.

3.10. Example Statistical Bounds on Training Error

Consider the left-most branch of the binary decision trees shown in [Figure 3.30](#). Observe that the left-most leaf node of TR has been expanded into two child nodes in TL. Before splitting, the training error rate of the node is $2/7=0.286$. By approximating a binomial distribution with a normal distribution, the following upper bound of the training error rate e can be derived:

$$eupper(N, e, \alpha) = e + z\alpha/222N + z\alpha/2e(1-e)N + z\alpha/224N^2 + z\alpha/22N, \quad (3.13)$$

where α is the confidence level, $z\alpha/2$ is the standardized value from a standard normal distribution, and N is the total number of training instances used to compute e . By replacing $\alpha=25\%$, $N=7$, and $e=2/7$, the upper bound for the error rate is $eupper(7, 2/7, 0.25)=0.503$, which corresponds to $7\times 0.503=3.521$ errors. If we expand the node into its child nodes as shown in TL, the training error rates for the child nodes are $1/4=0.250$ and $1/3=0.333$, respectively. Using **Equation (3.13)**, the upper bounds of these error rates are $eupper(4, 1/4, 0.25)=0.537$ and $eupper(3, 1/3, 0.25)=0.650$, respectively. The overall training error of the child nodes is $4\times 0.537+3\times 0.650=4.098$, which is larger than the estimated error for the corresponding node in TR, suggesting that it should not be split.

3.5.4 Model Selection for Decision Trees

Building on the generic approaches presented above, we present two commonly used model selection strategies for decision tree induction.

Prepruning (Early Stopping Rule)

In this approach, the tree-growing algorithm is halted before generating a fully grown tree that perfectly fits the entire training data. To do this, a more restrictive stopping condition must be used; e.g., stop expanding a leaf node when the observed gain in the generalization error estimate falls below a certain threshold. This estimate of the generalization error rate can be

computed using any of the approaches presented in the preceding three subsections, e.g., by using pessimistic error estimates, by using validation error estimates, or by using statistical bounds. The advantage of prepruning is that it avoids the computations associated with generating overly complex subtrees that overfit the training data. However, one major drawback of this method is that, even if no significant gain is obtained using one of the existing splitting criterion, subsequent splitting may result in better subtrees. Such subtrees would not be reached if prepruning is used because of the greedy nature of decision tree induction.

Post-pruning

In this approach, the decision tree is initially grown to its maximum size. This is followed by a tree-pruning step, which proceeds to trim the fully grown tree in a bottom-up fashion. Trimming can be done by replacing a subtree with (1) a new leaf node whose class label is determined from the majority class of instances affiliated with the subtree (approach known as **subtree replacement**), or (2) the most frequently used branch of the subtree (approach known as **subtree raising**). The tree-pruning step terminates when no further improvement in the generalization error estimate is observed beyond a certain threshold. Again, the estimates of generalization error rate can be computed using any of the approaches presented in the previous three subsections. Post-pruning tends to give better results than prepruning because it makes pruning decisions based on a fully grown tree, unlike prepruning, which can suffer from premature termination of the tree-growing process. However, for post-pruning, the additional computations needed to grow the full tree may be wasted when the subtree is pruned.

Figure 3.32  illustrates the simplified decision tree model for the web robot detection example given in **Section 3.3.5** . Notice that the subtree rooted at depth=1 has been replaced by one of its branches corresponding to

breadth ≤ 7 , width > 3 , and MultiP=1, using subtree raising. On the other hand, the subtree corresponding to depth > 1 and MultiAgent=0 has been replaced by a leaf node assigned to class 0, using subtree replacement. The subtree for depth > 1 and MultiAgent=1 remains intact.

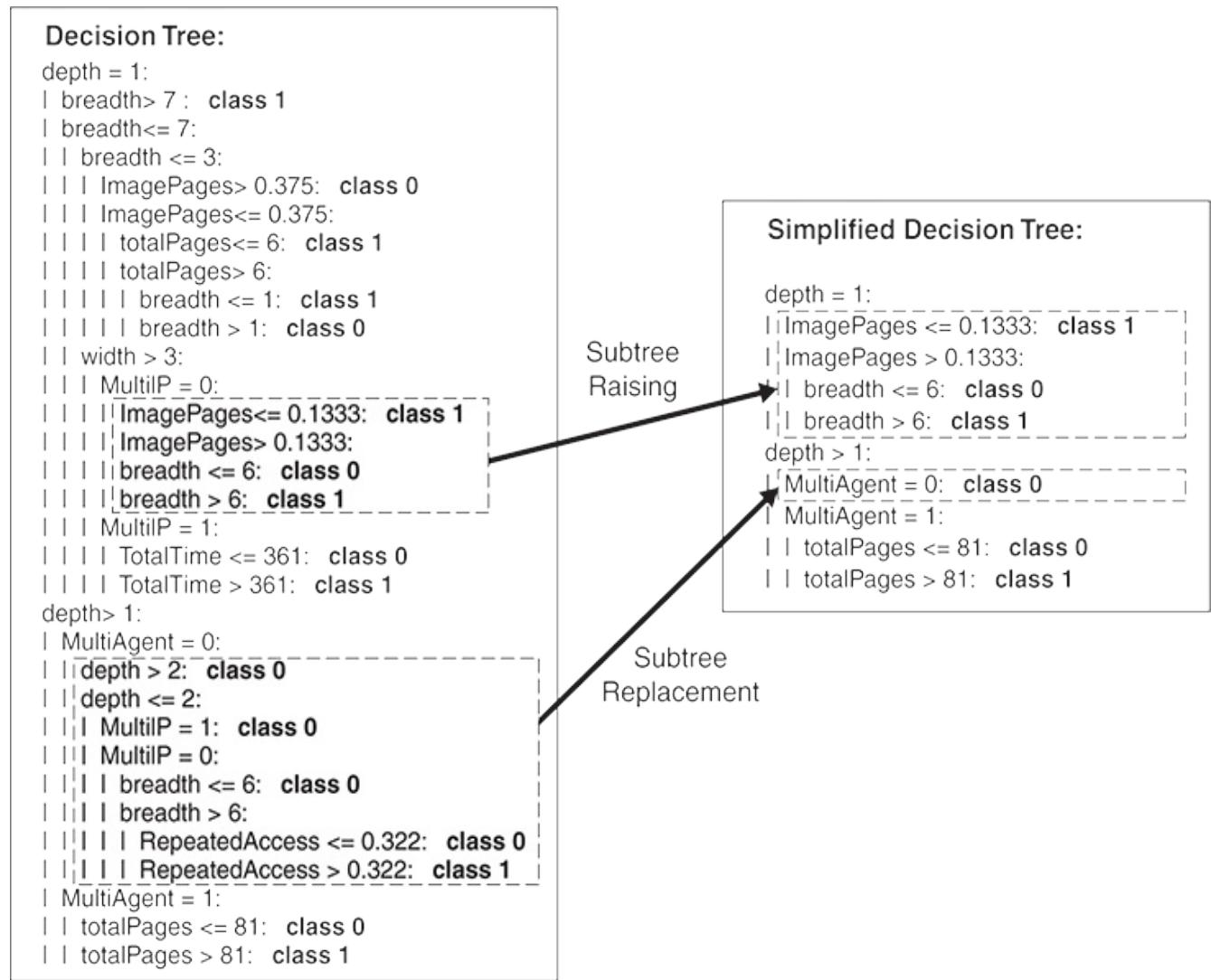


Figure 3.32.

Post-pruning of the decision tree for web robot detection.

3.6 Model Evaluation

The previous section discussed several approaches for model selection that can be used to learn a classification model from a training set $D.\text{train}$. Here we discuss methods for estimating its generalization performance, i.e. its performance on unseen instances outside of $D.\text{train}$. This process is known as **model evaluation**.

Note that model selection approaches discussed in [Section 3.5](#) also compute an estimate of the generalization performance using the training set $D.\text{train}$. However, these estimates are *biased* indicators of the performance on unseen instances, since they were used to guide the selection of classification model. For example, if we use the validation error rate for model selection (as described in [Section 3.5.1](#)), the resulting model would be deliberately chosen to minimize the errors on the validation set. The validation error rate may thus under-estimate the true generalization error rate, and hence cannot be reliably used for model evaluation.

A correct approach for model evaluation would be to assess the performance of a learned model on a labeled test set has not been used at any stage of model selection. This can be achieved by partitioning the entire set of labeled instances D , into two disjoint subsets, $D.\text{train}$, which is used for model selection and $D.\text{test}$, which is used for computing the test error rate, `errtest`. In the following, we present two different approaches for partitioning D into $D.\text{train}$ and $D.\text{test}$, and computing the test error rate, `errtest`.

3.6.1 Holdout Method

The most basic technique for partitioning a labeled data set is the holdout method, where the labeled set D is randomly partitioned into two disjoint sets, called the training set $D.\text{train}$ and the test set $D.\text{test}$. A classification model is then induced from $D.\text{train}$ using the model selection approaches presented in [Section 3.5](#), and its error rate on $D.\text{test}$, `errtest`, is used as an estimate of the generalization error rate. The proportion of data reserved for training and for testing is typically at the discretion of the analysts, e.g., two-thirds for training and one-third for testing.

Similar to the trade-off faced while partitioning $D.\text{train}$ into $D.\text{tr}$ and $D.\text{val}$ in [Section 3.5.1](#), choosing the right fraction of labeled data to be used for training and testing is non-trivial. If the size of $D.\text{train}$ is small, the learned classification model may be improperly learned using an insufficient number of training instances, resulting in a biased estimation of generalization performance. On the other hand, if the size of $D.\text{test}$ is small, `errtest` may be less reliable as it would be computed over a small number of test instances. Moreover, `errtest` can have a high variance as we change the random partitioning of D into $D.\text{train}$ and $D.\text{test}$.

The holdout method can be repeated several times to obtain a distribution of the test error rates, an approach known as **random subsampling** or repeated holdout method. This method produces a distribution of the error rates that can be used to understand the variance of `errtest`.

3.6.2 Cross-Validation

Cross-validation is a widely-used model evaluation method that aims to make effective use of all labeled instances in D for both training and testing. To illustrate this method, suppose that we are given a labeled set that we have

randomly partitioned into three equal-sized subsets, S_1 , S_2 , and S_3 , as shown in [Figure 3.33](#). For the first run, we train a model using subsets S_2 and S_3 (shown as empty blocks) and test the model on subset S_1 . The test error rate on S_1 , denoted as $\text{err}(S_1)$, is thus computed in the first run. Similarly, for the second run, we use S_1 and S_3 as the training set and S_2 as the test set, to compute the test error rate, $\text{err}(S_2)$, on S_2 . Finally, we use S_1 and S_2 for training in the third run, while S_3 is used for testing, thus resulting in the test error rate $\text{err}(S_3)$ for S_3 . The overall test error rate is obtained by summing up the number of errors committed in each test subset across all runs and dividing it by the total number of instances. This approach is called three-fold cross-validation.

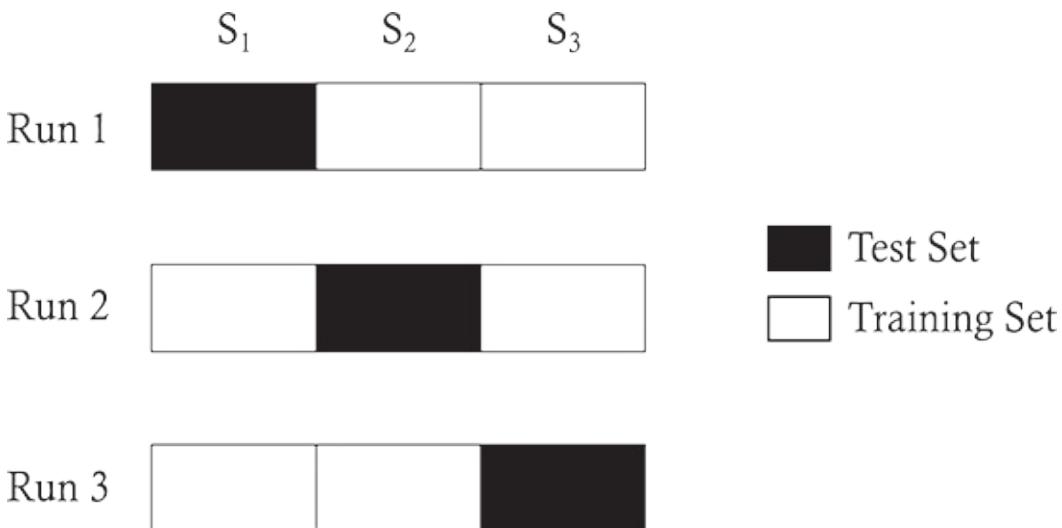


Figure 3.33.

Example demonstrating the technique of 3-fold cross-validation.

The k -fold cross-validation method generalizes this approach by segmenting the labeled data D (of size N) into k equal-sized partitions (or folds). During the i^{th} run, one of the partitions of D is chosen as $D.\text{test}(i)$ for testing, while the rest of the partitions are used as $D.\text{train}(i)$ for training. A model $m(i)$ is learned using $D.\text{train}(i)$ and applied on $D.\text{test}(i)$ to obtain the sum of test errors,

`errsum(i)`. This procedure is repeated k times. The total test error rate, `errest`, is then computed as

$$\text{errest} = \sum_{i=1}^k \text{errsum}(i)N. \quad (3.14)$$

Every instance in the data is thus used for testing exactly once and for training exactly $(k-1)$ times. Also, every run uses $(k-1)/k$ fraction of the data for training and $1/k$ fraction for testing.

The right choice of k in k -fold cross-validation depends on a number of characteristics of the problem. A small value of k will result in a smaller training set at every run, which will result in a larger estimate of generalization error rate than what is expected of a model trained over the entire labeled set. On the other hand, a high value of k results in a larger training set at every run, which reduces the bias in the estimate of generalization error rate. In the extreme case, when $k=N$, every run uses exactly one data instance for testing and the remainder of the data for training. This special case of k -fold cross-validation is called the **leave-one-out** approach. This approach has the advantage of utilizing as much data as possible for training. However, leave-one-out can produce quite misleading results in some special scenarios, as illustrated in Exercise 11. Furthermore, leave-one-out can be computationally expensive for large data sets as the cross-validation procedure needs to be repeated N times. For most practical applications, the choice of k between 5 and 10 provides a reasonable approach for estimating the generalization error rate, because each fold is able to make use of 80% to 90% of the labeled data for training.

The k -fold cross-validation method, as described above, produces a single estimate of the generalization error rate, without providing any information about the variance of the estimate. To obtain this information, we can run k -fold cross-validation for every possible partitioning of the data into k partitions,

and obtain a distribution of test error rates computed for every such partitioning. The average test error rate across all possible partitionings serves as a more robust estimate of generalization error rate. This approach of estimating the generalization error rate and its variance is known as the **complete cross-validation** approach. Even though such an estimate is quite robust, it is usually too expensive to consider all possible partitionings of a large data set into k partitions. A more practical solution is to repeat the cross-validation approach multiple times, using a different random partitioning of the data into k partitions at every time, and use the average test error rate as the estimate of generalization error rate. Note that since there is only one possible partitioning for the leave-one-out approach, it is not possible to estimate the variance of generalization error rate, which is another limitation of this method.

The k -fold cross-validation does not guarantee that the fraction of positive and negative instances in every partition of the data is equal to the fraction observed in the overall data. A simple solution to this problem is to perform a stratified sampling of the positive and negative instances into k partitions, an approach called **stratified cross-validation**.

In k -fold cross-validation, a different model is learned at every run and the performance of every one of the k models on their respective test folds is then aggregated to compute the overall test error rate, errtest. Hence, errtest does not reflect the generalization error rate of any of the k models. Instead, it reflects the *expected* generalization error rate of the *model selection approach*, when applied on a training set of the same size as one of the training folds ($N(k-1)/k$). This is different than the errtest computed in the holdout method, which exactly corresponds to the specific model learned over $D.\text{train}$. Hence, although effectively utilizing every data instance in D for training and testing, the errtest computed in the cross-validation method does not represent the performance of a single model learned over a specific $D.\text{train}$.

Nonetheless, in practice, `errtest` is typically used as an estimate of the generalization error of a model built on D . One motivation for this is that when the size of the training folds is closer to the size of the overall data (when k is large), then `errtest` resembles the expected performance of a model learned over a data set of the same size as D . For example, when k is 10, every training fold is 90% of the overall data. The `errtest` then should approach the expected performance of a model learned over 90% of the overall data, which will be close to the expected performance of a model learned over D .

3.7 Presence of Hyper-parameters

Hyper-parameters are parameters of learning algorithms that need to be determined before learning the classification model. For instance, consider the hyper-parameter α that appeared in [Equation 3.11](#), which is repeated here for convenience. This equation was used for estimating the generalization error for a model selection approach that used an explicit representations of model complexity. (See [Section 3.5.2](#).)

$\text{gen.error}(m) = \text{train.error}(m, D_{\text{train}}) + \alpha \times \text{complexity}(M)$

For other examples of hyper-parameters, see [Chapter 4](#).

Unlike regular model parameters, such as the test conditions in the internal nodes of a decision tree, hyper-parameters such as α do not appear in the final classification model that is used to classify unlabeled instances. However, the values of hyper-parameters need to be determined during model selection—a process known as **hyper-parameter selection**—and must be taken into account during model evaluation. Fortunately, both tasks can be effectively accomplished via slight modifications of the cross-validation approach described in the previous section.

3.7.1 Hyper-parameter Selection

In [Section 3.5.2](#), a validation set was used to select α and this approach is generally applicable for hyper-parameter selection. Let p be the hyper-parameter that needs to be selected from a finite range of values, $P =$

$\{p_1, p_2, \dots, p_n\}$. Partition $D.\text{train}$ into $D.\text{tr}$ and $D.\text{val}$. For every choice of hyper-parameter value p_i , we can learn a model m_i on $D.\text{tr}$, and apply this model on $D.\text{val}$ to obtain the validation error rate $\text{err}_{\text{val}}(p_i)$. Let p^* be the hyper-parameter value that provides the lowest validation error rate. We can then use the model m^* corresponding to p^* as the final choice of classification model.

The above approach, although useful, uses only a subset of the data, $D.\text{train}$, for training and a subset, $D.\text{val}$, for validation. The framework of cross-validation, presented in [Section 3.6.2](#), addresses both of those issues, albeit in the context of model evaluation. Here we indicate how to use a cross-validation approach for hyper-parameter selection. To illustrate this approach, let us partition $D.\text{train}$ into three folds as shown in [Figure 3.34](#). At every run, one of the folds is used as $D.\text{val}$ for validation, and the remaining two folds are used as $D.\text{tr}$ for learning a model, for every choice of hyper-parameter value p_i . The overall validation error rate corresponding to each p_i is computed by summing the errors across all the three folds. We then select the hyper-parameter value p^* that provides the lowest validation error rate, and use it to learn a model m^* on the entire training set $D.\text{train}$.

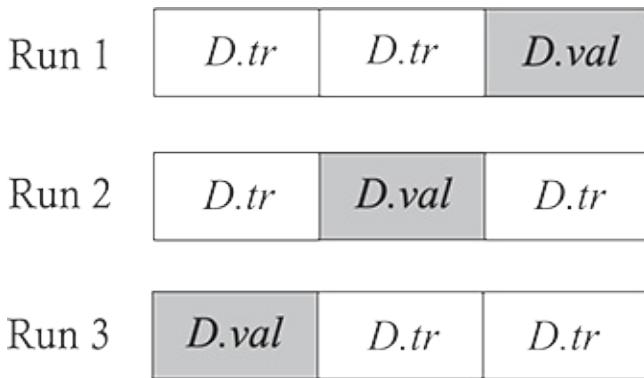


Figure 3.34.

Example demonstrating the 3-fold cross-validation framework for hyper-parameter selection using $D.\text{train}$.

Algorithm 3.2 generalizes the above approach using a k -fold cross-validation framework for hyper-parameter selection. At the i^{th} run of cross-validation, the data in the i^{th} fold is used as $D.\text{val}(i)$ for validation (Step 4), while the remainder of the data in $D.\text{train}$ is used as $D.\text{tr}(i)$ for training (Step 5). Then for every choice of hyper-parameter value π_i , a model is learned on $D.\text{tr}(i)$ (Step 7), which is applied on $D.\text{val}(i)$ to compute its validation error (Step 8). This is used to compute the validation error rate corresponding to models learning using π_i over all the folds (Step 11). The hyper-parameter value π^* that provides the lowest validation error rate (Step 12) is now used to learn the final model m^* on the entire training set $D.\text{train}$ (Step 13). Hence, at the end of this algorithm, we obtain the best choice of the hyper-parameter value as well as the final classification model (Step 14), both of which are obtained by making an effective use of every data instance in $D.\text{train}$.

Algorithm 3.2 Procedure model-select($k, P, D.\text{train}$)

```

1:  $N_{\text{train}} = |D.\text{train}|$  {Size of  $D.\text{train}.$ }
2: Divide  $D.\text{train}$  into  $k$  partitions,  $D.\text{train}_1$  to  $D.\text{train}_k$ .
3: for each run  $i = 1$  to  $k$  do
4:    $D.\text{val}(i) = D.\text{train}_i$ . {Partition used for validation.}
5:    $D.\text{tr}(i) = D.\text{train} \setminus D.\text{train}_i$ . {Partitions used for training.}
6:   for each parameter  $p \in P$  do
7:      $m = \text{model-train}(p, D.\text{tr}(i))$ . {Train model}
8:      $\text{err}_{\text{sum}}(p, i) = \text{model-test}(m, D.\text{val}(i))$ . {Sum of validation
errors.}
9:   end for
10: end for

```

```

11: errval(p) =  $\sum_{ik} \text{errsum}(p, i) / N_{\text{train}}$ . {Compute validation error
rate.}

12:  $p^* = \operatorname{argmin}_p \text{err}_{\text{val}}(p)$ . {Select best hyper-parameter value.}

13:  $m^* = \text{model-train}(p^*, D.\text{train})$ . {Learn final model on  $D.\text{train}$ }

14: return  $(p^*, m^*)$ .

```

3.7.2 Nested Cross-Validation

The approach of the previous section provides a way to effectively use all the instances in $D.\text{train}$ to learn a classification model when hyper-parameter selection is required. This approach can be applied over the entire data set D to learn the final classification model. However, applying [Algorithm 3.2](#) on D would only return the final classification model m^* but not an estimate of its generalization performance, errest . Recall that the validation error rates used in [Algorithm 3.2](#) cannot be used as estimates of generalization performance, since they are used to guide the selection of the final model m^* . However, to compute errest , we can again use a cross-validation framework for evaluating the performance on the entire data set D , as described originally in [Section 3.6.2](#). In this approach, D is partitioned into $D.\text{train}$ (for training) and $D.\text{test}$ (for testing) at every run of cross-validation. When hyper-parameters are involved, we can use [Algorithm 3.2](#) to train a model using $D.\text{train}$ at every run, thus “internally” using cross-validation for model selection. This approach is called **nested cross-validation** or double cross-validation. Algorithm 3.3 describes the complete approach for estimating errest using nested cross-validation in the presence of hyper-parameters.

As an illustration of this approach, see [Figure 3.35](#) where the labeled set D is partitioned into $D.\text{train}$ and $D.\text{test}$, using a 3-fold cross-validation method.

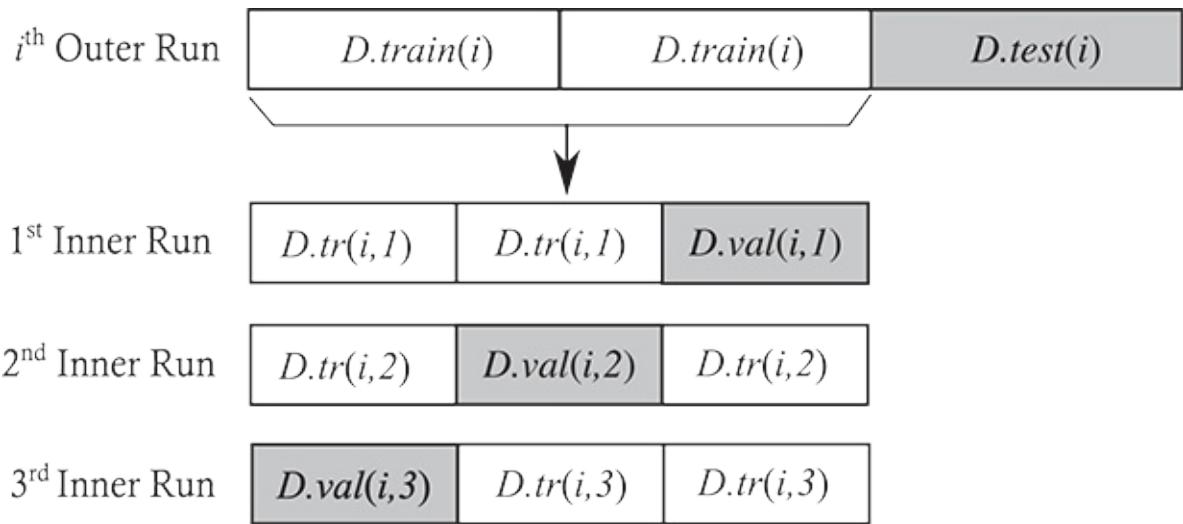


Figure 3.35.

Example demonstrating 3-fold nested cross-validation for computing errtest.

At the i^{th} run of this method, one of the folds is used as the test set, $D.\text{test}(i)$, while the remaining two folds are used as the training set, $D.\text{train}(i)$. This is represented in [Figure 3.35](#) as the i^{th} “outer” run. In order to select a model using $D.\text{train}(i)$, we again use an “inner” 3-fold cross-validation framework that partitions $D.\text{train}(i)$ into $D.\text{tr}$ and $D.\text{val}$ at every one of the three inner runs (iterations). As described in [Section 3.7](#), we can use the inner cross-validation framework to select the best hyper-parameter value $p^*(i)$ as well as its resulting classification model $m^*(i)$ learned over $D.\text{train}(i)$. We can then apply $m^*(i)$ on $D.\text{test}(i)$ to obtain the test error at the i^{th} outer run. By repeating this process for every outer run, we can compute the average test error rate, errtest , over the entire labeled set D . Note that in the above approach, the inner cross-validation framework is being used for model selection while the outer cross-validation framework is being used for model evaluation.

Algorithm 3.3 The nested cross-validation approach for computing errtest.

```

1: Divide  $D$  into  $k$  partitions,  $D_1$  to  $D_k$ .
2: for each outer run  $i = 1$  to  $k$  do
3:    $D.test(i) = D_i$ . {Partition used for testing.}
4:    $D.train(i) = D \setminus D_i$ . {Partitions used for model selection.}
5:    $(p^*, m^*(i)) = \text{model-select}(k, P, D.train(i))$ . {Inner cross-
validation.}
6:    $err_{sum}(i) = \text{model-test}(m^*(i), D.test(i))$ . {Sum of test errors.}
7: end for
8:  $\text{errtest} = \sum_i err_{sum}(i)/N$ . {Compute test error rate.}

```

3.8 Pitfalls of Model Selection and Evaluation

Model selection and evaluation, when used effectively, serve as excellent tools for learning classification models and assessing their generalization performance. However, when using them effectively in practical settings, there are several pitfalls that can result in improper and often misleading conclusions. Some of these pitfalls are simple to understand and easy to avoid, while others are quite subtle in nature and difficult to catch. In the following, we present two of these pitfalls and discuss best practices to avoid them.

3.8.1 Overlap between Training and Test Sets

One of the basic requirements of a *clean* model selection and evaluation setup is that the data used for model selection (*D.train*) must be kept separate from the data used for model evaluation (*D.test*). If there is any overlap between the two, the test error rate *errtest* computed over *D.test* cannot be considered representative of the performance on *unseen* instances.

Comparing the effectiveness of classification models using *errtest* can then be quite misleading, as an overly complex model can show an inaccurately low value of *errtest* due to model overfitting (see Exercise 12 at the end of this chapter).

To illustrate the importance of ensuring no overlap between $D.\text{train}$ and $D.\text{test}$, consider a labeled data set where all the attributes are irrelevant, i.e. they have no relationship with the class labels. Using such attributes, we should expect no classification model to perform better than random guessing. However, if the test set involves even a small number of data instances that were used for training, there is a possibility for an overly complex model to show better performance than random, even though the attributes are completely irrelevant. As we will see later in [Chapter 10](#), this scenario can actually be used as a criterion to detect overfitting due to improper setup of experiment. If a model shows better performance than a random classifier even when the attributes are irrelevant, it is an indication of a potential feedback between the training and test sets.

3.8.2 Use of Validation Error as Generalization Error

The validation error rate errval serves an important role during model selection, as it provides “out-of-sample” error estimates of models on $D.\text{val}$, which is not used for training the models. Hence, errval serves as a better metric than the training error rate for selecting models and hyper-parameter values, as described in [Sections 3.5.1](#) and [3.7](#), respectively. However, once the validation set has been used for selecting a classification model m^* , errval no longer reflects the performance of m^* on *unseen* instances.

To realize the pitfall in using validation error rate as an estimate of generalization performance, consider the problem of selecting a hyper-parameter value p from a range of values P , using a validation set $D.\text{val}$. If the number of possible values in P is quite large and the size of $D.\text{val}$ is small, it is

possible to select a hyper-parameter value p^* that shows favorable performance on $D.val$ just by random chance. Notice the similarity of this problem with the multiple comparisons problem discussed in [Section 3.4.1](#). Even though the classification model m^* learned using p^* would show a low validation error rate, it would lack generalizability on unseen test instances.

The correct approach for estimating the generalization error rate of a model m^* is to use an independently chosen test set $D.test$ that hasn't been used in any way to influence the selection of m^* . As a rule of thumb, the test set should never be examined during model selection, to ensure the absence of any form of overfitting. If the insights gained from any portion of a labeled data set help in improving the classification model even in an indirect way, then that portion of data must be discarded during testing.

3.9 Model Comparison*

One difficulty when comparing the performance of different classification models is whether the observed difference in their performance is statistically significant. For example, consider a pair of classification models, MA and MB. Suppose MA achieves 85% accuracy when evaluated on a test set containing 30 instances, while MB achieves 75% accuracy on a different test set containing 5000 instances. Based on this information, is MA a better model than MB? This example raises two key questions regarding the statistical significance of a performance metric:

1. Although MA has a higher accuracy than MB, it was tested on a smaller test set. How much confidence do we have that the accuracy for MA is actually 85%?
2. Is it possible to explain the difference in accuracies between MA and MB as a result of variations in the composition of their test sets?

The first question relates to the issue of estimating the confidence interval of model accuracy. The second question relates to the issue of testing the statistical significance of the observed deviation. These issues are investigated in the remainder of this section.

3.9.1 Estimating the Confidence Interval for Accuracy

To determine its confidence interval, we need to establish the probability distribution for sample accuracy. This section describes an approach for deriving the confidence interval by modeling the classification task as a binomial random experiment. The following describes characteristics of such an experiment:

1. The random experiment consists of N independent trials, where each trial has two possible outcomes: success or failure.
2. The probability of success, p , in each trial is constant.

An example of a binomial experiment is counting the number of heads that turn up when a coin is flipped N times. If X is the number of successes observed in N trials, then the probability that X takes a particular value is given by a binomial distribution with mean Np and variance $Np(1-p)$:

$$P(X=u) = \binom{N}{u} p^u (1-p)^{N-u}.$$

For example, if the coin is fair ($p=0.5$) and is flipped fifty times, then the probability that the head shows up 20 times is

$$P(X=20) = \binom{50}{20} 0.5^{20} (1-0.5)^{30} = 0.0419.$$

If the experiment is repeated many times, then the average number of heads expected to show up is $50 \times 0.5 = 25$, while its variance is $50 \times 0.5 \times 0.5 = 12.5$.

The task of predicting the class labels of test instances can also be considered as a binomial experiment. Given a test set that contains N instances, let X be the number of instances correctly predicted by a model and p be the true accuracy of the model. If the prediction task is modeled as a binomial experiment, then X has a binomial distribution with mean Np and variance $Np(1-p)$. It can be shown that the empirical accuracy, $\text{acc} = X/N$, also

has a binomial distribution with mean p and variance $p(1-p)/N$ (see Exercise 14). The binomial distribution can be approximated by a normal distribution when N is sufficiently large. Based on the normal distribution, the confidence interval for acc can be derived as follows:

$$P(-Z\alpha/2 \leq acc - p(1-p)/N \leq Z1-\alpha/2) = 1-\alpha, \quad (3.15)$$

where $Z\alpha/2$ and $Z1-\alpha/2$ are the upper and lower bounds obtained from a standard normal distribution at confidence level $(1-\alpha)$. Since a standard normal distribution is symmetric around $Z=0$, it follows that $Z\alpha/2 = Z1-\alpha/2$. Rearranging this inequality leads to the following confidence interval for p :

$$2 \times N \times acc \times Z\alpha/2 \pm Z\alpha/2 \sqrt{Z\alpha/2 + 4Nacc - 4Nacc^2/(N+Z\alpha/2)}. \quad (3.16)$$

The following table shows the values of $Z\alpha/2$ at different confidence levels:

$1-\alpha$	0.99	0.98	0.95	0.9	0.8	0.7	0.5
$Z\alpha/2$	2.58	2.33	1.96	1.65	1.28	1.04	0.67

3.11. Example Confidence Interval for Accuracy

Consider a model that has an accuracy of 80% when evaluated on 100 test instances. What is the confidence interval for its true accuracy at a 95% confidence level? The confidence level of 95% corresponds to $Z\alpha/2=1.96$ according to the table given above. Inserting this term into

Equation 3.16 yields a confidence interval between 71.1% and 86.7%.

The following table shows the confidence interval when the number of instances, N , increases:

N	20	50	100	500	1000	5000

Confidence	0.584	0.670	0.711	0.763	0.774	0.789
Interval	-0.919	-0.888	-0.867	-0.833	-0.824	-0.811

Note that the confidence interval becomes tighter when N increases.

3.9.2 Comparing the Performance of Two Models

Consider a pair of models, M1 and M2, which are evaluated on two independent test sets, D1 and D2. Let n_1 denote the number of instances in D1 and n_2 denote the number of instances in D2. In addition, suppose the error rate for M1 on D1 is e_1 and the error rate for M2 on D2 is e_2 . Our goal is to test whether the observed difference between e_1 and e_2 is statistically significant.

Assuming that n_1 and n_2 are sufficiently large, the error rates e_1 and e_2 can be approximated using normal distributions. If the observed difference in the error rate is denoted as $d = e_1 - e_2$, then d is also normally distributed with mean dt , its true difference, and variance, σ_d^2 . The variance of d can be computed as follows:

$$\sigma_d^2 \approx \sigma_d^2 = e_1(1-e_1)/n_1 + e_2(1-e_2)/n_2, \quad (3.17)$$

where $e_1(1-e_1)/n_1$ and $e_2(1-e_2)/n_2$ are the variances of the error rates. Finally, at the $(1-\alpha)\%$ confidence level, it can be shown that the confidence interval for the true difference dt is given by the following equation:

$$dt = d \pm z\alpha/2\sigma^d \quad (3.18)$$

3.12. Example Significance Testing

Consider the problem described at the beginning of this section. Model MA has an error rate of $e_1=0.15$ when applied to $N_1=30$ test instances, while model MB has an error rate of $e_2=0.25$ when applied to $N_2=5000$ test instances. The observed difference in their error rates is $d=|0.15-0.25|=0.1$. In this example, we are performing a two-sided test to check whether $dt=0$ or $dt\neq0$. The estimated variance of the observed difference in error rates can be computed as follows:

$$\sigma^d = \sqrt{0.15(1-0.15)30 + 0.25(1-0.25)5000} = 0.0043$$

or $\sigma^d=0.0655$. Inserting this value into [Equation 3.18](#), we obtain the following confidence interval for dt at 95% confidence level:

$$dt = 0.1 \pm 1.96 \times 0.0655 = 0.1 \pm 0.128.$$

As the interval spans the value zero, we can conclude that the observed difference is not statistically significant at a 95% confidence level.

At what confidence level can we reject the hypothesis that $dt=0$? To do this, we need to determine the value of $Z\alpha/2$ such that the confidence interval for dt does not span the value zero. We can reverse the preceding computation and look for the value $Z\alpha/2$ such that $d > Z\sigma/2\sigma^d$. Replacing the values of d and σ^d gives $Z\sigma/2 < 1.527$. This value first occurs when $(1-\alpha) \approx 0.936$ (for a two-sided test). The result suggests that the null hypothesis can be rejected at confidence level of 93.6% or lower.

3.10 Bibliographic Notes

Early classification systems were developed to organize various collections of objects, from living organisms to inanimate ones. Examples abound, from Aristotle's cataloguing of species to the Dewey Decimal and Library of Congress classification systems for books. Such a task typically requires considerable human efforts, both to identify properties of the objects to be classified and to organize them into well distinguished categories.

With the development of statistics and computing, automated classification has been a subject of intensive research. The study of classification in classical statistics is sometimes known as **discriminant analysis**, where the objective is to predict the group membership of an object based on its corresponding features. A well-known classical method is Fisher's linear discriminant analysis [142], which seeks to find a linear projection of the data that produces the best separation between objects from different classes.

Many pattern recognition problems also require the discrimination of objects from different classes. Examples include speech recognition, handwritten character identification, and image classification. Readers who are interested in the application of classification techniques for pattern recognition may refer to the survey articles by Jain et al. [150] and Kulkarni et al. [157] or classic pattern recognition books by Bishop [125], Duda et al. [137], and Fukunaga [143]. The subject of classification is also a major research topic in neural networks, statistical learning, and machine learning. An in-depth treatment on the topic of classification from the statistical and machine learning perspectives can be found in the books by Bishop [126], Cherkassky and Mulier [132], Hastie et al. [148], Michie et al. [162], Murphy [167], and Mitchell [165]. Recent years have also seen the release of many publicly available

software packages for classification, which can be embedded in programming languages such as Java (Weka [147]) and Python (scikit-learn [174]).

An overview of decision tree induction algorithms can be found in the survey articles by Buntine [129], Moret [166], Murthy [168], and Safavian et al. [179]. Examples of some well-known decision tree algorithms include CART [127], ID3 [175], C4.5 [177], and CHAID [153]. Both ID3 and C4.5 employ the entropy measure as their splitting function. An in-depth discussion of the C4.5 decision tree algorithm is given by Quinlan [177]. The CART algorithm was developed by Breiman et al. [127] and uses the Gini index as its splitting function. CHAID [153] uses the statistical χ^2 test to determine the best split during the tree-growing process.

The decision tree algorithm presented in this chapter assumes that the splitting condition at each internal node contains only one attribute. An oblique decision tree can use multiple attributes to form the attribute test condition in a single node [149, 187]. Breiman et al. [127] provide an option for using linear combinations of attributes in their CART implementation. Other approaches for inducing oblique decision trees were proposed by Heath et al. [149], Murthy et al. [169], Cantú-Paz and Kamath [130], and Utgoff and Brodley [187]. Although an oblique decision tree helps to improve the expressiveness of the model representation, the tree induction process becomes computationally challenging. Another way to improve the expressiveness of a decision tree without using oblique decision trees is to apply a method known as **constructive induction** [161]. This method simplifies the task of learning complex splitting functions by creating compound features from the original data.

Besides the top-down approach, other strategies for growing a decision tree include the bottom-up approach by Landeweerd et al. [159] and Pattipati and Alexandridis [173], as well as the bidirectional approach by Kim and

Landgrebe [154]. Schuermann and Doster [181] and Wang and Suen [193] proposed using a **soft splitting criterion** to address the data fragmentation problem. In this approach, each instance is assigned to different branches of the decision tree with different probabilities.

Model overfitting is an important issue that must be addressed to ensure that a decision tree classifier performs equally well on previously unlabeled data instances. The model overfitting problem has been investigated by many authors including Breiman et al. [127], Schaffer [180], Mingers [164], and Jensen and Cohen [151]. While the presence of noise is often regarded as one of the primary reasons for overfitting [164, 170], Jensen and Cohen [151] viewed overfitting as an artifact of failure to compensate for the multiple comparisons problem.

Bishop [126] and Hastie et al. [148] provide an excellent discussion of model overfitting, relating it to a well-known framework of theoretical analysis, known as bias-variance decomposition [146]. In this framework, the prediction of a learning algorithm is considered to be a function of the training set, which varies as the training set is changed. The generalization error of a model is then described in terms of its *bias* (the error of the average prediction obtained using different training sets), its *variance* (how different are the predictions obtained using different training sets), and *noise* (the irreducible error inherent to the problem). An underfit model is considered to have high bias but low variance, while an overfit model is considered to have low bias but high variance. Although the bias-variance decomposition was originally proposed for regression problems (where the target attribute is a continuous variable), a unified analysis that is applicable for classification has been proposed by Domingos [136]. The bias variance decomposition will be discussed in more detail while introducing ensemble learning methods in [Chapter 4](#).

Various learning principles, such as the Probably Approximately Correct (PAC) learning framework [188], have been developed to provide a theoretical framework for explaining the generalization performance of learning algorithms. In the field of statistics, a number of performance estimation methods have been proposed that make a trade-off between the goodness of fit of a model and the model complexity. Most noteworthy among them are the Akaike's Information Criterion [120] and the Bayesian Information Criterion [182]. They both apply corrective terms to the training error rate of a model, so as to penalize more complex models. Another widely-used approach for measuring the complexity of any general model is the VapnikChervonenkis (VC) Dimension [190]. The VC dimension of a class of functions C is defined as the maximum number of points that can be *shattered* (every point can be distinguished from the rest) by functions belonging to C , for any possible configuration of points. The VC dimension lays the foundation of the structural risk minimization principle [189], which is extensively used in many learning algorithms, e.g., support vector machines, which will be discussed in detail in [Chapter 4](#).

The Occam's razor principle is often attributed to the philosopher William of Occam. Domingos [135] cautioned against the pitfall of misinterpreting Occam's razor as comparing models with similar training errors, instead of generalization errors. A survey on decision tree-pruning methods to avoid overfitting is given by Breslow and Aha [128] and Esposito et al. [141]. Some of the typical pruning methods include reduced error pruning [176], pessimistic error pruning [176], minimum error pruning [171], critical value pruning [163], cost-complexity pruning [127], and error-based pruning [177]. Quinlan and Rivest proposed using the minimum description length principle for decision tree pruning in [178].

The discussions in this chapter on the significance of cross-validation error estimates is inspired from [Chapter 7](#) in Hastie et al. [148]. It is also an

excellent resource for understanding “the right and wrong ways to do cross-validation”, which is similar to the discussion on pitfalls in [Section 3.8](#) of this chapter. A comprehensive discussion of some of the common pitfalls in using cross-validation for model selection and evaluation is provided in Krstajic et al. [156].

The original cross-validation method was proposed independently by Allen [121], Stone [184], and Geisser [145] for model assessment (evaluation). Even though cross-validation can be used for model selection [194], its usage for model selection is quite different than when it is used for model evaluation, as emphasized by Stone [184]. Over the years, the distinction between the two usages has often been ignored, resulting in incorrect findings. One of the common mistakes while using cross-validation is to perform pre-processing operations (e.g., hyper-parameter tuning or feature selection) using the entire data set and not “within” the training fold of every cross-validation run. Ambroise et al., using a number of gene expression studies as examples, [124] provide an extensive discussion of the *selection bias* that arises when feature selection is performed outside cross-validation. Useful guidelines for evaluating models on microarray data have also been provided by Allison et al. [122].

The use of the cross-validation protocol for hyper-parameter tuning has been described in detail by Dudoit and van der Laan [138]. This approach has been called “grid-search cross-validation.” The correct approach in using cross-validation for both hyper-parameter selection and model evaluation, as discussed in [Section 3.7](#) of this chapter, is extensively covered by Varma and Simon [191]. This combined approach has been referred to as “nested cross-validation” or “double cross-validation” in the existing literature. Recently, Tibshirani and Tibshirani [185] have proposed a new approach for hyper-parameter selection and model evaluation. Tsamardinos et al. [186] compared this approach to nested cross-validation. The experiments they

performed found that, on average, both approaches provide conservative estimates of model performance with the Tibshirani and Tibshirani approach being more computationally efficient.

Kohavi [155] has performed an extensive empirical study to compare the performance metrics obtained using different estimation methods such as random subsampling and k -fold cross-validation. Their results suggest that the best estimation method is ten-fold, stratified cross-validation.

An alternative approach for model evaluation is the bootstrap method, which was presented by Efron in 1979 [139]. In this method, training instances are sampled *with replacement* from the labeled set, i.e., an instance previously selected to be part of the training set is equally likely to be drawn again. If the original data has N instances, it can be shown that, on average, a bootstrap sample of size N contains about 63.2% of the instances in the original data. Instances that are not included in the bootstrap sample become part of the test set. The bootstrap procedure for obtaining training and test sets is repeated b times, resulting in a different error rate on the test set, $\text{err}(i)$, at the i^{th} run. To obtain the overall error rate, errboot , the **.632 bootstrap** approach combines $\text{err}(i)$ with the error rate obtained from a training set containing all the labeled examples, errs , as follows:

$$\text{errboot} = \frac{1}{b} \sum_{i=1}^b (0.632) \times \text{err}(i) + 0.386 \times \text{errs}. \quad (3.19)$$

Efron and Tibshirani [140] provided a theoretical and empirical comparison between cross-validation and a bootstrap method known as the 632+ rule.

While the .632 bootstrap method presented above provides a robust estimate of the generalization performance with low variance in its estimate, it may produce misleading results for highly complex models in certain conditions, as demonstrated by Kohavi [155]. This is because the overall error rate is not

truly an out-of-sample error estimate as it depends on the training error rate, which can be quite small if there is overfitting.

Current techniques such as C4.5 require that the entire training data set fit into main memory. There has been considerable effort to develop parallel and scalable versions of decision tree induction algorithms. Some of the proposed algorithms include SLIQ by Mehta et al. [160], SPRINT by Shafer et al. [183], CMP by Wang and Zaniolo [192], CLOUDS by Alsabti et al. [123], RainForest by Gehrke et al. [144], and ScalParC by Joshi et al. [152]. A survey of parallel algorithms for classification and other data mining tasks is given in [158]. More recently, there has been extensive research to implement large-scale classifiers on the compute unified device architecture (CUDA) [131, 134] and MapReduce [133, 172] platforms.

Bibliography

- [120] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*, pages 199–213. Springer, 1998.
- [121] D. M. Allen. The relationship between variable selection and data augmentation and a method for prediction. *Technometrics*, 16(1):125–127, 1974.
- [122] D. B. Allison, X. Cui, G. P. Page, and M. Sabripour. Microarray data analysis: from disarray to consolidation and consensus. *Nature reviews genetics*, 7(1):55–65, 2006.
- [123] K. Alsabti, S. Ranka, and V. Singh. CLOUDS: A Decision Tree Classifier for Large Datasets. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 2–8, New York, NY, August 1998.
- [124] C. Ambroise and G. J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the national academy of sciences*, 99 (10):6562–6566, 2002.
- [125] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K., 1995.

- [126] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [127] L. Breiman, J. H. Friedman, R. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
- [128] L. A. Breslow and D. W. Aha. Simplifying Decision Trees: A Survey. *Knowledge Engineering Review*, 12(1):1–40, 1997.
- [129] W. Buntine. Learning classification trees. In *Artificial Intelligence Frontiers in Statistics*, pages 182–201. Chapman & Hall, London, 1993.
- [130] E. Cantú-Paz and C. Kamath. Using evolutionary algorithms to induce oblique decision trees. In *Proc. of the Genetic and Evolutionary Computation Conf.*, pages 1053–1060, San Francisco, CA, 2000.
- [131] B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th International Conference on Machine Learning*, pages 104–111, 2008.
- [132] V. Cherkassky and F. M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley, 2nd edition, 2007.
- [133] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.

- [134] A. Cotter, N. Srebro, and J. Keshet. A GPU-tailored Approach for Training Kernelized SVMs. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 805–813, San Diego, California, USA, 2011.
- [135] P. Domingos. The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.
- [136] P. Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning*, pages 231–238, 2000.
- [137] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.
- [138] S. Dudoit and M. J. van der Laan. Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. *Statistical Methodology*, 2(2):131–154, 2005.
- [139] B. Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in Statistics*, pages 569–593. Springer, 1992.
- [140] B. Efron and R. Tibshirani. Cross-validation and the Bootstrap: Estimating the Error Rate of a Prediction Rule. Technical report, Stanford University, 1995.

- [141] F. Esposito, D. Malerba, and G. Semeraro. A Comparative Analysis of Methods for Pruning Decision Trees. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.
- [142] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.
- [143] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, 1990.
- [144] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest—A Framework for Fast Decision Tree Construction of Large Datasets. *Data Mining and Knowledge Discovery*, 4(2/3):127–162, 2000.
- [145] S. Geisser. The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320–328, 1975.
- [146] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.
- [147] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009.
- [148] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

- [149] D. Heath, S. Kasif, and S. Salzberg. Induction of Oblique Decision Trees. In *Proc. of the 13th Intl. Joint Conf. on Artificial Intelligence*, pages 1002–1007, Chambery, France, August 1993.
- [150] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical Pattern Recognition: A Review. *IEEE Tran. Patt. Anal. and Mach. Intellig.*, 22(1):4–37, 2000.
- [151] D. Jensen and P. R. Cohen. Multiple Comparisons in Induction Algorithms. *Machine Learning*, 38(3):309–338, March 2000.
- [152] M. V. Joshi, G. Karypis, and V. Kumar. ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets. In *Proc. of 12th Intl. Parallel Processing Symp. (IPPS/SPDP)*, pages 573–579, Orlando, FL, April 1998.
- [153] G. V. Kass. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, 29:119–127, 1980.
- [154] B. Kim and D. Landgrebe. Hierarchical decision classifiers in high-dimensional and large class data. *IEEE Trans. on Geoscience and Remote Sensing*, 29(4):518–528, 1991.
- [155] R. Kohavi. A Study on Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence*, pages 1137–1145, Montreal, Canada, August 1995.

- [156] D. Krstajic, L. J. Buturovic, D. E. Leahy, and S. Thomas. Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of cheminformatics*, 6(1):1, 2014.
- [157] S. R. Kulkarni, G. Lugosi, and S. S. Venkatesh. Learning Pattern Classification—A Survey. *IEEE Tran. Inf. Theory*, 44(6):2178–2206, 1998.
- [158] V. Kumar, M. V. Joshi, E.-H. Han, P. N. Tan, and M. Steinbach. High Performance Data Mining. In *High Performance Computing for Computational Science (VECPAR 2002)*, pages 111–125. Springer, 2002.
- [159] G. Landeweerd, T. Timmers, E. Gersema, M. Bins, and M. Halic. Binary tree versus single level tree classification of white blood cells. *Pattern Recognition*, 16:571–577, 1983.
- [160] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A Fast Scalable Classifier for Data Mining. In *Proc. of the 5th Intl. Conf. on Extending Database Technology*, pages 18–32, Avignon, France, March 1996.
- [161] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–116, 1983.
- [162] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, 1994.

- [163] J. Mingers. Expert Systems—Rule Induction with Statistical Data. *J Operational Research Society*, 38:39–47, 1987.
- [164] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.
- [165] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [166] B. M. E. Moret. Decision Trees and Diagrams. *Computing Surveys*, 14(4):593–623, 1982.
- [167] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.
- [168] S. K. Murthy. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.
- [169] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *J of Artificial Intelligence Research*, 2:1–33, 1994.
- [170] T. Niblett. Constructing decision trees in noisy domains. In *Proc. of the 2nd European Working Session on Learning*, pages 67–78, Bled, Yugoslavia, May 1987.
- [171] T. Niblett and I. Bratko. Learning Decision Rules in Noisy Domains. In *Research and Development in Expert Systems III*, Cambridge, 1986.

Cambridge University Press.

- [172] I. Palit and C. K. Reddy. Scalable and parallel boosting with mapreduce. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1904–1916, 2012.
- [173] K. R. Pattipati and M. G. Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(4):872–887, 1990.
- [174] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [175] J. R. Quinlan. Discovering rules by induction from large collection of examples. In D. Michie, editor, *Expert Systems in the Micro Electronic Age*. Edinburgh University Press, Edinburgh, UK, 1979.
- [176] J. R. Quinlan. Simplifying Decision Trees. *Intl. J. Man-Machine Studies*, 27:221–234, 1987.
- [177] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann Publishers, San Mateo, CA, 1993.

- [178] J. R. Quinlan and R. L. Rivest. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation*, 80(3):227–248, 1989.
- [179] S. R. Safavian and D. Landgrebe. A Survey of Decision Tree Classifier Methodology. *IEEE Trans. Systems, Man and Cybernetics*, 22:660–674, May/June 1998.
- [180] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.
- [181] J. Schuermann and W. Doster. A decision-theoretic approach in hierarchical classifier design. *Pattern Recognition*, 17:359–369, 1984.
- [182] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2): 461–464, 1978.
- [183] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proc. of the 22nd VLDB Conf.*, pages 544–555, Bombay, India, September 1996.
- [184] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 111–147, 1974.
- [185] R. J. Tibshirani and R. Tibshirani. A bias correction for the minimum error rate in cross-validation. *The Annals of Applied Statistics*, pages 822–

829, 2009.

- [186] I. Tsamardinos, A. Rakhshani, and V. Lagani. Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. In *Hellenic Conference on Artificial Intelligence*, pages 1–14. Springer, 2014.
- [187] P. E. Utgoff and C. E. Brodley. An incremental method for finding multivariate splits for decision trees. In *Proc. of the 7th Intl. Conf. on Machine Learning*, pages 58–65, Austin, TX, June 1990.
- [188] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.
- [189] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.
- [190] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. Springer, 2015.
- [191] S. Varma and R. Simon. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7(1):1, 2006.
- [192] H. Wang and C. Zaniolo. CMP: A Fast Decision Tree Classifier Using Multivariate Predictions. In *Proc. of the 16th Intl. Conf. on Data Engineering*, pages 449–460, San Diego, CA, March 2000.

- [193] Q. R. Wang and C. Y. Suen. Large tree classifier with heuristic search and global training. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 9(1):91–102, 1987.
- [194] Y. Zhang and Y. Yang. Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187(1):95–112, 2015.

3.11 Exercises

1. Draw the full decision tree for the parity function of four Boolean attributes, A , B , C , and D . Is it possible to simplify the tree?
2. Consider the training examples shown in [Table 3.5](#) for a binary classification problem.

Table 3.5. Data set for Exercise 2.

Customer ID	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1

13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

- a. Compute the Gini index for the overall collection of training examples.
 - b. Compute the Gini index for the `Customer ID` attribute.
 - c. Compute the Gini index for the `Gender` attribute.
 - d. Compute the Gini index for the `Car Type` attribute using multiway split.
 - e. Compute the Gini index for the `Shirt Size` attribute using multiway split.
 - f. Which attribute is better, `Gender`, `Car Type`, or `Shirt Size`?
 - g. Explain why `Customer ID` should not be used as the attribute test condition even though it has the lowest Gini.
3. Consider the training examples shown in **Table 3.6** for a binary classification problem.

Table 3.6. Data set for Exercise 3.

Instance	a1	a2	a3	Target Class

1	T	T	1.0	+
2	T	T	6.0	+
3	T	F	5.0	-
4	F	F	4.0	+
5	F	T	7.0	-
6	F	T	3.0	-
7	F	F	8.0	-
8	T	F	7.0	+
9	F	T	5.0	-

- What is the entropy of this collection of training examples with respect to the class attribute?
- What are the information gains of a1 and a2 relative to these training examples?
- For a3, which is a continuous attribute, compute the information gain for every possible split.
- What is the best split (among a1, a2, and a3) according to the information gain?
- What is the best split (between a1 and a2) according to the misclassification error rate?
- What is the best split (between a1 and a2) according to the Gini index?

4. Show that the entropy of a node never increases after splitting it into smaller successor nodes.

5. Consider the following data set for a binary class problem.

A	B	Class Label
T	F	+
T	T	+
T	T	+
T	F	-
T	T	+
F	F	-
F	F	-
F	F	-
T	T	-
T	F	-

- Calculate the information gain when splitting on A and B. Which attribute would the decision tree induction algorithm choose?
- Calculate the gain in the Gini index when splitting on A and B. Which attribute would the decision tree induction algorithm choose?
- Figure 3.11** shows that entropy and the Gini index are both monotonically increasing on the range [0, 0.5] and they are both monotonically decreasing on the range [0.5, 1]. Is it possible that

information gain and the gain in the Gini index favor different attributes? Explain.

6. Consider splitting a parent node P into two child nodes, C1 and C2, using some attribute test condition. The composition of labeled training instances at every node is summarized in the Table below.

	P	C1	C2
Class 0	7	3	4
Class 1	3	0	3

- a. Calculate the Gini index and misclassification error rate of the parent node P .
- b. Calculate the weighted Gini index of the child nodes. Would you consider this attribute test condition if Gini is used as the impurity measure?
- c. Calculate the weighted misclassification rate of the child nodes. Would you consider this attribute test condition if misclassification rate is used as the impurity measure?

7. Consider the following set of training examples.

X	Y	Z	No. of Class C1 Examples	No. of Class C2 Examples
0	0	0	5	40
0	0	1	0	15
0	1	0	10	5
0	1	1	45	0

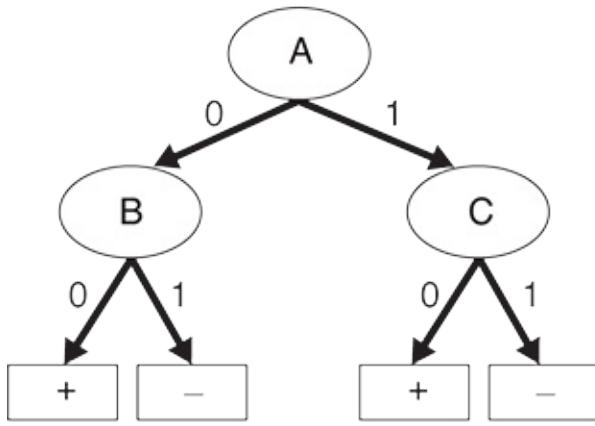
1	0	0	10	5
1	0	1	25	0
1	1	0	5	20
1	1	1	0	15

- a. Compute a two-level decision tree using the greedy approach described in this chapter. Use the classification error rate as the criterion for splitting. What is the overall error rate of the induced tree?
- b. Repeat part (a) using X as the first splitting attribute and then choose the best remaining attribute for splitting at each of the two successor nodes. What is the error rate of the induced tree?
- c. Compare the results of parts (a) and (b). Comment on the suitability of the greedy heuristic used for splitting attribute selection.
8. The following table summarizes a data set with three attributes A , B , C and two class labels +, -. Build a two-level decision tree.

A	B	C	Number of Instances	
			+	-
T	T	T	5	0
F	T	T	0	20
T	F	T	20	0
F	F	T	0	5
T	T	F	0	0

F	T	F	25	0
T	F	F	0	0
F	F	F	0	25

- a. According to the classification error rate, which attribute would be chosen as the first splitting attribute? For each attribute, show the contingency table and the gains in classification error rate.
 - b. Repeat for the two children of the root node.
 - c. How many instances are misclassified by the resulting decision tree?
 - d. Repeat parts (a), (b), and (c) using C as the splitting attribute.
 - e. Use the results in parts (c) and (d) to conclude about the greedy nature of the decision tree induction algorithm.
9. Consider the decision tree shown in [Figure 3.36](#).



Training:

Instance	A	B	C	Class
1	0	0	0	+
2	0	0	1	+
3	0	1	0	+
4	0	1	1	-
5	1	0	0	+
6	1	0	0	+
7	1	1	0	-
8	1	0	1	+
9	1	1	0	-
10	1	1	0	-

Validation:

Instance	A	B	C	Class
11	0	0	0	+
12	0	1	1	+
13	1	1	0	+
14	1	0	1	-
15	1	0	0	+

Figure 3.36.

Decision tree and data sets for Exercise 9.

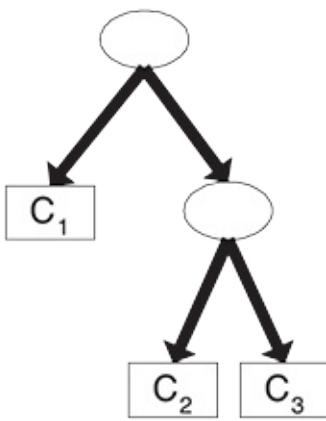
- Compute the generalization error rate of the tree using the optimistic approach.
 - Compute the generalization error rate of the tree using the pessimistic approach. (For simplicity, use the strategy of adding a factor of 0.5 to each leaf node.)
 - Compute the generalization error rate of the tree using the validation set shown above. This approach is known as **reduced error pruning**.
10. Consider the decision trees shown in [Figure 3.37](#). Assume they are generated from a data set that contains 16 binary attributes and 3 classes, C1, C2, and C3.

Compute the total description length of each decision tree according to the following formulation of the minimum description length principle.

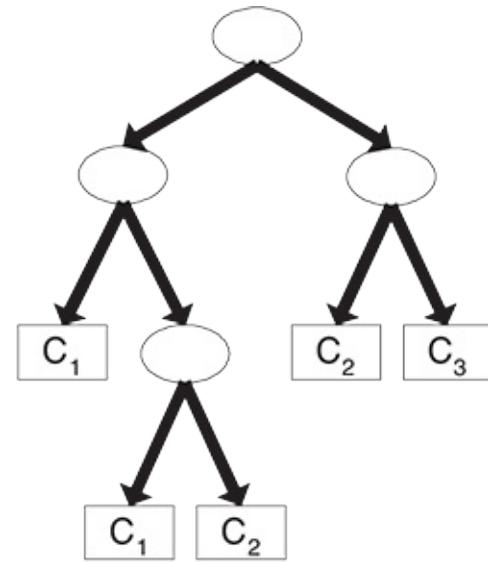
- The total description length of a tree is given by

$$\text{Cost(tree,data)} = \text{Cost(tree)} + \text{Cost(data|tree}).$$

- Each internal node of the tree is encoded by the ID of the splitting attribute. If there are m attributes, the cost of encoding each attribute is $\log_2 m$ bits.



(a) Decision tree with 7 errors



(b) Decision tree with 4 errors

Figure 3.37.

Decision trees for Exercise 10.

- Each leaf is encoded using the ID of the class it is associated with. If there are k classes, the cost of encoding a class is $\log_2 k$ bits.
- Cost(tree) is the cost of encoding all the nodes in the tree. To simplify the computation, you can assume that the total cost of the tree is obtained by adding up the costs of encoding each internal node and each leaf node.

- $\text{Cost}(\text{data}|\text{tree})$ is encoded using the classification errors the tree commits on the training set. Each error is encoded by $\log_2 n$ bits, where n is the total number of training instances.

Which decision tree is better, according to the MDL principle?

11. This exercise, inspired by the discussions in [155], highlights one of the known limitations of the leave-one-out model evaluation procedure. Let us consider a data set containing 50 positive and 50 negative instances, where the attributes are purely random and contain no information about the class labels. Hence, the generalization error rate of any classification model learned over this data is expected to be 0.5. Let us consider a classifier that assigns the majority class label of training instances (ties resolved by using the positive label as the default class) to any test instance, irrespective of its attribute values. We can call this approach as the *majority inducer* classifier. Determine the error rate of this classifier using the following methods.
- a. Leave-one-out.
 - b. 2-fold stratified cross-validation, where the proportion of class labels at every fold is kept same as that of the overall data.
 - c. From the results above, which method provides a more reliable evaluation of the classifier's generalization error rate?

12. Consider a labeled data set containing 100 data instances, which is randomly partitioned into two sets A and B , each containing 50 instances. We use A as the training set to learn two decision trees, T_{10} with 10 leaf nodes and T_{100} with 100 leaf nodes. The accuracies of the two decision trees on data sets A and B are shown in [Table 3.7](#).

Table 3.7. Comparing the test accuracy of decision trees T_{10} and T_{100} .

	Accuracy

Data Set	T10	T100
A	0.86	0.97
B	0.84	0.77

- a. Based on the accuracies shown in [Table 3.7](#), which classification model would you expect to have better performance on unseen instances?
- b. Now, you tested T10 and T100 on the entire data set (A+B) and found that the classification accuracy of T10 on data set (A+B) is 0.85, whereas the classification accuracy of T100 on the data set (A+B) is 0.87. Based on this new information and your observations from [Table 3.7](#), which classification model would you finally choose for classification?
13. Consider the following approach for testing whether a classifier A beats another classifier B. Let N be the size of a given dataset, p_A be the accuracy of classifier A, p_B be the accuracy of classifier B, and $p=(p_A+p_B)/2$ be the average accuracy for both classifiers. To test whether classifier A is significantly better than B, the following Z-statistic is used:

$$Z=p_A-p_B\sqrt{2p(1-p)}N.$$

Classifier A is assumed to be better than classifier B if $Z>1.96$.

[Table 3.8](#) compares the accuracies of three different classifiers, decision tree classifiers, naïve Bayes classifiers, and support vector machines, on various data sets. (The latter two classifiers are described in [Chapter 4](#).)

Summarize the performance of the classifiers given in [Table 3.8](#) using the following 3×3 table:

win-loss-draw	Decision tree	Naïve Bayes	Support vector machine
---------------	---------------	-------------	------------------------

Decision tree	0 - 0 - 23		
Naïve Bayes		0 - 0 - 23	
Support vector machine			0 - 0 - 23

Table 3.8. Comparing the accuracy of various classification methods.

Data Set	Size(N)	Decision Tree (%)	naïve Bayes (%)	Support vector machine (%)
Anneal	898	92.09	79.62	87.19
Australia	690	85.51	76.81	84.78
Auto	205	81.95	58.05	70.73
Breast	699	95.14	95.99	96.42
Cleve	303	76.24	83.50	84.49
Credit	690	85.80	77.54	85.07
Diabetes	768	72.40	75.91	76.82
German	1000	70.90	74.70	74.40
Glass	214	67.29	48.59	59.81
Heart	270	80.00	84.07	83.70
Hepatitis	155	81.94	83.23	87.10
Horse	368	85.33	78.80	82.61
Ionosphere	351	89.17	82.34	88.89

Iris	150	94.67	95.33	96.00
Labor	57	78.95	94.74	92.98
Led7	3200	73.34	73.16	73.56
Lymphography	148	77.03	83.11	86.49
Pima	768	74.35	76.04	76.95
Sonar	208	78.85	69.71	76.92
Tic-tac-toe	958	83.72	70.04	98.33
Vehicle	846	71.04	45.04	74.94
Wine	178	94.38	96.63	98.88
Zoo	101	93.07	93.07	96.04

Each cell in the table contains the number of wins, losses, and draws when comparing the classifier in a given row to the classifier in a given column.

14. Let X be a binomial random variable with mean Np and variance $Np(1-p)$. Show that the ratio X/N also has a binomial distribution with mean p and variance $p(1-p)N$.

4 Classification: Alternative Techniques

The previous chapter introduced the classification problem and presented a technique known as the decision tree classifier. Issues such as model overfitting and model evaluation were also discussed. This chapter presents alternative techniques for building classification models—from simple techniques such as rule-based and nearest neighbor classifiers to more sophisticated techniques such as artificial neural networks, deep learning, support vector machines, and ensemble methods. Other practical issues such as the class imbalance and multiclass problems are also discussed at the end of the chapter.

4.1 Types of Classifiers

Before presenting specific techniques, we first categorize the different types of classifiers available. One way to distinguish classifiers is by considering the characteristics of their output.

Binary versus Multiclass

Binary classifiers assign each data instance to one of two possible labels, typically denoted as $+1$ and -1 . The positive class usually refers to the category we are more interested in predicting correctly compared to the negative class (e.g., the `spam` category in email classification problems). If there are more than two possible labels available, then the technique is known as a multiclass classifier. As some classifiers were designed for binary classes only, they must be adapted to deal with multiclass problems. Techniques for transforming binary classifiers into multiclass classifiers are described in [Section 4.12](#).

Deterministic versus Probabilistic

A deterministic classifier produces a discrete-valued label to each data instance it classifies whereas a probabilistic classifier assigns a continuous score between 0 and 1 to indicate how likely it is that an instance belongs to a particular class, where the probability scores for all the classes sum up to 1. Some examples of probabilistic classifiers include the naïve Bayes classifier, Bayesian networks, and logistic regression. Probabilistic classifiers provide additional information about the confidence in assigning an instance to a class than deterministic classifiers. A data instance is typically assigned to the class

with the highest probability score, except when the cost of misclassifying the class with lower probability is significantly higher. We will discuss the topic of cost-sensitive classification with probabilistic outputs in [Section 4.11.2](#).

Another way to distinguish the different types of classifiers is based on their technique for discriminating instances from different classes.

Linear versus Nonlinear

A linear classifier uses a linear separating hyperplane to discriminate instances from different classes whereas a nonlinear classifier enables the construction of more complex, nonlinear decision surfaces. We illustrate an example of a linear classifier (perceptron) and its nonlinear counterpart (multi-layer neural network) in [Section 4.7](#). Although the linearity assumption makes the model less flexible in terms of fitting complex data, linear classifiers are thus less susceptible to model overfitting compared to nonlinear classifiers. Furthermore, one can transform the original set of attributes, $x = (x_1, x_2, \dots, x_d)$, into a more complex feature set, e.g., $\Phi(x) = (x_1, x_2, x_1x_2, x_{12}, x_{22}, \dots)$, before applying the linear classifier. Such feature transformation allows the linear classifier to fit data sets with nonlinear decision surfaces (see [Section 4.9.4](#)).

Global versus Local

A global classifier fits a single model to the entire data set. Unless the model is highly nonlinear, this one-size-fits-all strategy may not be effective when the relationship between the attributes and the class labels varies over the input space. In contrast, a local classifier partitions the input space into smaller regions and fits a distinct model to training instances in each region. The k -nearest neighbor classifier (see [Section 4.3](#)) is a classic example of local classifiers. While local classifiers are more flexible in terms of fitting complex

decision boundaries, they are also more susceptible to the model overfitting problem, especially when the local regions contain few training examples.

Generative versus Discriminative

Given a data instance \underline{x} , the primary objective of any classifier is to predict the class label, y , of the data instance. However, apart from predicting the class label, we may also be interested in describing the underlying mechanism that *generates* the instances belonging to every class label. For example, in the process of classifying spam email messages, it may be useful to understand the typical characteristics of email messages that are labeled as spam, e.g., specific usage of keywords in the subject or the body of the email. Classifiers that learn a generative model of every class in the process of predicting class labels are known as generative classifiers. Some examples of generative classifiers include the naïve Bayes classifier and Bayesian networks. In contrast, discriminative classifiers directly predict the class labels without explicitly describing the distribution of every class label. They solve a simpler problem than generative models since they do not have the onus of deriving insights about the generative mechanism of data instances. They are thus sometimes preferred over generative models, especially when it is not crucial to obtain information about the properties of every class. Some examples of discriminative classifiers include decision trees, rule-based classifier, nearest neighbor classifier, artificial neural networks, and support vector machines.

4.2 Rule-Based Classifier

A rule-based classifier uses a collection of “if ...then...” rules (also known as a **rule set**) to classify data instances. [Table 4.1](#) shows an example of a rule set generated for the vertebrate classification problem described in the previous chapter. Each classification rule in the rule set can be expressed in the following way:

$$r_i: (\text{Condition}_i) \rightarrow y_i. \quad (4.1)$$

The left-hand side of the rule is called the **rule antecedent** or **precondition**. It contains a conjunction of attribute test conditions:

$$\text{Condition}_i = (A_1 \text{ op } v_1) \wedge (A_2 \text{ op } v_2) \dots (A_k \text{ op } v_k), \quad (4.2)$$

where (A_j, v_j) is an attribute-value pair and op is a comparison operator chosen from the set $\{=, \neq, <, >, \leq, \geq\}$. Each attribute test $(A_j \text{ op } v_j)$ is also known as a conjunct. The right-hand side of the rule is called the **rule consequent**, which contains the predicted class y_i .

A rule r covers a data instance x if the precondition of r matches the attributes of x . r is also said to be fired or triggered whenever it covers a given instance. For an illustration, consider the rule r_1 given in [Table 4.1](#) and the following attributes for two vertebrates: hawk and grizzly bear.

Table 4.1. Example of a rule set for the vertebrate classification problem.

r1:(Gives Birth=no) ∧ (Aerial Creature=yes) → Birds	r2:(Gives Birth=no) ∧ (Aquatic Creature=yes) → Fishes	r3:(Gives Birth=yes) ∧ (Body Temperature=warm-blooded) → Mammals	r4:(Gives Birth=no) ∧ (Aerial Creature=no) → Reptiles	r5:(Aquatic Creature=semi) → Amphibians
---	---	--	---	---

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates
hawk	warm-blooded	feather	no	no	yes	yes	no
grizzly bear	warm-blooded	fur	yes	no	no	yes	yes

r1 covers the first vertebrate because its precondition is satisfied by the hawk's attributes. The rule does not cover the second vertebrate because grizzly bears give birth to their young and cannot fly, thus violating the precondition of r1.

The quality of a classification rule can be evaluated using measures such as coverage and accuracy. Given a data set D and a classification rule $r : A \rightarrow y$, the coverage of the rule is the fraction of instances in D that trigger the rule r . On the other hand, its accuracy or confidence factor is the fraction of instances triggered by r whose class labels are equal to y . The formal definitions of these measures are

$$\text{Coverage}(r) = |A| / |D| \quad \text{Coverage}(r) = |A \cap y| / |A|, \quad (4.3)$$

where $|A|$ is the number of instances that satisfy the rule antecedent, $|A \cap y|$ is the number of instances that satisfy both the antecedent and consequent, and $|D|$ is the total number of instances.

Example 4.1.

Consider the data set shown in [Table 4.2](#). The rule

$(\text{Gives Birth}=\text{yes}) \wedge (\text{Body Temperature}=\text{warm-blooded}) \rightarrow \text{Mammals}$

has a coverage of 33% since five of the fifteen instances support the rule antecedent. The rule accuracy is 100% because all five vertebrates covered by the rule are mammals.

Table 4.2. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	Mammals
python	cold-blooded	scales	no	no	no	no	yes	Reptiles
salmon	cold-blooded	scales	no	yes	no	no	no	Fishes
whale	warm-blooded	hair	yes	yes	no	no	no	Mammals
frog	cold-blooded	none	no	semi	no	yes	yes	Amphibians
komodo dragon	cold-blooded	scales	no	no	no	yes	no	Reptiles
bat	warm-blooded	hair	yes	no	yes	yes	yes	Mammals
pigeon	warm-blooded	feathers	no	no	yes	yes	no	Birds
cat	warm-blooded	fur	yes	no	no	yes	no	Mammals
guppy	cold-blooded	scales	yes	yes	no	no	no	Fishes
alligator	cold-blooded	scales	no	semi	no	yes	no	Reptiles
penguin	warm-blooded	feathers	no	semi	no	yes	no	Birds
porcupine	warm-blooded	quills	yes	no	no	yes	yes	Mammals

eel	cold-blooded	scales	no	yes	no	no	no	Fishes
salamander	cold-blooded	none	no	semi	no	yes	yes	Amphibians

4.2.1 How a Rule-Based Classifier Works

A rule-based classifier classifies a test instance based on the rule triggered by the instance. To illustrate how a rule-based classifier works, consider the rule set shown in [Table 4.1](#) and the following vertebrates:

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates
lemur	warm-blooded	fur	yes	no	no	yes	yes
turtle	cold-blooded	scales	no	semi	no	yes	no
dogfish shark	cold-blooded	scales	yes	yes	no	no	no

- The first vertebrate, which is a lemur, is warm-blooded and gives birth to its young. It triggers the rule r3, and thus, is classified as a mammal.
- The second vertebrate, which is a turtle, triggers the rules r4 and r5. Since the classes predicted by the rules are contradictory (reptiles versus amphibians), their conflicting classes must be resolved.
- None of the rules are applicable to a dogfish shark. In this case, we need to determine what class to assign to such a test instance.

4.2.2 Properties of a Rule Set

The rule set generated by a rule-based classifier can be characterized by the following two properties.

Definition 4.1 (Mutually Exclusive Rule Set).

The rules in a rule set R are mutually exclusive if no two rules in R are triggered by the same instance. This property ensures that every instance is covered by at most one rule in R .

Definition 4.2 (Exhaustive Rule Set).

A rule set R has exhaustive coverage if there is a rule for each combination of attribute values. This property ensures that every instance is covered by at least one rule in R .

Table 4.3. Example of a mutually exclusive and exhaustive rule set.

r1: (Body Temperature=cold-blooded)→Non-mammals r2: (Body Temperature=warm-blooded) ∧ (Gives Birth=yes)→Mammals r3: (Body Temperature=warm-blooded) ∧ (Gives Birth=no)→Non-mammals
--

blooded) \wedge (Gives Birth=no) \rightarrow Non-mammals

Together, these two properties ensure that every instance is covered by exactly one rule. An example of a mutually exclusive and exhaustive rule set is shown in [Table 4.3](#). Unfortunately, many rule-based classifiers, including the one shown in [Table 4.1](#), do not have such properties. If the rule set is not exhaustive, then a default rule, $rd: () \rightarrow yd$, must be added to cover the remaining cases. A default rule has an empty antecedent and is triggered when all other rules have failed. yd is known as the default class and is typically assigned to the majority class of training instances not covered by the existing rules. If the rule set is not mutually exclusive, then an instance can be covered by more than one rule, some of which may predict conflicting classes.

Definition 4.3 (Ordered Rule Set).

The rules in an ordered rule set R are ranked in decreasing order of their priority. An ordered rule set is also known as a **decision list**.

The rank of a rule can be defined in many ways, e.g., based on its accuracy or total description length. When a test instance is presented, it will be classified by the highest-ranked rule that covers the instance. This avoids the problem of having conflicting classes predicted by multiple classification rules if the rule set is not mutually exclusive.

An alternative way to handle a non-mutually exclusive rule set without ordering the rules is to consider the consequent of each rule triggered by a test instance as a vote for a particular class. The votes are then tallied to determine the class label of the test instance. The instance is usually assigned to the class that receives the highest number of votes. The vote may also be weighted by the rule's accuracy. Using unordered rules to build a rule-based classifier has both advantages and disadvantages. Unordered rules are less susceptible to errors caused by the wrong rule being selected to classify a test instance unlike classifiers based on ordered rules, which are sensitive to the choice of rule-ordering criteria. Model building is also less expensive because the rules do not need to be kept in sorted order. Nevertheless, classifying a test instance can be quite expensive because the attributes of the test instance must be compared against the precondition of every rule in the rule set.

In the next two sections, we present techniques for extracting an ordered rule set from data. A rule-based classifier can be constructed using (1) direct methods, which extract classification rules directly from data, and (2) indirect methods, which extract classification rules from more complex classification models, such as decision trees and neural networks. Detailed discussions of these methods are presented in [Sections 4.2.3](#) and [4.2.4](#), respectively.

4.2.3 Direct Methods for Rule Extraction

To illustrate the direct method, we consider a widely-used rule induction algorithm called RIPPER. This algorithm scales almost linearly with the number of training instances and is particularly suited for building models from

data sets with imbalanced class distributions. RIPPER also works well with noisy data because it uses a validation set to prevent model overfitting.

RIPPER uses the **sequential covering** algorithm to extract rules directly from data. Rules are grown in a greedy fashion one class at a time. For binary class problems, RIPPER chooses the majority class as its default class and learns the rules to detect instances from the minority class. For multiclass problems, the classes are ordered according to their prevalence in the training set. Let (y_1, y_2, \dots, y_c) be the ordered list of classes, where y_1 is the least prevalent class and y_c is the most prevalent class. All training instances that belong to y_1 are initially labeled as positive examples, while those that belong to other classes are labeled as negative examples. The sequential covering algorithm learns a set of rules to discriminate the positive from negative examples. Next, all training instances from y_2 are labeled as positive, while those from classes y_3, y_4, \dots, y_c are labeled as negative. The sequential covering algorithm would learn the next set of rules to distinguish y_2 from other remaining classes. This process is repeated until we are left with only one class, y_c , which is designated as the default class.

Example 4.1. Sequential covering algorithm.

```
1: Let  $E$  be the training instances and  $A$  be the set of attribute-value pairs,  $\{(A_j, v_j)\}$ .  
2: Let  $Y_o$  be an ordered set of classes  $\{y_1, y_2, \dots, y_k\}$ .  
3: Let  $R = \{\}$  be the initial rule list.  
4: for each class  $y \in Y_o - \{y_k\}$  do  
5:   while stopping condition is not met do  
6:      $r \leftarrow \text{Learn-One-Rule}(E, A, y)$ .  
7:     Remove training instances from  $E$  that are covered by  $r$ .  
8:   Add  $r$  to the bottom of the rule list:  $R \leftarrow R \vee r$ .
```

```

9: end while
10: end for
11: Insert the default rule, {} →  $y_k$ , to the bottom of the rule list
R.

```

A summary of the sequential covering algorithm is shown in [Algorithm 4.1](#). The algorithm starts with an empty decision list, R , and extracts rules for each class based on the ordering specified by the class prevalence. It iteratively extracts the rules for a given class y using the Learn-One-Rule function. Once such a rule is found, all the training instances covered by the rule are eliminated. The new rule is added to the bottom of the decision list R . This procedure is repeated until the stopping criterion is met. The algorithm then proceeds to generate rules for the next class.

[Figure 4.1](#) demonstrates how the sequential covering algorithm works for a data set that contains a collection of positive and negative examples. The rule R_1 , whose coverage is shown in [Figure 4.1\(b\)](#), is extracted first because it covers the largest fraction of positive examples. All the training instances covered by R_1 are subsequently removed and the algorithm proceeds to look for the next best rule, which is R_2 .

Learn-One-Rule Function

Finding an optimal rule is computationally expensive due to the exponential search space to explore. The Learn-One-Rule function addresses this problem by growing the rules in a greedy fashion. It generates an initial rule $r: \{\} \rightarrow +$, where the left-hand side is an empty set and the right-hand side corresponds to the positive class. It then refines the rule until a certain stopping criterion is met. The accuracy of the initial rule may be poor because some of the training instances covered by the rule belong to the negative

class. A new conjunct must be added to the rule antecedent to improve its accuracy.

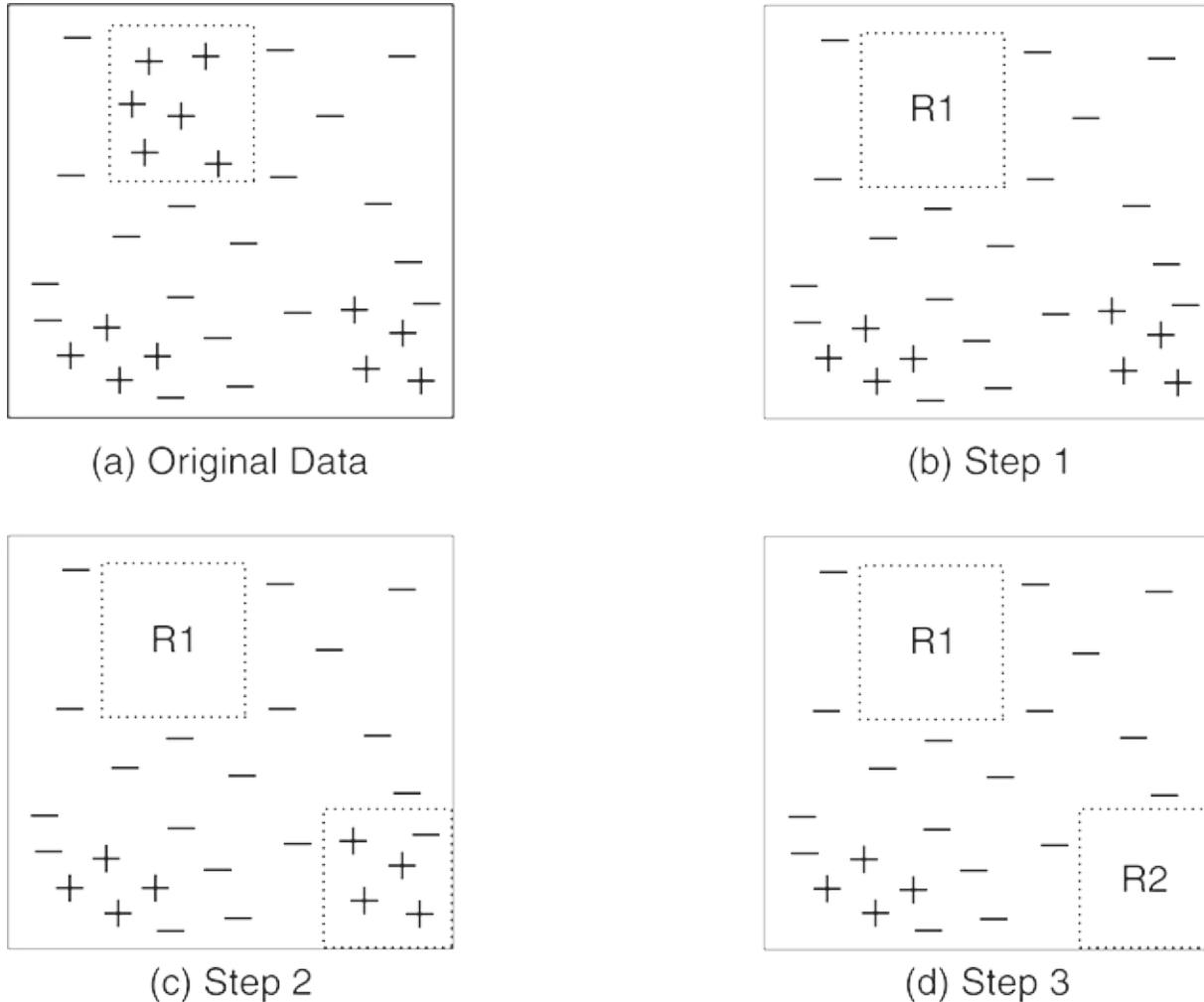


Figure 4.1.

An example of the sequential covering algorithm.

RIPPER uses the FOIL's information gain measure to choose the best conjunct to be added into the rule antecedent. The measure takes into consideration both the gain in accuracy and support of a candidate rule, where support is defined as the number of positive examples covered by the rule. For example, suppose the rule $r: A \rightarrow +$ initially covers p_0 positive examples and n_0 negative examples. After adding a new conjunct B , the extended rule $r': A \wedge B \rightarrow +$ covers p_1 positive examples and n_1 negative

examples. The FOIL's information gain of the extended rule is computed as follows:

$$\text{FOIL's information gain} = p_1 \times (\log_2 p_1 p_1 + n_1 - \log_2 p_0 p_0 + n_0). \quad (4.4)$$

RIPPER chooses the conjunct with highest FOIL's information gain to extend the rule, as illustrated in the next example.

Example 4.2. [Foil's Information Gain]

Consider the training set for the vertebrate classification problem shown in [Table 4.2](#). Suppose the target class for the Learn-One-Rule function is mammals. Initially, the antecedent of the rule $\{\} \rightarrow \text{Mammals}$ covers 5 positive and 10 negative examples. Thus, the accuracy of the rule is only 0.333. Next, consider the following three candidate conjuncts to be added to the left-hand side of the rule: Skin cover=hair, Body temperature=warm, and Has legs>No. The number of positive and negative examples covered by the rule after adding each conjunct along with their respective accuracy and FOIL's information gain are shown in the following table.

Candidate rule	p1	n1	Accuracy	Info Gain
{Skin Cover=hair} \rightarrow mammals	3	0	1.000	4.755
{Body temperature=warm} \rightarrow mammals	5	1	0.714	5.498
{Has legs>No} \rightarrow mammals	2	4	0.200	-0.737

Although Skin cover=hair has the highest accuracy among the three candidates, the conjunct Body temperature=warm has the highest FOIL's information gain. Thus, it is chosen to extend the rule (see [Figure 4.2](#)).

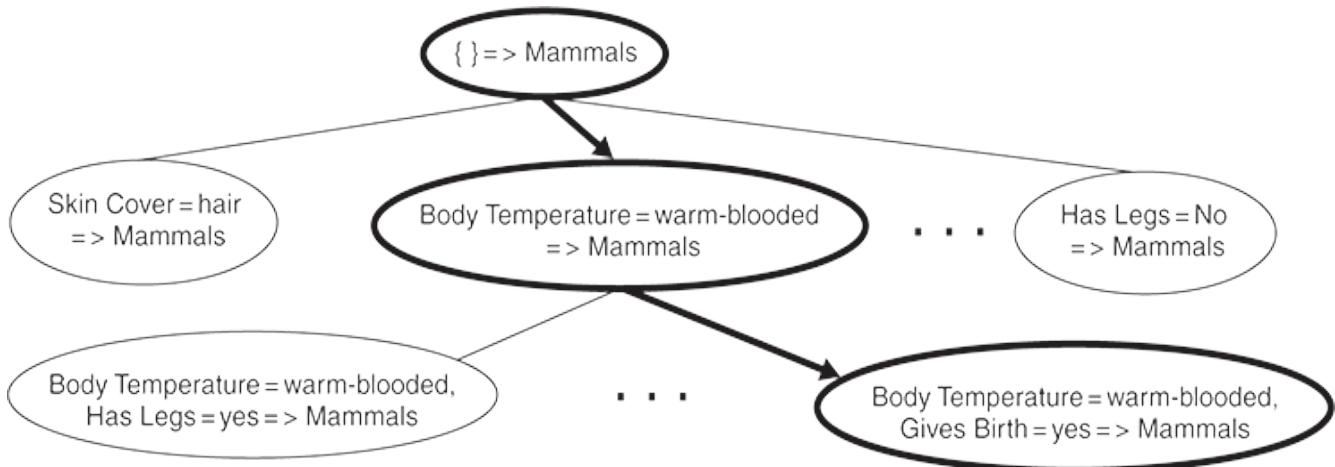
This process continues until adding new conjuncts no longer improves the information gain measure.

Rule Pruning

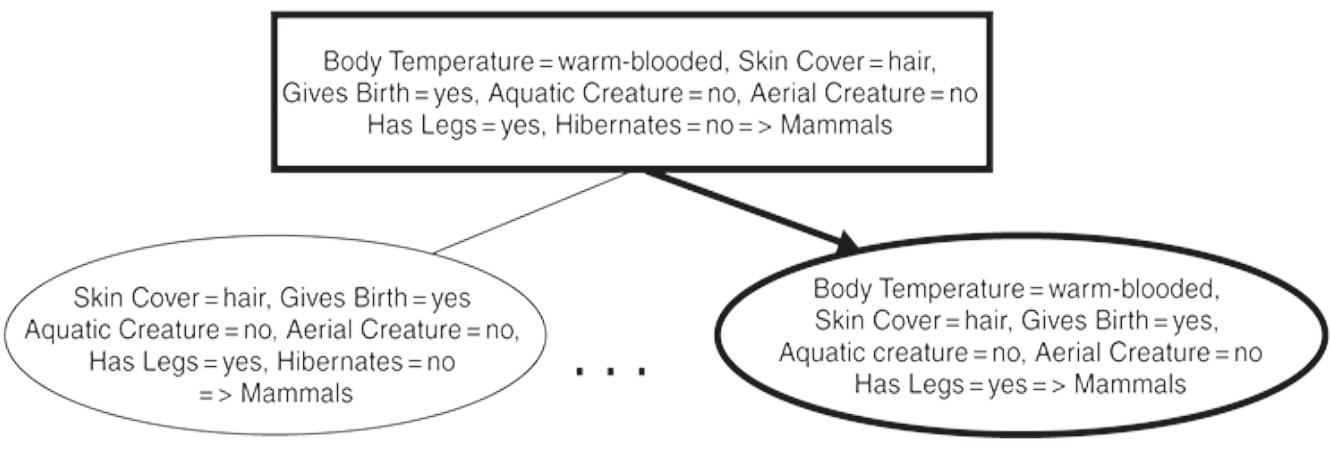
The rules generated by the Learn-One-Rule function can be pruned to improve their generalization errors. RIPPER prunes the rules based on their performance on the validation set. The following metric is computed to determine whether pruning is needed: $(p-n)/(p+n)$, where $p(n)$ is the number of positive (negative) examples in the validation set covered by the rule. This metric is monotonically related to the rule's accuracy on the validation set. If the metric improves after pruning, then the conjunct is removed. Pruning is done starting from the last conjunct added to the rule. For example, given a rule $ABCD \rightarrow y$, RIPPER checks whether D should be pruned first, followed by CD , BCD , etc. While the original rule covers only positive examples, the pruned rule may cover some of the negative examples in the training set.

Building the Rule Set

After generating a rule, all the positive and negative examples covered by the rule are eliminated. The rule is then added into the rule set as long as it does not violate the stopping condition, which is based on the minimum description length principle. If the new rule increases the total description length of the rule set by at least d bits, then RIPPER stops adding rules into its rule set (by default, d is chosen to be 64 bits). Another stopping condition used by RIPPER is that the error rate of the rule on the validation set must not exceed 50%.



(a) General-to-specific



(b) Specific-to-general

Figure 4.2.

General-to-specific and specific-to-general rule-growing strategies.

RIPPER also performs additional optimization steps to determine whether some of the existing rules in the rule set can be replaced by better alternative rules. Readers who are interested in the details of the optimization method may refer to the reference cited at the end of this chapter.

Instance Elimination

After a rule is extracted, RIPPER eliminates the positive and negative examples covered by the rule. The rationale for doing this is illustrated in the next example.

Figure 4.3 shows three possible rules, R_1 , R_2 , and R_3 , extracted from a training set that contains 29 positive examples and 21 negative examples. The accuracies of R_1 , R_2 , and R_3 are 12/15 (80%), 7/10 (70%), and 8/12 (66.7%), respectively. R_1 is generated first because it has the highest accuracy. After generating R_1 , the algorithm must remove the examples covered by the rule so that the next rule generated by the algorithm is different than R_1 . The question is, should it remove the positive examples only, negative examples only, or both? To answer this, suppose the algorithm must choose between generating R_2 or R_3 after R_1 . Even though R_2 has a higher accuracy than R_3 (70% versus 66.7%), observe that the region covered by R_2 is disjoint from R_1 , while the region covered by R_3 overlaps with R_1 . As a result, R_1 and R_3 together cover 18 positive and 5 negative examples (resulting in an overall accuracy of 78.3%), whereas R_1 and R_2 together cover 19 positive and 6 negative examples (resulting in a lower overall accuracy of 76%). If the positive examples covered by R_1 are not removed, then we may overestimate the effective accuracy of R_3 . If the negative examples covered by R_1 are not removed, then we may underestimate the accuracy of R_3 . In the latter case, we might end up preferring R_2 over R_3 even though half of the false positive errors committed by R_3 have already been accounted for by the preceding rule, R_1 . This example shows that the effective accuracy after adding R_2 or R_3 to the rule set becomes evident only when both positive and negative examples covered by R_1 are removed.

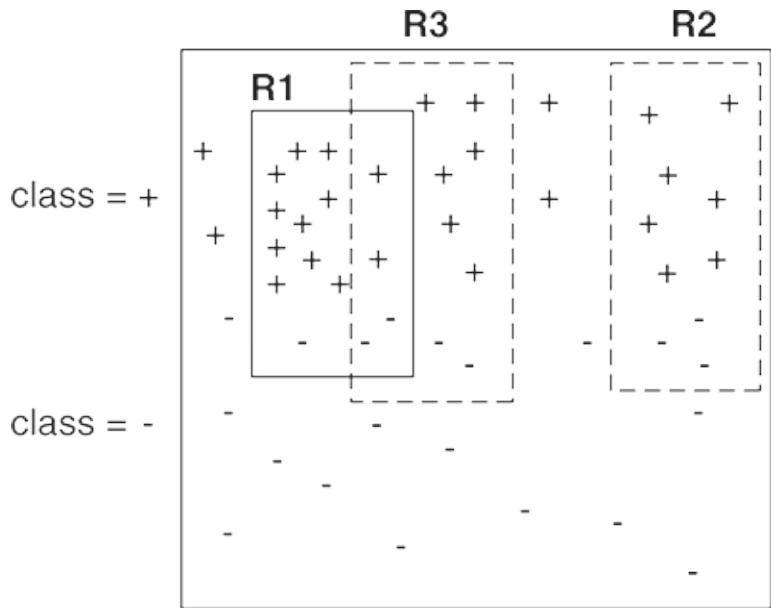


Figure 4.3.

Elimination of training instances by the sequential covering algorithm. $R1$, $R2$, and $R3$ represent regions covered by three different rules.

4.2.4 Indirect Methods for Rule Extraction

This section presents a method for generating a rule set from a decision tree. In principle, every path from the root node to the leaf node of a decision tree can be expressed as a classification rule. The test conditions encountered along the path form the conjuncts of the rule antecedent, while the class label at the leaf node is assigned to the rule consequent. [Figure 4.4](#) shows an example of a rule set generated from a decision tree. Notice that the rule set is exhaustive and contains mutually exclusive rules. However, some of the rules can be simplified as shown in the next example.

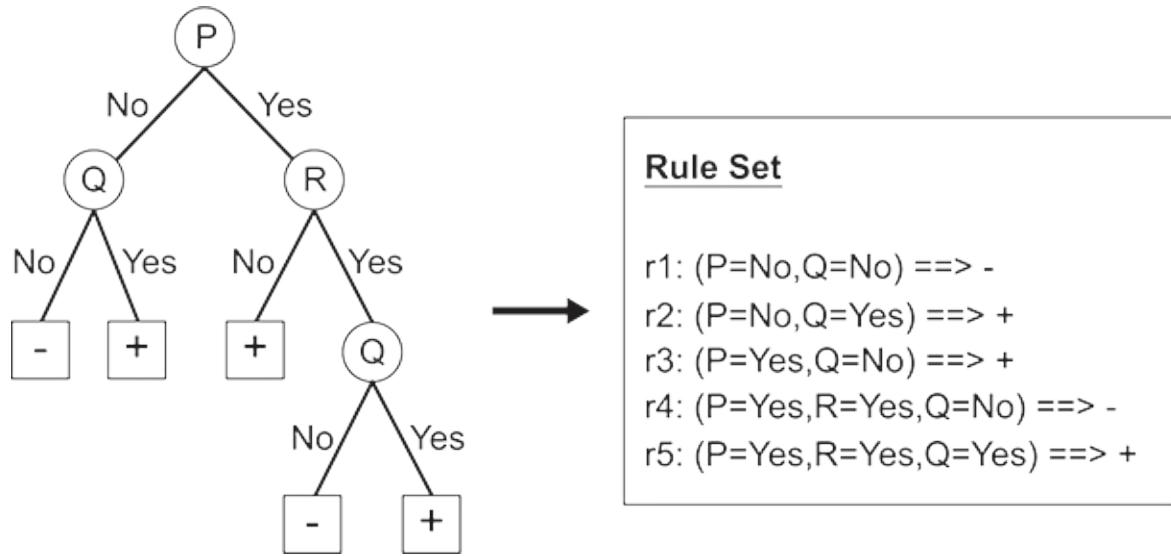


Figure 4.4.

Converting a decision tree into classification rules.

Example 4.3.

Consider the following three rules from [Figure 4.4](#):

$$\begin{aligned} r2: (P=\text{No}) \wedge (Q=\text{Yes}) &\rightarrow + \\ r3: (P=\text{Yes}) \wedge (R=\text{No}) &\rightarrow + \\ r5: (P=\text{Yes}) \wedge (R=\text{Yes}) \wedge (Q=\text{Yes}) &\rightarrow +. \end{aligned}$$

Observe that the rule set always predicts a positive class when the value of Q is Yes. Therefore, we may simplify the rules as follows:

$$r2': (Q=\text{Yes}) \rightarrow +$$

r3 is retained to cover the remaining instances of the positive class. Although the rules obtained after simplification are no longer mutually exclusive, they are less complex and are easier to interpret.

In the following, we describe an approach used by the C4.5rules algorithm to generate a rule set from a decision tree. [Figure 4.5](#) shows the decision tree

and resulting classification rules obtained for the data set given in [Table 4.2](#).

Rule Generation

Classification rules are extracted for every path from the root to one of the leaf nodes in the decision tree. Given a classification rule $r:A \rightarrow y$, we consider a simplified rule, $r':A' \rightarrow y$ where A' is obtained by removing one of the conjuncts in A . The simplified rule with the lowest pessimistic error rate is retained provided its error rate is less than that of the original rule. The rule-pruning step is repeated until the pessimistic error of the rule cannot be improved further. Because some of the rules may become identical after pruning, the duplicate rules are discarded.

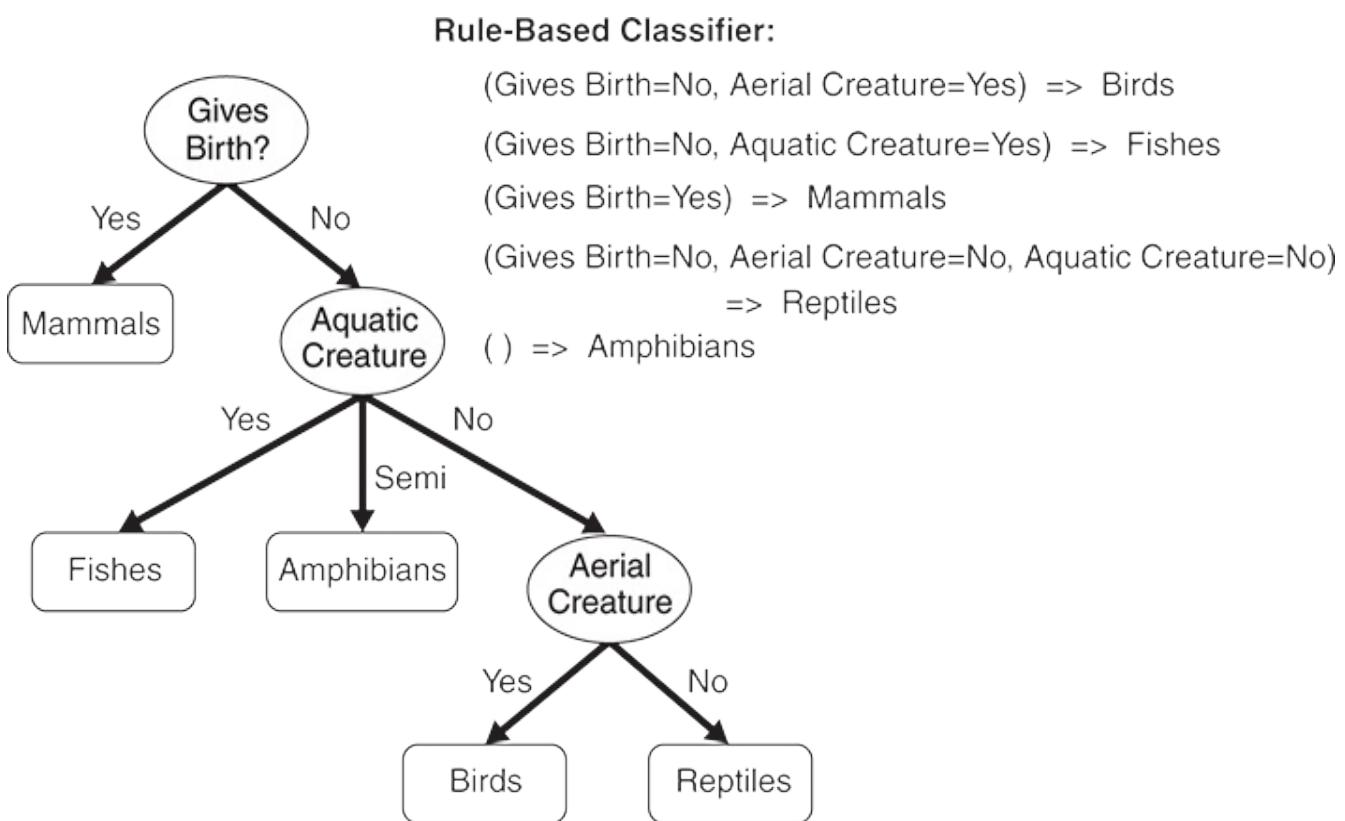


Figure 4.5.

Classification rules extracted from a decision tree for the vertebrate classification problem.

Rule Ordering

After generating the rule set, C4.5rules uses the class-based ordering scheme to order the extracted rules. Rules that predict the same class are grouped together into the same subset. The total description length for each subset is computed, and the classes are arranged in increasing order of their total description length. The class that has the smallest description length is given the highest priority because it is expected to contain the best set of rules. The total description length for a class is given by $L_{\text{exception}} + g \times L_{\text{model}}$, where $L_{\text{exception}}$ is the number of bits needed to encode the misclassified examples, L_{model} is the number of bits needed to encode the model, and g is a tuning parameter whose default value is 0.5. The tuning parameter depends on the number of redundant attributes present in the model. The value of the tuning parameter is small if the model contains many redundant attributes.

4.2.5 Characteristics of Rule-Based Classifiers

1. Rule-based classifiers have very similar characteristics as decision trees. The expressiveness of a rule set is almost equivalent to that of a decision tree because a decision tree can be represented by a set of mutually exclusive and exhaustive rules. Both rule-based and decision tree classifiers create rectilinear partitions of the attribute space and assign a class to each partition. However, a rule-based classifier can

allow multiple rules to be triggered for a given instance, thus enabling the learning of more complex models than decision trees.

2. Like decision trees, rule-based classifiers can handle varying types of categorical and continuous attributes and can easily work in multiclass classification scenarios. Rule-based classifiers are generally used to produce descriptive models that are easier to interpret but give comparable performance to the decision tree classifier.
3. Rule-based classifiers can easily handle the presence of redundant attributes that are highly correlated with one other. This is because once an attribute has been used as a conjunct in a rule antecedent, the remaining redundant attributes would show little to no FOIL's information gain and would thus be ignored.
4. Since irrelevant attributes show poor information gain, rule-based classifiers can avoid selecting irrelevant attributes if there are other relevant attributes that show better information gain. However, if the problem is complex and there are interacting attributes that can collectively distinguish between the classes but individually show poor information gain, it is likely for an irrelevant attribute to be accidentally favored over a relevant attribute just by random chance. Hence, rule-based classifiers can show poor performance in the presence of interacting attributes, when the number of irrelevant attributes is large.
5. The class-based ordering strategy adopted by RIPPER, which emphasizes giving higher priority to rare classes, is well suited for handling training data sets with imbalanced class distributions.
6. Rule-based classifiers are not well-suited for handling missing values in the test set. This is because the position of rules in a rule set follows a certain ordering strategy and even if a test instance is covered by multiple rules, they can assign different class labels depending on their position in the rule set. Hence, if a certain rule involves an attribute that is missing in a test instance, it is difficult to ignore the rule and proceed

to the subsequent rules in the rule set, as such a strategy can result in incorrect class assignments.

4.3 Nearest Neighbor Classifiers

The classification framework shown in [Figure 3.3](#) involves a two-step process:

- (1) an inductive step for constructing a classification model from data, and
- (2) a deductive step for applying the model to test examples. Decision tree and rule-based classifiers are examples of **eager learners** because they are designed to learn a model that maps the input attributes to the class label as soon as the training data becomes available. An opposite strategy would be to delay the process of modeling the training data until it is needed to classify the test instances. Techniques that employ this strategy are known as **lazy learners**. An example of a lazy learner is the **Rote classifier**, which memorizes the entire training data and performs classification only if the attributes of a test instance match one of the training examples exactly. An obvious drawback of this approach is that some test instances may not be classified because they do not match any training example.

One way to make this approach more flexible is to find all the training examples that are relatively similar to the attributes of the test instances. These examples, which are known as **nearest neighbors**, can be used to determine the class label of the test instance. The justification for using nearest neighbors is best exemplified by the following saying: *“If it walks like a duck, quacks like a duck, and looks like a duck, then it’s probably a duck.”* A nearest neighbor classifier represents each example as a data point in a d -dimensional space, where d is the number of attributes. Given a test instance, we compute its proximity to the training instances according to one of the proximity measures described in [Section 2.4](#) on page 71. The k -nearest

neighbors of a given test instance z refer to the k training examples that are closest to z .

Figure 4.6 illustrates the 1-, 2-, and 3-nearest neighbors of a test instance located at the center of each circle. The instance is classified based on the class labels of its neighbors. In the case where the neighbors have more than one label, the test instance is assigned to the majority class of its nearest neighbors. In **Figure 4.6(a)**, the 1-nearest neighbor of the instance is a negative example. Therefore the instance is assigned to the negative class. If the number of nearest neighbors is three, as shown in **Figure 4.6(c)**, then the neighborhood contains two positive examples and one negative example. Using the majority voting scheme, the instance is assigned to the positive class. In the case where there is a tie between the classes (see **Figure 4.6(b)**), we may randomly choose one of them to classify the data point.

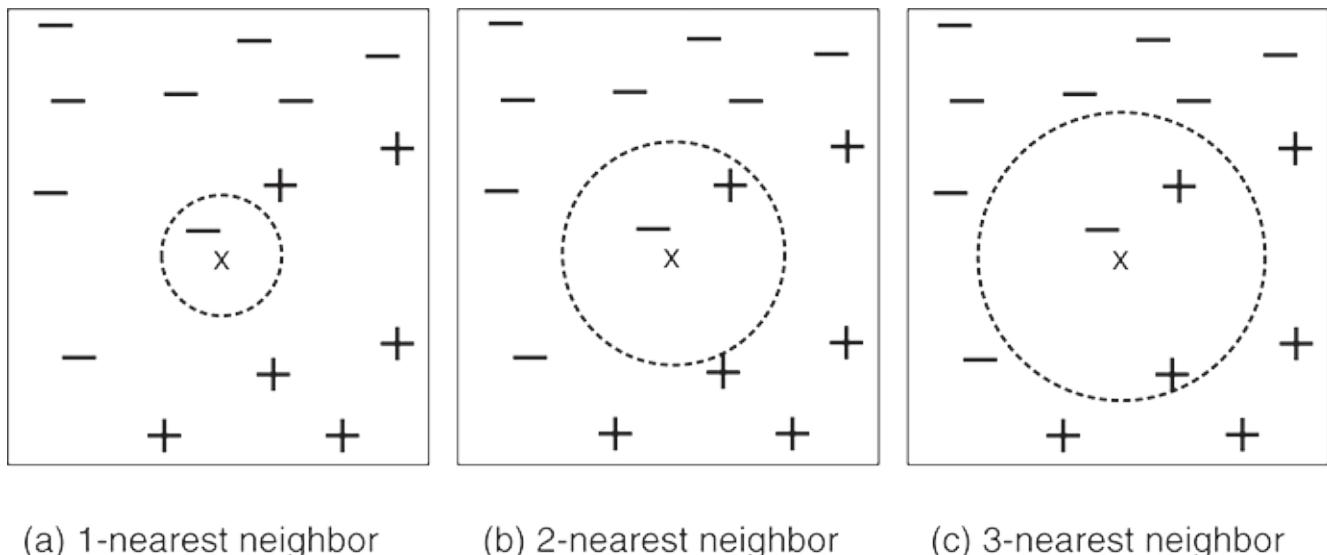


Figure 4.6.
The 1-, 2-, and 3-nearest neighbors of an instance.

The preceding discussion underscores the importance of choosing the right value for k . If k is too small, then the nearest neighbor classifier may be susceptible to overfitting due to noise, i.e., mislabeled examples in the training

data. On the other hand, if k is too large, the nearest neighbor classifier may misclassify the test instance because its list of nearest neighbors includes training examples that are located far away from its neighborhood (see [Figure 4.7](#)).

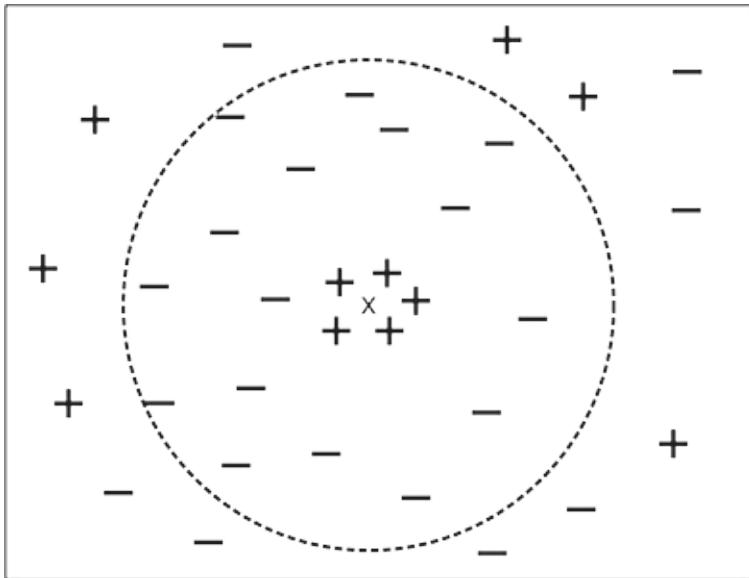


Figure 4.7.
 k -nearest neighbor classification with large k .

4.3.1 Algorithm

A high-level summary of the nearest neighbor classification method is given in [Algorithm 4.2](#). The algorithm computes the distance (or similarity) between each test instance $z=(x', y')$ and all the training examples $(x, y) \in D$ to determine its nearest neighbor list, D_z . Such computation can be costly if the number of training examples is large. However, efficient indexing techniques are available to reduce the computation needed to find the nearest neighbors of a test instance.

Algorithm 4.2 The k-nearest neighbor classifier.

```
1: Let  $k$  be the number of nearest neighbors and  $D$  be the set of  
training examples.  
2: for each test instance  $z=(x', y')$  do  
3:   Compute  $z=(x', x)$ , the distance between  $z$  and every example,  
 $(x, y) \in D$ .  
4:   Select  $D_z \subseteq D$ , the set of  $k$  closest training examples to  $z$ .  
5:    $y' = \text{argmax}_{v \in \sum(x_i, y_i) \in D_z} I(v=y_i)$   
6: end for
```

Once the nearest neighbor list is obtained, the test instance is classified based on the majority class of its nearest neighbors:

$$\text{Majority Voting: } y' = \text{argmax}_{v \in \sum(x_i, y_i) \in D_z} I(v=y_i), \quad (4.5)$$

where v is a class label, y_i is the class label for one of the nearest neighbors, and $I(\cdot)$ is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

In the majority voting approach, every neighbor has the same impact on the classification. This makes the algorithm sensitive to the choice of k , as shown in [Figure 4.6](#). One way to reduce the impact of k is to weight the influence of each nearest neighbor x_i according to its distance: $w_i = 1/d(x', x_i)^2$. As a result, training examples that are located far away from z have a weaker impact on the classification compared to those that are located close to z . Using the distance-weighted voting scheme, the class label can be determined as follows:

$$\text{Distance-Weighted Voting: } y' = \text{argmax}_{v \in \sum(x_i, y_i) \in D_z} w_i \times I(v=y_i). \quad (4.6)$$

4.3.2 Characteristics of Nearest Neighbor Classifiers

1. Nearest neighbor classification is part of a more general technique known as instance-based learning, which does not build a global model, but rather uses the training examples to make predictions for a test instance. (Thus, such classifiers are often said to be “model free.”) Such algorithms require a proximity measure to determine the similarity or distance between instances and a classification function that returns the predicted class of a test instance based on its proximity to other instances.
2. Although lazy learners, such as nearest neighbor classifiers, do not require model building, classifying a test instance can be quite expensive because we need to compute the proximity values individually between the test and training examples. In contrast, eager learners often spend the bulk of their computing resources for model building. Once a model has been built, classifying a test instance is extremely fast.
3. Nearest neighbor classifiers make their predictions based on local information. (This is equivalent to building a local model for each test instance.) By contrast, decision tree and rule-based classifiers attempt to find a global model that fits the entire input space. Because the classification decisions are made locally, nearest neighbor classifiers (with small values of k) are quite susceptible to noise.
4. Nearest neighbor classifiers can produce decision boundaries of arbitrary shape. Such boundaries provide a more flexible model representation compared to decision tree and rule-based classifiers that are often constrained to rectilinear decision boundaries. The decision boundaries of nearest neighbor classifiers also have high

variability because they depend on the composition of training examples in the local neighborhood. Increasing the number of nearest neighbors may reduce such variability.

5. Nearest neighbor classifiers have difficulty handling missing values in both the training and test sets since proximity computations normally require the presence of all attributes. Although, the subset of attributes present in two instances can be used to compute a proximity, such an approach may not produce good results since the proximity measures may be different for each pair of instances and thus hard to compare.
6. Nearest neighbor classifiers can handle the presence of interacting attributes, i.e., attributes that have more predictive power taken in combination than by themselves, by using appropriate proximity measures that can incorporate the effects of multiple attributes together.
7. The presence of irrelevant attributes can distort commonly used proximity measures, especially when the number of irrelevant attributes is large. Furthermore, if there are a large number of redundant attributes that are highly correlated with each other, then the proximity measure can be overly biased toward such attributes, resulting in improper estimates of distance. Hence, the presence of irrelevant and redundant attributes can adversely affect the performance of nearest neighbor classifiers.
8. Nearest neighbor classifiers can produce wrong predictions unless the appropriate proximity measure and data preprocessing steps are taken. For example, suppose we want to classify a group of people based on attributes such as height (measured in meters) and weight (measured in pounds). The height attribute has a low variability, ranging from 1.5 m to 1.85 m, whereas the weight attribute may vary from 90 lb. to 250 lb. If the scale of the attributes are not taken into consideration, the proximity measure may be dominated by differences in the weights of a person.

4.4 Naïve Bayes Classifier

Many classification problems involve uncertainty. First, the observed attributes and class labels may be unreliable due to imperfections in the measurement process, e.g., due to the limited precision of sensor devices. Second, the set of attributes chosen for classification may not be fully representative of the target class, resulting in uncertain predictions. To illustrate this, consider the problem of predicting a person's risk for heart disease based on a model that uses their diet and workout frequency as attributes. Although most people who eat healthily and exercise regularly have less chance of developing heart disease, they may still be at risk due to other *latent factors*, such as heredity, excessive smoking, and alcohol abuse, that are not captured in the model. Third, a classification model learned over a finite training set may not be able to fully capture the true relationships in the overall data, as discussed in the context of model overfitting in the previous chapter. Finally, uncertainty in predictions may arise due to the inherent *random nature* of real-world systems, such as those encountered in weather forecasting problems.

In the presence of uncertainty, there is a need to not only make predictions of class labels but also provide a measure of confidence associated with every prediction. Probability theory offers a systematic way for quantifying and manipulating uncertainty in data, and thus, is an appealing framework for assessing the confidence of predictions. Classification models that make use of probability theory to represent the relationship between attributes and class labels are known as **probabilistic classification models**. In this section, we present the naïve Bayes classifier, which is one of the simplest and most widely-used probabilistic classification models.

4.4.1 Basics of Probability Theory

Before we discuss how the naïve Bayes classifier works, we first introduce some basics of probability theory that will be useful in understanding the probabilistic classification models presented in this chapter. This involves defining the notion of probability and introducing some common approaches for manipulating probability values.

Consider a variable X , which can take any discrete value from the set $\{x_1, \dots, x_k\}$. When we have multiple observations of that variable, such as in a data set where the variable describes some characteristic of data objects, then we can compute the relative frequency with which each value occurs. Specifically, suppose that X has the value x_i for n_i data objects. The relative frequency with which we observe the event $X=x_i$ is then n_i/N , where N denotes the total number of occurrences ($N=\sum_{i=1}^k n_i$). These relative frequencies characterize the uncertainty that we have with respect to what value X may take for an unseen observation and motivates the notion of probability.

More formally, the **probability** of an event e , e.g., $P(X=x_i)$, measures how likely it is for the event e to occur. The most traditional view of probability is based on relative frequency of events (frequentist), while the Bayesian viewpoint (described later) takes a more flexible view of probabilities. In either case, a probability is always a number between 0 and 1. Further, the sum of probability values of all possible events, e.g., outcomes of a variable X is equal to 1. Variables that have probabilities associated with each possible outcome (values) are known as **random variables**.

Now, let us consider two random variables, X and Y , that can each take k discrete values. Let n_{ij} be the number of times we observe $X=x_i$ and $Y=y_j$, out

of a total number of N occurrences. The **joint probability** of observing $X=x_i$ and $Y=y_j$ together can be estimated as

$$P(X=x_i, Y=y_j) = n_{ij}N. \quad (4.7)$$

(This is an estimate since we typically have only a finite subset of all possible observations.) Joint probabilities can be used to answer questions such as “what is the probability that there will be a surprise quiz today AND I will be late for the class.” Joint probabilities are symmetric, i.e., $P(X=x, Y=y) = P(Y=y, X=x)$. For joint probabilities, it is useful to consider their sum with respect to one of the random variables, as described in the following equation:

$$\sum_{j=1}^k P(X=x_i, Y=y_j) = \sum_{j=1}^k n_{ij}N = n_iN = P(X=x_i), \quad (4.8)$$

where n_i is the total number of times we observe $X=x_i$ irrespective of the value of Y . Notice that n_i/N is essentially the probability of observing $X=x_i$. Hence, by summing out the joint probabilities with respect to a random variable Y , we obtain the probability of observing the remaining variable X . This operation is called **marginalization** and the probability value $P(X=x_i)$ obtained by marginalizing out Y is sometimes called the **marginal probability** of X . As we will see later, joint probability and marginal probability form the basic building blocks of a number of probabilistic classification models discussed in this chapter.

Notice that in the previous discussions, we used $P(X=x_i)$ to denote the probability of a particular outcome of a random variable X . This notation can easily become cumbersome when a number of random variables are involved. Hence, in the remainder of this section, we will use $P(X)$ to denote the probability of any generic outcome of the random variable X , while $P(x_i)$ will be used to represent the probability of the specific outcome x_i .

Bayes Theorem

Suppose you have invited two of your friends Alex and Martha to a dinner party. You know that Alex attends 40% of the parties he is invited to. Further, if Alex is going to a party, there is an 80% chance of Martha coming along. On the other hand, if Alex is not going to the party, the chance of Martha coming to the party is reduced to 30%. If Martha has responded that she will be coming to your party, what is the probability that Alex will also be coming?

Bayes theorem presents the statistical principle for answering questions like the previous one, where evidence from multiple sources has to be combined with prior beliefs to arrive at predictions. Bayes theorem can be briefly described as follows.

Let $P(Y|X)$ denote the **conditional probability** of observing the random variable Y whenever the random variable X takes a particular value. $P(Y|X)$ is often read as the probability of observing Y *conditioned* on the outcome of X . Conditional probabilities can be used for answering questions such as “given that it is going to rain today, what will be the probability that I will go to the class.” Conditional probabilities of X and Y are related to their joint

probability in the following way:

$$P(Y|X)=P(X, Y)P(X), \text{ which implies} \quad (4.9)$$

$$P(X, Y)=P(Y|X)\times P(X)=P(X|Y)\times P(Y). \quad (4.10)$$

Rearranging the last two expressions in [Equation 4.10](#) leads to [Equation 4.11](#), which is known as **Bayes theorem**:

$$P(Y|X)=P(X|Y)P(Y)P(X). \quad (4.11)$$

Bayes theorem provides a relationship between the conditional probabilities $P(Y|X)$ and $P(X|Y)$. Note that the denominator in [Equation 4.11](#) involves the marginal probability of X , which can also be represented as

$$P(X) = \sum_{i=1}^k P(X, y_i) = \sum_{i=1}^k P(X|y_i) \times P(y_i).$$

Using the previous expression for $P(X)$, we can obtain the following equation for $P(Y|X)$ solely in terms of $P(X|Y)$ and $P(Y)$:

$$P(Y|X) = P(X|Y)P(Y) / \sum_{i=1}^k P(X|y_i)P(y_i). \quad (4.12)$$

Example 4.4. [Bayes Theorem]

Bayes theorem can be used to solve a number of inferential questions about random variables. For example, consider the problem stated at the beginning on inferring whether Alex will come to the party. Let $P(A=1)$ denote the probability of Alex going to a party, while $P(A=0)$ denotes the probability of him not going to a party. We know that

$$P(A=1) = 0.4, \text{ and } P(A=0) = 1 - P(A=1) = 0.6.$$

Further, let $P(M=1|A)$ denote the conditional probability of Martha going to a party conditioned on whether Alex is going to the party. $P(M=1|A)$ takes the following values:

$$P(M=1|A=1) = 0.8, \text{ and } P(M=1|A=0) = 0.3.$$

We can use the above values of $P(M|A)$ and $P(A)$ to compute the probability of Alex going to the party given Martha is going to the party, $P(A=1|M=1)$, as follows:

$$P(A=1|M=1) = P(M=1|A=1)P(A=1)P(M=1|A=0)P(A=0) + P(M=1|A=1)P(A \neq 1|A=0)$$

Notice that even though the prior probability $P(A)$ of Alex going to the party is low, the observation that Martha is going, $M=1$, affects the conditional probability $P(A=1|M=1)$. This shows the value of Bayes theorem in combining prior assumptions with observed outcomes to make predictions. Since $P(A=1|M=1) > 0.5$, it is more likely for Alex to join if Martha is going to the party.

Using Bayes Theorem for Classification

For the purpose of classification, we are interested in computing the probability of observing a class label y for a data instance given its set of attribute values \underline{x} . This can be represented as $P(y|\underline{x})$, which is known as the **posterior probability** of the target class. Using the Bayes Theorem, we can represent the posterior probability as

$$P(y|\underline{x}) = P(\underline{x}|y)P(y)P(\underline{x}) \quad (4.14)$$

Note that the numerator of the previous equation involves two terms, $P(\underline{x}|y)$ and $P(y)$, both of which contribute to the posterior probability $P(y|\underline{x})$. We describe both of these terms in the following.

The first term $P(\underline{x}|y)$ is known as the **class-conditional** probability of the attributes given the class label. $P(\underline{x}|y)$ measures the likelihood of observing \underline{x} from the distribution of instances belonging to y . If \underline{x} indeed belongs to class y , then we should expect $P(\underline{x}|y)$ to be high. From this point of view, the use of class-conditional probabilities attempts to capture the process from which the data instances were generated. Because of this interpretation, probabilistic classification models that involve computing class-conditional probabilities are

known as **generative classification models**. Apart from their use in computing posterior probabilities and making predictions, class-conditional probabilities also provide insights about the underlying mechanism behind the generation of attribute values.

The second term in the numerator of [Equation 4.14](#) is the **prior probability** $P(y)$. The prior probability captures our prior beliefs about the distribution of class labels, independent of the observed attribute values. (This is the Bayesian viewpoint.) For example, we may have a prior belief that the likelihood of any person to suffer from a heart disease is α , irrespective of their diagnosis reports. The prior probability can either be obtained using expert knowledge, or inferred from historical distribution of class labels.

The denominator in [Equation 4.14](#) involves the probability of evidence, $P(\underline{x})$. Note that this term does not depend on the class label and thus can be treated as a normalization constant in the computation of posterior probabilities. Further, the value of $P(\underline{x})$ can be calculated as
$$P(x) = \sum_i P(x|y_i)P(y_i).$$

Bayes theorem provides a convenient way to combine our prior beliefs with the likelihood of obtaining the observed attribute values. During the training phase, we are required to learn the parameters for $P(y)$ and $P(x|y)$. The prior probability $P(y)$ can be easily estimated from the training set by computing the fraction of training instances that belong to each class. To compute the class-conditional probabilities, one approach is to consider the fraction of training instances of a given class for every possible combination of attribute values. For example, suppose that there are two attributes X_1 and X_2 that can each take a discrete value from c_1 to c_k . Let n_0 denote the number of training instances belonging to class 0, out of which n_{ij0} number of training instances have $X_1=c_i$ and $X_2=c_j$. The class-conditional probability can then be given as

$$P(X_1=c_i, X_2=c_j|Y=0)=n_{ij}0n_0.$$

This approach can easily become computationally prohibitive as the number of attributes increase, due to the exponential growth in the number of attribute value combinations. For example, if every attribute can take k discrete values, then the number of attribute value combinations is equal to k^d , where d is the number of attributes. The large number of attribute value combinations can also result in poor estimates of class-conditional probabilities, since every combination will have fewer training instances when the size of training set is small.

In the following, we present the naïve Bayes classifier, which makes a simplifying assumption about the class-conditional probabilities, known as the **naïve Bayes assumption**. The use of this assumption significantly helps in obtaining reliable estimates of class-conditional probabilities, even when the number of attributes are large.

4.4.2 Naïve Bayes Assumption

The naïve Bayes classifier assumes that the class-conditional probability of all attributes \mathbf{x} can be factored as a product of class-conditional probabilities of every attribute x_i , as described in the following equation:

$$P(\mathbf{x}|y) = \prod_{i=1}^d P(x_i|y), \quad (4.15)$$

where every data instance \mathbf{x} consists of d attributes, $\{x_1, x_2, \dots, x_d\}$. The basic assumption behind the previous equation is that the attribute values x_i are **conditionally independent** of each other, given the class label y . This means that the attributes are influenced only by the target class and if we

know the class label, then we can consider the attributes to be independent of each other. The concept of conditional independence can be formally stated as follows.

Conditional Independence

Let X_1 , X_2 , and Y denote three sets of random variables. The variables in X_1 are said to be conditionally independent of X_2 , given Y , if the following condition holds:

$$P(X_1|X_2, Y) = P(X_1|Y). \quad (4.16)$$

This means that conditioned on Y , the distribution of X_1 is not influenced by the outcomes of X_2 , and hence is conditionally independent of X_2 . To illustrate the notion of conditional independence, consider the relationship between a person's arm length (X_1) and his or her reading skills (X_2). One might observe that people with longer arms tend to have higher levels of reading skills, and thus consider X_1 and X_2 to be related to each other. However, this relationship can be explained by another factor, which is the age of the person (Y). A young child tends to have short arms and lacks the reading skills of an adult. If the age of a person is fixed, then the observed relationship between arm length and reading skills disappears. Thus, we can conclude that arm length and reading skills are not directly related to each other and are conditionally independent when the age variable is fixed.

Another way of describing conditional independence is to consider the joint conditional probability, $P(X_1, X_2|Y)$, as follows:

$$P(X_1, X_2|Y) = P(X_1, X_2, Y)P(Y) = P(X_1, X_2, Y)P(X_2, Y) \times P(X_2, Y)P(Y) = P(X_1|Y)P(X_2|Y)$$

where [Equation 4.16](#) was used to obtain the last line of [Equation 4.17](#). The previous description of conditional independence is quite useful from an operational perspective. It states that the joint conditional probability of X_1 and X_2 given Y can be factored as the product of conditional probabilities of X_1 and X_2 considered separately. This forms the basis of the naïve Bayes assumption stated in [Equation 4.15](#).

How a Naïve Bayes Classifier Works

Using the naïve Bayes assumption, we only need to estimate the conditional probability of each x_i given Y separately, instead of computing the class-conditional probability for every combination of attribute values. For example, if n_{i0} and n_{j0} denote the number of training instances belonging to class 0 with $X_1=c_i$ and $X_2=c_j$, respectively, then the class-conditional probability can be estimated as

$$P(X_1=c_i, X_2=x_j|Y=0) = n_{i0}n_0 \times n_{j0}n_0.$$

In the previous equation, we only need to count the number of training instances for every one of the k values of an attribute X , irrespective of the values of other attributes. Hence, the number of parameters needed to learn class-conditional probabilities is reduced from d_k to dk . This greatly simplifies the expression for the class-conditional probability and makes it more amenable to learning parameters and making predictions, even in high-dimensional settings.

The naïve Bayes classifier computes the posterior probability for a test instance \underline{x} by using the following equation:

$$P(y|x) = P(y) \prod_{i=1}^d P(x_i|y)P(x) \quad (4.18)$$

Since $P(\textcolor{teal}{x})$ is fixed for every y and only acts as a normalizing constant to ensure that $P(y|x) \in [0, 1]$, we can write

$$P(y|x) \propto P(y) \prod_{i=1}^d P(x_i|y).$$

Hence, it is sufficient to choose the class that maximizes $P(y) \prod_{i=1}^d P(x_i|y)$.

One of the useful properties of the naïve Bayes classifier is that it can easily work with incomplete information about data instances, when only a subset of attributes are observed at every instance. For example, if we only observe p out of d attributes at a data instance, then we can still compute $P(y) \prod_{i=1}^p P(x_i|y)$ using those p attributes and choose the class with the maximum value. The naïve Bayes classifier can thus naturally handle missing values in test instances. In fact, in the extreme case where no attributes are observed, we can still use the prior probability $P(y)$ as an estimate of the posterior probability. As we observe more attributes, we can keep refining the posterior probability to better reflect the likelihood of observing the data instance.

In the next two subsections, we describe several approaches for estimating the conditional probabilities $P(x_i|y)$ for categorical and continuous attributes from the training set.

Estimating Conditional Probabilities for Categorical Attributes

For a categorical attribute X_i , the conditional probability $P(X_i=c|y)$ is estimated according to the fraction of training instances in class y where X_i takes on a particular categorical value c .

$$P(X_i=c|y)=ncn,$$

where n is the number of training instances belonging to class y , out of which nc number of instances have $X_i=c$. For example, in the training set given in [Figure 4.8](#), seven people have the class label Defaulted Borrower=No, out of which three people have Home Owner=Yes while the remaining four have Home Owner=No. As a result, the conditional probability for $P(\text{Home Owner}=Yes|\text{Defaulted Borrower}=No)$ is equal to 3/7. Similarly, the conditional probability for defaulted borrowers with Marital Status=Single is given by $P(\text{Marital Status}=Single|\text{Defaulted Borrower}=Yes)=2/3$. Note that the sum of conditional probabilities over all possible outcomes of X_i is equal to one, i.e., $\sum c P(X_i=c|y)=1$.

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Figure 4.8.

Training set for predicting the loan default problem.

Estimating Conditional Probabilities for Continuous Attributes

There are two ways to estimate the class-conditional probabilities for continuous attributes:

1. We can discretize each continuous attribute and then replace the continuous values with their corresponding discrete intervals. This approach transforms the continuous attributes into ordinal attributes, and the simple method described previously for computing the conditional probabilities of categorical attributes can be employed. Note that the estimation error of this method depends on the discretization strategy (as described in [Section 2.3.6](#) on [page 63](#)), as well as the number of discrete intervals. If the number of intervals is too large, every interval may have an insufficient number of training instances to provide a reliable estimate of $P(X_i|Y)$. On the other hand, if the number of intervals is too small, then the discretization process may lose information about the true distribution of continuous values, and thus result in poor predictions.
2. We can assume a certain form of probability distribution for the continuous variable and estimate the parameters of the distribution using the training data. For example, we can use a Gaussian distribution to represent the conditional probability of continuous attributes. The Gaussian distribution is characterized by two parameters, the mean, μ , and the variance, σ^2 . For each class y_j , the class-conditional probability for attribute X_i is

$$P(X_i=x_i|Y=y_j)=\frac{1}{2\pi\sigma_{ij}^2}\exp[-(x_i-\mu_{ij})^2/2\sigma_{ij}^2]. \quad (4.19)$$

The parameter μ_{ij} can be estimated using the sample mean of $X_i(x^-)$ for all training instances that belong to y_j . Similarly, σ_{ij}^2 can be estimated from the sample variance (s^2) of such training instances. For example, consider the annual income attribute shown in [Figure 4.8](#). The sample mean and variance for this attribute with respect to the class [No](#) are

$$\bar{x} = 125 + 100 + 70 + \dots + 757 = 100 \\ s^2 = (125 - 110)^2 + (100 - 110)^2 + \dots \\ (75 - 110)^2 = 2975 \\ s = \sqrt{2975} = 54.54.$$

Given a test instance with taxable income equal to \$120K, we can use the following value as its conditional probability given class [No](#):

$$P(\text{Income}=120|\text{No}) = \frac{1}{2\pi(54.54)} \exp^{-(120-110)^2/2 \times 2975} = 0.0072.$$

Example 4.5. [Naïve Bayes Classifier]

Consider the data set shown in [Figure 4.9\(a\)](#), where the target class is Defaulted Borrower, which can take two values Yes and No. We can compute the class-conditional probability for each categorical attribute and the sample mean and variance for the continuous attribute, as summarized in [Figure 4.9\(b\)](#).

We are interested in predicting the class label of a test instance $x = (\text{Home Owner}=\text{No}, \text{Marital Status}=\text{Married}, \text{Annual Income}=\$120K)$. To do this, we first compute the prior probabilities by counting the number of training instances belonging to every class. We thus obtain $P(\text{Yes})=0.3$ and $P(\text{No})=0.7$. Next, we can compute the class-conditional probability as follows:

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

(a)

$P(\text{Home Owner}=\text{Yes}|\text{No}) = 3/7$
 $P(\text{Home Owner}=\text{No}|\text{No}) = 4/7$
 $P(\text{Home Owner}=\text{Yes}|\text{Yes}) = 0$
 $P(\text{Home Owner}=\text{No}|\text{Yes}) = 1$
 $P(\text{Marital Status}=\text{Single}|\text{No}) = 2/7$
 $P(\text{Marital Status}=\text{Divorced}|\text{No}) = 1/7$
 $P(\text{Marital Status}=\text{Married}|\text{No}) = 4/7$
 $P(\text{Marital Status}=\text{Single}|\text{Yes}) = 2/3$
 $P(\text{Marital Status}=\text{Divorced}|\text{Yes}) = 1/3$
 $P(\text{Marital Status}=\text{Married}|\text{Yes}) = 0$

For Annual Income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

(b)

Figure 4.9.

The naïve Bayes classifier for the loan classification problem.

$$P(x|\text{NO}) = P(\text{Home Owner}=\text{No}|\text{No}) \times P(\text{Status}=\text{Married}|\text{No}) \times P(\text{Annual Income}$$

Notice that the class-conditional probability for class `Yes` has become 0 because there are no instances belonging to class `Yes` with `Status=Married` in the training set. Using these class-conditional probabilities, we can estimate the posterior probabilities as

$$P(\text{No}|x) = 0.7 \times 0.0024 P(x) = 0.0016 \alpha. P(\text{Yes}|x) = 0.3 \times 0 P(x) = 0.$$

where $\alpha = 1/P(x)$ is a normalizing constant. Since $P(\text{No}|x) > P(\text{Yes}|x)$, the instance is classified as `No`.

Handling Zero Conditional Probabilities

The preceding example illustrates a potential problem with using the naïve Bayes assumption in estimating class-conditional probabilities. If the conditional probability for any of the attributes is zero, then the entire expression for the class-conditional probability becomes zero. Note that zero conditional probabilities arise when the number of training instances is small and the number of possible values of an attribute is large. In such cases, it may happen that a combination of attribute values and class labels are never observed, resulting in a zero conditional probability.

In a more extreme case, if the training instances do not cover some combinations of attribute values and class labels, then we may not be able to even classify some of the test instances. For example, if

$P(\text{Marital Status}=\text{Divorced}|\text{No})$ is zero instead of 1/7, then a data instance with attribute set $x=$

(Home Owner=Yes, Marital Status=Divorced, Income=\$120K) has the following class-conditional probabilities:

$$P(x|\text{No}) = 3/7 \times 0 \times 0.0072 = 0. P(x|\text{Yes}) = 0 \times 1/3 \times 1.2 \times 10^{-9} = 0.$$

Since both the class-conditional probabilities are 0, the naïve Bayes classifier will not be able to classify the instance. To address this problem, it is important to adjust the conditional probability estimates so that they are not as brittle as simply using fractions of training instances. This can be achieved by using the following alternate estimates of conditional probability:

$$\text{Laplace estimate: } P(X_i=c|y) = \frac{n_{ci} + 1}{n + v}, \quad (4.20)$$

$$\text{m-estimate: } P(X_i=c|y) = \frac{n_{ci} + m}{n + mp_n + m}, \quad (4.21)$$

where n is the number of training instances belonging to class y , n_c is the number of training instances with $X_i=c$ and $Y=y$, v is the total number of attribute values that X_i can take, p is some initial estimate of $P(X_i=c|y)$ that is known a priori, and m is a hyper-parameter that indicates our confidence in using p when the fraction of training instances is too brittle. Note that even if $n_c=0$, both Laplace and m-estimate provide non-zero values of conditional probabilities. Hence, they avoid the problem of vanishing class-conditional probabilities and thus generally provide more robust estimates of posterior probabilities.

Characteristics of Naïve Bayes Classifiers

1. Naïve Bayes classifiers are probabilistic classification models that are able to quantify the uncertainty in predictions by providing posterior probability estimates. They are also generative classification models as they treat the target class as the causative factor for generating the data instances. Hence, apart from computing posterior probabilities, naïve Bayes classifiers also attempt to capture the underlying mechanism behind the generation of data instances belonging to every class. They are thus useful for gaining predictive as well as descriptive insights.
2. By using the naïve Bayes assumption, they can easily compute class-conditional probabilities even in high-dimensional settings, provided that the attributes are conditionally independent of each other given the class labels. This property makes naïve Bayes classifier a simple and effective classification technique that is commonly used in diverse application problems, such as text classification.
3. Naïve Bayes classifiers are robust to isolated noise points because such points are not able to significantly impact the conditional probability estimates, as they are often averaged out during training.

4. Naïve Bayes classifiers can handle missing values in the training set by ignoring the missing values of every attribute while computing its conditional probability estimates. Further, naïve Bayes classifiers can effectively handle missing values in a test instance, by using only the non-missing attribute values while computing posterior probabilities. If the frequency of missing values for a particular attribute value depends on class label, then this approach will not accurately estimate posterior probabilities.
5. Naïve Bayes classifiers are robust to irrelevant attributes. If X_i is an irrelevant attribute, then $P(X_i|Y)$ becomes almost uniformly distributed for every class y . The class-conditional probabilities for every class thus receive similar contributions of $P(X_i|Y)$, resulting in negligible impact on the posterior probability estimates.
6. Correlated attributes can degrade the performance of naïve Bayes classifiers because the naïve Bayes assumption of conditional independence no longer holds for such attributes. For example, consider the following probabilities:

$$P(A=0|Y=0)=0.4, P(A=1|Y=0)=0.6, P(A=0|Y=1)=0.6, P(A=1|Y=1)=0.4,$$

where A is a binary attribute and Y is a binary class variable. Suppose there is another binary attribute B that is perfectly correlated with A when $Y=0$, but is independent of A when $Y=1$. For simplicity, assume that the conditional probabilities for B are the same as for A . Given an instance with attributes $A=0$, $B=0$, and assuming conditional independence, we can compute its posterior probabilities as follows:

$$P(Y=0|A=0, B=0)=P(A=0|Y=0)P(B=0|Y=0)P(Y=0)P(A=0, B=0)=0.16\times P(Y=0)$$

If $P(Y=0)=P(Y=1)$, then the naïve Bayes classifier would assign the instance to class 1. However, the truth is,

$$P(A=0, B=0|Y=0)=P(A=0|Y=0)=0.4,$$

because A and B are perfectly correlated when $Y=0$. As a result, the posterior probability for $Y=0$ is

$$P(Y=0|A=0, B=0) = P(A=0, B=0|Y=0)P(Y=0)P(A=0, B=0) = 0.4 \times P(Y=0)P(A=0, B=0)$$

which is larger than that for $Y=1$. The instance should have been classified as class 0. Hence, the naïve Bayes classifier can produce incorrect results when the attributes are not conditionally independent given the class labels. Naïve Bayes classifiers are thus not well-suited for handling redundant or interacting attributes.

4.5 Bayesian Networks

The conditional independence assumption made by naïve Bayes classifiers may seem too rigid, especially for classification problems where the attributes are dependent on each other even after conditioning on the class labels. We thus need an approach to relax the naïve Bayes assumption so that we can capture more generic representations of conditional independence among attributes.

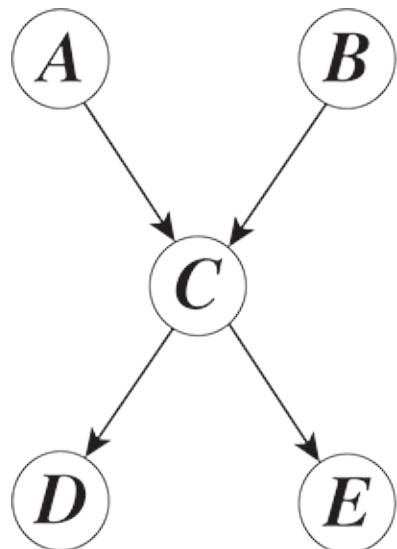
In this section, we present a flexible framework for modeling probabilistic relationships between attributes and class labels, known as **Bayesian Networks**. By building on concepts from probability theory and graph theory, Bayesian networks are able to capture more generic forms of conditional independence using simple schematic representations. They also provide the necessary computational structure to perform inferences over random variables in an efficient way. In the following, we first describe the basic representation of a Bayesian network, and then discuss methods for performing inference and learning model parameters in the context of classification.

4.5.1 Graphical Representation

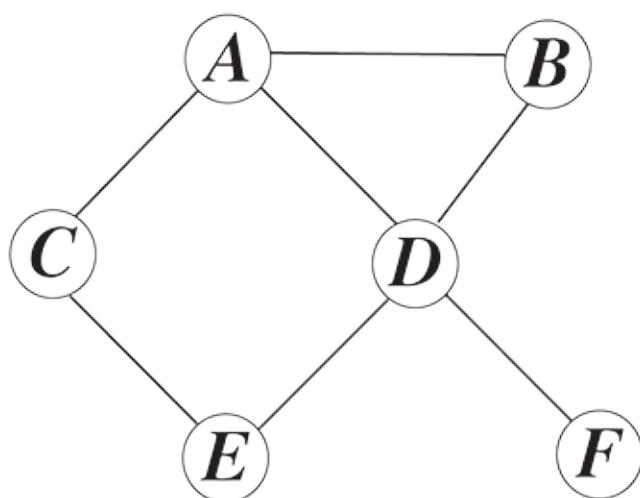
Bayesian networks belong to a broader family of models for capturing probabilistic relationships among random variables, known as **probabilistic graphical models**. The basic concept behind these models is to use graphical representations where the nodes of the graph correspond to random variables and the edges between the nodes express probabilistic

relationships. [Figures 4.10\(a\)](#) and [4.10\(b\)](#) show examples of probabilistic graphical models using directed edges (with arrows) and undirected edges (without arrows), respectively. Directed graphical models are also known as Bayesian networks while undirected graphical models are known as **Markov random fields**. The two approaches use different semantics for expressing relationships among random variables and are thus useful in different contexts. In the following, we briefly describe Bayesian networks that are useful in the context of classification.

A Bayesian network (also referred to as a **belief network**) involves directed edges between nodes, where every edge represents a direction of influence among random variables. For example, [Figure 4.10\(a\)](#) shows a Bayesian network where variable *C* depends upon the values of variables *A* and *B*, as indicated by the arrows pointing toward *C* from *A* and *B*. Consequently, the variable *C* influences the values of variables *D* and *E*. Every edge in a Bayesian network thus encodes a dependence relationship between random variables with a particular directionality.



(a) Directed graphical model (Bayesian Network).



(b) Undirected graphical model (Markov Random Field).

Figure 4.10.

Illustrations of two basic types of graphical models.

Bayesian networks are directed acyclic graphs (DAG) because they do not contain any directed cycles such that the influence of a node loops back to the same node. [Figure 4.11](#) shows some examples of Bayesian networks that capture different types of dependence structures among random variables. In a directed acyclic graph, if there is a directed edge from X to Y , then X is called the **parent** of Y and Y is called the **child** of X . Note that a node can have multiple parents in a Bayesian network, e.g., node D has two parent nodes, B and C , in [Figure 4.11\(a\)](#). Furthermore, if there is a directed path in the network from X to Z , then X is an **ancestor** of Z , while Z is a **descendant** of X . For example, in the diagram shown in [Figure 4.11\(b\)](#), A is a descendant of D and D is an ancestor of B . Note that there can be multiple directed paths between two nodes of a directed acyclic graph, as is the case for nodes A and D in [Figure 4.11\(a\)](#).

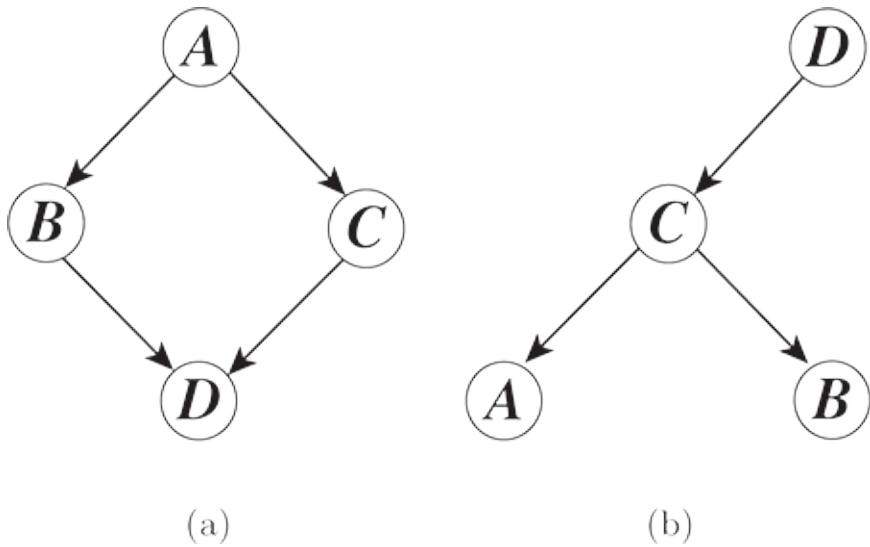


Figure 4.11.

Examples of Bayesian networks.

Conditional Independence

An important property of a Bayesian network is its ability to represent varying forms of conditional independence among random variables. There are several ways of describing the conditional independence assumptions captured by Bayesian networks. One of the most generic ways of expressing conditional independence is the concept of ***d-separation***, which can be used to determine if any two sets of nodes A and B are conditionally independent given another set of nodes C . Another useful concept is that of the **Markov blanket** of a node Y , which denotes the minimal set of nodes X that makes Y independent of the other nodes in the graph, when conditioned on X . (See Bibliographic Notes for more details on *d-separation* and Markov blanket.) However, for the purpose of classification, it is sufficient to describe a simpler expression of conditional independence in Bayesian networks, known as the **local Markov property**.

Property 1 (Local Markov Property).

A node in a Bayesian network is conditionally independent of its non-descendants, if its parents are known.

To illustrate the local Markov property, consider the Bayes network shown in [Figure 4.11\(b\)](#). We can state that A is conditionally independent of both B and D given C , because C is the parent of A and nodes B and D are non-descendants of A . The local Markov property helps in interpreting parent-child relationships in Bayesian networks as representations of conditional probabilities. Since a node is conditionally independent of its non-descendants

given its parents, the conditional independence assumptions imposed by a Bayesian network is often sparse in structure. Nonetheless, Bayesian networks are able to express a richer class of conditional independence statements among attributes and class labels than the naïve Bayes classifier. In fact, the naïve Bayes classifier can be viewed as a special type of Bayesian network, where the target class Y is at the root of a tree and every attribute X_i is connected to the root node by a directed edge, as shown in [Figure 4.12\(a\)](#).

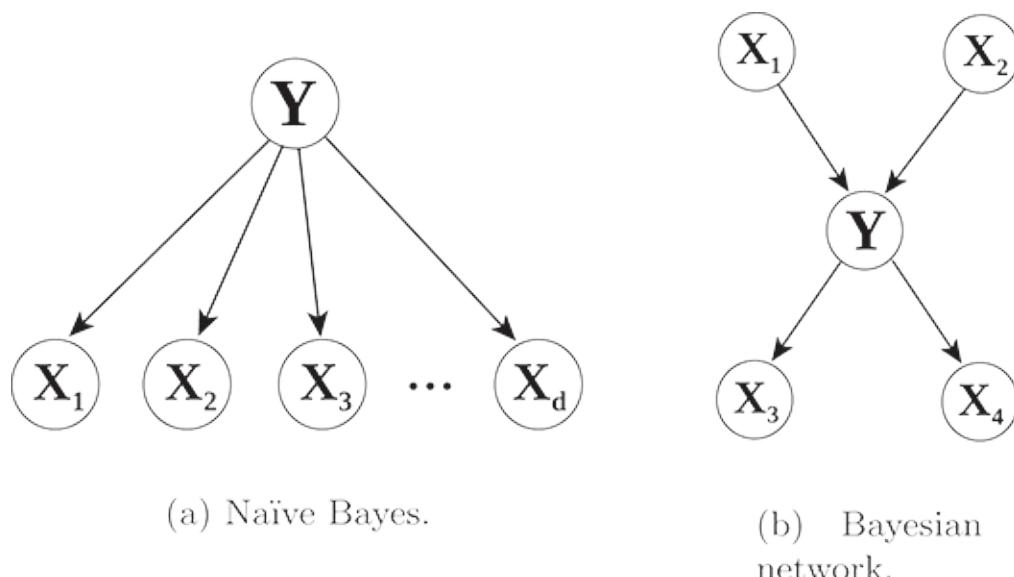


Figure 4.12.

Comparing the graphical representation of a naïve Bayes classifier with that of a generic Bayesian network.

Note that in a naïve Bayes classifier, every directed edge points from the target class to the observed attributes, suggesting that the class label is a factor behind the generation of attributes. Inferring the class label can thus be viewed as *diagnosing* the root cause behind the observed attributes. On the other hand, Bayesian networks provide a more generic structure of probabilistic relationships, since the target class is not required to be at the root of a tree but can appear anywhere in the graph, as shown in [Figure 4.12\(b\)](#).

4.12(b) In this diagram, inferring Y not only helps in diagnosing the factors influencing X_3 and X_4 , but also helps in *predicting* the influence of X_1 and X_2 .

Joint Probability

The local Markov property can be used to succinctly express the joint probability of the set of random variables involved in a Bayesian network. To realize this, let us first consider a Bayesian network consisting of d nodes, X_1 to X_d , where the nodes have been numbered in such a way that X_i is an ancestor of X_j only if $i < j$. The joint probability of $X = \{X_1, \dots, X_d\}$ can be generically factorized using the chain rule of probability as

$$P(X) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2) \dots P(X_d|X_1, \dots, X_{d-1}) = \prod_{i=1}^d P(X_i|X_{i-1}) \quad (4.22)$$

By the way we have constructed the graph, note that the set $\{X_1, \dots, X_{i-1}\}$ contains only non-descendants of X_i . Hence, by using the local Markov property, we can write $P(X_i|X_1, \dots, X_{i-1})$ as $P(X_i|\text{pa}(X_i))$, where $\text{pa}(X_i)$ denotes the parents of X_i . The joint probability can then be represented as

$$P(X) = \prod_{i=1}^d P(X_i|\text{pa}(X_i)) \quad (4.23)$$

It is thus sufficient to represent the probability of every node X_i in terms of its parent nodes, $\text{pa}(X_i)$, for computing $P(\text{X})$. This is achieved with the help of **probability tables** that associate every node to its parent nodes as follows:

1. The probability table for node X_i contains the conditional probability values $P(X_i|\text{pa}(X_i))$ for every combination of values in X_i and $\text{pa}(X_i)$.
2. If X_i has no parents ($\text{pa}(X_i)=\emptyset$), then the table contains only the prior probability $P(X_i)$.

Example 4.6. [Probability Tables]

Figure 4.13 shows an example of a Bayesian network for modeling the relationships between a patient's symptoms and risk factors. The probability tables are shown at the side of every node in the figure. The probability tables associated with the risk factors (Exercise and Diet) contain only the prior probabilities, whereas the tables for heart disease, heartburn, blood pressure, and chest pain, contain the conditional probabilities.

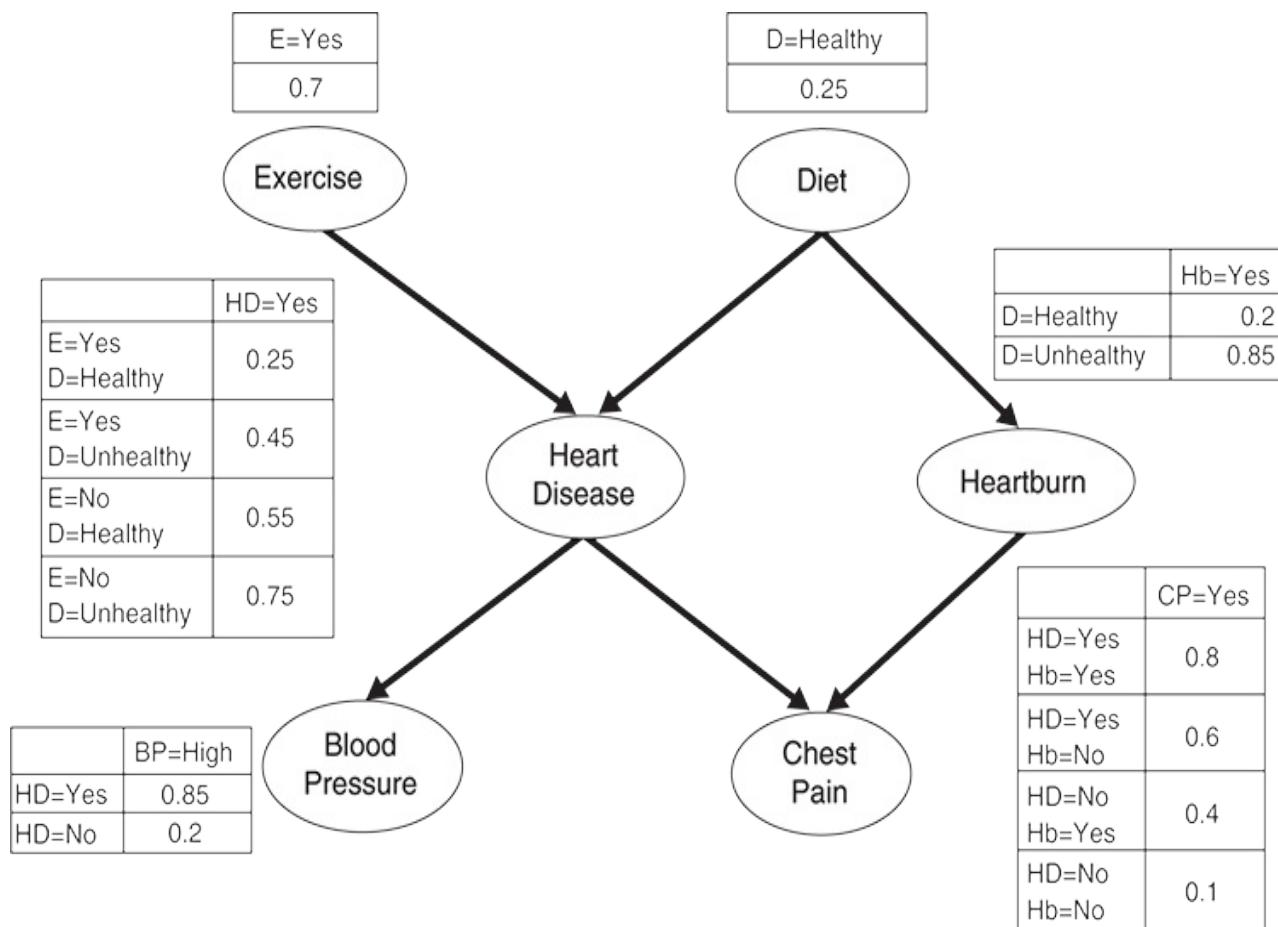


Figure 4.13.

A Bayesian network for detecting heart disease and heartburn in patients.

Use of Hidden Variables

A Bayesian network typically involves two types of variables: *observed variables* that are clamped to specific observed values, and *unobserved variables*, whose values are not known and need to be inferred from the network. To distinguish between these two types of variables, observed variables are generally represented using shaded nodes while unobserved variables are represented using empty nodes. [Figure 4.14](#) shows an example of a Bayesian network with observed variables (A , B , and E) and unobserved variables (C and D).

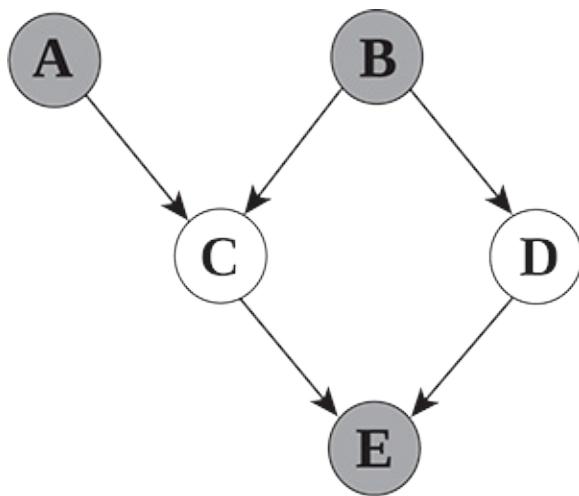


Figure 4.14.

Observed and unobserved variables are represented using unshaded and shaded circles, respectively.

In the context of classification, the observed variables correspond to the set of attributes \mathbf{X} , while the target class is represented using an unobserved variable Y that needs to be inferred during testing. However, note that a generic Bayesian network may contain many other unobserved variables apart from the target class, as represented in [Figure 4.15](#) as the set of variables \mathbf{H} . These unobserved variables represent hidden or confounding factors that affect the probabilities of attributes and class labels, although they are never directly observed. The use of hidden variables enhances the expressive power of Bayesian networks in representing complex probabilistic

relationships between attributes and class labels. This is one of the key distinguishing properties of Bayesian networks as compared to naïve Bayes classifiers.

4.5.2 Inference and Learning

Given the probability tables corresponding to every node in a Bayesian network, the problem of inference corresponds to computing the probabilities of different sets of random variables. In the context of classification, one of the key inference problems is to compute the probability of a target class Y taking on a specific value y , given the set of observed attributes at a data instance, \underline{x} . This can be represented using the following conditional probability:

$$P(Y=y|\underline{x}) = \sum_{\underline{y}'} P(y', \underline{x}) \quad (4.24)$$

The previous equation involves marginal probabilities of the form $P(y, \underline{x})$. They can be computed by marginalizing out the hidden variables \mathbf{H} from the joint probability as follows:

$$P(y, \underline{x}) = \sum_{\mathbf{H}} P(y, \underline{x}, \mathbf{H}), \quad (4.25)$$

where the joint probability $P(y, \underline{x}, \mathbf{H})$ can be obtained by using the factorization described in [Equation 4.23](#). To understand the nature of computations involved in estimating $P(y, \underline{x})$, consider the example Bayesian network shown in [Figure 4.15](#), which involves a target class, Y , three observed attributes, X_1 to X_3 , and four hidden variables, H_1 to H_4 . For this network, we can express $P(y, \underline{x})$ as

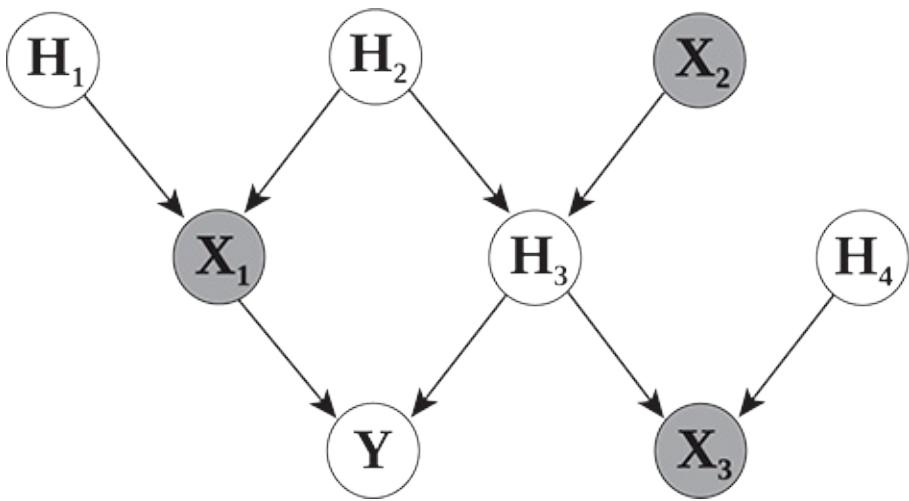


Figure 4.15.

An example of a Bayesian network with four hidden variables, H_1 to H_4 , three observed attributes, X_1 to X_3 , and one target class Y .

$$P(y, x) = \sum h_1 \sum h_2 \sum h_3 \sum h_4 P(y, x_1, x_2, h_1, h_2, h_3, h_4) = \sum h_1 \sum h_2 \sum h_3 \sum h_4 [P(h_1)P(h_2)P(x_2)P(h_4)P(x_1|h_1, h_2) \times P(h_3|x_2, h_2)P(y|x_1, h_3)P(x_3|h_3, h_4)], \quad (4.26)$$

$$= \sum h_1 \sum h_2 \sum h_3 \sum h_4 f(h_1, h_2, h_3, h_4), \quad (4.27)$$

where f is a factor that depends on the values of h_1 to h_4 . In the previous simplistic expression of $P(y, \underline{x})$, a different summand is considered for every combination of values, h_1 to h_4 , in the hidden variables, H_1 to H_4 . If we assume that every variable in the network can take k discrete values, then the summation has to be carried out for a total number of k^4 times. The computational complexity of this approach is thus $O(k^4)$. Moreover, the number of computations grows exponentially with the number of hidden variables, making it difficult to use this approach with networks that have a large number of hidden variables. In the following, we present different computational techniques for efficiently performing inferences in Bayesian networks.

Variable Elimination

To reduce the number of computations involved in estimating $P(y, \underline{x})$, let us closely examine the expressions in [Equations 4.26](#) and [4.27](#). Notice that although $f(h_1, h_2, h_3, h_4)$ depends on the values of all four hidden variables, it can be decomposed as a product of several smaller factors, where every factor involves only a small number of hidden variables. For example, the factor $P(h_4)$ depends only on the value of h_4 , and thus acts as a constant multiplicative term when summations are performed over h_1 , h_2 , or h_3 . Hence, if we place $P(h_4)$ outside the summations of h_1 to h_3 , we can save some repeated multiplications occurring inside every summand.

In general, we can push every summation as far inside as possible, so that the factors that do not depend on the summing variable are placed outside the summation. This will help reduce the number of wasteful computations by using smaller factors at every summation. To illustrate this process, consider the following sequence of steps for computing $P(y, \underline{x})$, by rearranging the order

of summations in [Equation 4.26](#).

$$P(y, x) = P(x_2) \sum h_4 P(h_4) \sum h_3 P(y|x_1, h_3) P(x_3|h_3, h_4) \times \sum h_2 P(h_2) P(h_3|x_2, h_2) \quad (4.28)$$

$$= P(x_2) \sum h_4 P(h_4) \sum h_3 P(y|x_1, h_3) P(x_3|h_3, h_4) \times \sum h_2 P(h_2) P(h_3|x_2, h_2) f_1(h_2) \quad (4.29)$$

$$= P(x_2) \sum h_4 P(h_4) \sum h_3 P(y|x_1, h_3) P(x_3|h_3, h_4) f_2(h_3) \quad (4.30)$$

$$= P(x_2) \sum h_4 P(h_4) f_3(h_4) \quad (4.31)$$

where f_i represents the intermediate factor term obtained by summing out h_i . To check if the previous rearrangements provide any improvements in

computational efficiency, let us count the number of computations occurring at every step of the process. At the first step ([Equation 4.28](#)), we perform a summation over h_1 using factors that depend on h_1 and h_2 . This requires considering every pair of values in h_1 and h_2 , resulting in $O(k^2)$ computations. Similarly, the second step ([Equation 4.29](#)) involves summing out h_2 using factors of h_2 and h_3 , leading to $O(k^2)$ computations. The third step ([Equation 4.30](#)) again requires $O(k^2)$ computations as it involves summing out h_3 over factors depending on h_3 and h_4 . Finally, the fourth step ([Equation 4.31](#)) involves summing out h_4 using factors depending on h_4 , resulting in $O(k)$ computations.

The overall complexity of the previous approach is thus $O(k^2)$, which is considerably smaller than the $O(k^4)$ complexity of the basic approach. Hence, by merely rearranging summations and using algebraic manipulations, we are able to improve the computational efficiency in computing $P(y, \underline{x})$. This procedure is known as **variable elimination**.

The basic concept that variable elimination exploits to reduce the number of computations is the distributive nature of multiplication over addition operations. For example, consider the following multiplication and addition operations:

$$a.(b+c+d)=a.b+a.c+a.d$$

Notice that the right-hand side of the previous equation involves three multiplications and three additions, while the left-hand side involves only one multiplication and three additions, thus saving on two arithmetic operations. This property is utilized by variable elimination in pushing out constant terms outside the summation, such that they are multiplied only once.

Note that the efficiency of variable elimination depends on the order of hidden variables used for performing summations. Hence, we would ideally like to find the optimal order of variables that result in the smallest number of computations. Unfortunately, finding the optimal order of summations for a generic Bayesian network is an NP-Hard problem, i.e., there does not exist an efficient algorithm for finding the optimal ordering that can run in polynomial time. However, there exists efficient techniques for handling special types of Bayesian networks, e.g., those involving *tree-like* graphs, as described in the following.

Sum-Product Algorithm for Trees

Note that in [Equations 4.28](#) and [4.29](#), whenever a variable h_i is eliminated during marginalization, it results in the creation of a factor f_i that depends on the neighboring nodes of h_i . f_i is then absorbed in the factors of neighboring variables and the process is repeated until all unobserved variables have marginalized. This phenomena of variable elimination can be viewed as transmitting a *local message* from the variable being marginalized to its neighboring nodes. This idea of message passing utilizes the structure of the graph for performing computations, thus making it possible to use graph-theoretic approaches for making effective inferences. The **sum-product algorithm** builds on the concept of message passing for computing marginal and conditional probabilities on tree-based graphs.

[Figure 4.16](#) shows an example of a tree involving five variables, X_1 to X_5 . A key characteristic of a tree is that every node in the tree has exactly one parent, and there is only one directed edge between any two nodes in the tree. For the purpose of illustration, let us consider the problem of estimating the marginal probability of X_2 , $P(X_2)$. This can be obtained by marginalizing out every variable in the graph except X_2 and rearranging the summations to obtain the following expression:

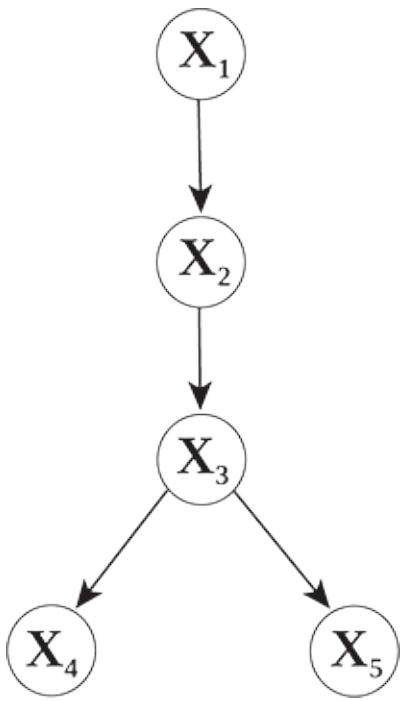


Figure 4.16.

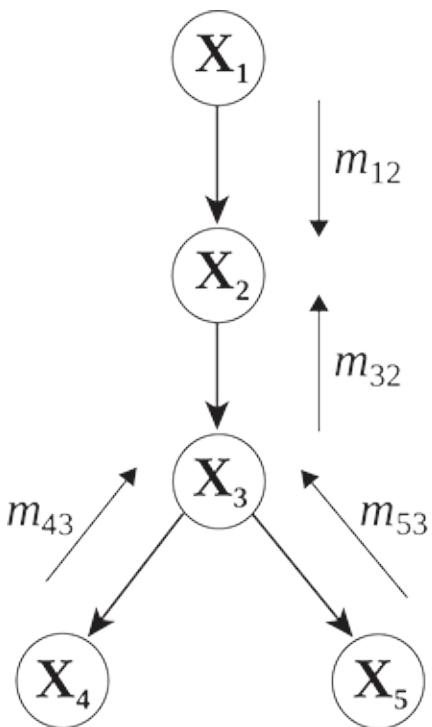
An example of a Bayesian network with a tree structure.

$$\begin{aligned}
 P(x_2) &= \sum_{x_1} \sum_{x_3} \sum_{x_4} \sum_{x_5} P(x_1) P(x_2|x_1) P(x_3|x_2) P(x_4|x_3) P(x_5|x_3), = \\
 &(\sum_{x_1} P(x_1) P(x_2|x_1)) \rightsquigarrow m_{12}(x_2) (\sum_{x_3} P(x_3|x_2) (\sum_{x_4} P(x_4|x_3)) \rightsquigarrow m_{43}(x_3) \\
 &(\sum_{x_5} P(x_5|x_3)) \rightsquigarrow m_{53}(x_3)), \rightsquigarrow m_{32}(x_2)
 \end{aligned}$$

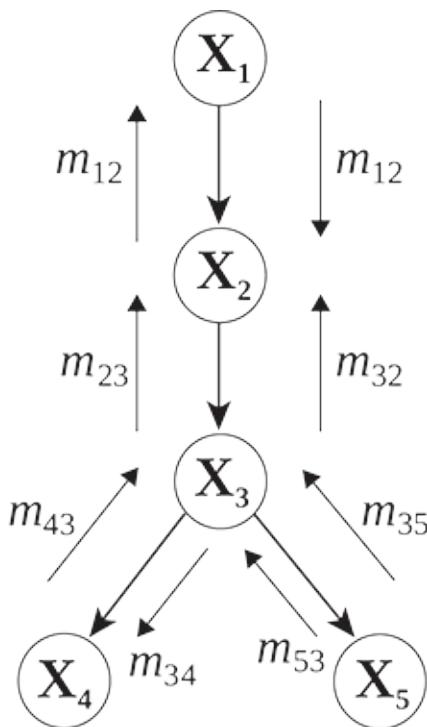
where $m_{ij}(x_j)$ has been conveniently chosen to represent the factor of x_j that is obtained by summing out x_i . We can view $m_{ij}(x_j)$ as a local message passed from node x_i to node x_j , as shown using arrows in [Figure 4.17\(a\)](#). These local messages capture the influence of eliminating nodes on the marginal probabilities of neighboring nodes.

Before we formally describe the formula for computing $m_{ij}(x_j)$ and $P(x_j)$, we first define a potential function $\psi(\cdot)$ that is associated every node and edge of the graph. We can define the potential of a node X_i as

$$\psi(X_i) = \{P(X_i), \text{if } X_i \text{ is the root node. } 1, \text{otherwise.} \quad (4.32)$$



(a) Message passing
for computing $P(x_2)$.



(b) Complete message
passing approach.

Figure 4.17.

Illustration of message passing in the sum-product algorithm.

Similarly, we can define the potential of an edge between nodes X_i and X_j (where X_i is the parent of X_j) as

$$\psi(X_i, X_j) = P(X_j|X_i).$$

Using $\psi(X_i)$ and $\psi(X_i, X_j)$, we can represent $m_{ij}(x_j)$ using the following equation:

$$m_{ij}(x_j) = \sum_{x_i} (\psi(x_i) \psi(x_i, x_j) \prod_{k \in N(i) \setminus m_{ki}(x_i)}), \quad (4.33)$$

where $N(i)$ represents the set of neighbors of node X_i . The message m_{ij} that is transmitted from X_i to X_j can thus be recursively computed using the

messages incident on X_i from its neighboring nodes excluding X_i . Note that the formula for m_{ij} involves taking a sum over all possible values of X_j , after multiplying the factors obtained from the neighbors of X_j . This approach of message passing is thus called the “sum-product” algorithm. Further, since m_{ij} represents a notion of “belief” propagated from X_i to X_j , this algorithm is also known as **belief propagation**. The marginal probability of a node X_i

is then given as

$$P(x_i) = \psi(x_i) \prod_{j \in N(i)} m_{ji}(x_i). \quad (4.34)$$

A useful property of the sum-product algorithm is that it allows the messages to be reused for computing a different marginal probability in the future. For example, if we had to compute the marginal probability for node X_3 , we would require the following messages from its neighboring nodes: $m_{23}(x_3)$, $m_{43}(x_3)$, and $m_{53}(x_3)$. However, note that $m_{43}(x_3)$, and $m_{53}(x_3)$ have already been computed in the process of computing the marginal probability of X_2 and thus can be reused.

Notice that the basic operations of the sum-product algorithm resemble a message passing protocol over the edges of the network. A node sends out a message to all its neighboring nodes only after it has received incoming messages from all its neighbors. Hence, we can initialize the message passing protocol from the leaf nodes, and transmit messages till we reach the root node. We can then run a second pass of messages from the root node back to the leaf nodes. In this way, we can compute the messages for every edge in both directions, using just $O(2|E|)$ operations, where $|E|$ is the number of edges. Once we have transmitted all possible messages as shown in [Figure 4.17\(b\)](#), we can easily compute the marginal probability of every node in the graph using [Equation 4.34](#).

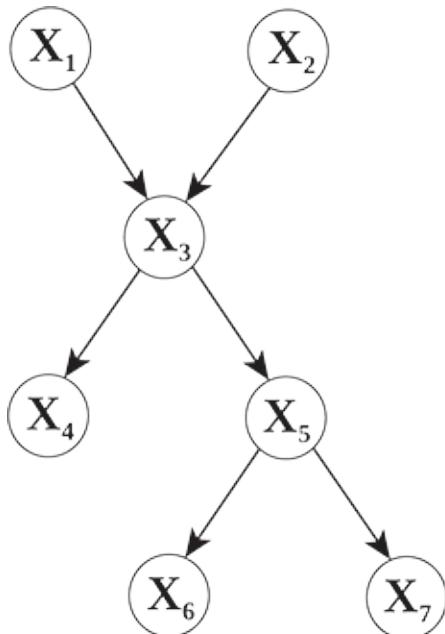
In the context of classification, the sum-product algorithm can be easily modified for computing the conditional probability of the class label y given the set of observed attributes x^\wedge , i.e., $P(y|x^\wedge)$. This basically amounts to computing $P(y, X=x^\wedge)$ in [Equation 4.24](#), where \mathbf{X} is clamped to the observed values x^\wedge . To handle the scenario where some of the random variables are fixed and do not need to be normalized, we consider the following modification.

If X_i is a random variable that is fixed to a specific value x^i , then we can simply modify $\psi(X_i)$ and $\psi(X_i, X_j)$ as follows:

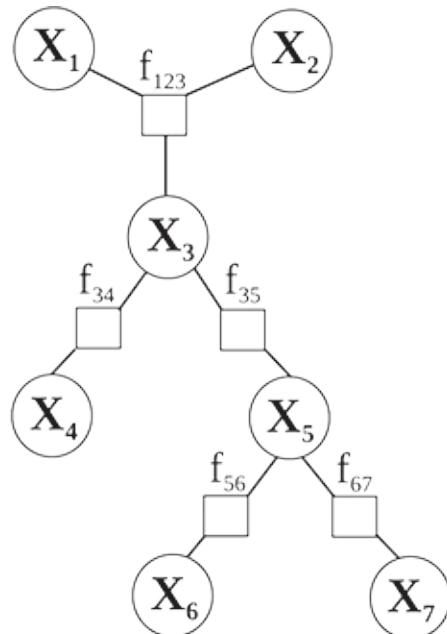
$$\psi(X_i) = \begin{cases} 1, & \text{if } X_i = x^i \\ 0, & \text{otherwise.} \end{cases} \quad (4.35)$$

$$\psi(X_i, X_j) = \begin{cases} P(X_i|x^i), & \text{if } X_i = x^i \\ 0, & \text{otherwise.} \end{cases} \quad (4.36)$$

We can run the sum-product algorithm using these modified values for every observed variable and thus compute $P(y, X=x^\wedge)$.



(a) Poly-tree.



(b) Factor graph.

Figure 4.18.

Example of a poly-tree and its corresponding factor graph.

Generalizations for Non-Tree Graphs

The sum-product algorithm is guaranteed to optimally converge in the case of trees using a single run of message passing in both directions of every edge. This is because any two nodes in a tree have a unique path for the transmission of messages. Furthermore, since every node in a tree has a single parent, the joint probability involves only factors of at most two variables. Hence, it is sufficient to consider potentials over edges and not other generic substructures in the graph.

Both of the previous properties are violated in graphs that are not trees, thus making it difficult to directly apply the sum-product algorithm for making inferences. However, a number of variants of the sum-product algorithm have been devised to perform inferences on a broader family of graphs than trees. Many of these variants transform the original graph into an alternative tree-based representation, and then apply the sum-product algorithm on the transformed tree. In this section, we briefly discuss one such transformations known as **factor graphs**.

Factor graphs are useful for making inferences over graphs that violate the condition that every node has a single parent. Nonetheless, they still require the absence of multiple paths between any two nodes, to guarantee convergence. Such graphs are known as **poly-trees**. An example of a poly-tree is shown in [Figure 4.18\(a\)](#).

A poly-tree can be transformed into a tree-based representation with the help of factor graphs. These graphs consist of two types of nodes, variables nodes (that are represented using circles) and factor nodes (that are represented

using squares). The factor nodes represent conditional independence relationships among the variables of the poly-tree. In particular, every probability table can be represented as a factor node. The edges in a factor graph are undirected in nature and relate a variable node to a factor node if the variable is involved in the probability table corresponding to the factor node. [Figure 4.18\(b\)](#) presents the factor graph representation of the poly-tree shown in [Figure 4.18\(a\)](#).

Note that the factor graph of a poly-tree always forms a tree-like structure, where there is a unique path of influence between any two nodes in the factor graph. Hence, we can apply a modified form of sum-product algorithm to transmit messages between variable nodes and factor nodes, which is guaranteed to converge to optimal values.

Learning Model Parameters

In all our previous discussions on Bayesian networks, we had assumed that the topology of the Bayesian network and the values in the probability tables of every node were already known. In this section, we discuss approaches for learning both the topology and the probability table values of a Bayesian network from the training data.

Let us first consider the case where the topology of the network is known and we are only required to compute the probability tables. If there are no unobserved variables in the training data, then we can easily compute the probability table for $P(X_i|pa(X_i))$, by counting the fraction of training instances for every value of X_i and every combination of values in $pa(X_i)$. However, if there are unobserved variables in X_i or $pa(X_i)$, then computing the fraction of training instances for such variables is non-trivial and requires the use of advanced techniques such as the Expectation-Maximization algorithm (described later in [Chapter 8](#)).

Learning the structure of the Bayesian network is a much more challenging task than learning the probability tables. Although there are some scoring approaches that attempt to find a graph structure that maximizes the training likelihood, they are often computationally infeasible when the graph is large. Hence, a common approach for constructing Bayesian networks is to use the subjective knowledge of domain experts.

4.5.3 Characteristics of Bayesian Networks

1. Bayesian networks provide a powerful approach for representing probabilistic relationships between attributes and class labels with the help of graphical models. They are able to capture complex forms of dependencies among variables. Apart from encoding prior beliefs, they are also able to model the presence of latent (unobserved) factors as hidden variables in the graph. Bayesian networks are thus quite expressive and provide predictive as well as descriptive insights about the behavior of attributes and class labels.
2. Bayesian networks can easily handle the presence of correlated or redundant attributes, as opposed to the naïve Bayes classifier. This is because Bayesian networks do not use the naïve Bayes assumption about conditional independence, but instead are able to express richer forms of conditional independence.
3. Similar to the naïve Bayes classifier, Bayesian networks are also quite robust to the presence of noise in the training data. Further, they can handle missing values during training as well as testing. If a test instance contains an attribute X_i with a missing value, then a Bayesian network can perform inference by treating X_i as an unobserved node

and marginalizing out its effect on the target class. Hence, Bayesian networks are well-suited for handling incompleteness in the data, and can work with partial information. However, unless the pattern with which missing values occurs is completely random, then their presence will likely introduce some degree of error and/or bias into the analysis.

4. Bayesian networks are robust to irrelevant attributes that contain no discriminatory information about the class labels. Such attributes show no impact on the conditional probability of the target class, and are thus rightfully ignored.
5. Learning the structure of a Bayesian network is a cumbersome task that often requires assistance from expert knowledge. However, once the structure has been decided, learning the parameters of the network can be quite straightforward, especially if all the variables in the network are observed.
6. Due to its additional ability of representing complex forms of relationships, Bayesian networks are more susceptible to overfitting as compared to the naïve Bayes classifier. Furthermore, Bayesian networks typically require more training instances for effectively learning the probability tables than the naïve Bayes classifier.
7. Although the sum-product algorithm provides computationally efficient techniques for performing inference over tree-like graphs, the complexity of the approach increase significantly when dealing with generic graphs of large sizes. In situations where exact inference is computationally infeasible, it is quite common to use approximate inference techniques.

4.6 Logistic Regression

The naïve Bayes and the Bayesian network classifiers described in the previous sections provide different ways of estimating the conditional probability of an instance \underline{x} given class y , $P(x|y)$. Such models are known as **probabilistic generative models**. Note that the conditional probability $P(x|y)$ essentially describes the behavior of instances in the attribute space that are *generated* from class y . However, for the purpose of making predictions, we are finally interested in computing the posterior probability $P(y|x)$. For example, computing the following ratio of posterior probabilities is sufficient for inferring class labels in a binary classification problem:

$$P(y=1|x)P(y=0|x)$$

This ratio is known as the **odds**. If this ratio is greater than 1, then \underline{x} is classified as $y=1$. Otherwise, it is assigned to class $y=0$. Hence, one may simply learn a model of the odds based on the attribute values of training instances, without having to compute $P(x|y)$ as an intermediate quantity in the Bayes theorem.

Classification models that directly assign class labels without computing class-conditional probabilities are called **discriminative models**. In this section, we present a probabilistic discriminative model known as **logistic regression**, which directly estimates the odds of a data instance \underline{x} using its attribute values. The basic idea of logistic regression is to use a linear predictor, $z=w^T x + b$, for representing the odds of \underline{x} as follows:

$$P(y=1|x)P(y=0|x)=ez=ew^T x+b, \quad (4.37)$$

where w and b are the parameters of the model and a^T denotes the transpose of a vector a . Note that if $w^T x + b > 0$, then x belongs to class 1 since its odds is greater than 1. Otherwise, x belongs to class 0.

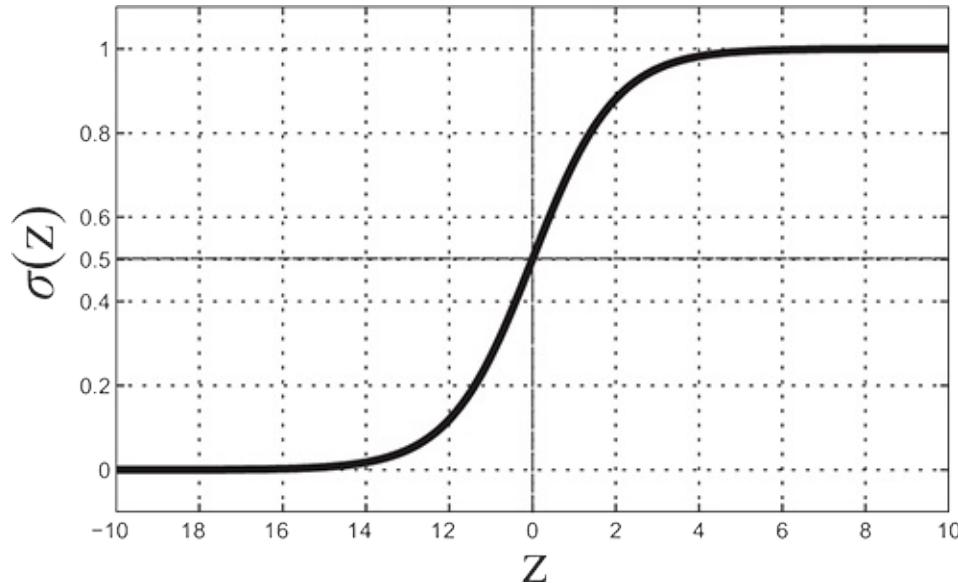


Figure 4.19.

Plot of sigmoid (logistic) function, $\sigma(z)$.

Since $P(y=0|x) + P(y=1|x) = 1$, we can re-write [Equation 4.37](#) as

$$P(y=1|x) \cdot 1 - P(y=1|x) = e^z.$$

This can be further simplified to express $P(y=1|x)$ as a function of z .

$$P(y=1|x) = 1/(1 + e^{-z}) = \sigma(z), \quad (4.38)$$

where the function $\sigma(\cdot)$ is known as the logistic or sigmoid function. [Figure 4.19](#) shows the behavior of the sigmoid function as we vary z . We can see that $\sigma(z) \geq 0.5$ only when $z \geq 0$. We can also derive $P(y=0|x)$ using $\sigma(z)$ as follows:

$$P(y=0|x) = 1 - \sigma(z) = 1/(1 + e^{-z}) \quad (4.39)$$

Hence, if we have learned a suitable value of parameters w and b , we can use Equations 4.38 and 4.39 to estimate the posterior probabilities of any data instance x and determine its class label.

4.6.1 Logistic Regression as a Generalized Linear Model

Since the posterior probabilities are real-valued, their estimation using the previous equations can be viewed as solving a regression problem. In fact, logistic regression belongs to a broader family of statistical regression models, known as **generalized linear models** (GLM). In these models, the target variable y is considered to be generated from a probability distribution $P(y|x)$, whose mean μ can be estimated using a link function $g(\cdot)$ as follows:

$$g(\mu) = z = w^T x + b. \quad (4.40)$$

For binary classification using logistic regression, y follows a Bernoulli distribution (y can either be 0 or 1) and μ is equal to $P(y=1|x)$. The link function $g(\cdot)$ of logistic regression, called the logit function, can thus be represented as

$$g(\mu) = \ln(\mu/(1-\mu)).$$

Depending on the choice of link function $g(\cdot)$ and the form of probability distribution $P(y|x)$, GLMs are able to represent a broad family of regression models, such as linear regression and Poisson regression. They require

different approaches for estimating their model parameters, (w , b). In this chapter, we will only discuss approaches for estimating the model parameters of logistic regression, although methods for estimating parameters of other types of GLMs are often similar (and sometimes even simpler). (See Bibliographic Notes for more details on GLMs.)

Note that even though logistic regression has relationships with regression models, it is a classification model since the computed posterior probabilities are eventually used to determine the class label of a data instance.

4.6.2 Learning Model Parameters

The parameters of logistic regression, (w , b), are estimated during training using a statistical approach known as the maximum likelihood estimation (MLE) method. This method involves computing the likelihood of observing the training data given (w , b), and then determining the model parameters (w^* , b^*) that yield maximum likelihood.

Let $D.\text{train}=\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ denote a set of n training instances, where y_i is a binary variable (0 or 1). For a given training instance x_i , we can compute its posterior probabilities using [Equations 4.38](#) and [4.39](#). We can then express the likelihood of observing y_i given x_i , w , and b as

$$P(y_i|x_i, w, b) = P(y=1|x_i)y_i \times P(y=0|x_i)1-y_i, = (\sigma(z_i))y_i \times (1-\sigma(z_i))1-y_i, = (\sigma(w^T x_i + b))y_i \times (1-\sigma(w^T x_i + b))1-y_i, \quad (4.41)$$

where $\sigma(\cdot)$ is the sigmoid function as described above, [Equation 4.41](#) basically means that the likelihood $P(y_i|x_i, w, b)$ is equal to $P(y=1|x_i)$ when

$y_i=1$, and equal to $P(y=0|x_i)$ when $y_i=0$. The likelihood of all training instances, $L(w, b)$, can then be computed by taking the product of individual likelihoods (assuming independence among training instances) as follows:

$$L(w, b) = \prod_{i=1}^n P(y_i|x_i, w, b) = \prod_{i=1}^n P(y_i=1|x_i)^{y_i} \times P(y_i=0|x_i)^{1-y_i}. \quad (4.42)$$

The previous equation involves multiplying a large number of probability values, each of which are smaller than or equal to 1. Since this naïve computation can easily become numerically unstable when n is large, a more practical approach is to consider the negative logarithm (to base e) of the likelihood function, also known as the **cross entropy function**:

$$\begin{aligned} -\log L(w, b) &= -\sum_{i=1}^n y_i \log(P(y_i=1|x_i)) + (1-y_i) \log(P(y_i=0|x_i)). = \\ &= -\sum_{i=1}^n y_i \log(\sigma(w^T x_i + b)) + (1-y_i) \log(1-\sigma(w^T x_i + b)). \end{aligned}$$

The cross entropy is a loss function that measures how unlikely it is for the training data to be generated from the logistic regression model with parameters (w, b) . Intuitively, we would like to find model parameters (w^*, b^*) that result in the lowest cross entropy, $-\log L(w^*, b^*)$.

$$(w^*, b^*) = \operatorname{argmin}_{(w, b)} E(w, b) = \operatorname{argmin}_{(w, b)} -\log L(w, b) \quad (4.43)$$

where $E(w, b) = -\log L(w, b)$ is the loss function. It is worth emphasizing that $E(w, b)$ is a convex function, i.e., any minima of $E(w, b)$ will be a global minima. Hence, we can use any of the standard convex optimization techniques to solve **Equation 4.43**, which are mentioned in Appendix E. Here, we briefly describe the Newton-Raphson method that is commonly used for estimating the parameters of logistic regression. For ease of representation, we will use a single vector to describe $\tilde{w} = (w^T \ b)^T$, which is of size one greater than w . Similarly, we will consider the concatenated feature vector $\tilde{x} = (x^T \ 1)^T$, such that the linear predictor $z = w^T x + b$ can be succinctly

written as $z = w^T x$. Also, the concatenation of all training labels, y_1 to y_n , will be represented as y , the set consisting of $\sigma(z_1)$ to $\sigma(z_n)$ will be represented as σ , and the concatenation of x^1 to x^n will be represented as X .

The Newton-Raphson is an iterative method for finding w^* that uses the following equation to update the model parameters at every iteration:

$$w^{\text{new}} = w^{\text{old}} - H^{-1} \nabla E(w^{\text{old}}), \quad (4.44)$$

where $\nabla E(w^{\text{old}})$ and H are the first- and second-order derivatives of the loss function $E(w^{\text{old}})$ with respect to w^{old} , respectively. The key intuition behind

Equation 4.44 is to move the model parameters in the direction of maximum gradient, such that w^{old} takes larger steps when $\nabla E(w^{\text{old}})$ is large. When w^{old} arrives at a minima after some number of iterations, then $\nabla E(w^{\text{old}})$ would become equal to 0 and thus result in convergence. Hence, we start with some initial values of w^{old} (either randomly assigned or set to 0) and use **Equation 4.44** to iteratively update w^{old} till there are no significant changes in its value (beyond a certain threshold).

The first-order derivative of $E(w^{\text{old}})$ is given by

$$\nabla E(w^{\text{old}}) = -\sum_{i=1}^n y_i x^i (1 - \sigma(w^T x^i)) - (1 - y_i) x^i \sigma(w^T x^i), = -\sum_{i=1}^n n(\sigma(w^T x^i) - y_i) x^i, = X^T (\sigma - y), \quad (4.45)$$

where we have used the fact that $d\sigma(z)/dz = \sigma(z)(1 - \sigma(z))$. Using $\nabla E(w^{\text{old}})$, we can compute the second-order derivative of $E(w^{\text{old}})$ as

$$H = \nabla \nabla E(w^{\text{old}}) = \sum_{i=1}^n n \sigma(w^T x^i) (1 - \sigma(w^T x^i)) x^i x^i T = X^T R X, \quad (4.46)$$

where R is a diagonal matrix whose i^{th} diagonal element $R_{ii} = \sigma_i(1 - \sigma_i)$. We can now use the first- and second-order derivatives of $E(w^{\text{old}})$ in **Equation 4.44** to

obtain the following update equation at the k^{th} iteration:

$$\tilde{w}(k+1) = \tilde{w}(k) - (\tilde{X}^T R_k \tilde{X})^{-1} \tilde{X}^T (\sigma_k - y) \quad (4.47)$$

where the subscript k under R_k and σ_k refers to using $\tilde{w}(k)$ to compute both terms.

4.6.3 Characteristics of Logistic Regression

1. Logistic Regression is a discriminative model for classification that directly computes the poster probabilities without making any assumption about the class conditional probabilities. Hence, it is quite generic and can be applied in diverse applications. It can also be easily extended to multiclass classification, where it is known as **multinomial logistic regression**. However, its expressive power is limited to learning only linear decision boundaries.
2. Because there are different weights (parameters) for every attribute, the learned parameters of logistic regression can be analyzed to understand the relationships between attributes and class labels.
3. Because logistic regression does not involve computing densities and distances in the attribute space, it can work more robustly even in high-dimensional settings than distance-based methods such as nearest neighbor classifiers. However, the objective function of logistic regression does not involve any term relating to the complexity of the model. Hence, logistic regression does not provide a way to make a trade-off between model complexity and training performance, as compared to other classification models such as support vector

machines. Nevertheless, variants of logistic regression can easily be developed to account for model complexity, by including appropriate terms in the objective function along with the cross entropy function.

4. Logistic regression can handle irrelevant attributes by learning weight parameters close to 0 for attributes that do not provide any gain in performance during training. It can also handle interacting attributes since the learning of model parameters is achieved in a joint fashion by considering the effects of all attributes together. Furthermore, if there are redundant attributes that are duplicates of each other, then logistic regression can learn equal weights for every redundant attribute, without degrading classification performance. However, the presence of a large number of irrelevant or redundant attributes in high-dimensional settings can make logistic regression susceptible to model overfitting.
5. Logistic regression cannot handle data instances with missing values, since the posterior probabilities are only computed by taking a weighted sum of all the attributes. If there are missing values in a training instance, it can be discarded from the training set. However, if there are missing values in a test instance, then logistic regression would fail to predict its class label.

4.7 Artificial Neural Network (ANN)

Artificial neural networks (ANN) are powerful classification models that are able to learn highly complex and nonlinear decision boundaries purely from the data. They have gained widespread acceptance in several applications such as vision, speech, and language processing, where they have been repeatedly shown to outperform other classification models (and in some cases even human performance). Historically, the study of artificial neural networks was inspired by attempts to emulate biological neural systems. The human brain consists primarily of nerve cells called **neurons**, linked together with other neurons via strands of fiber called **axons**. Whenever a neuron is stimulated (e.g., in response to a stimuli), it transmits nerve activations via axons to other neurons. The receptor neurons collect these nerve activations using structures called **dendrites**, which are extensions from the cell body of the neuron. The strength of the contact point between a dendrite and an axon, known as a **synapse**, determines the connectivity between neurons. Neuroscientists have discovered that the human brain learns by changing the strength of the synaptic connection between neurons upon repeated stimulation by the same impulse.

The human brain consists of approximately 100 billion neurons that are interconnected in complex ways, making it possible for us to learn new tasks and perform regular activities. Note that a single neuron only performs a simple modular function, which is to respond to the nerve activations coming from sender neurons connected at its dendrite, and transmit its activation to receptor neurons via axons. However, it is the composition of these simple functions that together is able to express complex functions. This idea is at the basis of constructing artificial neural networks.

Analogous to the structure of a human brain, an artificial neural network is composed of a number of processing units, called nodes, that are connected with each other via directed links. The nodes correspond to neurons that perform the basic units of computation, while the directed links correspond to connections between neurons, consisting of axons and dendrites. Further, the weight of a directed link between two neurons represents the strength of the synaptic connection between neurons. As in biological neural systems, the primary objective of ANN is to adapt the weights of the links until they fit the input-output relationships of the underlying data.

The basic motivation behind using an ANN model is to extract useful features from the original attributes that are most relevant for classification. Traditionally, feature extraction has been achieved by using dimensionality reduction techniques such as PCA (introduced in Chapter 2), which show limited success in extracting nonlinear features, or by using hand-crafted features provided by domain experts. By using a complex combination of inter-connected nodes, ANN models are able to extract much richer sets of features, resulting in good classification performance. Moreover, ANN models provide a natural way of representing features at multiple levels of abstraction, where complex features are seen as compositions of simpler features. In many classification problems, modeling such a hierarchy of features turns out to be very useful. For example, in order to detect a human face in an image, we can first identify low-level features such as sharp edges with different gradients and orientations. These features can then be combined to identify facial parts such as eyes, nose, ears, and lips. Finally, an appropriate arrangement of facial parts can be used to correctly identify a human face. ANN models provide a powerful architecture to represent a hierarchical abstraction of features, from lower levels of abstraction (e.g., edges) to higher levels (e.g., facial parts).

Artificial neural networks have had a long history of developments spanning over five decades of research. Although classical models of ANN suffered from several challenges that hindered progress for a long time, they have re-emerged with widespread popularity because of a number of recent developments in the last decade, collectively known as **deep learning**. In this section, we examine classical approaches for learning ANN models, starting from the simplest model called **perceptrons** to more complex architectures called **multi-layer neural networks**. In the next section, we discuss some of the recent advancements in the area of ANN that have made it possible to effectively learn modern ANN models with deep architectures.

4.7.1 Perceptron

A perceptron is a basic type of ANN model that involves two types of nodes: input nodes, which are used to represent the input attributes, and an output node, which is used to represent the model output. [Figure 4.20](#) illustrates the basic architecture of a perceptron that takes three input attributes, x_1 , x_2 , and x_3 , and produces a binary output y . The input node corresponding to an attribute x_i is connected via a weighted link w_i to the output node. The weighted link is used to emulate the strength of a synaptic connection between neurons.

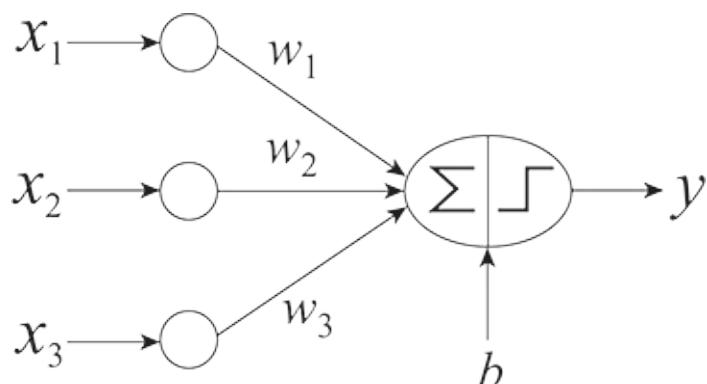


Figure 4.20.

Basic architecture of a perceptron.

The output node is a mathematical device that computes a weighted sum of its inputs, adds a bias factor b to the sum, and then examines the sign of the result to produce the output y^{\wedge} as follows:

$$y^{\wedge} = \begin{cases} 1, & \text{if } w^T x + b > 0 \\ -1, & \text{otherwise.} \end{cases} \quad (4.48)$$

To simplify notations, w and b can be concatenated to form $\tilde{w} = (w^T \ b)^T$, while x can be appended with 1 at the end to form $\tilde{x} = (x^T \ 1)^T$. The output of the perceptron y^{\wedge} can then be written:

$$y^{\wedge} = \text{sign}(\tilde{w}^T \tilde{x}),$$

where the sign function acts as an **activation function** by providing an output value of +1 if the argument is positive and -1 if its argument is negative.

Learning the Perceptron

Given a training set, we are interested in learning parameters \tilde{w} such that y^{\wedge} closely resembles the true y of training instances. This is achieved by using the perceptron learning algorithm given in [Algorithm 4.3](#). The key computation for this algorithm is the iterative weight update formula given in Step 8 of the algorithm:

$$w_j(k+1) = w_j(k) + \lambda(y_i - y_i^{\wedge}(k))x_{ij}, \quad (4.49)$$

where $w(k)$ is the weight parameter associated with the i^{th} input link after the k^{th} iteration, λ is a parameter known as the **learning rate**, and x_{ij} is the value

of the j^{th} attribute of the training example x_i . The justification for [Equation 4.49](#) is rather intuitive. Note that $(y_i - \hat{y}_i)$ captures the discrepancy between y_i and \hat{y}_i , such that its value is 0 only when the true label and the predicted output match. Assume x_{ij} is positive. If $\hat{y} = 0$ and $y = 1$, then w_j is increased at the next iteration so that $w^T x_i$ can become positive. On the other hand, if $\hat{y} = 1$ and $y = 0$, then w_j is decreased so that $w^T x_i$ can become negative. Hence, the weights are modified at every iteration to reduce the discrepancies between \hat{y} and y across all training instances. The learning rate λ , a parameter whose value is between 0 and 1, can be used to control the amount of adjustments made in each iteration. The algorithm halts when the average number of discrepancies are smaller than a threshold γ .

Algorithm 4.3 Perceptron learning algorithm.

```

1: Let  $D.\text{train}=\{(x^i, y_i) | i=1, 2, \dots, n\}$  be the set of training
instances.

2: Set  $k \leftarrow 0$ .

3: Initialize the weight vector  $w^*(0)$  with random values.

4: repeat
5:   for each training instance  $(x^i, y_i) \in D.\text{train}$  do
6:     Compute the predicted output  $\hat{y}_i(k)$  using  $w^*(k)$ .
7:     for each weight component  $w_j$  do
8:       Update the weight,  $w_j(k+1) = w_j(k) + \lambda(y_i - \hat{y}_i(k))x_{ij}$ .
9:     end for
10:    Update  $k \leftarrow k + 1$ .
11: until  $\sum_{i=1}^n |y_i - \hat{y}_i(k)| / n$  is less than a threshold  $\gamma$ 
```

The perceptron is a simple classification model that is designed to learn linear decision boundaries in the attribute space. [Figure 4.21](#) shows the decision

boundary obtained by applying the perceptron learning algorithm to the data set provided on the left of the figure. However, note that there can be multiple decision boundaries that can separate the two classes, and the perceptron arbitrarily learns one of these boundaries depending on the random initial values of parameters. (The selection of the optimal decision boundary is a problem that will be revisited in the context of support vector machines in [Section 4.9](#).) Further, the perceptron learning algorithm is only guaranteed to converge when the classes are linearly separable. However, if the classes are not linearly separable, the algorithm fails to converge. [Figure 4.22](#) shows an example of a nonlinearly separable data given by the XOR function. The perceptron cannot find the right solution for this data because there is no linear decision boundary that can perfectly separate the training instances. Thus, the stopping condition at line 12 of [Algorithm 4.3](#) would never be met and hence, the perceptron learning algorithm would fail to converge. This is a major limitation of perceptrons since real-world classification problems often involve nonlinearly separable classes.

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

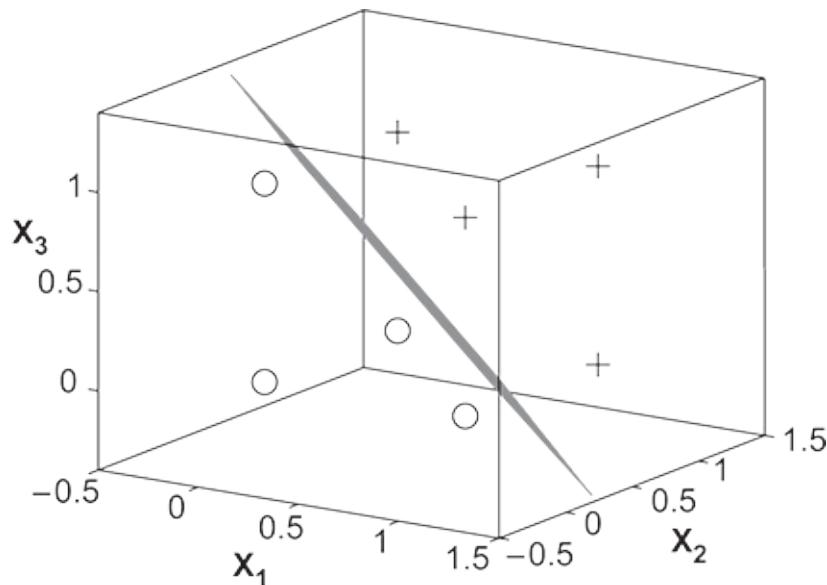


Figure 4.21.

Perceptron decision boundary for the data given on the left (+ represents a positively labeled instance while o represents a negatively labeled instance).

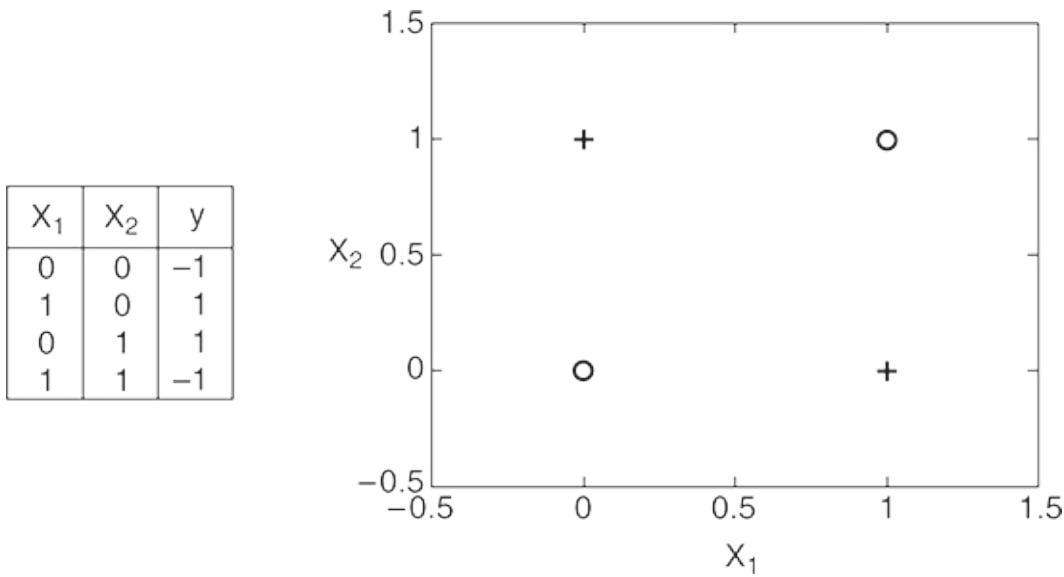


Figure 4.22.

XOR classification problem. No linear hyperplane can separate the two classes.

4.7.2 Multi-layer Neural Network

A multi-layer neural network generalizes the basic concept of a perceptron to more complex architectures of nodes that are capable of learning nonlinear decision boundaries. A generic architecture of a multi-layer neural network is shown in [Figure 4.23](#) where the nodes are arranged in groups called layers. These layers are commonly organized in the form of a chain such that every layer operates on the outputs of its preceding layer. In this way, the layers represent different levels of *abstraction* that are applied on the input features in a sequential manner. The composition of these abstractions generates the final output at the last layer, which is used for making predictions. In the following, we briefly describe the three types of layers used in multi-layer neural networks.

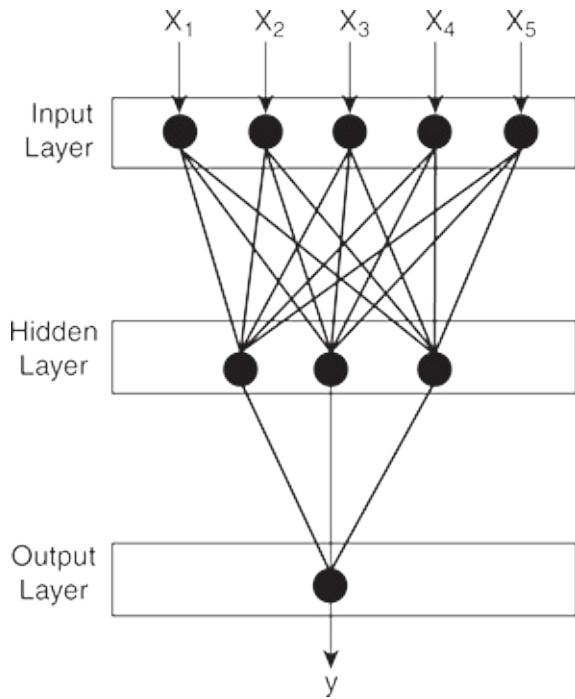
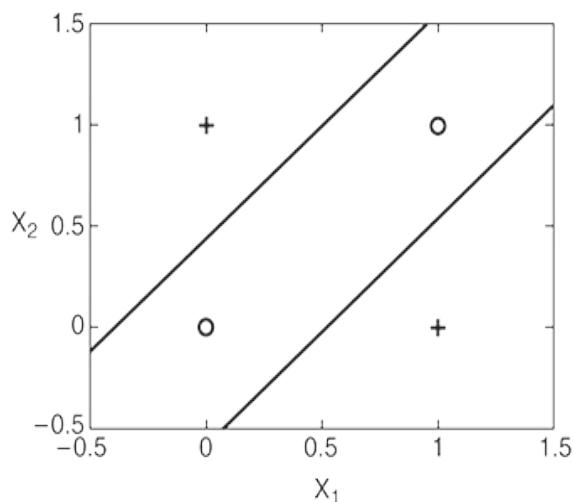


Figure 4.23.

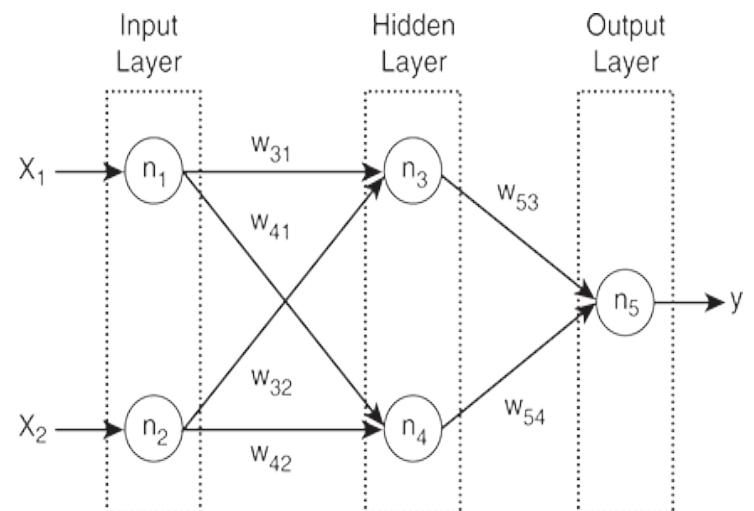
Example of a multi-layer artificial neural network (ANN).

The first layer of the network, called the **input layer**, is used for representing inputs from attributes. Every numerical or binary attribute is typically represented using a single node on this layer, while a categorical attribute is either represented using a different node for each categorical value, or by encoding the k -ary attribute using $\lceil \log_2 k \rceil$ input nodes. These inputs are fed into intermediary layers known as **hidden layers**, which are made up of processing units known as hidden nodes. Every hidden node operates on signals received from the input nodes or hidden nodes at the preceding layer, and produces an activation value that is transmitted to the next layer. The final layer is called the **output layer** and processes the activation values from its preceding layer to produce predictions of output variables. For binary classification, the output layer contains a single node representing the binary class label. In this architecture, since the signals are propagated only in the forward direction from the input layer to the output layer, they are also called **feedforward neural networks**.

A major difference between multi-layer neural networks and perceptrons is the inclusion of hidden layers, which dramatically improves their ability to represent arbitrarily complex decision boundaries. For example, consider the XOR problem described in the previous section. The instances can be classified using two hyperplanes that partition the input space into their respective classes, as shown in [Figure 4.24\(a\)](#). Because a perceptron can create only one hyperplane, it cannot find the optimal solution. However, this problem can be addressed by using a hidden layer consisting of two nodes, as shown in [Figure 4.24\(b\)](#). Intuitively, we can think of each hidden node as a perceptron that tries to construct one of the two hyperplanes, while the output node simply combines the results of the perceptrons to yield the decision boundary shown in [Figure 4.24\(a\)](#).



(a) Decision boundary.



(b) Neural network topology.

Figure 4.24.

A two-layer neural network for the XOR problem.

The hidden nodes can be viewed as learning *latent representations* or *features* that are useful for distinguishing between the classes. While the first hidden layer directly operates on the input attributes and thus captures simpler features, the subsequent hidden layers are able to combine them and

construct more complex features. From this perspective, multi-layer neural networks learn a hierarchy of features at different levels of abstraction that are finally combined at the output nodes to make predictions. Further, there are combinatorially many ways we can combine the features learned at the hidden layers of ANN, making them highly expressive. This property chiefly distinguishes ANN from other classification models such as decision trees, which can learn partitions in the attribute space but are unable to combine them in exponential ways.

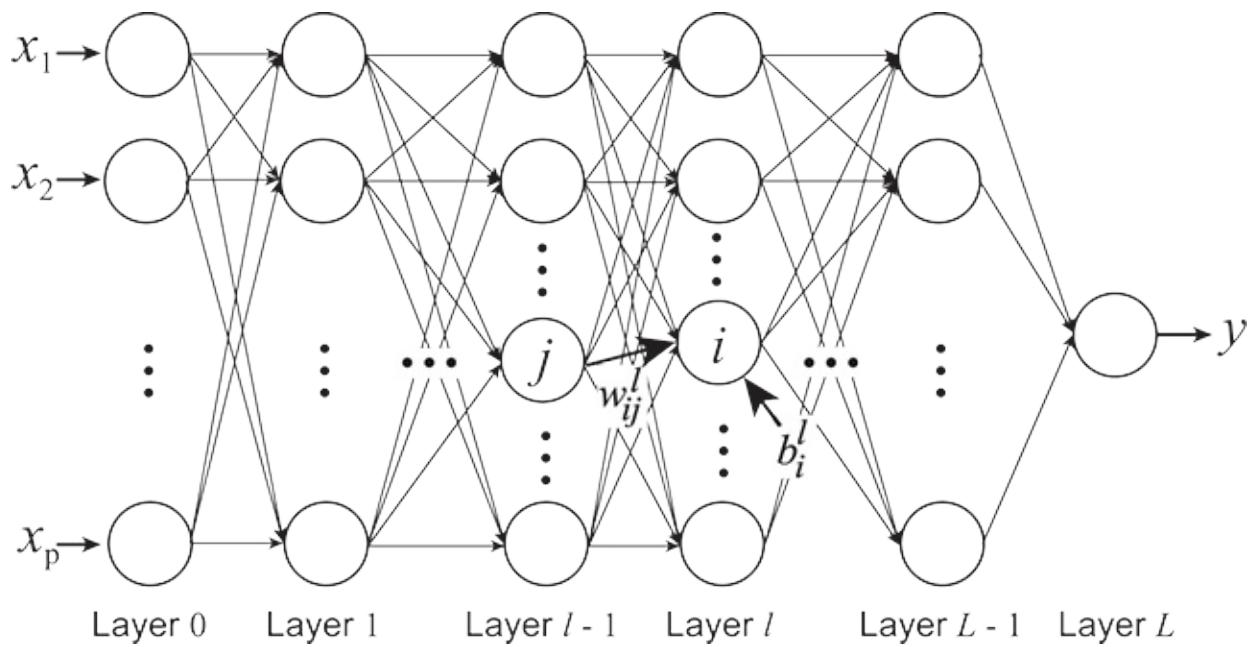


Figure 4.25.

Schematic illustration of the parameters of an ANN model with $(L-1)$ hidden layers.

To understand the nature of computations happening at the hidden and output nodes of ANN, consider the i^{th} node at the l^{th} layer of the network ($l > 0$), where the layers are numbered from 0 (input layer) to L (output layer), as shown in [Figure 4.25](#). The activation value generated at this node, a_{il} , can be represented as a function of the inputs received from nodes at the preceding layer. Let w_{jl} represent the weight of the connection from the j^{th} node at layer

$(l-1)$ to the i^{th} node at layer l . Similarly, let us denote the bias term at this node as b_{jl} . The activation value a_{il} can then be expressed as

$$a_{il} = f(z_{il}) = f(\sum_j w_{ijl} a_{jl} - b_{il}),$$

where z is called the *linear predictor* and $f(\cdot)$ is the activation function that converts z to a . Further, note that, by definition, $a_{j0} = x_j$ at the input layer and $a_L = y^*$ at the output node.

There are a number of alternate activation functions apart from the sign function that can be used in multi-layer neural networks. Some examples include linear, sigmoid (logistic), and hyperbolic tangent functions, as shown in [Figure 4.26](#). These functions are able to produce real-valued and nonlinear activation values. Among these activation functions, the sigmoid $\sigma(\cdot)$ has been widely used in many ANN models, although the use of other types of activation functions in the context of deep learning will be discussed in [Section 4.8](#). We can thus represent a_{il} as

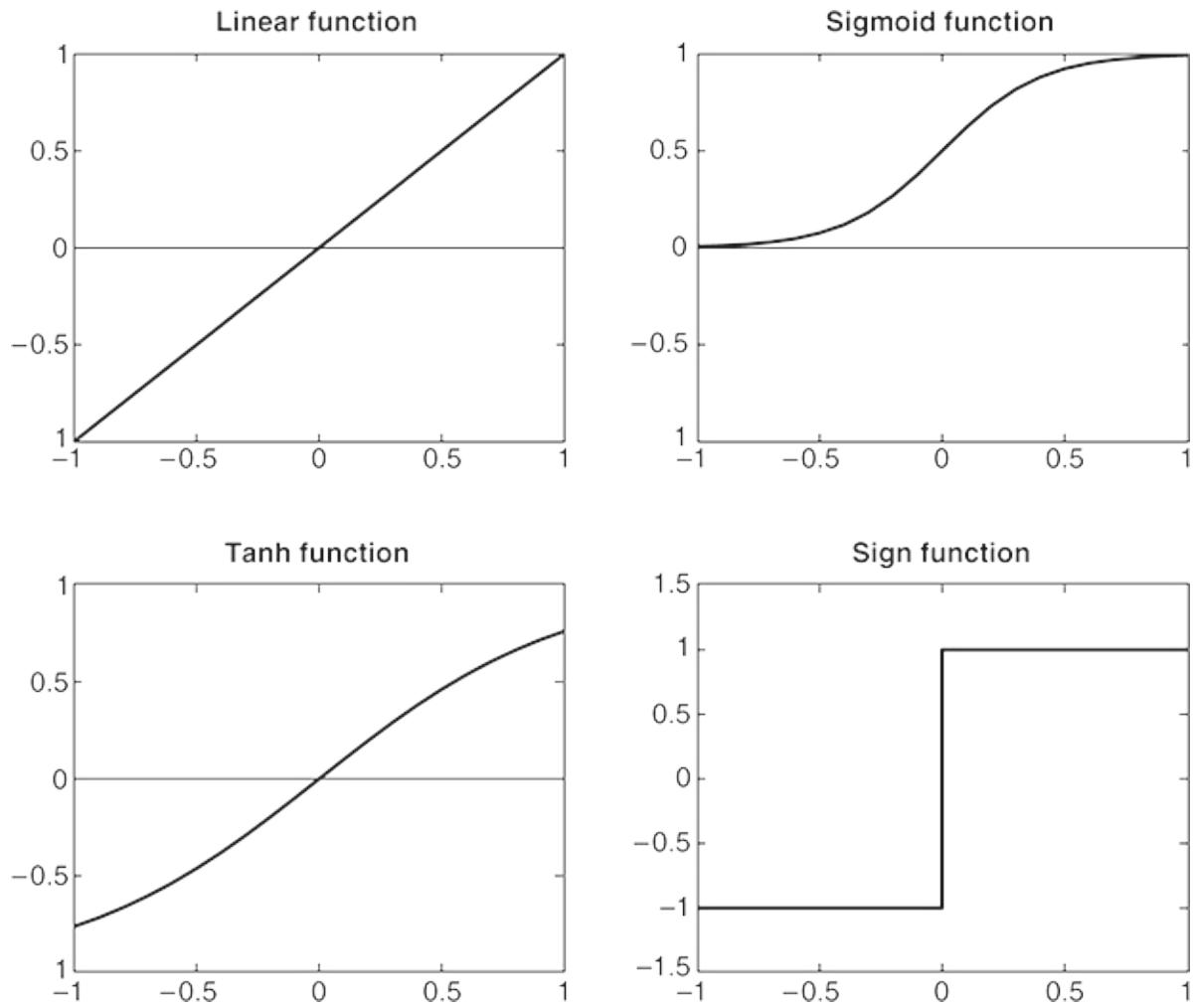


Figure 4.26.

Types of activation functions used in multi-layer neural networks.

$$a_{il} = \sigma(z_{il}) = \frac{1}{1 + e^{-z_{il}}} \quad (4.50)$$

Learning Model Parameters

The weights and bias terms (w , b) of the ANN model are learned during training so that the predictions on training instances match the true labels. This is achieved by using a loss function

$$E(w, b) = \sum_{k=1}^n \text{Loss}(y_k, \hat{y}_k) \quad (4.51)$$

where y_k is the true label of the k th training instance and y^k is equal to a_L , produced by using x_k . A typical choice of the loss function is the **squared loss function**:

$$\text{Loss } (y_k, y^k) = (y_k - y^k)^2. \quad (4.52)$$

Note that $E(w, b)$ is a function of the model parameters (w, b) because the output activation value a_L depends on the weights and bias terms. We are interested in choosing (w, b) that minimizes the training loss $E(w, b)$.

Unfortunately, because of the use of hidden nodes with nonlinear activation functions, $E(w, b)$ is not a convex function of w and b , which means that $E(w, b)$ can have local minima that are not globally optimal. However, we can still apply standard optimization techniques such as the **gradient descent method** to arrive at a locally optimal solution. In particular, the weight parameter w_{ijl} and the bias term b_{il} can be iteratively updated using the following equations:

$$w_{ijl} \leftarrow w_{ijl} - \lambda \partial E \partial w_{ijl}, \quad (4.53)$$

$$b_{il} \leftarrow b_{il} - \lambda \partial E \partial b_{il}, \quad (4.54)$$

where λ is a hyper-parameter known as the learning rate. The intuition behind this equation is to move the weights in a direction that reduces the training loss. If we arrive at a minima using this procedure, the gradient of the training loss will be close to 0, eliminating the second term and resulting in the convergence of weights. The weights are commonly initialized with values drawn randomly from a Gaussian or a uniform distribution.

A necessary tool for updating weights in [Equation 4.53](#) is to compute the partial derivative of E with respect to w_{ijl} . This computation is nontrivial especially at hidden layers ($i < L$), since w_{ijl} does not directly affect $y^k = a_L$ (and

hence the training loss), but has complex chains of influences via activation values at subsequent layers. To address this problem, a technique known as **backpropagation** was developed, which propagates the derivatives backward from the output layer to the hidden layers. This technique can be described as follows.

Recall that the training loss E is simply the sum of individual losses at training instances. Hence the partial derivative of E can be decomposed as a sum of partial derivatives of individual losses.

$$\partial E \partial w_{jl} = \sum_{k=1}^n \partial \text{Loss}(y_k, y^k) \partial w_{jl}.$$

To simplify discussions, we will consider only the derivatives of the loss at the k^{th} training instance, which will be generically represented as $\text{Loss}(y, a_L)$. By using the chain rule of differentiation, we can represent the partial derivatives of the loss with respect to w_{jl} as

$$\partial \text{Loss} \partial w_{jl} = \partial \text{Loss} \partial a_l \times \partial a_l \partial z_{il} \times \partial z_{il} \partial w_{jl}. \quad (4.55)$$

The last term of the previous equation can be written as

$$\partial z_{il} \partial w_{jl} = \partial (\sum_j w_{jl} a_j - 1 + b_l) \partial w_{jl} = a_j - 1.$$

Also, if we use the sigmoid activation function, then

$$\partial a_l \partial z_{il} = \partial \sigma(z_{il}) \partial z_{il} = a_l(1 - a_l).$$

Equation 4.55 can thus be simplified as

$$\partial \text{Loss} \partial w_{jl} = \delta_{il} \times a_l(1 - a_l) \times a_j - 1, \text{ where } \delta_{il} = \partial \text{Loss} \partial a_l. \quad (4.56)$$

A similar formula for the partial derivatives with respect to the bias terms b_{li} is given by

$$\partial \text{Loss} \partial b_{li} = \delta_{il} \times a_{il}(1 - a_{il}). \quad (4.57)$$

Hence, to compute the partial derivatives, we only need to determine δ_{il} . Using a squared loss function, we can easily write δ_L at the output node as

$$\delta_L = \partial \text{Loss} \partial a_L = \partial (y - a_L)^2 \partial a_L = 2(a_L - y). \quad (4.58)$$

However, the approach for computing δ_{jl} at hidden nodes ($l < L$) is more involved. Notice that a_{jl} affects the activation values a_{il+1} of all nodes at the next layer, which in turn influences the loss. Hence, again using the chain rule of differentiation, δ_{jl} can be represented as

$$\delta_{jl} = \partial \text{Loss} \partial a_{jl} = \sum_i (\partial \text{Loss} \partial a_{il+1} \times \partial a_{il+1} \partial a_{jl}) = \sum_i (\partial \text{Loss} \partial a_{il+1} \times \partial a_{il+1} \partial z_{il+1} \times \partial z_{il+1} \partial a_{jl})$$

The previous equation provides a concise representation of the δ_{jl} values at layer l in terms of the δ_{jl+1} values computed at layer $l+1$. Hence, proceeding backward from the output layer L to the hidden layers, we can recursively apply [Equation 4.59](#) to compute δ_{il} at every hidden node. δ_{il} can then be used in [Equations 4.56](#) and [4.57](#) to compute the partial derivatives of the loss with respect to w_{ijl} and b_{il} , respectively. [Algorithm 4.4](#) summarizes the complete approach for learning the model parameters of ANN using backpropagation and gradient descent method.

Algorithm 4.4 Learning ANN using backpropagation and gradient descent.

```

1: Let  $D.\text{train} = \{(x_k, y_k) \mid k = 1, 2, \dots, n\}$  be the set of training
instances.

2: Set counter  $c \leftarrow 0$ .

3: Initialize the weight and bias terms  $(w^{(0)}, b^{(0)})$  with random
values.

4: repeat

5:   for each training instance  $(x_k, y_k) \in D.\text{train}$  do

6:     Compute the set of activations  $(a_{il})_k$  by making a forward
pass using  $x_k$ .

7:     Compute the set  $(\delta_{il})_k$  by backpropagation using Equations
4.58  and 4.59 .
```

8: Compute $(\partial \text{Loss}/\partial w_{ijl}, \partial \text{Loss}/\partial b_{il})_k$ using Equations 4.56

and 4.57 .

9: end for

10: Compute $\partial E/\partial w_{ijl} \leftarrow \sum_{k=1}^n (\partial \text{Loss}/\partial w_{ijl})_k$.

11: Compute $\partial E/\partial b_{il} \leftarrow \sum_{k=1}^n (\partial \text{Loss}/\partial b_{il})_k$.

12: Update $(w^{(c+1)}, b^{(c+1)})$ by gradient descent using Equations
4.53 and 4.54 .

13: Update $c \leftarrow c + 1$.

14: until $(w^{(c+1)}, b^{(c+1)})$ and $(w^{(c)}, b^{(c)})$ converge to the same value

4.7.3 Characteristics of ANN

1. Multi-layer neural networks with at least one hidden layer are **universal approximators**; i.e., they can be used to approximate any target function. They are thus highly expressive and can be used to learn complex decision boundaries in diverse applications. ANN can also be used for multiclass classification and regression problems, by

appropriately modifying the output layer. However, the high model complexity of classical ANN models makes it susceptible to overfitting, which can be overcome to some extent by using deep learning techniques discussed in [Section 4.8.3](#).

2. ANN provides a natural way to represent a hierarchy of features at multiple levels of abstraction. The outputs at the final hidden layer of the ANN model thus represent features at the highest level of abstraction that are most useful for classification. These features can also be used as inputs in other supervised classification models, e.g., by replacing the output node of the ANN by any generic classifier.
3. ANN represents complex high-level features as compositions of simpler lower-level features that are easier to learn. This provides ANN the ability to gradually increase the complexity of representations, by adding more hidden layers to the architecture. Further, since simpler features can be combined in combinatorial ways, the number of complex features learned by ANN is much larger than traditional classification models. This is one of the main reasons behind the high expressive power of deep neural networks.
4. ANN can easily handle irrelevant attributes, by using zero weights for attributes that do not help in improving the training loss. Also, redundant attributes receive similar weights and do not degrade the quality of the classifier. However, if the number of irrelevant or redundant attributes is large, the learning of the ANN model may suffer from overfitting, leading to poor generalization performance.
5. Since the learning of ANN model involves minimizing a non-convex function, the solutions obtained by gradient descent are not guaranteed to be globally optimal. For this reason, ANN has a tendency to get stuck in local minima, a challenge that can be addressed by using deep learning techniques discussed in [Section 4.8.4](#).
6. Training an ANN is a time consuming process, especially when the number of hidden nodes is large. Nevertheless, test examples can be

classified rapidly.

7. Just like logistic regression, ANN can learn in the presence of interacting variables, since the model parameters are jointly learned over all variables together. In addition, ANN cannot handle instances with missing values in the training or testing phase.

4.8 Deep Learning

As described above, the use of hidden layers in ANN is based on the general belief that complex high-level features can be constructed by combining simpler lower-level features. Typically, the greater the number of hidden layers, the deeper the hierarchy of features learned by the network. This motivates the learning of ANN models with long chains of hidden layers, known as **deep neural networks**. In contrast to “shallow” neural networks that involve only a small number of hidden layers, deep neural networks are able to represent features at multiple levels of abstraction and often require far fewer nodes per layer to achieve generalization performance similar to shallow networks.

Despite the huge potential in learning deep neural networks, it has remained challenging to learn ANN models with a large number of hidden layers using classical approaches. Apart from reasons related to limited computational resources and hardware architectures, there have been a number of algorithmic challenges in learning deep neural networks. First, learning a deep neural network with low training error has been a daunting task because of the saturation of sigmoid activation functions, resulting in slow convergence of gradient descent. This problem becomes even more serious as we move away from the output node to the hidden layers, because of the compounded effects of saturation at multiple layers, known as the **vanishing gradient problem**. Because of this reason, classical ANN models have suffered from slow and ineffective learning, leading to poor training and test performance. Second, the learning of deep neural networks is quite sensitive to the initial values of model parameters, chiefly because of the non-convex nature of the optimization function and the slow convergence of gradient descent. Third, deep neural networks with a large number of hidden layers have high model

complexity, making them susceptible to overfitting. Hence, even if a deep neural network has been trained to show low training error, it can still suffer from poor generalization performance.

These challenges have deterred progress in building deep neural networks for several decades and it is only recently that we have started to unlock their immense potential with the help of a number of advances being made in the area of deep learning. Although some of these advances have been around for some time, they have only gained mainstream attention in the last decade, with deep neural networks continually beating records in various competitions and solving problems that were too difficult for other classification approaches.

There are two factors that have played a major role in the emergence of deep learning techniques. First, the availability of larger labeled data sets, e.g., the ImageNet data set contains more than 10 million labeled images, has made it possible to learn more complex ANN models than ever before, without falling easily into the traps of model overfitting. Second, advances in computational abilities and hardware infrastructures, such as the use of graphical processing units (GPU) for distributed computing, have greatly helped in experimenting with deep neural networks with larger architectures that would not have been feasible with traditional resources.

In addition to the previous two factors, there have been a number of algorithmic advancements to overcome the challenges faced by classical methods in learning deep neural networks. Some examples include the use of more responsive combinations of loss functions and activation functions, better initialization of model parameters, novel regularization techniques, more agile architecture designs, and better techniques for model learning and hyper-parameter selection. In the following, we describe some of the deep learning advances made to address the challenges in learning deep neural

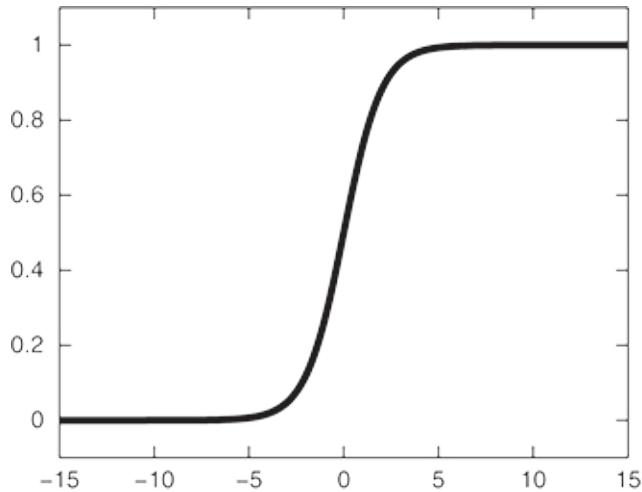
networks. Further details on recent developments in deep learning can be obtained from the Bibliographic Notes.

4.8.1 Using Synergistic Loss Functions

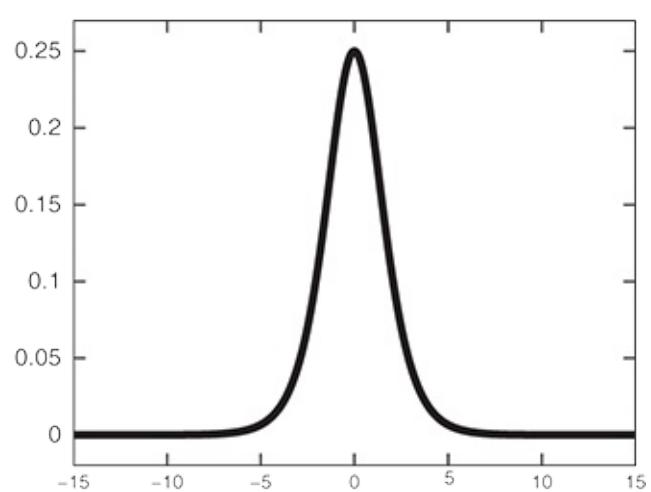
One of the major realizations leading to deep learning has been the importance of choosing appropriate combinations of activation and loss functions. Classical ANN models commonly made use of the sigmoid activation function at the output layer, because of its ability to produce real-valued outputs between 0 and 1, which was combined with a squared loss objective to perform gradient descent. It was soon noticed that this particular combination of activation and loss function resulted in the saturation of output activation values, which can be described as follows.

Saturation of Outputs

Although the sigmoid has been widely-used as an activation function, it easily saturates at high and low values of inputs that are far away from 0. Observe from [Figure 4.27\(a\)](#) that $\sigma(z)$ shows variance in its values only when z is close to 0. For this reason, $\partial\sigma(z)/\partial z$ is non-zero for only a small range of z around 0, as shown in [Figure 4.27\(b\)](#). Since $\partial\sigma(z)/\partial z$ is one of the components in the gradient of loss (see [Equation 4.55](#)), we get a diminishing gradient value when the activation values are far from 0.



(a) $\sigma(z)$.



(b) $\partial\sigma(z)/\partial z$.

Figure 4.27.

Plots of sigmoid function and its derivative.

To illustrate the effect of saturation on the learning of model parameters at the output node, consider the partial derivative of loss with respect to the weight w_{jL} at the output node. Using the squared loss function, we can write this as

$$\frac{\partial \text{Loss}}{\partial w_j} = 2(a_L - y) \times \sigma(z_L)(1 - \sigma(z_L)) \times a_j L - 1. \quad (4.60)$$

In the previous equation, notice that when zL is highly negative, $\sigma(zL)$ (and hence the gradient) is close to 0. On the other hand, when zL is highly positive, $(1-\sigma(zL))$ becomes close to 0, nullifying the value of the gradient. Hence, irrespective of whether the prediction aL matches the true label y or not, the gradient of the loss with respect to the weights is close to 0 whenever zL is highly positive or negative. This causes an unnecessarily slow convergence of the model parameters of the ANN model, often resulting in poor learning.

Note that it is the combination of the squared loss function and the sigmoid activation function at the output node that together results in diminishing

gradients (and thus poor learning) upon saturation of outputs. It is thus important to choose a synergistic combination of loss function and activation function that does not suffer from the saturation of outputs.

Cross entropy loss function

The cross entropy loss function, which was described in the context of logistic regression in [Section 4.6.2](#), can significantly avoid the problem of saturating outputs when used in combination with the sigmoid activation function. The cross entropy loss function of a real-valued prediction $y^{\wedge} \in (0, 1)$ on a data instance with binary label $y \in \{0, 1\}$ can be defined as

$$\text{Loss}(y, y^{\wedge}) = -y \log(y^{\wedge}) - (1-y) \log(1-y^{\wedge}), \quad (4.61)$$

where \log represents the natural logarithm (to base e) and $0 \log(0)=0$ for convenience. The cross entropy function has foundations in information theory and measures the amount of disagreement between y and y^{\wedge} . The partial derivative of this loss function with respect to $y^{\wedge}=a_L$ can be given as

$$\delta L = \partial \text{Loss} / \partial a_L = -ya_L + (1-y)(1-a_L). = (a_L - y)a_L(1-a_L). \quad (4.62)$$

Using this value of δL in [Equation 4.56](#), we can obtain the partial derivative of the loss with respect to the weight w_{jL} at the output node as

$$\partial \text{Loss} / \partial w_{jL} = (a_L - y)a_L(1-a_L) \times a_L(1-a_L) \times a_{jL} - 1. = (a_L - y) \times a_{jL} - 1. \quad (4.63)$$

Notice the simplicity of the previous formula using the cross entropy loss function. The partial derivatives of the loss with respect to the weights at the output node depend only on the difference between the prediction a_L and the true label y . In contrast to [Equation 4.60](#), it does not involve terms such as $\sigma(z_L)(1-\sigma(z_L))$ that can be impacted by saturation of z_L . Hence, the gradients

are high whenever $(aL - y)$ is large, promoting effective learning of the model parameters at the output node. This has been a major breakthrough in the learning of modern ANN models and it is now a common practice to use the cross entropy loss function with sigmoid activations at the output node.

4.8.2 Using Responsive Activation Functions

Even though the cross entropy loss function helps in overcoming the problem of saturating outputs, it still does not solve the problem of saturation at hidden layers, arising due to the use of sigmoid activation functions at hidden nodes. In fact, the effect of saturation on the learning of model parameters is even more aggravated at hidden layers, a problem known as the vanishing gradient problem. In the following, we describe the vanishing gradient problem and the use of a more responsive activation function, called the **rectified linear output unit (ReLU)**, to overcome this problem.

Vanishing Gradient Problem

The impact of saturating activation values on the learning of model parameters increases at deeper hidden layers that are farther away from the output node. Even if the activation in the output layer does not saturate, the repeated multiplications performed as we backpropagate the gradients from the output layer to the hidden layers may lead to decreasing gradients in the hidden layers. This is called the vanishing gradient problem, which has been one of the major hindrances in learning deep neural networks.

To illustrate the vanishing gradient problem, consider an ANN model that consists of a single node at every hidden layer of the network, as shown in [Figure 4.28](#). This simplified architecture involves a single chain of hidden nodes where a single weighted link w_l connects the node at layer $l-1$ to the node at layer l . Using [Equations 4.56](#) and [4.59](#), we can represent the partial derivative of the loss with respect to w_l as

$$\frac{\partial \text{Loss}}{\partial w_l} = \delta_l \times a_l(1-a_l) \times a_{l-1}, \text{ where } \delta_l = 2(a_L - y) \times \prod_{r=1}^{l-1} (a_r + 1)(1-a_r + 1) \times w_r \quad (4.164)$$

Notice that if any of the linear predictors $a_r + 1$ saturates at subsequent layers, then the term $a_r + 1(1-a_r + 1)$ becomes close to 0, thus diminishing the overall gradient. The saturation of activations thus gets compounded and has multiplicative effects on the gradients at hidden layers, making them highly unstable and thus, unsuitable for use with gradient descent. Even though the previous discussion only pertains to the simplified architecture involving a single chain of hidden nodes, a similar argument can be made for any generic ANN architecture involving multiple chains of hidden nodes. Note that the vanishing gradient problem primarily arises because of the use of sigmoid activation function at hidden nodes, which is known to easily saturate especially after repeated multiplications.

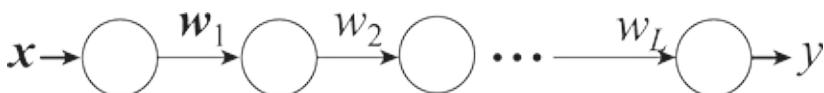


Figure 4.28.

An example of an ANN model with only one node at every hidden layer.

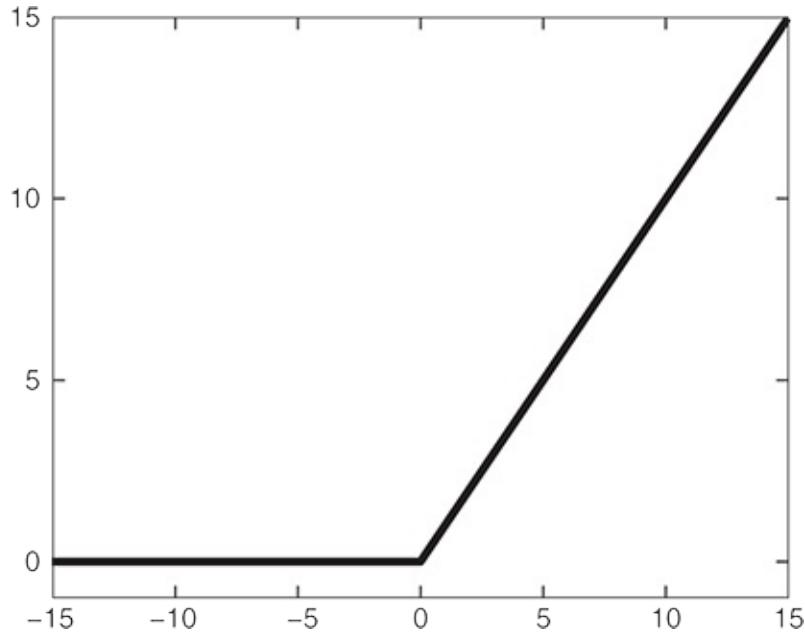


Figure 4.29.

Plot of the rectified linear unit (ReLU) activation function.

Rectified Linear Units (ReLU)

To overcome the vanishing gradient problem, it is important to use an activation function $f(z)$ at the hidden nodes that provides a stable and significant value of the gradient whenever a hidden node is active, i.e., $z>0$. This is achieved by using rectified linear units (ReLU) as activation functions at hidden nodes, which can be defined as

$$a=f(z)=\{z, \text{if } z>0.0, \text{otherwise.} \quad (4.65)$$

The idea of ReLU has been inspired from biological neurons, which are either in an inactive state ($f(z)=0$) or show an activation value proportional to the input. [Figure 4.29](#) shows a plot of the ReLU function. We can see that it is linear with respect to z when $z>0$. Hence, the gradient of the activation value with respect to z can be written as

$$\frac{\partial a}{\partial z} = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases} \quad (4.66)$$

Although $f(z)$ is not differentiable at 0, it is common practice to use $\frac{\partial a}{\partial z} = 0$ when $z=0$. Since the gradient of the ReLU activation function is equal to 1 whenever $z>0$, it avoids the problem of saturation at hidden nodes, even after repeated multiplications. Using ReLU, the partial derivatives of the loss with respect to the weight and bias parameters can be given by

$$\frac{\partial \text{Loss}}{\partial w_{ijl}} = \delta_{il} \times I(z_{il}) \times a_{jl} - 1, \quad (4.67)$$

$$\frac{\partial \text{Loss}}{\partial b_{il}} = \delta_{il} \times I(z_{il}), \text{ where } \delta_{il} = \sum_{j=1}^n (\delta_{il} + 1) \times I(z_{il}+1) \times w_{ijl} + 1, \text{ and } I(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (4.68)$$

Notice that ReLU shows a linear behavior in the activation values whenever a node is active, as compared to the nonlinear properties of the sigmoid function. This linearity promotes better flows of gradients during backpropagation, and thus simplifies the learning of ANN model parameters. The ReLU is also highly responsive at large values of z away from 0, as opposed to the sigmoid activation function, making it more suitable for gradient descent. These differences give ReLU a major advantage over the sigmoid function. Indeed, ReLU is used as the preferred choice of activation function at hidden layers in most modern ANN models.

4.8.3 Regularization

A major challenge in learning deep neural networks is the high model complexity of ANN models, which grows with the addition of hidden layers in the network. This can become a serious concern, especially when the training set is small, due to the phenomena of model overfitting. To overcome this

challenge, it is important to use techniques that can help in reducing the complexity of the learned model, known as **regularization techniques**. Classical approaches for learning ANN models did not have an effective way to promote regularization of the learned model parameters. Hence, they had often been sidelined by other classification methods, such as support vector machines (SVM), which have in-built regularization mechanisms. (SVMs will be discussed in more detail in [Section 4.9](#)).

One of the major advancements in deep learning has been the development of novel regularization techniques for ANN models that are able to offer significant improvements in generalization performance. In the following, we discuss one of the regularization techniques for ANN, known as the **dropout** method, that have gained a lot of attention in several applications.

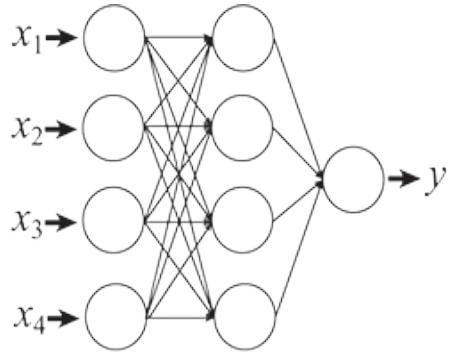
Dropout

The main objective of dropout is to avoid the learning of spurious features at hidden nodes, occurring due to model overfitting. It uses the basic intuition that spurious features often “co-adapt” themselves such that they show good training performance only when used in highly selective combinations. On the other hand, relevant features can be used in a diversity of feature combinations and hence are quite resilient to the removal or modification of other features. The dropout method uses this intuition to break complex “co-adaptations” in the learned features by randomly dropping input and hidden nodes in the network during training.

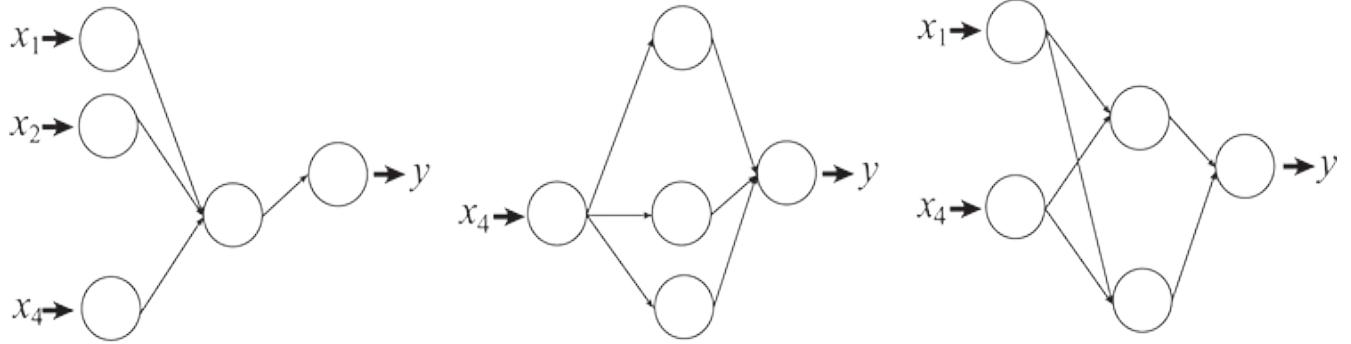
Dropout belongs to a family of regularization techniques that uses the criteria of resilience to random perturbations as a measure of the robustness (and hence, simplicity) of a model. For example, one approach to regularization is to inject noise in the input attributes of the training set and learn a model with the noisy training instances. If a feature learned from the training data is

indeed generalizable, it should not be affected by the addition of noise. Dropout can be viewed as a similar regularization approach that perturbs the information content of the training set not only at the level of attributes but also at multiple levels of abstractions, by dropping input and hidden nodes.

The dropout method draws inspiration from the biological process of gene swapping in sexual reproduction, where half of the genes from both parents are combined together to create the genes of the offspring. This favors the selection of parent genes that are not only useful but can also inter-mingle with diverse combinations of genes coming from the other parent. On the other hand, co-adapted genes that function only in highly selective combinations are soon eliminated in the process of evolution. This idea is used in the dropout method for eliminating spurious co-adapted features. A simplified description of the dropout method is provided in the rest of this section.



(a) Original network.



(b) Sub-networks.

Figure 4.30.

Examples of sub-networks generated in the dropout method using $\gamma=0.5$.

Let (w_k, b_k) represent the model parameters of the ANN model at the k^{th} iteration of the gradient descent method. At every iteration, we randomly select a fraction γ of input and hidden nodes to be dropped from the network, where $\gamma \in (0, 1)$ is a hyper-parameter that is typically chosen to be 0.5. The weighted links and bias terms involving the dropped nodes are then eliminated, resulting in a “thinned” sub-network of smaller size. The model parameters of the sub-network (w_{sk}, b_{sk}) are then updated by computing activation values and performing backpropagation on this smaller sub-network. These updated values are then added back in the original network to

obtain the updated model parameters, (w_{k+1}, b_{k+1}) , to be used in the next iteration.

Figure 4.30 shows some examples of sub-networks that can be generated at different iterations of the dropout method, by randomly dropping input and hidden nodes. Since every sub-network has a different architecture, it is difficult to learn complex co-adaptations in the features that can result in overfitting. Instead, the features at the hidden nodes are learned to be more agile to random modifications in the network structure, thus improving their generalization ability. The model parameters are updated using a different random sub-network at every iteration, till the gradient descent method converges.

Let $(w_{k\max}, b_{k\max})$ denote the model parameters at the last iteration $k\max$ of the gradient descent method. These parameters are finally scaled down by a factor of $(1-\gamma)$, to produce the weights and bias terms of the final ANN model, as follows:

$$(w^*, b^*) = ((1-\gamma) \times w_{k\max}, (1-\gamma) \times b_{k\max})$$

We can now use the complete neural network with model parameters (w^*, b^*) for testing. The dropout method has been shown to provide significant improvements in the generalization performance of ANN models in a number of applications. It is computationally cheap and can be applied in combination with any of the other deep learning techniques. It also has a number of similarities with a widely-used ensemble learning method known as **bagging**, which learns multiple models using random subsets of the training set, and then uses the *average output* of all the models to make predictions. (Bagging will be presented in more detail later in **Section 4.10.4**). In a similar vein, it can be shown that the predictions of the final network learned using dropout approximates the average output of all possible 2^n sub-networks that can be

formed using n nodes. This is one of the reasons behind the superior regularization abilities of dropout.

4.8.4 Initialization of Model Parameters

Because of the non-convex nature of the loss function used by ANN models, it is possible to get stuck in locally optimal but globally inferior solutions. Hence, the initial choice of model parameter values plays a significant role in the learning of ANN by gradient descent. The impact of poor initialization is even more aggravated when the model is complex, the network architecture is deep, or the classification task is difficult. In such cases, it is often advisable to first learn a simpler model for the problem, e.g., using a single hidden layer, and then incrementally increase the complexity of the model, e.g., by adding more hidden layers. An alternate approach is to train the model for a simpler task and then use the learned model parameters as initial parameter choices in the learning of the original task. The process of initializing ANN model parameters before the actual training process is known as **pretraining**.

Pretraining helps in initializing the model to a suitable region in the parameter space that would otherwise be inaccessible by random initialization.

Pretraining also reduces the variance in the model parameters by fixing the starting point of gradient descent, thus reducing the chances of overfitting due to multiple comparisons. The models learned by pretraining are thus more consistent and provide better generalization performance.

Supervised Pretraining

A common approach for pretraining is to incrementally train the ANN model in a layer-wise manner, by adding one hidden layer at a time. This approach,

known as **supervised pretraining**, ensures that the parameters learned at every layer are obtained by solving a simpler problem, rather than learning all model parameters together. These parameter values thus provide a good choice for initializing the ANN model. The approach for supervised pretraining can be briefly described as follows.

We start the supervised pretraining process by considering a reduced ANN model with only a single hidden layer. By applying gradient descent on this simple model, we are able to learn the model parameters of the first hidden layer. At the next run, we add another hidden layer to the model and apply gradient descent to learn the parameters of the newly added hidden layer, while keeping the parameters of the first layer fixed. This procedure is recursively applied such that while learning the parameters of the l^{th} hidden layer, we consider a reduced model with only l hidden layers, whose first $(l-1)$ hidden layers are not updated on the l^{th} run but are instead fixed using pretrained values from previous runs. In this way, we are able to learn the model parameters of all $(L-1)$ hidden layers. These pretrained values are used to initialize the hidden layers of the final ANN model, which is fine-tuned by applying a final round of gradient descent over all the layers.

Unsupervised Pretraining

Supervised pretraining provides a powerful way to initialize model parameters, by gradually growing the model complexity from shallower to deeper networks. However, supervised pretraining requires a sufficient number of labeled training instances for effective initialization of the ANN model. An alternate pretraining approach is **unsupervised pretraining**, which initializes model parameters by using unlabeled instances that are often abundantly available. The basic idea of unsupervised pretraining is to initialize the ANN

model in such a way that the learned features capture the latent structure in the unlabeled data.

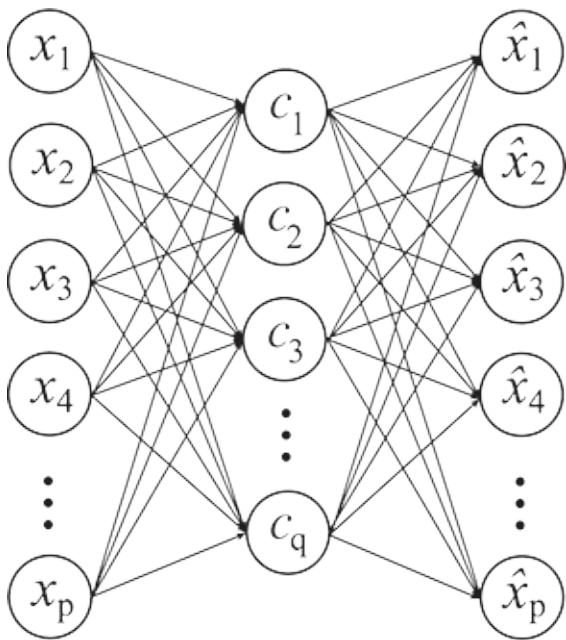


Figure 4.31.

The basic architecture of a single-layer autoencoder.

Unsupervised pretraining relies on the assumption that learning the distribution of the input data can indirectly help in learning the classification model. It is most helpful when the number of labeled examples is small and the features for the supervised problem bear resemblance to the factors generating the input data. Unsupervised pretraining can be viewed as a different form of regularization, where the focus is not explicitly toward finding simpler features but instead toward finding features that can best explain the input data. Historically, unsupervised pretraining has played an important role in reviving the area of deep learning, by making it possible to train any generic deep neural network without requiring specialized architectures.

Use of Autoencoders

One simple and commonly used approach for unsupervised pretraining is to use an unsupervised ANN model known as an **autoencoder**. The basic architecture of an autoencoder is shown in [Figure 4.31](#). An autoencoder attempts to learn a *reconstruction* of the input data by mapping the attributes x to latent features c , and then re-projecting c back to the original attribute space to create the reconstruction x^{\wedge} . The latent features are represented using a hidden layer of nodes, while the input and output layers represent the attributes and contain the same number of nodes. During training, the goal is to learn an autoencoder model that provides the lowest reconstruction error, $RE(x, x^{\wedge})$, on all input data instances. A typical choice of the reconstruction error is the squared loss function:

$$RE(x, x^{\wedge}) = \|x - x^{\wedge}\|_2^2.$$

The model parameters of the autoencoder can be learned by using a similar gradient descent method as the one used for learning supervised ANN models for classification. The key difference is the use of the reconstruction error on all training instances as the training loss. Autoencoders that have multiple layers of hidden layers are known as **stacked autoencoders**.

Autoencoders are able to capture complex representations of the input data by the use of hidden nodes. However, if the number of hidden nodes is large, it is possible for an autoencoder to learn the identity relationship, where the input x is just copied and returned as the output x^{\wedge} , resulting in a trivial solution. For example, if we use as many hidden nodes as the number of attributes, then it is possible for every hidden node to copy an attribute and simply pass it along to an output node, without extracting any useful information. To avoid this problem, it is common practice to keep the number of hidden nodes smaller than the number of input attributes. This forces the autoencoder to learn a compact and useful *encoding* of the input data, similar to a dimensionality reduction technique. An alternate approach is to corrupt

the input instances by adding random noise, and then learn the autoencoder to reconstruct the original instance from the noisy input. This approach is known as the **denoising autoencoder**, which offers strong regularization capabilities and is often used to learn complex features even in the presence of a large number of hidden nodes.

To use an autoencoder for unsupervised pretraining, we can follow a similar layer-wise approach like supervised pretraining. In particular, to pretrain the model parameters of the l^{th} hidden layer, we can construct a reduced ANN model with only l hidden layers and an output layer containing the same number of nodes as the attributes and is used for reconstruction. The parameters of the l^{th} hidden layer of this network are then learned using a gradient descent method to minimize the reconstruction error. The use of unlabeled data can be viewed as providing *hints* to the learning of parameters at every layer that aid in generalization. The final model parameters of the ANN model are then learned by applying gradient descent over all the layers, using the initial values of parameters obtained from pretraining.

Hybrid Pretraining

Unsupervised pretraining can also be combined with supervised pretraining by using two output layers at every run of pretraining, one for reconstruction and the other for supervised classification. The parameters of the l^{th} hidden layer are then learned by jointly minimizing the losses on both output layers, usually weighted by a trade-off hyper-parameter α . Such a combined approach often shows better generalization performance than either of the approaches, since it provides a way to balance between the competing objectives of representing the input data and improving classification performance.

4.8.5 Characteristics of Deep Learning

Apart from the basic characteristics of ANN discussed in [Section 4.7.3](#), the use of deep learning techniques provides the following additional characteristics:

1. An ANN model trained for some task can be easily re-used for a different task that involves the same attributes, by using pretraining strategies. For example, we can use the learned parameters of the original task as initial parameter choices for the target task. In this way, ANN promotes *re-usability* of learning, which can be quite useful when the target application has a smaller number of labeled training instances.
2. Deep learning techniques for regularization, such as the dropout method, help in reducing the model complexity of ANN and thus promoting good generalization performance. The use of regularization techniques is especially useful in high-dimensional settings, where the number of training labels is small but the classification problem is inherently difficult.
3. The use of an autoencoder for pretraining can help eliminate irrelevant attributes that are not related to other attributes. Further, it can help reduce the impact of redundant attributes by representing them as copies of the same attribute.
4. Although the learning of an ANN model can succumb to finding inferior and locally optimal solutions, there are a number of deep learning techniques that have been proposed to ensure adequate learning of an ANN. Apart from the methods discussed in this section, some other techniques involve novel architecture designs such as skip connections between the output layer and lower layers, which aids the easy flow of gradients during backpropagation.

5. A number of specialized ANN architectures have been designed to handle a variety of input data sets. Some examples include **convolutional neural networks** (CNN) for two-dimensional gridded objects such as images, and **recurrent neural network** (RNN) for sequences. While CNNs have been extensively used in the area of computer vision, RNNs have found applications in processing speech and language.

4.9 Support Vector Machine (SVM)

A support vector machine (SVM) is a discriminative classification model that learns linear or nonlinear decision boundaries in the attribute space to separate the classes. Apart from maximizing the separability of the two classes, SVM offers strong regularization capabilities, i.e., it is able to control the complexity of the model in order to ensure good generalization performance. Due to its unique ability to innately regularize its learning, SVM is able to learn highly expressive models without suffering from overfitting. It has thus received considerable attention in the machine learning community and is commonly used in several practical applications, ranging from handwritten digit recognition to text categorization. SVM has strong roots in statistical learning theory and is based on the principle of structural risk minimization. Another unique aspect of SVM is that it represents the decision boundary using only a subset of the training examples that are most difficult to classify, known as the **support vectors**. Hence, it is a discriminative model that is impacted only by training instances near the boundary of the two classes, in contrast to learning the generative distribution of every class.

To illustrate the basic idea behind SVM, we first introduce the concept of the margin of a separating hyperplane and the rationale for choosing such a hyperplane with maximum margin. We then describe how a linear SVM can be trained to explicitly look for this type of hyperplane. We conclude by showing how the SVM methodology can be extended to learn nonlinear decision boundaries by using kernel functions.

4.9.1 Margin of a Separating

Hyperplane

The generic equation of a separating hyperplane can be written as

$$\mathbf{w}^T \mathbf{x} + b = 0,$$

where \mathbf{x} represents the attributes and (\mathbf{w}, b) represent the parameters of the hyperplane. A data instance x_i can belong to either side of the hyperplane depending on the sign of $(\mathbf{w}^T \mathbf{x}_i + b)$. For the purpose of binary classification, we are interested in finding a hyperplane that places instances of both classes on opposite sides of the hyperplane, thus resulting in a *separation* of the two classes. If there exists a hyperplane that can perfectly separate the classes in the data set, we say that the data set is **linearly separable**. [Figure 4.32](#) shows an example of linearly separable data involving two classes, squares and circles. Note that there can be infinitely many hyperplanes that can separate the classes, two of which are shown in [Figure 4.32](#) as lines B1 and B2. Even though every such hyperplane will have zero training error, they can provide different results on previously unseen instances. Which separating hyperplane should we thus finally choose to obtain the best generalization performance? Ideally, we would like to choose a *simple* hyperplane that is robust to small perturbations. This can be achieved by using the concept of the margin of a separating hyperplane, which can be briefly described as follows.

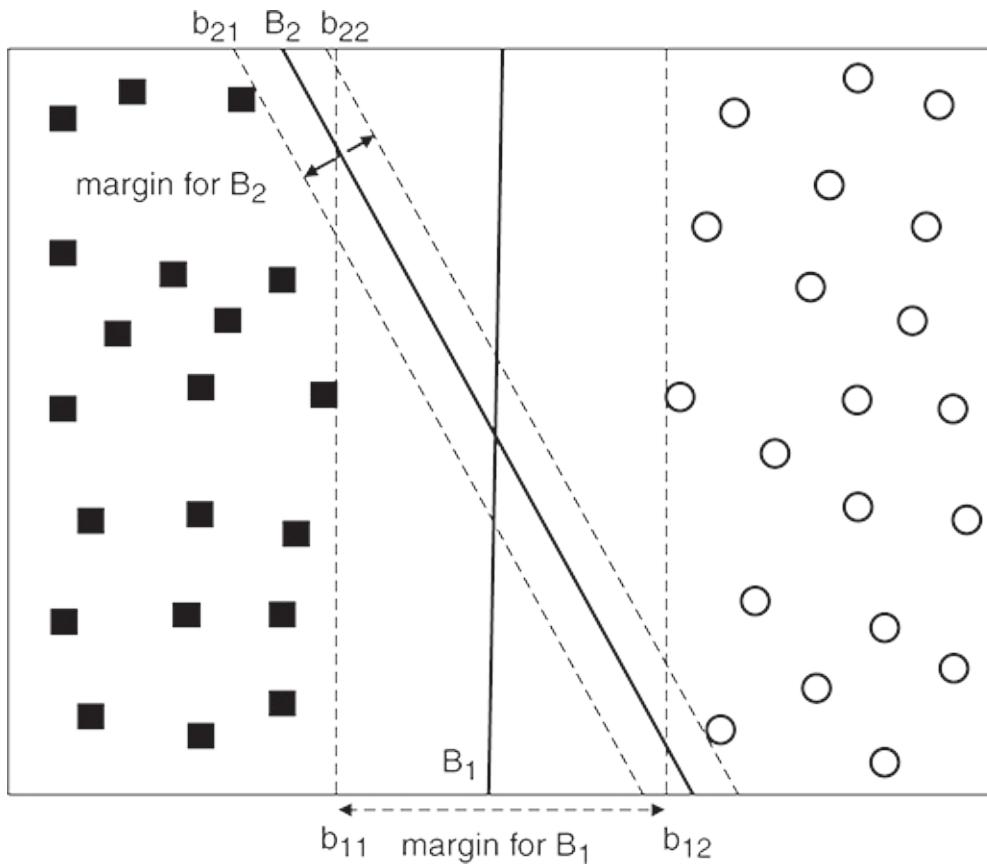


Figure 4.32.

Margin of a hyperplane in a two-dimensional data set.

For every separating hyperplane B_i , let us associate a pair of parallel hyperplanes, b_{i1} and b_{i2} , such that they touch the closest instances of both classes, respectively. For example, if we move B_1 parallel to its direction, we can touch the first square using b_{11} and the first circle using b_{12} . b_{i1} and b_{i2} are known as the **margin hyperplanes** of B_i and the distance between them is known as the **margin** of the separating hyperplane B_i . From the diagram shown in [Figure 4.32](#), notice that the margin for B_1 is considerably larger than that for B_2 . In this example, b_1 turns out to be the separating hyperplane with the maximum margin, known as the **maximum margin hyperplane**.

Rationale for Maximum Margin

Hyperplanes with large margins tend to have better generalization performance than those with small margins. Intuitively, if the margin is small, then any slight perturbation in the hyperplane or the training instances located at the boundary can have quite an impact on the classification performance. Small margin hyperplanes are thus more susceptible to overfitting, as they are barely able to separate the classes with a very narrow room to allow perturbations. On the other hand, a hyperplane that is farther away from training instances of both classes has sufficient leeway to be robust to minor modifications in the data, and thus shows superior generalization performance.

The idea of choosing the maximum margin separating hyperplane also has strong foundations in statistical learning theory. It can be shown that the margin of such a hyperplane is inversely related to the VC-dimension of the classifier, which is a commonly used measure of the complexity of a model. As discussed in [Section 3.4](#) of the last chapter, a simpler model should be preferred over a more complex model if they both show similar training performance. Hence, maximizing the margin results in the selection of a separating hyperplane with the lowest model complexity, which is expected to show better generalization performance.

4.9.2 Linear SVM

A linear SVM is a classifier that searches for a separating hyperplane with the largest margin, which is why it is often known as a **maximal margin classifier**. The basic idea of SVM can be described as follows.

Consider a binary classification problem consisting of n training instances, where every training instance x_i is associated with a binary label $y_i \in \{-1, 1\}$.

Let $w^T x + b = 0$ be the equation of a separating hyperplane that separates the two classes by placing them on opposite sides. This means that

$$w^T x_i + b > 0 \text{ if } y_i = 1, w^T x_i + b < 0 \text{ if } y_i = -1.$$

The distance of any point x from the hyperplane is then given by

$$D(x) = |w^T x + b| \|w\|$$

where $|\cdot|$ denotes the absolute value and $\|\cdot\|$ denotes the length of a vector.

Let the distance of the closest point from the hyperplane with $y=1$ be $k+>0$.

Similarly, let $k->0$ denote the distance of the closest point from class -1 .

This can be represented using the following constraints:

$$w^T x_i + b \|w\| \geq k+ \text{ if } y_i = 1, w^T x_i + b \|w\| \leq -k- \text{ if } y_i = -1, \quad (4.69)$$

The previous equations can be succinctly represented by using the product of y_i and $(w^T x_i + b)$ as

$$y_i(w^T x_i + b) \geq M \|w\| \quad (4.70)$$

where M is a parameter related to the margin of the hyperplane, i.e., if $k+=k-=M$, then margin $=k+-k-=2M$. In order to find the maximum margin hyperplane that adheres to the previous constraints, we can consider the following optimization problem:

$$\max_w, b \text{ subject to } y_i(w^T x_i + b) \geq M \|w\|. \quad (4.71)$$

To find the solution to the previous problem, note that if w and b satisfy the constraints of the previous problem, then any scaled version of w and b would

satisfy them too. Hence, we can conveniently choose $\|w\|=1/M$ to simplify the right-hand side of the inequalities. Furthermore, maximizing M amounts to minimizing $\|w\|^2$. Hence, the optimization problem of SVM is commonly represented in the following form:

$$\min_w, b \quad \text{subject to } y_i(w^T x_i + b) \geq 1. \quad (4.72)$$

Learning Model Parameters

Equation 4.72 represents a constrained optimization problem with linear inequalities. Since the objective function is convex and quadratic with respect to w , it is known as a quadratic programming problem (QPP), which can be solved using standard optimization techniques, as described in Appendix E. In the following, we present a brief sketch of the main ideas for learning the model parameters of SVM.

First, we rewrite the objective function in a form that takes into account the constraints imposed on its solutions. The new objective function is known as the **Lagrangian primal problem**, which can be represented as follows,

$$LP = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \lambda_i (y_i(w^T x_i + b) - 1), \quad (4.73)$$

where the parameters $\lambda_i \geq 0$ correspond to the constraints and are called the **Lagrange multipliers**. Next, to minimize the Lagrangian, we take the derivative of LP with respect to w and b and set them equal to zero:

$$\partial LP / \partial w = 0 \Rightarrow w = \sum_{i=1}^n \lambda_i y_i x_i, \quad (4.74)$$

$$\partial LP / \partial b = 0 \Rightarrow \sum_{i=1}^n \lambda_i y_i = 0. \quad (4.75)$$

Note that using [Equation 4.74](#), we can represent w completely in terms of the Lagrange multipliers. There is another relationship between (w, b) and λ_i that is derived from the Karush-Kuhn-Tucker (KKT) conditions, a commonly used technique for solving QPP. This relationship can be described as

$$\lambda_i[y_i(w^T x_i + b) - 1] = 0. \quad (4.76)$$

[Equation 4.76](#) is known as the **complementary slackness condition**, which sheds light on a valuable property of SVM. It states that the Lagrange multiplier λ_i is strictly greater than 0 only when x_i satisfies the equation $y_i(w^T x_i + b) = 1$, which means that x_i lies exactly on a margin hyperplane. However, if x_i is farther away from the margin hyperplanes such that $y_i(w^T x_i + b) > 1$, then λ_i is necessarily 0. Hence, $\lambda_i > 0$ for only a small number of instances that are closest to the separating hyperplane, which are known as **support vectors**. [Figure 4.33](#) shows the support vectors of a hyperplane as filled circles and squares. Further, if we look at [Equation 4.74](#), we will observe that training instances with $\lambda_i = 0$ do not contribute to the weight parameter w . This suggests that w can be concisely represented only in terms of the support vectors in the training data, which are quite fewer than the overall number of training instances. This ability to represent the decision function only in terms of the support vectors is what gives this classifier the name support vector machines.

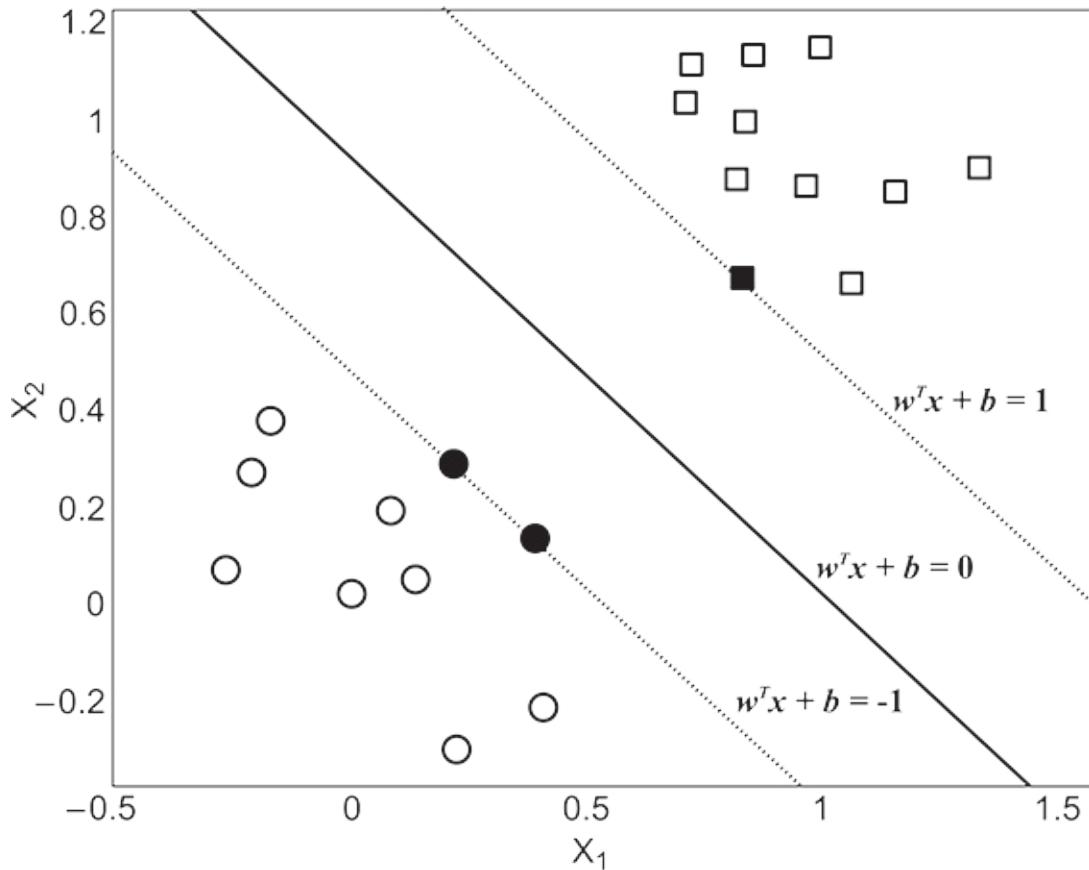


Figure 4.33.

Support vectors of a hyperplane shown as filled circles and squares.

Using equations 4.74, 4.75, and 4.76 in Equation 4.73, we obtain the following optimization problem in terms of the Lagrange multipliers λ_i :

$$\max \lambda_i \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j x_i^T x_j \text{ subject to } \sum_{i=1}^n \lambda_i y_i = 0, \lambda_i \geq 0. \quad (4.77)$$

The previous optimization problem is called the **dual optimization problem**. Maximizing the dual problem with respect to λ_i is equivalent to minimizing the primal problem with respect to w and b .

The key differences between the dual and primal problems are as follows:

1. Solving the dual problem helps us identify the support vectors in the data that have non-zero values of λ_i . Further, the solution of the dual problem is influenced only by the support vectors that are closest to the decision boundary of SVM. This helps in summarizing the learning of SVM solely in terms of its support vectors, which are easier to manage computationally. Further, it represents a unique ability of SVM to be dependent only on the instances closest to the boundary, which are harder to classify, rather than the distribution of instances farther away from the boundary.
2. The objective of the dual problem involves only terms of the form $x_i^T x_j$, which are basically inner products in the attribute space. As we will see later in [Section 4.9.4](#), this property will prove to be quite useful in learning nonlinear decision boundaries using SVM.

Because of these differences, it is useful to solve the dual optimization problem using any of the standard solvers for QPP. Having found an optimal solution for λ_i , we can use [Equation 4.74](#) to solve for w . We can then use [Equation 4.76](#) on the support vectors to solve for b as follows:

$$b = \frac{1}{n_S} \sum_{i \in S} y_i w^T x_i - \frac{1}{2} \sum_{i \in S} \sum_{j \in S} y_i y_j w^T x_i w^T x_j \quad (4.78)$$

where S represents the set of support vectors ($S = \{i | \lambda_i > 0\}$) and n_S is the number of support vectors. The maximum margin hyperplane can then be expressed as

$$f(x) = \left(\sum_{i=1}^{n_S} \lambda_i y_i x_i^T x \right) + b = 0. \quad (4.79)$$

Using this separating hyperplane, a test instance x can be assigned a class label using the sign of $f(x)$.

Example 4.7.

Consider the two-dimensional data set shown in [Figure 4.34](#), which contains eight training instances. Using quadratic programming, we can solve the optimization problem stated in [Equation 4.77](#) to obtain the Lagrange multiplier λ_i for each training instance. The Lagrange multipliers are depicted in the last column of the table. Notice that only the first two instances have non-zero Lagrange multipliers. These instances correspond to the support vectors for this data set.

Let $w=(w_1, w_2)$ and b denote the parameters of the decision boundary. Using [Equation 4.74](#), we can solve for w_1 and w_2 in the following way:

$$w_1 = \sum i \lambda_i y_i x_i = 65.5261 \times 1 \times 0.3858 + 65.5261 \times -1 \times 0.4871 = -6.64 \\ w_2 = \sum i \lambda_i y_i x_i = 65.5261 \times 1 \times 0.4687 + 65.5261 \times -1 \times 0.611 = -9.32.$$

x_1	x_2	y	Lagrange Multiplier
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

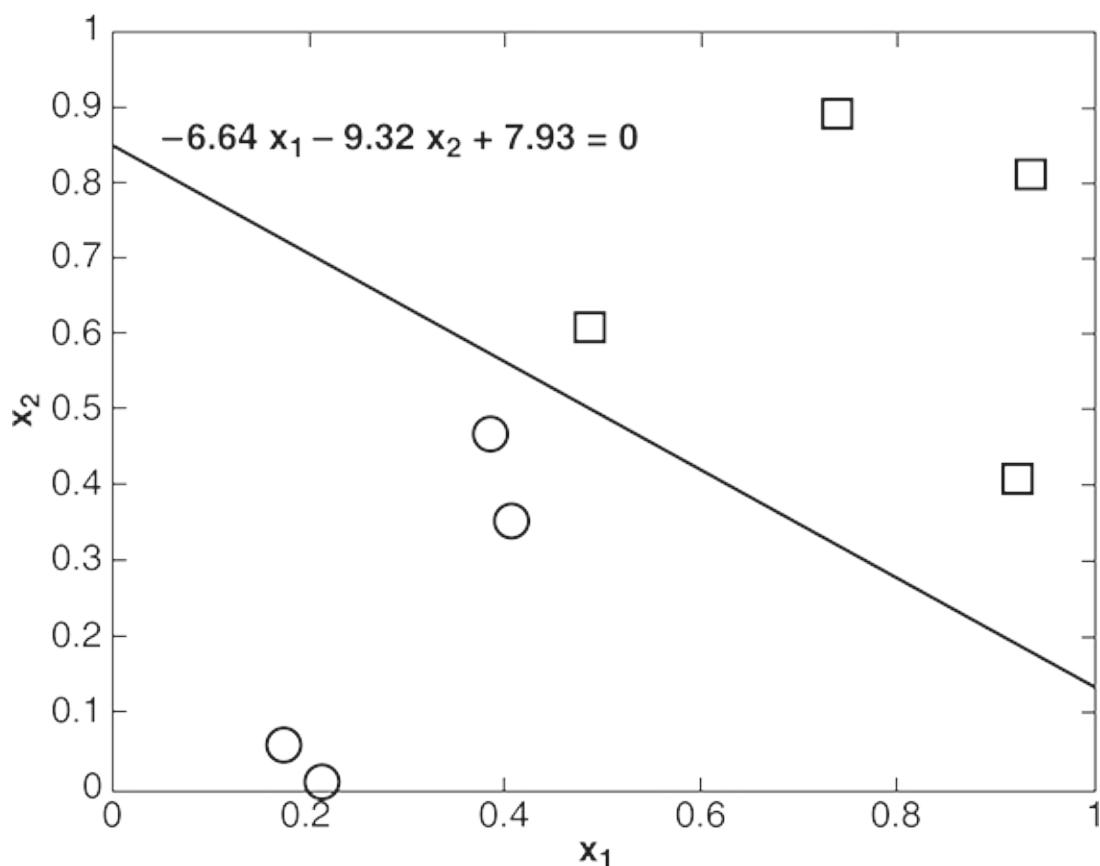


Figure 4.34.

Example of a linearly separable data set.

The bias term b can be computed using [Equation 4.76](#) for each support vector:

$$b(1)=1-w \cdot x_1=1-(-6.64)(0.3858)-(-9.32)(0.4687)=7.9300.b(2)=1-w \cdot x_2=-1-(-6.64)(0.4871)-(-9.32)(0.611)=7.9289.$$

Averaging these values, we obtain $b=7.93$. The decision boundary corresponding to these parameters is shown in [Figure 4.34](#).

4.9.3 Soft-margin SVM

[Figure 4.35](#) shows a data set that is similar to [Figure 4.32](#), except it has two new examples, P and Q . Although the decision boundary B_1 misclassifies the new examples, while B_2 classifies them correctly, this does not mean that B_2 is a better decision boundary than B_1 because the new examples may correspond to noise in the training data. B_1 should still be preferred over B_2 because it has a wider margin, and thus, is less susceptible to overfitting. However, the SVM formulation presented in the previous section only constructs decision boundaries that are mistake-free.

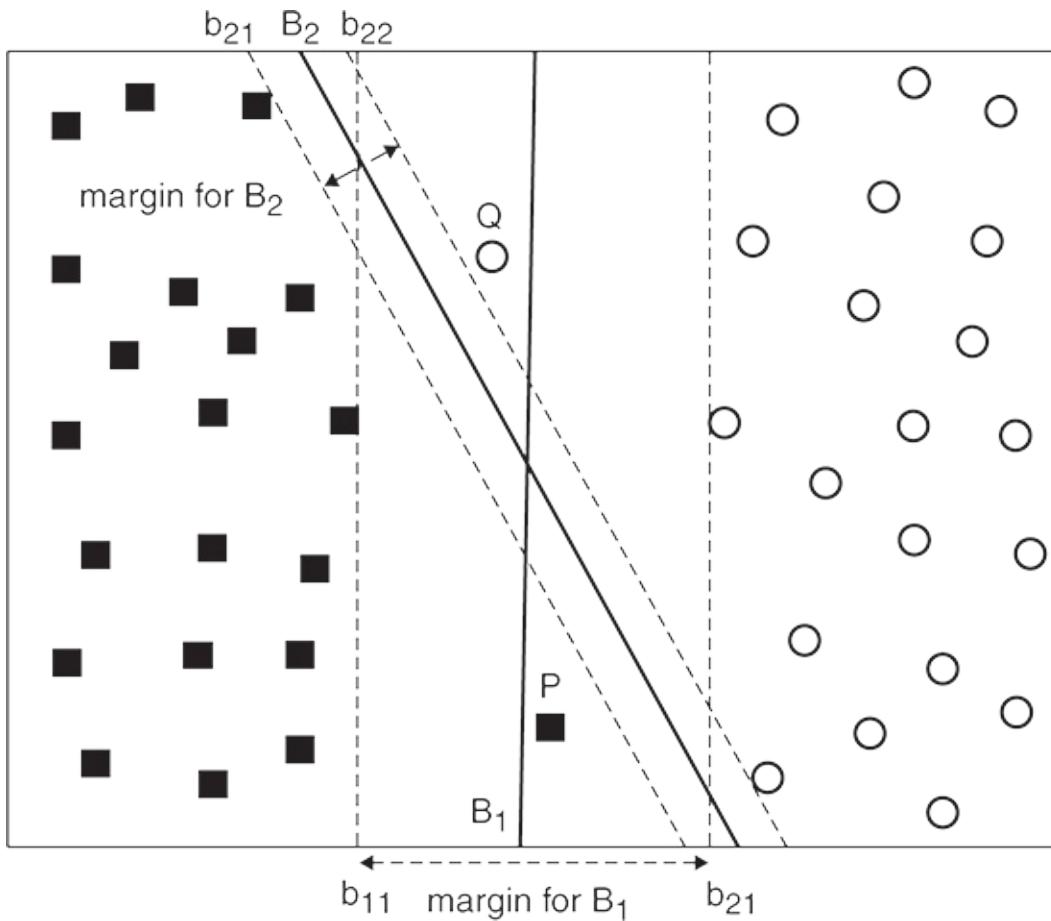


Figure 4.35.

Decision boundary of SVM for the non-separable case.

This section examines how the formulation of SVM can be modified to learn a separating hyperplane that is tolerable to small number of training errors using a method known as the **soft-margin** approach. More importantly, the method presented in this section allows SVM to learn linear hyperplanes even in situations where the classes are not linearly separable. To do this, the learning algorithm in SVM must consider the trade-off between the width of the margin and the number of training errors committed by the linear hyperplane.

To introduce the concept of training errors in the SVM formulation, let us relax the inequality constraints to accommodate for some violations on a small number of training instances. This can be done by introducing a **slack variable** $\xi \geq 0$ for every training instance x_i as follows:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad (4.80)$$

The variable ξ_i allows for some slack in the inequalities of the SVM such that every instance x_i does not need to strictly satisfy $y_i(w^T x_i + b) \geq 1$. Further, ξ_i is non-zero only if the margin hyperplanes are not able to place x_i on the same side as the rest of the instances belonging to y_i . To illustrate this, [Figure 4.36](#) shows a circle P that falls on the opposite side of the separating hyperplane as the rest of the circles, and thus satisfies $w^T x + b = -1 + \xi$. The distance between P and the margin hyperplane $w^T x + b = -1$ is equal to $\xi / \|w\|$. Hence, ξ_i provides a measure of the error of SVM in representing x_i using soft inequality constraints.

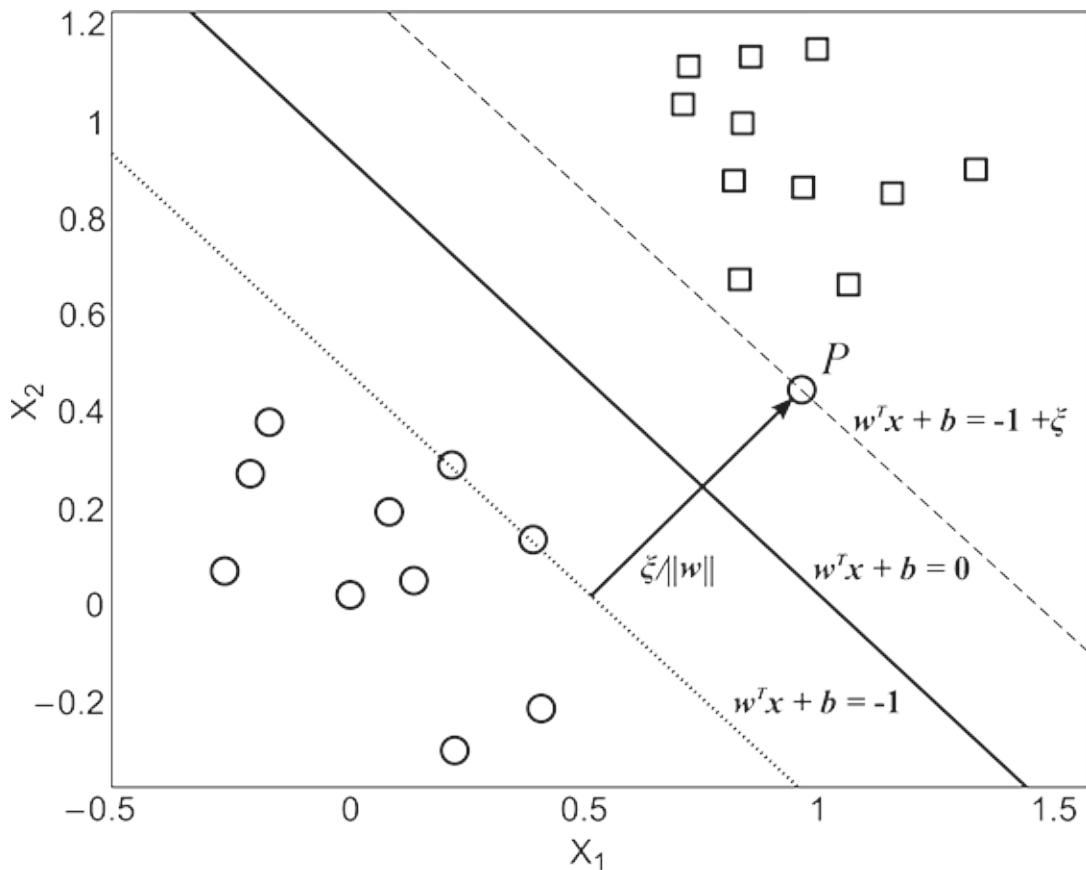


Figure 4.36.
Slack variables used in soft-margin SVM.

In the presence of slack variables, it is important to learn a separating hyperplane that jointly maximizes the margin (ensuring good generalization performance) and minimizes the values of slack variables (ensuring low training error). This can be achieved by modifying the optimization problem of SVM as follows:

$$\min_{w, b, \xi} \|w\|^2 + C \sum_{i=1}^n \xi_i \text{ subject to } y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0. \quad (4.81)$$

where C is a hyper-parameter that makes a trade-off between maximizing the margin and minimizing the training error. A large value of C pays more emphasis on minimizing the training error than maximizing the margin. Notice the similarity of the previous equation with the generic formula of generalization error rate introduced in [Section 3.4](#) of the previous chapter. Indeed, SVM provides a natural way to balance between model complexity and training error in order to maximize generalization performance.

To solve [Equation 4.81](#) we apply the Lagrange multiplier method and convert the primal problem to its corresponding dual problem, similar to the approach described in the previous section. The Lagrangian primal problem of [Equation 4.81](#) can be written as follows:

$$LP = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \lambda_i (y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^n \mu_i (\xi_i), \quad (4.82)$$

where $\lambda_i \geq 0$ and $\mu_i \geq 0$ are the Lagrange multipliers corresponding to the inequality constraints of [Equation 4.81](#). Setting the derivative of LP with respect to w , b , and ξ_i equal to 0, we obtain the following equations:

$$\partial L \partial w = 0 \Rightarrow w = \sum_{i=1}^n \lambda_i y_i x_i. \quad (4.83)$$

$$\partial L \partial b = 0 \Rightarrow \sum_{i=1}^n \lambda_i y_i = 0. \quad (4.84)$$

$$\partial L / \partial \xi_j = 0 \Rightarrow \lambda_i + \mu_i = C. \quad (4.85)$$

We can also obtain the complementary slackness conditions by using the following KKT conditions:

$$\lambda_i(y_i(w^T x_i + b) - 1 + \xi_i) = 0, \quad (4.86)$$

$$\mu_i \xi_i = 0. \quad (4.87)$$

Equation 4.86 suggests that λ_i is zero for all training instances except those that reside on the margin hyperplanes $w^T x_i + b = \pm 1$, or have $\xi_i > 0$. These instances with $\lambda_i > 0$ are known as support vectors. On the other hand, μ_i given in **Equation 4.87** is zero for any training instance that is misclassified, i.e., $\xi_i > 0$. Further, λ_i and μ_i are related with each other by **Equation 4.85**. This results in the following three configurations of (λ_i, μ_i) :

1. If $\lambda_i = 0$ and $\mu_i = C$, then x_i does not reside on the margin hyperplanes and is correctly classified on the same side as other instances belonging to y_i .
2. If $\lambda_i = C$ and $\mu_i = 0$, then x_i is misclassified and has a non-zero slack variable ξ_i .
3. If $0 < \lambda_i < C$ and $0 < \mu_i < C$, then x_i resides on one of the margin hyperplanes.

Substituting **Equations 4.83** to **4.87** into **Equation 4.82**, we obtain the following dual optimization problem:

$$\max_{\lambda} \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j x_i^T x_j \text{ subject to } \sum_{i=1}^n \lambda_i y_i = 0, 0 \leq \lambda_i \leq C. \quad (4.88)$$

Notice that the previous problem looks almost identical to the dual problem of SVM for the linearly separable case (**Equation 4.77**), except that λ_i is

required to not only be greater than 0 but also smaller than a constant value C. Clearly, when C reaches infinity, the previous optimization problem becomes equivalent to [Equation 4.77](#), where the learned hyperplane perfectly separates the classes (with no training errors). However, by capping the values of λ_i to C, the learned hyperplane is able to tolerate a few training errors that have $\xi_i > 0$.

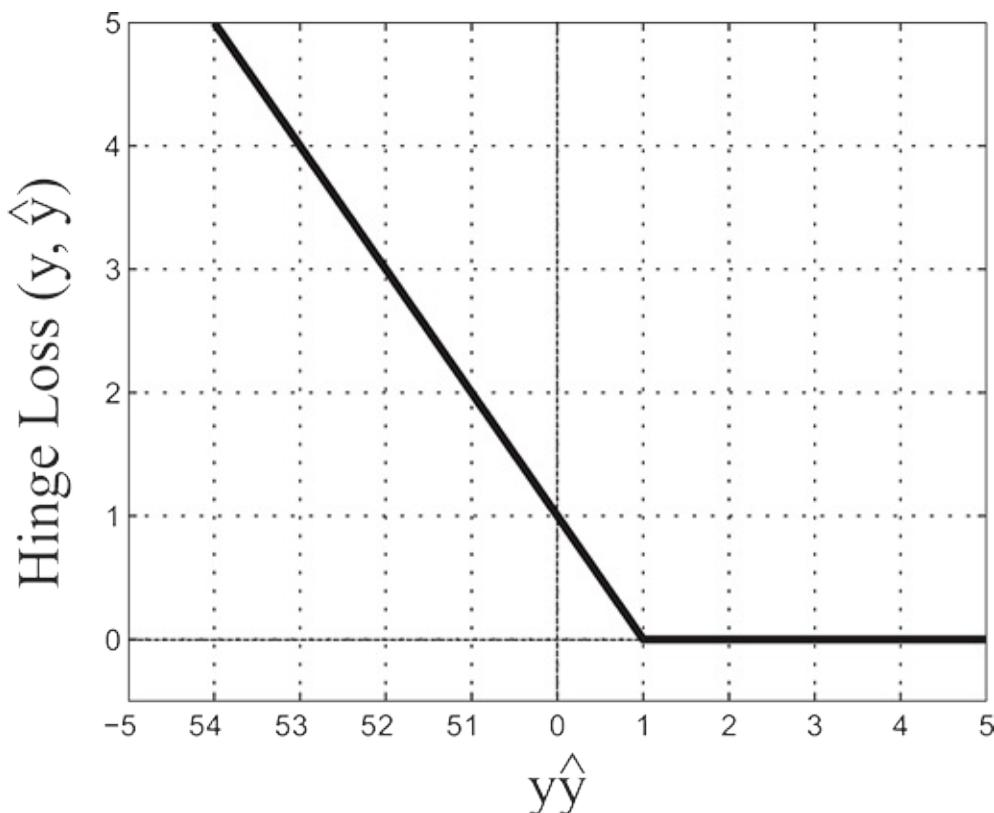


Figure 4.37.

Hinge loss as a function of $y\hat{y}$.

As before, [Equation 4.88](#) can be solved by using any of the standard solvers for QPP, and the optimal value of w can be obtained by using [Equation 4.83](#). To solve for b , we can use [Equation 4.86](#) on the support vectors that reside on the margin hyperplanes as follows:

$$b = \frac{1}{n} \sum_{i \in S} (1 - y_i w^T x_i) \quad (4.89)$$

where S represents the set of support vectors residing on the margin hyperplanes ($S=\{i|0<\lambda_i < C\}$) and n_S is the number of elements in S .

SVM as a Regularizer of Hinge Loss

SVM belongs to a broad class of regularization techniques that use a loss function to represent the training errors and a norm of the model parameters to represent the model complexity. To realize this, notice that the slack variable ξ , used for measuring the training errors in SVM, is equivalent to the hinge loss function, which can be defined as follows:

$$\text{Loss}(y, y^{\wedge}) = \max(0, 1 - yy^{\wedge}),$$

where $y \in \{+1, -1\}$. In the case of SVM, y^{\wedge} corresponds to $w^T x + b$. [Figure 4.37](#) shows a plot of the hinge loss function as we vary yy^{\wedge} . We can see that the hinge loss is equal to 0 as long as y and y^{\wedge} have the same sign and $|y^{\wedge}| \geq 1$. However, the hinge loss grows linearly with $|y^{\wedge}|$ whenever y and y^{\wedge} are of the opposite sign or $|y^{\wedge}| < 1$. This is similar to the notion of the slack variable ξ , which is used to measure the distance of a point from its margin hyperplane. Hence, the optimization problem of SVM can be represented in the following equivalent form:

$$\min_w, \min_b \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \text{Loss}(y_i, w^T x_i + b) \quad (4.90)$$

Note that using the hinge loss ensures that the optimization problem is convex and can be solved using standard optimization techniques. However, if we use a different loss function, such as the squared loss function that was introduced in [Section 4.7](#) on ANN, it will result in a different optimization problem that may or may not remain convex. Nevertheless, different loss functions can be explored to capture varying notions of training error, depending on the characteristics of the problem.

Another interesting property of SVM that relates it to a broader class of regularization techniques is the concept of a margin. Although minimizing $\|w\|_2$ has the geometric interpretation of maximizing the margin of a separating hyperplane, it is essentially the squared L2 norm of the model parameters, $\|w\|_2^2$. In general, the Lq norm of w , $\|w\|_q$, is equal to the Minkowski distance of order q from w to the origin, i.e.,

$$\|w\|_q = (\sum_i p_i w_i^q)^{1/q}$$

Minimizing the Lq norm of w to achieve lower model complexity is a generic regularization concept that has several interpretations. For example, minimizing the L2 norm amounts to finding a solution on a hypersphere of smallest radius that shows suitable training performance. To visualize this in two-dimensions, [Figure 4.38\(a\)](#) shows the plot of a circle with constant radius r , where every point has the same L2 norm. On the other hand, using the L1 norm ensures that the solution lies on the surface of a hypercube with smallest size, with vertices along the axes. This is illustrated in [Figure 4.38\(b\)](#) as a square with vertices on the axes at a distance of r from the origin. The L1 norm is commonly used as a regularizer to obtain sparse model parameters with only a small number of non-zero parameter values, such as the use of Lasso in regression problems (see Bibliographic Notes).

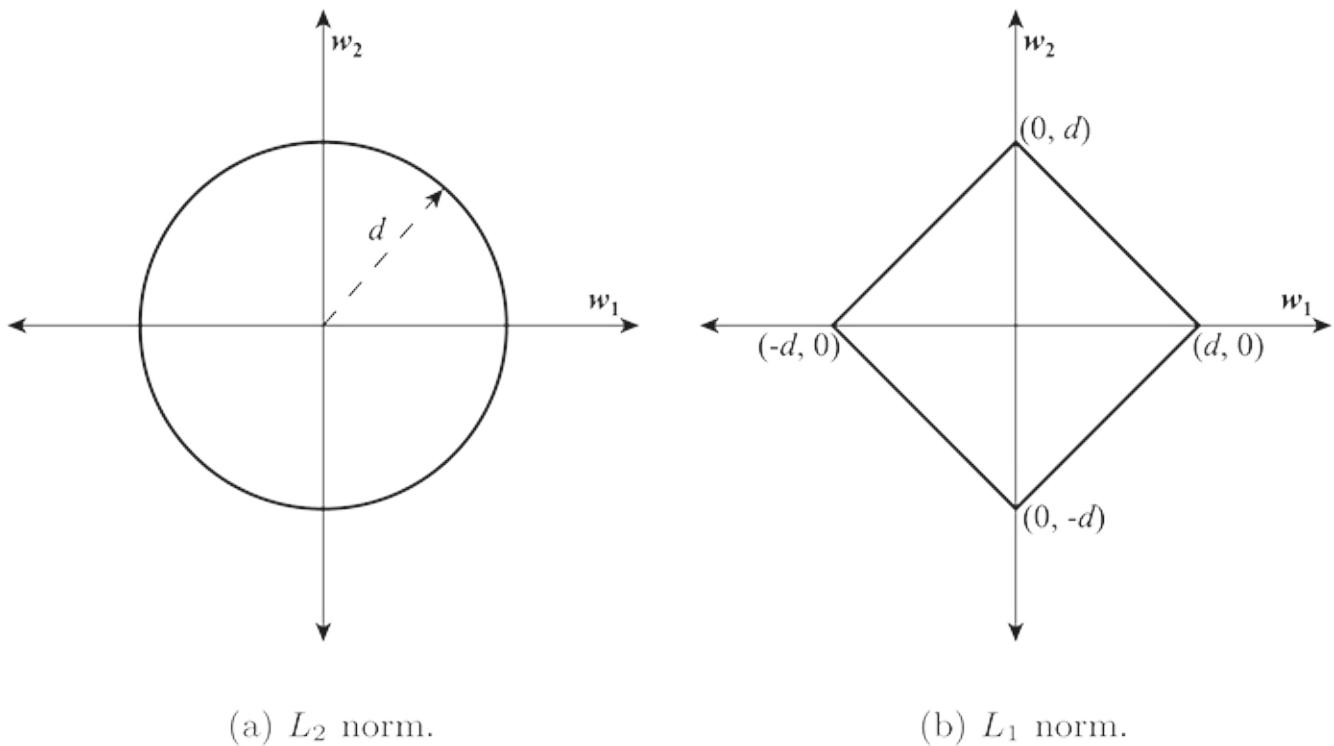


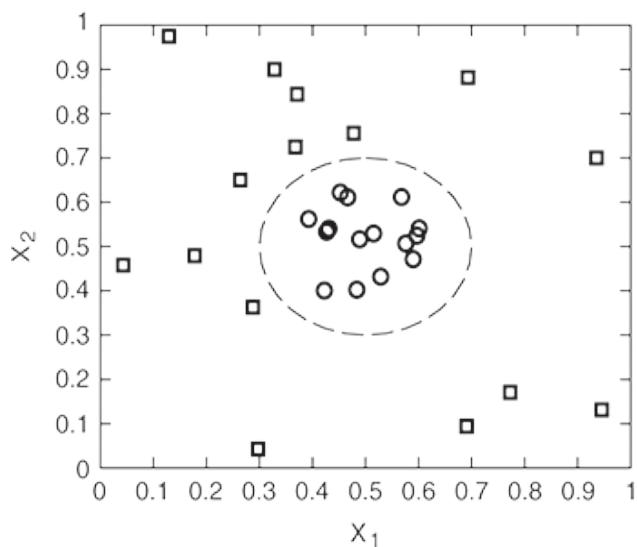
Figure 4.38.

Plots showing the behavior of two-dimensional solutions with constant L2 and L1 norms.

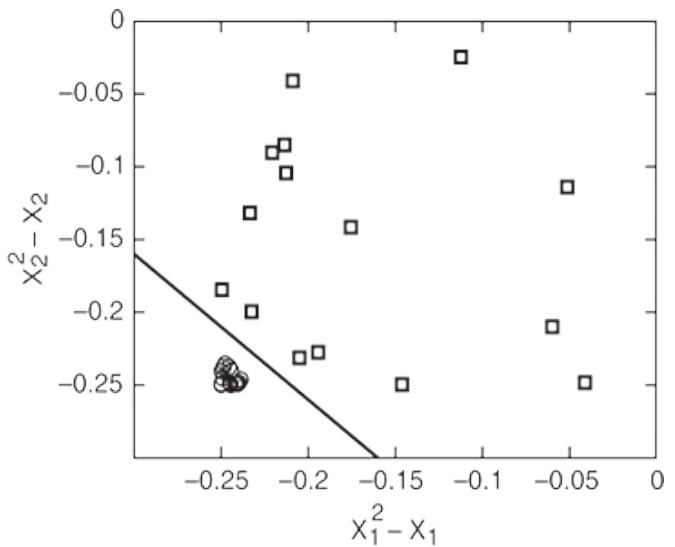
In general, depending on the characteristics of the problem, different combinations of L_q norms and training loss functions can be used for learning the model parameters, each requiring a different optimization solver. This forms the backbone of a wide range of modeling techniques that attempt to improve the generalization performance by jointly minimizing training error and model complexity. However, in this section, we focus only on the squared L2 norm and the hinge loss function, resulting in the classical formulation of SVM.

4.9.4 Nonlinear SVM

The SVM formulations described in the previous sections construct a linear decision boundary to separate the training examples into their respective classes. This section presents a methodology for applying SVM to data sets that have nonlinear decision boundaries. The basic idea is to transform the data from its original attribute space in \mathbf{x} into a new space $\varphi(\mathbf{x})$ so that a linear hyperplane can be used to separate the instances in the transformed space, using the SVM approach. The learned hyperplane can then be projected back to the original attribute space, resulting in a nonlinear decision boundary.



(a) Decision boundary in the original two-dimensional space.



(b) Decision boundary in the transformed space.

Figure 4.39.

Classifying data with a nonlinear decision boundary.

Attribute Transformation

To illustrate how attribute transformation can lead to a linear decision boundary, **Figure 4.39(a)** shows an example of a two-dimensional data set consisting of squares (classified as $y=1$) and circles (classified as $y=-1$). The

data set is generated in such a way that all the circles are clustered near the center of the diagram and all the squares are distributed farther away from the center. Instances of the data set can be classified using the following equation:

$$y = \begin{cases} 1 & \text{if } (x_1 - 0.5)^2 + (x_2 - 0.5)^2 > 0.2, \\ -1 & \text{otherwise.} \end{cases} \quad (4.91)$$

The decision boundary for the data can therefore be written as follows:

$$(x_1 - 0.5)^2 + (x_2 - 0.5)^2 > 0.2,$$

which can be further simplified into the following quadratic equation:

$$x_{12} - x_1 + x_{22} - x_2 = -0.46.$$

A nonlinear transformation φ is needed to map the data from its original attribute space into a new space such that a linear hyperplane can separate the classes. This can be achieved by using the following simple transformation:

$$\varphi: (x_1, x_2) \rightarrow (x_{12} - x_1, x_{22} - x_2). \quad (4.92)$$

Figure 4.39(b) shows the points in the transformed space, where we can see that all the circles are located in the lower left-hand side of the diagram. A linear hyperplane with parameters w and b can therefore be constructed in the transformed space, to separate the instances into their respective classes.

One may think that because the nonlinear transformation possibly increases the dimensionality of the input space, this approach can suffer from the curse of dimensionality that is often associated with high-dimensional data.

However, as we will see in the following section, nonlinear SVM is able to avoid this problem by using kernel functions.

Learning a Nonlinear SVM Model

Using a suitable function, $\phi(\cdot)$, we can transform any data instance x to $\phi(x)$. (The details on how to choose $\phi(\cdot)$ will become clear later.) The linear hyperplane in the transformed space can be expressed as $w^T\phi(x) + b = 0$. To learn the optimal separating hyperplane, we can substitute $\phi(x)$ for x in the formulation of SVM to obtain the following optimization problem:

$$\min_w, b, \xi \text{ s.t. } \|w\|^2 + C \sum_{i=1}^n \xi_i \text{ subject to } y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0. \quad (4.93)$$

Using Lagrange multipliers λ_i , this can be converted into a dual optimization problem: max

$$\begin{aligned} & \max \lambda \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \\ & \text{subject to } \sum_{i=1}^n \lambda_i y_i = 0, 0 \leq \lambda_i \leq C, \end{aligned} \quad (4.94)$$

where $\langle a, b \rangle$ denotes the inner product between vectors a and b . Also, the equation of the hyperplane in the transformed space can be represented using

λ_i as follows:

$$\sum_{i=1}^n \lambda_i y_i \langle \phi(x_i), \phi(x) \rangle + b = 0. \quad (4.95)$$

Further, b is given by

$$b = \min_{i \in S} (\sum_{j \in S} \lambda_j y_j \langle \phi(x_i), \phi(x_j) \rangle y_j) \quad (4.96)$$

where $S = \{i | 0 > \lambda_i < C\}$ is the set of support vectors residing on the margin hyperplanes and n_S is the number of elements in S .

Note that in order to solve the dual optimization problem in [Equation 4.94](#), or to use the learned model parameters to make predictions using [Equations 4.95](#) and [4.96](#), we need only inner products of $\varphi(x)$. Hence, even though $\varphi(x)$ may be nonlinear and high-dimensional, it suffices to use a function of the inner products of $\varphi(x)$ in the transformed space. This can be achieved by using a kernel trick, which can be described as follows.

The inner product between two vectors is often regarded as a measure of similarity between the vectors. For example, the cosine similarity described in [Section 2.4.5](#) on [page 79](#) can be defined as the dot product between two vectors that are normalized to unit length. Analogously, the inner product $\varphi(x_i), \varphi(x_j)$ can also be regarded as a measure of similarity between two instances, x_i and x_j , in the transformed space. The **kernel trick** is a method for computing this similarity as a function of the original attributes. Specifically, the kernel function $K(u, v)$ between two instances u and v can be defined as follows:

$$K(u, v) = \langle \varphi(u), \varphi(v) \rangle = f(u, v) \quad (4.97)$$

where $f(\cdot)$ is a function that follows certain conditions as stated by the Mercer's Theorem. Although the details of this theorem are outside the scope of the book, we provide a list of some of the commonly used kernel functions:

$$\text{Polynomial kernel } K(u, v) = (u^T v + 1)^p \quad (4.98)$$

$$\text{Radial Basis Function } K(u, v) = e^{-\|u - v\|^2 / (2\sigma^2)} \quad (4.99)$$

$$\text{Sigmoid kernel } K(u, v) = \tanh(k u^T v - \delta) \quad (4.100)$$

By using a kernel function, we can directly work with inner products in the transformed space without dealing with the exact forms of the nonlinear transformation function φ . Specifically, this allows us to use high-dimensional transformations (sometimes even involving infinitely many dimensions), while performing calculations only in the original attribute space. Computing the inner products using kernel functions is also considerably cheaper than using the transformed attribute set $\varphi(x)$. Hence, the use of kernel functions provides a significant advantage in representing nonlinear decision boundaries, without suffering from the curse of dimensionality. This has been one of the major reasons behind the widespread usage of SVM in highly complex and nonlinear problems.

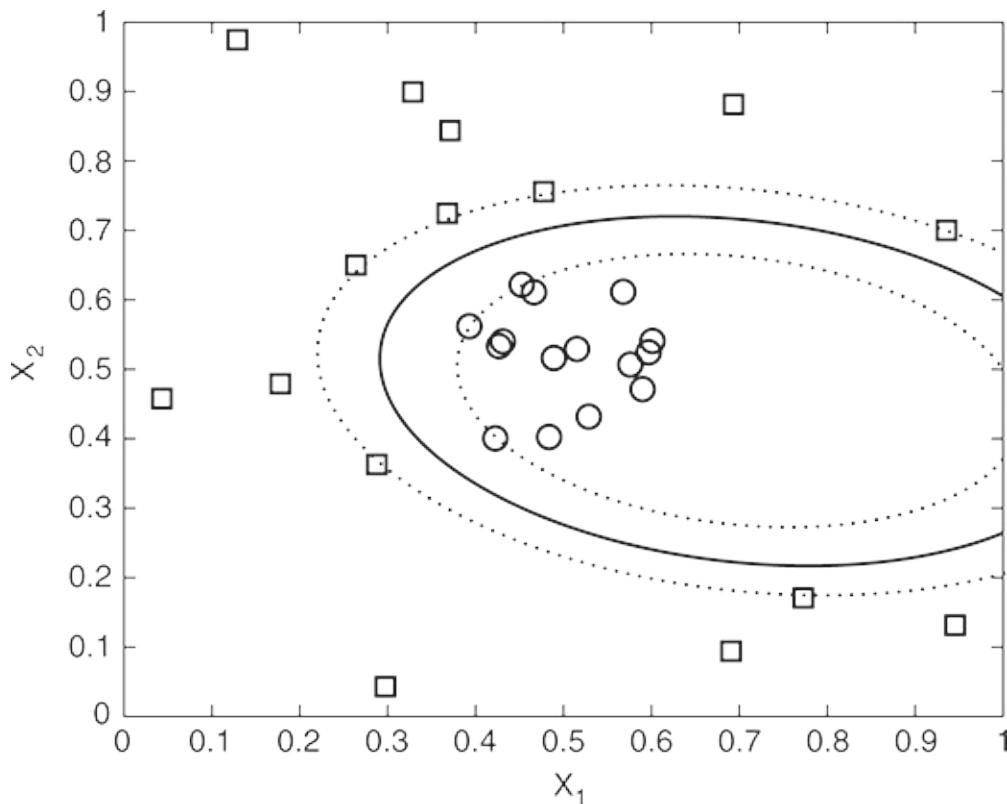


Figure 4.40.

Decision boundary produced by a nonlinear SVM with polynomial kernel.

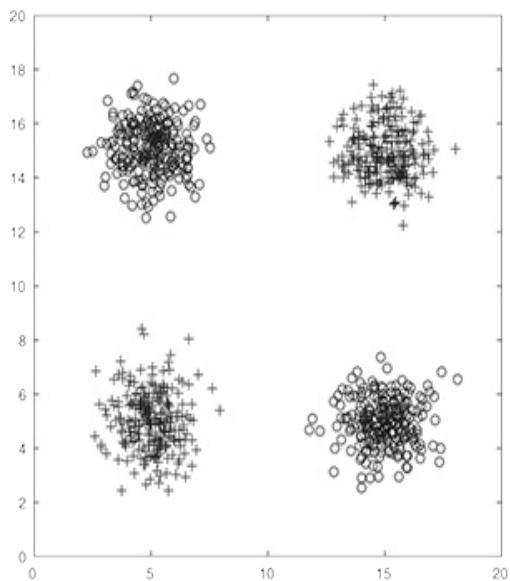
Figure 4.40 shows the nonlinear decision boundary obtained by SVM using the polynomial kernel function given in [Equation 4.98](#). We can see that the

learned decision boundary is quite close to the true decision boundary shown in [Figure 4.39\(a\)](#). Although the choice of kernel function depends on the characteristics of the input data, a commonly used kernel function is the radial basis function (RBF) kernel, which involves a single hyper-parameter σ , known as the standard deviation of the RBF kernel.

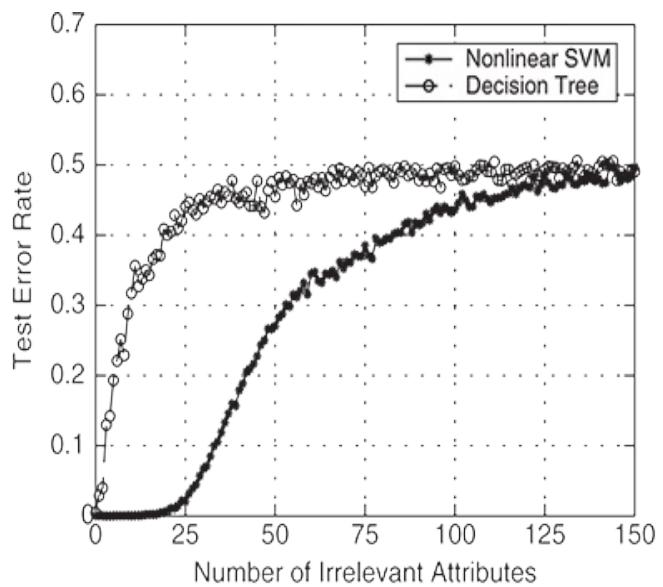
4.9.5 Characteristics of SVM

1. The SVM learning problem can be formulated as a convex optimization problem, in which efficient algorithms are available to find the global minimum of the objective function. Other classification methods, such as rule-based classifiers and artificial neural networks, employ a greedy strategy to search the hypothesis space. Such methods tend to find only locally optimum solutions.
2. SVM provides an effective way of regularizing the model parameters by maximizing the margin of the decision boundary. Furthermore, it is able to create a balance between model complexity and training errors by using a hyper-parameter C . This trade-off is generic to a broader class of model learning techniques that capture the model complexity and the training loss using different formulations.
3. Linear SVM can handle irrelevant attributes by learning zero weights corresponding to such attributes. It can also handle redundant attributes by learning similar weights for the duplicate attributes. Furthermore, the ability of SVM to regularize its learning makes it more robust to the presence of a large number of irrelevant and redundant attributes than other classifiers, even in high-dimensional settings. For this reason, nonlinear SVMs are less impacted by irrelevant and redundant attributes than other highly expressive classifiers that can learn nonlinear decision boundaries such as decision trees.

To compare the effect of irrelevant attributes on the performance of nonlinear SVMs and decision trees, consider the two-dimensional data set shown in [Figure 4.41\(a\)](#) containing 500+ and 500o instances, where the two classes can be easily separated using a nonlinear decision boundary. We incrementally add irrelevant attributes to this data set and compare the performance of two classifiers: decision tree and nonlinear SVM (using radial basis function kernel), using 70% of the data for training and the rest for testing. [Figure 4.41\(b\)](#) shows the test error rates of the two classifiers as we increase the number of irrelevant attributes. We can see that the test error rate of decision trees swiftly reaches 0.5 (same as random guessing) in the presence of even a small number of irrelevant attributes. This can be attributed to the problem of multiple comparisons while choosing splitting attributes at internal nodes as discussed in [Example 3.7](#) of the previous chapter. On the other hand, nonlinear SVM shows a more robust and steady performance even after adding a moderately large number of irrelevant attributes. Its test error rate gradually declines and eventually reaches close to 0.5 after adding 125 irrelevant attributes, at which point it becomes difficult to discern the discriminative information in the original two attributes from the noise in the remaining attributes for learning nonlinear decision boundaries.



(a) Original data with two attributes.



(b) Test error rates after adding irrelevant attributes to the original data.

Figure 4.41.

Comparing the effect of adding irrelevant attributes on the performance of nonlinear SVMs and decision trees.

4. SVM can be applied to categorical data by introducing dummy variables for each categorical attribute value present in the data. For example, if `Marital Status` has three values {Single, Married, Divorced}, we can introduce a binary variable for each of the attribute values.
5. The SVM formulation presented in this chapter is for binary class problems. However, multiclass extensions of SVM have also been proposed.
6. Although the training time of an SVM model can be large, the learned parameters can be succinctly represented with the help of a small number of support vectors, making the classification of test instances quite fast.

4.10 Ensemble Methods

This section presents techniques for improving classification accuracy by aggregating the predictions of multiple classifiers. These techniques are known as **ensemble** or **classifier combination** methods. An ensemble method constructs a set of **base classifiers** from training data and performs classification by taking a vote on the predictions made by each base classifier. This section explains why ensemble methods tend to perform better than any single classifier and presents techniques for constructing the classifier ensemble.

4.10.1 Rationale for Ensemble Method

The following example illustrates how an ensemble method can improve a classifier's performance.

Example 4.8.

Consider an ensemble of 25 binary classifiers, each of which has an error rate of $\epsilon=0.35$. The ensemble classifier predicts the class label of a test example by taking a majority vote on the predictions made by the base classifiers. If the base classifiers are identical, then all the base classifiers will commit the same mistakes. Thus, the error rate of the ensemble remains 0.35. On the other hand, if the base classifiers are independent—i.e., their errors are uncorrelated—then the ensemble makes a wrong prediction only if more than half of the base classifiers predict incorrectly. In this case, the error rate of the ensemble classifier is

$$\text{eensemble} = \sum_{i=1}^{25} (25i) \in_i (1 - \epsilon)^{25-i} = 0.06, \quad (4.101)$$

which is considerably lower than the error rate of the base classifiers.

Figure 4.42 shows the error rate of an ensemble of 25 binary classifiers (eensemble) for different base classifier error rates (ϵ). The diagonal line represents the case in which the base classifiers are identical, while the solid line represents the case in which the base classifiers are independent. Observe that the ensemble classifier performs worse than the base classifiers when ϵ is larger than 0.5.

The preceding example illustrates two necessary conditions for an ensemble classifier to perform better than a single classifier: (1) the base classifiers should be independent of each other, and (2) the base classifiers should do better than a classifier that performs random guessing. In practice, it is difficult to ensure total independence among the base classifiers. Nevertheless, improvements in classification accuracies have been observed in ensemble methods in which the base classifiers are somewhat correlated.

4.10.2 Methods for Constructing an Ensemble Classifier

A logical view of the ensemble method is presented in **Figure 4.43**. The basic idea is to construct multiple classifiers from the original data and then aggregate their predictions when classifying unknown examples. The ensemble of classifiers can be constructed in many ways:

1. **By manipulating the training set.** In this approach, multiple training sets are created by resampling the original data according to some sampling distribution and constructing a classifier from each training set. The sampling distribution determines how likely it is that an example will be selected for training, and it may vary from one trial to another. **Bagging** and **boosting** are two examples of ensemble methods that manipulate their training sets. These methods are described in further detail in [Sections 4.10.4](#) and [4.10.5](#).

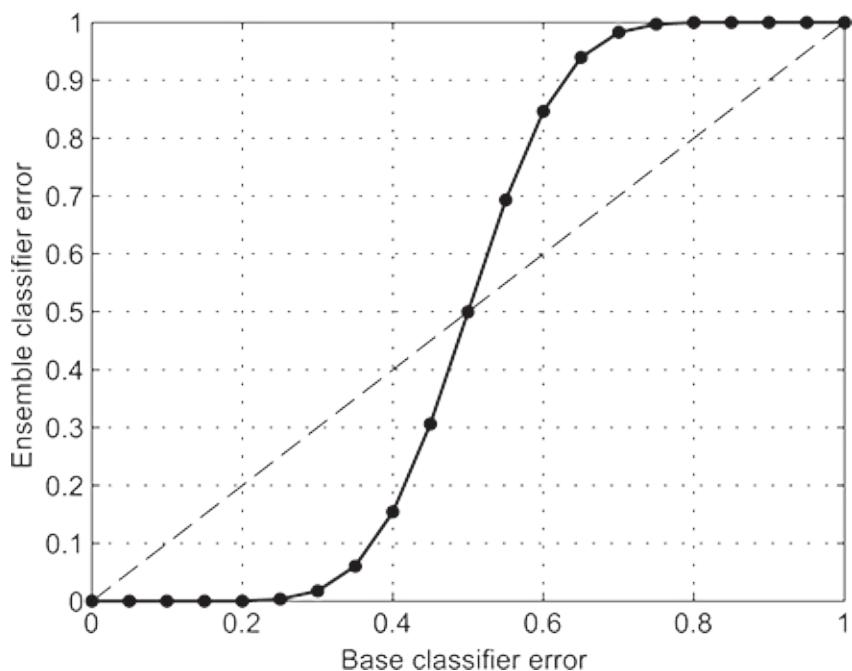


Figure 4.42.

Comparison between errors of base classifiers and errors of the ensemble classifier.

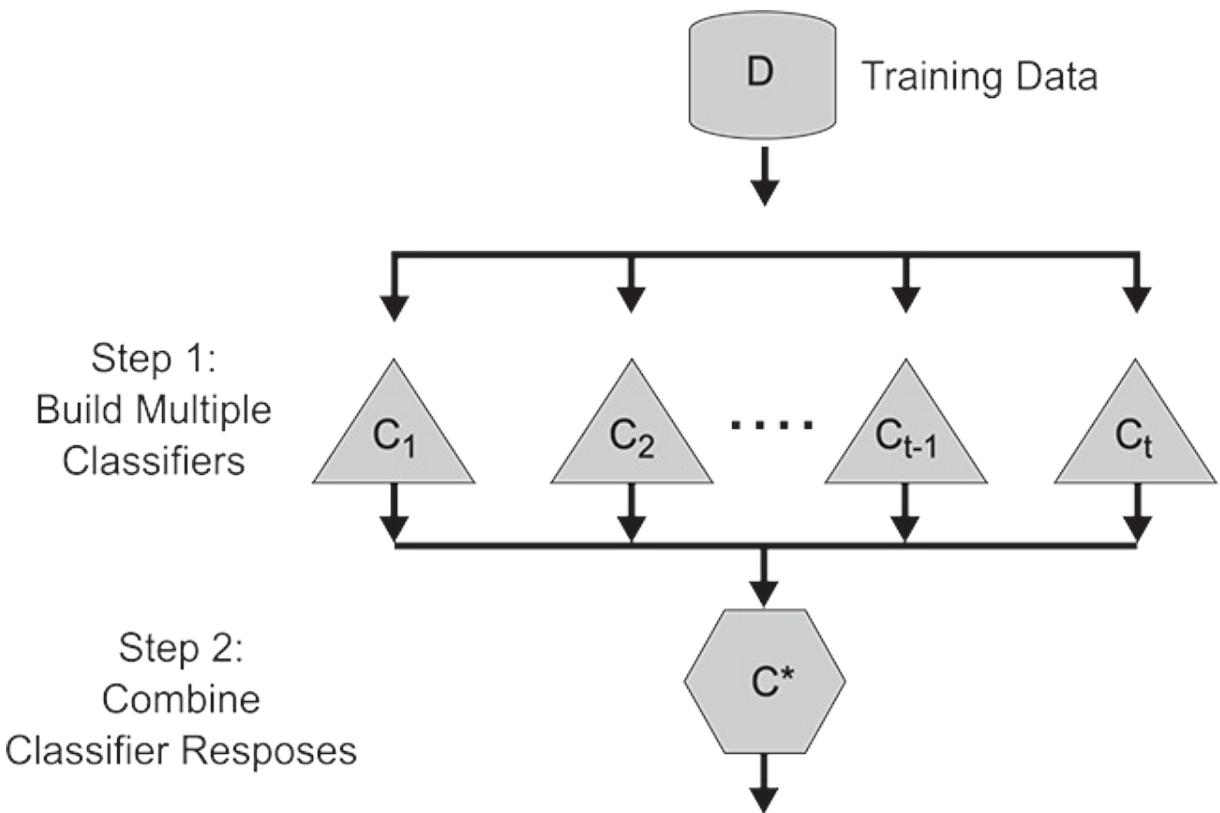


Figure 4.43.

A logical view of the ensemble learning method.

2. **By manipulating the input features.** In this approach, a subset of input features is chosen to form each training set. The subset can be either chosen randomly or based on the recommendation of domain experts. Some studies have shown that this approach works very well with data sets that contain highly redundant features. **Random forest**, which is described in [Section 4.10.6](#), is an ensemble method that manipulates its input features and uses decision trees as its base classifiers.
3. **By manipulating the class labels.** This method can be used when the number of classes is sufficiently large. The training data is transformed into a binary class problem by randomly partitioning the class labels into two disjoint subsets, A0 and A1. Training examples whose class

label belongs to the subset A0 are assigned to class 0, while those that belong to the subset A1 are assigned to class 1. The relabeled examples are then used to train a base classifier. By repeating this process multiple times, an ensemble of base classifiers is obtained. When a test example is presented, each base classifier C_i is used to predict its class label. If the test example is predicted as class 0, then all the classes that belong to A0 will receive a vote. Conversely, if it is predicted to be class 1, then all the classes that belong to A1 will receive a vote. The votes are tallied and the class that receives the highest vote is assigned to the test example. An example of this approach is the **error-correcting output coding** method described on page 331.

4. **By manipulating the learning algorithm.** Many learning algorithms can be manipulated in such a way that applying the algorithm several times on the same training data will result in the construction of different classifiers. For example, an artificial neural network can change its network topology or the initial weights of the links between neurons. Similarly, an ensemble of decision trees can be constructed by injecting randomness into the tree-growing procedure. For example, instead of choosing the best splitting attribute at each node, we can randomly choose one of the top k attributes for splitting.

The first three approaches are generic methods that are applicable to any classifier, whereas the fourth approach depends on the type of classifier used. The base classifiers for most of these approaches can be generated sequentially (one after another) or in parallel (all at once). Once an ensemble of classifiers has been learned, a test example x is classified by combining the predictions made by the base classifiers $C_i(x)$:

$$C^*(x) = f(C_1(x), C_2(x), \dots, C_k(x)).$$

where f is the function that combines the ensemble responses. One simple approach for obtaining $C^*(x)$ is to take a majority vote of the individual predictions. An alternate approach is to take a *weighted* majority vote, where the weight of a base classifier denotes its accuracy or relevance.

Ensemble methods show the most improvement when used with **unstable classifiers**, i.e., base classifiers that are sensitive to minor perturbations in the training set, because of high model complexity. Although unstable classifiers may have a low *bias* in finding the optimal decision boundary, their predictions have a high *variance* for minor changes in the training set or model selection. This trade-off between bias and variance is discussed in detail in the next section. By aggregating the responses of multiple unstable classifiers, ensemble learning attempts to minimize their variance without worsening their bias.

4.10.3 Bias-Variance Decomposition

Bias-variance decomposition is a formal method for analyzing the generalization error of a predictive model. Although the analysis is slightly different for classification than regression, we first discuss the basic intuition of this decomposition by using an analogue of a regression problem.

Consider the illustrative task of reaching a target y by firing projectiles from a starting position \underline{x} , as shown in [Figure 4.44](#). The target corresponds to the desired output at a test instance, while the starting position corresponds to its observed attributes. In this analogy, the projectile represents the model used for predicting the target using the observed attributes. Let \hat{y} denote the point where the projectile hits the ground, which is analogous of the prediction of the model.

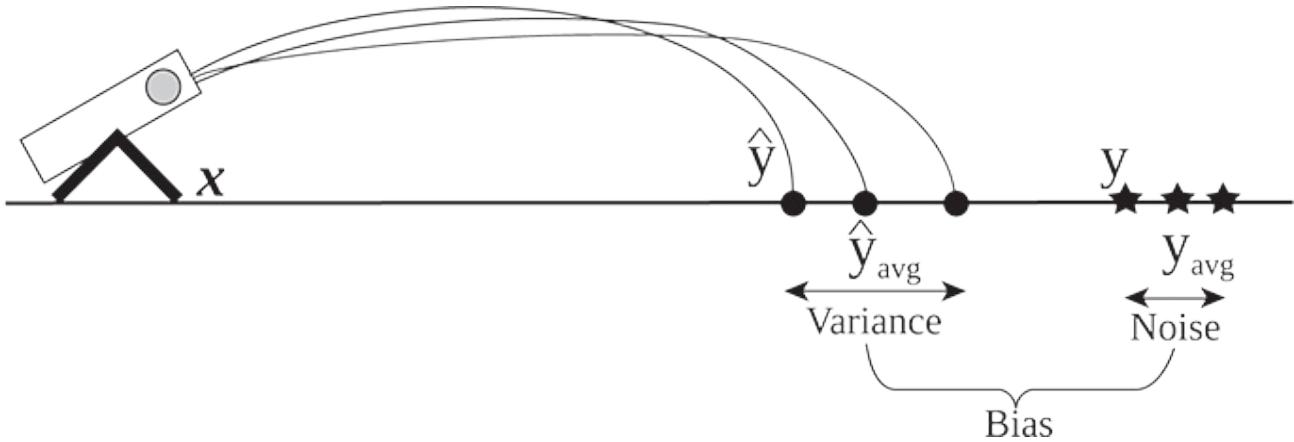


Figure 4.44.

Bias-variance decomposition.

Ideally, we would like our predictions to be as close to the true target as possible. However, note that different trajectories of projectiles are possible based on differences in the training data or in the approach used for model selection. Hence, we can observe a **variance** in the predictions \hat{y} over different runs of projectile. Further, the target in our example is not fixed but has some freedom to move around, resulting in a **noise** component in the true target. This can be understood as the non-deterministic nature of the output variable, where the same set of attributes can have different output values. Let \hat{y}^{avg} represent the *average* prediction of the projectile over multiple runs, and y^{avg} denote the average target value. The difference between \hat{y}^{avg} and y^{avg} is known as the **bias** of the model.

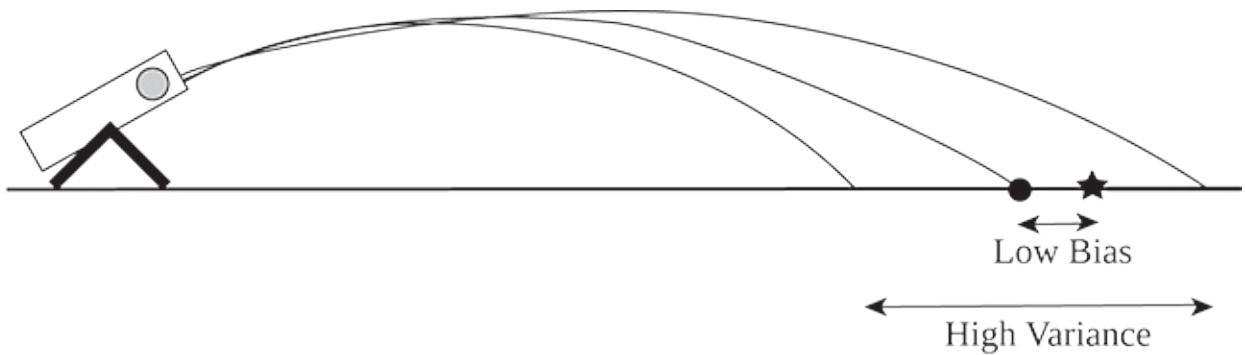
In the context of classification, it can be shown that the generalization error of a classification model m can be decomposed into terms involving the bias, variance, and noise components of the model in the following way:

$$\text{gen.error}(m) = c_1 \times \text{noise} + \text{bias}(m) + c_2 \times \text{variance}(m)$$

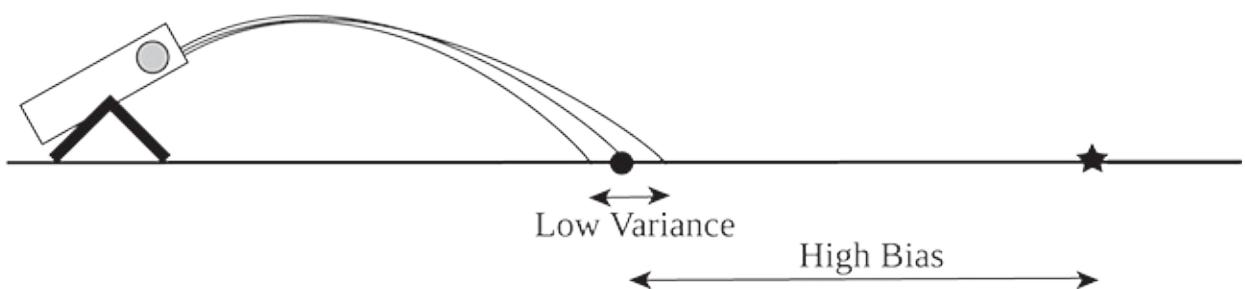
where c_1 and c_2 are constants that depend on the characteristics of training and test sets. Note that while the noise term is intrinsic to the target class, the

bias and variance terms depend on the choice of the classification model. The bias of a model represents how close the average prediction of the model is to the average target. Models that are able to learn complex decision boundaries, e.g., models produced by k -nearest neighbor and multi-layer ANN, generally show low bias. The variance of a model captures the stability of its predictions in response to minor perturbations in the training set or the model selection approach.

We can say that a model shows better generalization performance if it has a lower bias and lower variance. However, if the complexity of a model is high but the training size is small, we generally expect to see a lower bias but *higher variance*, resulting in the phenomena of overfitting. This phenomena is pictorially represented in [Figure 4.45\(a\)](#). On the other hand, an overly simplistic model that suffers from underfitting may show a lower variance but would suffer from a *high bias*, as shown in [Figure 4.45\(b\)](#). Hence, the trade-off between bias and variance provides a useful way for interpreting the effects of underfitting and overfitting on the generalization performance of a model.



(a) Phenomena of Overfitting.



(b) Phenomena of Underfitting.

Figure 4.45.

Plots showing the behavior of two-dimensional solutions with constant L2 and L1 norms.

The bias-variance trade-off can be used to explain why ensemble learning improves the generalization performance of unstable classifiers. If a base classifier shows low bias but high variance, it can become susceptible to overfitting, as even a small change in the training set will result in different predictions. However, by combining the responses of multiple base classifiers, we can expect to reduce the overall variance. Hence, ensemble learning methods show better performance primarily by lowering the variance in the predictions, although they can even help in reducing the bias. One of the simplest approaches for combining predictions and reducing their variance is to compute their average. This forms the basis of the bagging method, described in the following subsection.

4.10.4 Bagging

Bagging, which is also known as bootstrap aggregating, is a technique that repeatedly samples (with replacement) from a data set according to a uniform probability distribution. Each bootstrap sample has the same size as the original data. Because the sampling is done with replacement, some instances may appear several times in the same training set, while others may be omitted from the training set. On average, a bootstrap sample D_i contains approximately 63% of the original training data because each sample has a probability $1 - (1 - 1/N)^N$ of being selected in each D_i . If N is sufficiently large, this probability converges to $1 - 1/e \approx 0.632$. The basic procedure for bagging is summarized in [Algorithm 4.5](#). After training the k classifiers, a test instance is assigned to the class that receives the highest number of votes.

To illustrate how bagging works, consider the data set shown in [Table 4.4](#). Let x denote a one-dimensional attribute and y denote the class label. Suppose we use only one-level binary decision trees, with a test condition $x \leq k$, where k is a split point chosen to minimize the entropy of the leaf nodes. Such a tree is also known as a **decision stump**.

Table 4.4. Example of data set used to construct an ensemble of bagging classifiers.

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Without bagging, the best decision stump we can produce splits the instances at either $x \leq 0.35$ or $x \leq 0.75$. Either way, the accuracy of the tree is at most 70%. Suppose we apply the bagging procedure on the data set using 10 bootstrap samples. The examples chosen for training in each bagging round are shown

in [Figure 4.46](#). On the right-hand side of each table, we also describe the decision stump being used in each round.

We classify the entire data set given in [Table 4.4](#) by taking a majority vote among the predictions made by each base classifier. The results of the predictions are shown in [Figure 4.47](#). Since the class labels are either -1 or $+1$, taking the majority vote is equivalent to summing up the predicted values of y and examining the sign of the resulting sum (refer to the second to last row in [Figure 4.47](#)). Notice that the ensemble classifier perfectly classifies all 10 examples in the original data.

Algorithm 4.5 Bagging algorithm.

```
1: Let  $k$  be the number of bootstrap samples.  
2: for  $i = 1$  to  $k$  do  
3:   Create a bootstrap sample of size  $N$ ,  $D_i$ .  
4:   Train a base classifier  $C_i$  on the bootstrap sample  $D_i$ .  
5: end for  
6:  $C^*(x) = \text{argmax}_{y \in \{-1, +1\}} \sum_i \delta(C_i(x) = y)$ .  
   { $\delta(\cdot) = 1$  if its argument is true and 0 otherwise.}
```

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$$x \leq 0.35 \Rightarrow y = 1$$

$$x > 0.35 \Rightarrow y = -1$$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

$$x \leq 0.65 \Rightarrow y = 1$$

$$x > 0.65 \Rightarrow y = -1$$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$$x \leq 0.35 \Rightarrow y = 1$$

$$x > 0.35 \Rightarrow y = -1$$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$$x \leq 0.3 \Rightarrow y = 1$$

$$x > 0.3 \Rightarrow y = -1$$

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$$x \leq 0.35 \Rightarrow y = 1$$

$$x > 0.35 \Rightarrow y = -1$$

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \Rightarrow y = -1$$

$$x > 0.75 \Rightarrow y = 1$$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$$x \leq 0.75 \Rightarrow y = -1$$

$$x > 0.75 \Rightarrow y = 1$$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \Rightarrow y = -1$$

$$x > 0.75 \Rightarrow y = 1$$

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \Rightarrow y = -1$$

$$x > 0.75 \Rightarrow y = 1$$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$$x \leq 0.05 \Rightarrow y = -1$$

$$x > 0.05 \Rightarrow y = 1$$

Figure 4.46.

Example of bagging.

The preceding example illustrates another advantage of using ensemble methods in terms of enhancing the representation of the target function. Even though each base classifier is a decision stump, combining the classifiers can lead to a decision boundary that mimics a decision tree of depth 2.

Bagging improves generalization error by reducing the variance of the base classifiers. The performance of bagging depends on the stability of the base classifier. If a base classifier is unstable, bagging helps to reduce the errors associated with random fluctuations in the training data. If a base classifier is stable, i.e., robust to minor perturbations in the training set, then the error of the ensemble is primarily caused by bias in the base classifier. In this situation, bagging may not be able to improve the performance of the base classifiers significantly. It may even degrade the classifier's performance because the effective size of each training set is about 37% smaller than the original data.

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 4.47.

Example of combining classifiers constructed using the bagging approach.

4.10.5 Boosting

Boosting is an iterative procedure used to adaptively change the distribution of training examples for learning base classifiers so that they increasingly focus on examples that are hard to classify. Unlike bagging, boosting assigns a weight to each training example and may adaptively change the weight at the end of each boosting round. The weights assigned to the training examples can be used in the following ways:

1. They can be used to inform the sampling distribution used to draw a set of bootstrap samples from the original data.
2. They can be used to learn a model that is biased toward examples with higher weight.

This section describes an algorithm that uses weights of examples to determine the sampling distribution of its training set. Initially, the examples are assigned equal weights, $1/N$, so that they are equally likely to be chosen for training. A sample is drawn according to the sampling distribution of the training examples to obtain a new training set. Next, a classifier is built from the training set and used to classify all the examples in the original data. The weights of the training examples are updated at the end of each boosting round. Examples that are classified incorrectly will have their weights increased, while those that are classified correctly will have their weights decreased. This forces the classifier to focus on examples that are difficult to classify in subsequent iterations.

The following table shows the examples chosen during each boosting round, when applied to the data shown in [Table 4.4](#).

Boosting (Round 1):	7	3	2	8	7	9	4	10	6	3
---------------------	---	---	---	---	---	---	---	----	---	---

Boosting (Round 2):	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3):	4	4	8	10	4	5	4	6	3	4

Initially, all the examples are assigned the same weights. However, some examples may be chosen more than once, e.g., examples 3 and 7, because the sampling is done with replacement. A classifier built from the data is then used to classify all the examples. Suppose example 4 is difficult to classify. The weight for this example will be increased in future iterations as it gets misclassified repeatedly. Meanwhile, examples that were not chosen in the previous round, e.g., examples 1 and 5, also have a better chance of being selected in the next round since their predictions in the previous round were likely to be wrong. As the boosting rounds proceed, examples that are the hardest to classify tend to become even more prevalent. The final ensemble is obtained by aggregating the base classifiers obtained from each boosting round.

Over the years, several implementations of the boosting algorithm have been developed. These algorithms differ in terms of (1) how the weights of the training examples are updated at the end of each boosting round, and (2) how the predictions made by each classifier are combined. An implementation called AdaBoost is explored in the next section.

AdaBoost

Let $\{(x_j, y_j) | j=1, 2, \dots, N\}$ denote a set of N training examples. In the AdaBoost algorithm, the importance of a base classifier C_i depends on its

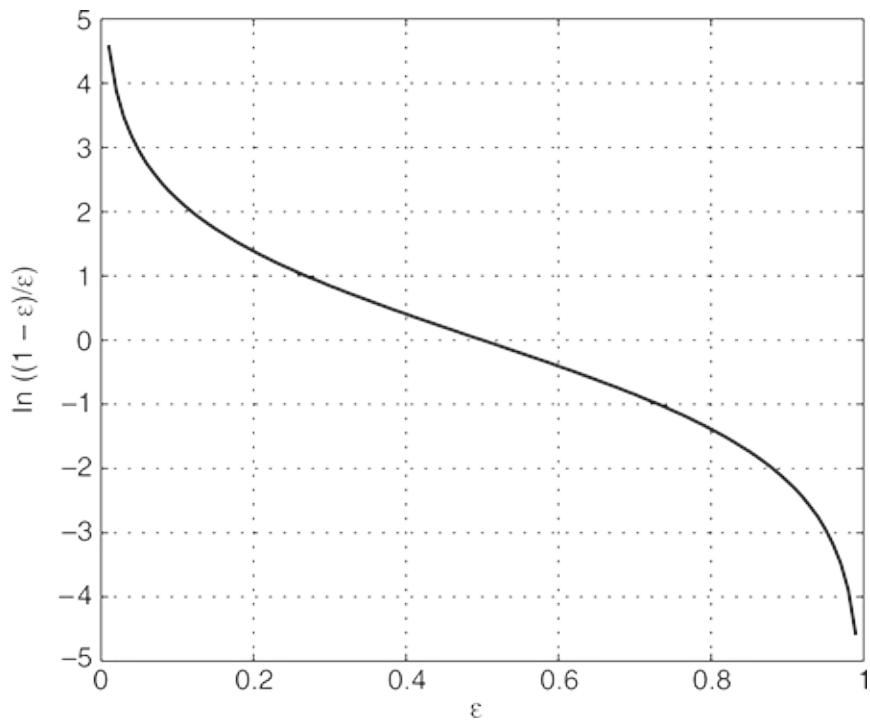


Figure 4.48.

Plot of α as a function of training error ϵ .

error rate, which is defined as

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^N w_j I(C_i(x_j) \neq y_j), \quad (4.102)$$

where $I(p)=1$ if the predicate p is true, and 0 otherwise. The importance of a classifier C_i is given by the following parameter,

$$\alpha_i = 12 \ln(1 - \epsilon_i).$$

Note that α_i has a large positive value if the error rate is close to 0 and a large negative value if the error rate is close to 1, as shown in [Figure 4.48](#).

The α_i parameter is also used to update the weight of the training examples. To illustrate, let $w_i(j)$ denote the weight assigned to example (x_i, y_i) during the

j^{th} boosting round. The weight update mechanism for AdaBoost is given by the equation:

$$w_i(j+1) = w_i(j) Z_j \times \begin{cases} e^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ e^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases} \quad (4.103)$$

where Z_j is the normalization factor used to ensure that $\sum_i w_i(j+1) = 1$. The weight update formula given in [Equation 4.103](#) increases the weights of incorrectly classified examples and decreases the weights of those classified correctly.

Instead of using a majority voting scheme, the prediction made by each classifier C_j is weighted according to α_j . This approach allows AdaBoost to penalize models that have poor accuracy, e.g., those generated at the earlier boosting rounds. In addition, if any intermediate rounds produce an error rate higher than 50%, the weights are reverted back to their original uniform values, $w_i = 1/N$, and the resampling procedure is repeated. The AdaBoost algorithm is summarized in [Algorithm 4.6](#).

Algorithm 4.6 AdaBoost algorithm.

```

1: w = { $w_j = 1/N \mid j = 1, 2, \dots, N$ } . {Initialize the weights for all  $N$ 
examples.}

2: Let  $k$  be the number of boosting rounds.

3: for  $i = 1$  to  $k$  do

4:   Create training set  $D_i$  by sampling (with replacement) from  $D$ 
according to w.

5:   Train a base classifier  $C_i$  on  $D_i$ .

6:   Apply  $C_i$  to all examples in the original training set,  $D$ .

7:    $\epsilon_i = \frac{1}{N} \sum_j w_j \delta(C_i(x_j) \neq y_j)$  {Calculate the weighted error.}

8:   if  $\epsilon_i > 0.5$  then

```

```

9:      w = {wj = 1/N | j = 1, 2, ..., N}. {Reset the weights for all N
examples.}

10:     Go back to Step 4.

11:   end if

12:   αi=12ln1-εiεi.

13:   Update the weight of each example according to Equation 4.103.
14: end for

15: C*(x)=argmaxyΣj=1Tαjδ(Cj(x)=y) ...

```

Let us examine how the boosting approach works on the data set shown in [Table 4.4](#). Initially, all the examples have identical weights. After three boosting rounds, the examples chosen for training are shown in [Figure 4.49\(a\)](#). The weights for each example are updated at the end of each boosting round using [Equation 4.103](#), as shown in [Figure 4.50\(b\)](#).

Without boosting, the accuracy of the decision stump is, at best, 70%. With AdaBoost, the results of the predictions are given in [Figure 4.50\(b\)](#). The final prediction of the ensemble classifier is obtained by taking a weighted average of the predictions made by each base classifier, which is shown in the last row of [Figure 4.50\(b\)](#). Notice that AdaBoost perfectly classifies all the examples in the training data.

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting.

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records.

Figure 4.49.

Example of boosting.

An important analytical result of boosting shows that the training error of the ensemble is bounded by the following expression:

$$e_{\text{ensemble}} \leq \prod_{i=1}^n e_i [1 - e_i], \quad (4.104)$$

where e_i is the error rate of each base classifier i . If the error rate of the base classifier is less than 50%, we can write $e_i = 0.5 - \gamma_i$, where γ_i measures how much better the classifier is than random guessing. The bound on the training error of the ensemble becomes

$$\text{eensemble} \leq \prod_{i=1}^n \gamma_i^2 \leq \exp(-2 \sum_i \gamma_i^2). \quad (4.105)$$

Hence, the training error of the ensemble decreases exponentially, which leads to the fast convergence of the algorithm. By focusing on examples that are difficult to classify by base classifiers, it is able to reduce the bias of the final predictions along with the variance. AdaBoost has been shown to provide significant improvements in performance over base classifiers on a range of data sets. Nevertheless, because of its tendency to focus on training examples that are wrongly classified, the boosting technique can be susceptible to overfitting, resulting in poor generalization performance in some scenarios.

Round	Split Point	Left Class	Right Class	α
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

(a)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

(b)

Figure 4.50.

Example of combining classifiers constructed using the AdaBoost approach.

4.10.6 Random Forests

Random forests attempt to improve the generalization performance by constructing an ensemble of *decorrelated* decision trees. Random forests build on the idea of bagging to use a different bootstrap sample of the training data for learning decision trees. However, a key distinguishing feature of random forests from bagging is that at every internal node of a tree, the best splitting criterion is chosen among a small set of randomly selected attributes. In this way, random forests construct ensembles of decision trees by not only manipulating training instances (by using bootstrap samples similar to bagging), but also the input attributes (by using different subsets of attributes at every internal node).

Given a training set D consisting of n instances and d attributes, the basic procedure of training a random forest classifier can be summarized using the following steps:

1. Construct a bootstrap sample D_i of the training set by randomly sampling n instances (with replacement) from D .
2. Use D_i to learn a decision tree T_i as follows. At every internal node of T_i , randomly sample a set of p attributes and choose an attribute from this subset that shows the maximum reduction in an impurity measure for splitting. Repeat this procedure till every leaf is pure, i.e., containing instances from the same class.

Once an ensemble of decision trees have been constructed, their average prediction (majority vote) on a test instance is used as the final prediction of the random forest. Note that the decision trees involved in a random forest are *unpruned* trees, as they are allowed to grow to their largest possible size till every leaf is pure. Hence, the base classifiers of random forest represent

unstable classifiers that have low bias but high variance, because of their large size.

Another property of the base classifiers learned in random forests is the *lack of correlation* among their model parameters and test predictions. This can be attributed to the use of an independently sampled data set D_i for learning every decision tree T_i , similar to the bagging approach. However, random forests have the additional advantage of choosing a splitting criterion at every internal node using a different (and randomly selected) subset of attributes. This property significantly helps in breaking the correlation structure, if any, among the decision trees T_i .

To realize this, consider a training set involving a large number of attributes, where only a small subset of attributes are strong predictors of the target class, whereas other attributes are weak indicators. Given such a training set, even if we consider different bootstrap samples D_i for learning T_i , we would mostly be choosing the same attributes for splitting at internal nodes, because the weak attributes would be largely overlooked when compared with the strong predictors. This can result in a considerable correlation among the trees. However, if we restrict the choice of attributes at every internal node to a random subset of attributes, we can ensure the selection of both strong and weak predictors, thus promoting diversity among the trees. This principle is utilized by random forests for creating decorrelated decision trees.

By aggregating the predictions of an ensemble of strong and decorrelated decision trees, random forests are able to reduce the variance of the trees without negatively impacting their low bias. This makes random forests quite robust to overfitting. Additionally, because of their ability to consider only a small subset of attributes at every internal node, random forests are computationally fast and robust even in high-dimensional settings.

The number of attributes to be selected at every node, p , is a hyper-parameter of the random forest classifier. A small value of p can reduce the correlation among the classifiers but may also reduce their strength. A large value can improve their strength but may result in correlated trees similar to bagging. Although common suggestions for p in the literature include d and $\log_2 d + 1$, a suitable value of p for a given training set can always be selected by tuning it over a validation set, as described in the previous chapter. However, there is an alternative way for selecting hyper-parameters in random forests, which does not require using a separate validation set. It involves computing a reliable estimate of the generalization error rate directly during training, known as the **out-of-bag (oob)** error estimate. The oob estimate can be computed for any generic ensemble learning method that builds independent base classifiers using bootstrap samples of the training set, e.g., bagging and random forests. The approach for computing oob estimate can be described as follows.

Consider an ensemble learning method that uses an independent base classifier T_i built on a bootstrap sample of the training set D_i . Since every training instance x will be used for training approximately 63% of base classifiers, we can call x as an **out-of-bag sample** for the remaining 27% of base classifiers that did not use it for training. If we use these remaining 27% classifiers to make predictions on x , we can obtain the **oob error** on x by taking their majority vote and comparing it with its class label. Note that the oob error estimates the error of 27% classifiers on an instance that was not used for training those classifiers. Hence, the oob error can be considered as a reliable estimate of generalization error. By taking the average of oob errors of all training instances, we can compute the overall oob error estimate. This can be used as an alternative to the validation error rate for selecting hyper-parameters. Hence, random forests do not need to use a separate partition of the training set for validation, as it can simultaneously train the base classifiers and compute generalization error estimates on the same data set.

Random forests have been empirically found to provide significant improvements in generalization performance that are often comparable, if not superior, to the improvements provided by the AdaBoost algorithm. Random forests are also more robust to overfitting and run much faster than the AdaBoost algorithm.

4.10.7 Empirical Comparison among Ensemble Methods

Table 4.5 shows the empirical results obtained when comparing the performance of a decision tree classifier against bagging, boosting, and random forest. The base classifiers used in each ensemble method consist of 50 decision trees. The classification accuracies reported in this table are obtained from tenfold cross-validation. Notice that the ensemble classifiers generally outperform a single decision tree classifier on many of the data sets.

Table 4.5. Comparing the accuracy of a decision tree classifier against three ensemble methods.

Data Set	Number of (Attributes, Classes, Instances)	Decision Tree (%)	Bagging(%)	Boosting(%)	RF(%)
Anneal	(39, 6, 898)	92.09	94.43	95.43	95.43
Australia	(15, 2, 690)	85.51	87.10	85.22	85.80
Auto	(26, 7, 205)	81.95	85.37	85.37	84.39
Breast	(11, 2, 699)	95.14	96.42	97.28	96.14
Cleve	(14, 2, 303)	76.24	81.52	82.18	82.18

Credit	(16, 2, 690)	85.8	86.23	86.09	85.8
Diabetes	(9, 2, 768)	72.40	76.30	73.18	75.13
German	(21, 2, 1000)	70.90	73.40	73.00	74.5
Glass	(10, 7, 214)	67.29	76.17	77.57	78.04
Heart	(14, 2, 270)	80.00	81.48	80.74	83.33
Hepatitis	(20, 2, 155)	81.94	81.29	83.87	83.23
Horse	(23, 2, 368)	85.33	85.87	81.25	85.33
Ionosphere	(35, 2, 351)	89.17	92.02	93.73	93.45
Iris	(5, 3, 150)	94.67	94.67	94.00	93.33
Labor	(17, 2, 57)	78.95	84.21	89.47	84.21
Led7	(8, 10, 3200)	73.34	73.66	73.34	73.06
Lymphography	(19, 4, 148)	77.03	79.05	85.14	82.43
Pima	(9, 2, 768)	74.35	76.69	73.44	77.60
Sonar	(61, 2, 208)	78.85	78.85	84.62	85.58
Tic-tac-toe	(10, 2, 958)	83.72	93.84	98.54	95.82
Vehicle	(19, 4, 846)	71.04	74.11	78.25	74.94
Waveform	(22, 3, 5000)	76.44	83.30	83.90	84.04
Wine	(14, 3, 178)	94.38	96.07	97.75	97.75
Zoo	(17, 7, 101)	93.07	93.07	95.05	97.03

4.11 Class Imbalance Problem

In many data sets there are a disproportionate number of instances that belong to different classes, a property known as **skew** or **class imbalance**. For example, consider a health-care application where diagnostic reports are used to decide whether a person has a rare disease. Because of the infrequent nature of the disease, we can expect to observe a smaller number of subjects who are positively diagnosed. Similarly, in credit card fraud detection, fraudulent transactions are greatly outnumbered by legitimate transactions.

The degree of imbalance between the classes varies across different applications and even across different data sets from the same application. For example, the risk for a rare disease may vary across different populations of subjects depending on their dietary and lifestyle choices. However, despite their infrequent occurrences, a correct classification of the rare class often has greater value than a correct classification of the majority class. For example, it may be more dangerous to ignore a patient suffering from a disease than to misdiagnose a healthy person.

More generally, class imbalance poses two challenges for classification. First, it can be difficult to find sufficiently many labeled samples of a rare class. Note that many of the classification methods discussed so far work well only when the training set has a balanced representation of both classes. Although some classifiers are more effective at handling imbalance in the training data than others, e.g., rule-based classifiers and k -NN, they are all impacted if the minority class is not well-represented in the training set. In general, a classifier trained over an imbalanced data set shows a bias toward improving its performance over the majority class, which is often not the desired behavior.

As a result, many existing classification models, when trained on an imbalanced data set, may not effectively detect instances of the rare class.

Second, accuracy, which is the traditional measure for evaluating classification performance, is not well-suited for evaluating models in the presence of class imbalance in the test data. For example, if 1% of the credit card transactions are fraudulent, then a trivial model that predicts every transaction as legitimate will have an accuracy of 99% even though it fails to detect any of the fraudulent activities. Thus, there is a need to use alternative evaluation metrics that are sensitive to the skew and can capture different criteria of performance than accuracy.

In this section, we first present some of the generic methods for building classifiers when there is class imbalance in the training set. We then discuss methods for evaluating classification performance and adapting classification decisions in the presence of a skewed test set. In the remainder of this section, we will consider binary classification problems for simplicity, where the minority class is referred as the positive (+) class while the majority class is referred as the negative (-) class.

4.11.1 Building Classifiers with Class Imbalance

There are two primary considerations for building classifiers in the presence of class imbalance in the training set. First, we need to ensure that the learning algorithm is trained over a data set that has adequate representation of both the majority as well as the minority classes. Some common approaches for ensuring this includes the methodologies of oversampling and undersampling

the training set. Second, having learned a classification model, we need a way to adapt its classification decisions (and thus create an appropriately tuned classifier) to best match the requirements of the imbalanced test set. This is typically done by converting the outputs of the classification model to real-valued scores, and then selecting a suitable threshold on the classification score to match the needs of a test set. Both these considerations are discussed in detail in the following.

Oversampling and Undersampling

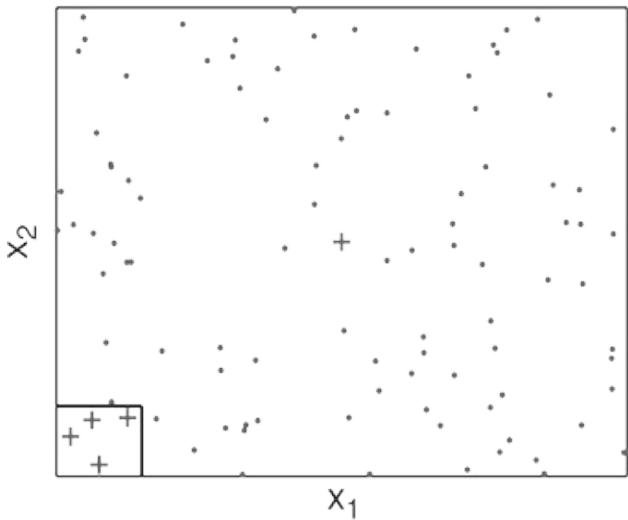
The first step in learning with imbalanced data is to transform the training set to a balanced training set, where both classes have nearly equal representation. The balanced training set can then be used with any of the existing classification techniques (without making any modifications in the learning algorithm) to learn a model that gives equal emphasis to both classes. In the following, we present some of the common techniques for transforming an imbalanced training set to a balanced one.

A basic approach for creating balanced training sets is to generate a *sample* of training instances where the rare class has adequate representation. There are two types of sampling methods that can be used to enhance the representation of the minority class: (a) **undersampling**, where the frequency of the majority class is reduced to match the frequency of the minority class, and (b) **oversampling**, where artificial examples of the minority class are created to make them equal in proportion to the number of negative instances.

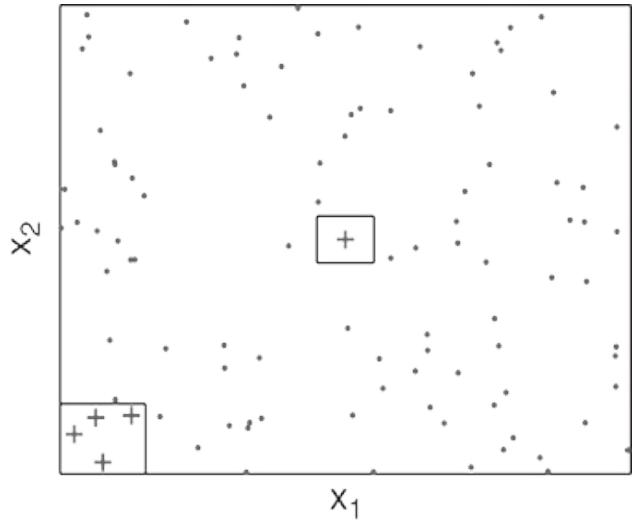
To illustrate undersampling, consider a training set that contains 100 positive examples and 1000 negative examples. To overcome the skew among the classes, we can select a random sample of 100 examples from the negative class and use them with the 100 positive examples to create a balanced training set. A classifier built over the resultant balanced set will then be

unbiased toward both classes. However, one limitation of undersampling is that some of the useful negative examples (e.g., those closer to the actual decision boundary) may not be chosen for training, therefore, resulting in an inferior classification model. Another limitation is that the smaller sample of 100 negative instances may have a higher variance than the larger set of 1000.

Oversampling attempts to create a balanced training set by artificially generating new positive examples. A simple approach for oversampling is to duplicate every positive instance $n-/n+$ times, where $n+$ and $n-$ are the numbers of positive and negative training instances, respectively. **Figure 4.51** illustrates the effect of oversampling on the learning of a decision boundary using a classifier such as a decision tree. Without oversampling, only the positive examples at the bottom right-hand side of **Figure 4.51(a)** are classified correctly. The positive example in the middle of the diagram is misclassified because there are not enough examples to justify the creation of a new decision boundary to separate the positive and negative instances. Oversampling provides the additional examples needed to ensure that the decision boundary surrounding the positive example is not pruned, as illustrated in **Figure 4.51(b)**. Note that duplicating a positive instance is analogous to doubling its weight during the training stage. Hence, the effect of oversampling can be alternatively achieved by assigning higher weights to positive instances than negative instances. This method of weighting instances can be used with a number of classifiers such as logistic regression, ANN, and SVM.



(a) Without oversampling.



(b) With oversampling.

Figure 4.51.

Illustrating the effect of oversampling of the rare class.

One limitation of the duplication method for oversampling is that the replicated positive examples have an artificially lower variance when compared with their true distribution in the overall data. This can bias the classifier to the specific distribution of training instances, which may not be representative of the overall distribution of test instances, leading to poor generalizability. To overcome this limitation, an alternative approach for oversampling is to generate *synthetic* positive instances in the neighborhood of existing positive instances. In this approach, called the Synthetic Minority Oversampling Technique (SMOTE), we first determine the k -nearest positive neighbors of every positive instance $\textcolor{teal}{x}$, and then generate a synthetic positive instance at some intermediate point along the line segment joining $\textcolor{teal}{x}$ to one of its randomly chosen k -nearest neighbor, x_k . This process is repeated until the desired number of positive instances is reached. However, one limitation of this approach is that it can only generate new positive instances in the convex hull of the existing positive class. Hence, it does not help improve the representation of the positive class outside the boundary of existing positive

instances. Despite their complementary strengths and weaknesses, undersampling and oversampling provide useful directions for generating balanced training sets in the presence of class imbalance.

Assigning Scores to Test Instances

If a classifier returns an ordinal score $s(\underline{x})$ for every test instance \underline{x} such that a higher score denotes a greater likelihood of \underline{x} belonging to the positive class, then for every possible value of score threshold, s_T , we can create a new binary classifier where a test instance \underline{x} is classified positive only if $s(x) > s_T$. Thus, every choice of s_T can potentially lead to a different classifier, and we are interested in finding the classifier that is best suited for our needs.

Ideally, we would like the classification score to vary monotonically with the actual posterior probability of the positive class, i.e., if $s(x_1)$ and $s(x_2)$ are the scores of any two instances, x_1 and x_2 , then

$s(x_1) \geq s(x_2) \Rightarrow P(y=1|x_1) \geq P(y=1|x_2)$. However, this is difficult to guarantee in practice as the properties of the classification score depends on several factors such as the complexity of the classification algorithm and the representative power of the training set. In general, we can only expect the classification score of a reasonable algorithm to be weakly related to the actual posterior probability of the positive class, even though the relationship may not be strictly monotonic. Most classifiers can be easily modified to produce such a real valued score. For example, the signed distance of an instance from the positive margin hyperplane of SVM can be used as a classification score. As another example, test instances belonging to a leaf in a decision tree can be assigned a score based on the fraction of training instances labeled as positive in the leaf. Also, probabilistic classifiers such as naïve Bayes, Bayesian networks, and logistic regression naturally output estimates of posterior probabilities, $P(y=1|x)$. Next, we discuss some

evaluation measures for assessing the goodness of a classifier in the presence of class imbalance.

Table 4.6. A confusion matrix for a binary classification problem in which the classes are not equally important.

		Predicted Class	
		+	-
Actual class	+	f++ (TP)	f+- (FN)
	-	f-+ (FP)	f-- (TN)

4.11.2 Evaluating Performance with Class Imbalance

The most basic approach for representing a classifier's performance on a test set is to use a **confusion matrix**, as shown in [Table 4.6](#). This table is essentially the same as [Table 3.4](#), which was introduced in the context of evaluating classification performance in [Section 3.2](#). A confusion matrix summarizes the number of instances predicted correctly or incorrectly by a classifier using the following four counts:

- True positive (TP) or f++, which corresponds to the number of positive examples correctly predicted by the classifier.
- False positive (FP) or f-+ (also known as Type I error), which corresponds to the number of negative examples wrongly predicted as positive by the classifier.

- False negative (FN) or $f+-$ (also known as Type II error), which corresponds to the number of positive examples wrongly predicted as negative by the classifier.
- True negative (TN) or $f--$, which corresponds to the number of negative examples correctly predicted by the classifier.

The confusion matrix provides a concise representation of classification performance on a given test data set. However, it is often difficult to interpret and compare the performance of classifiers using the four-dimensional representations (corresponding to the four counts) provided by their confusion matrices. Hence, the counts in the confusion matrix are often summarized using a number of **evaluation measures**. Accuracy is an example of one such measure that combines these four counts into a single value, which is used extensively when classes are balanced. However, the accuracy measure is not suitable for handling data sets with imbalanced class distributions as it tends to favor classifiers that correctly classify the majority class. In the following, we describe other possible measures that capture different criteria of performance when working with imbalanced classes.

A basic evaluation measure is the **true positive rate** (TPR), which is defined as the fraction of positive test instances correctly predicted by the classifier:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

In the medical community, TPR is also known as **sensitivity**, while in the information retrieval literature, it is also called **recall** (r). A classifier with a high TPR has a high chance of correctly identifying the positive instances of the data.

Analogously to TPR, the **true negative rate** (TNR) (also known as **specificity**) is defined as the fraction of negative test instances correctly

predicted by the classifier, i.e.,

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

A high TNR value signifies that the classifier correctly classifies any randomly chosen negative instance in the test set. A commonly used evaluation measure that is closely related to TNR is the **false positive rate** (FPR), which is defined as $1 - \text{TNR}$.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Similarly, we can define **false negative rate** (FNR) as $1 - \text{TPR}$.

$$\text{FNR} = \frac{\text{FN}}{\text{FN} + \text{TP}}$$

Note that the evaluation measures defined above do not take into account the **skew** among the classes, which can be formally defined as $\alpha = P/(P+N)$, where P and N denote the number of actual positives and actual negatives, respectively. As a result, changing the relative numbers of P and N will have no effect on TPR, TNR, FPR, or FNR, since they depend only on the fraction of correct classifications for every class, independently of the other class. Furthermore, knowing the values of TPR and TNR (and consequently FNR and FPR) does not by itself help us uniquely determine all four entries of the confusion matrix. However, together with information about the skew factor, α , and the total number of instances, N , we can compute the entire confusion matrix using TPR and TNR, as shown in [Table 4.7](#).

Table 4.7. Entries of the confusion matrix in terms of the TPR, TNR, skew, α , and total number of instances, N .

	Predicted +	Predicted -	

Actual +	TPR×α×N	(1-TPR)×α×N	α×N
Actual -	(1-TNR)×(1-α)×N	TNR×(1-α)×N	(1-α)×N
			N

An evaluation measure that is sensitive to the skew is **precision**, which can be defined as the fraction of correct predictions of the positive class over the total number of positive predictions, i.e.,

Precision, $p = \frac{\text{TP}}{\text{TP} + \text{FP}}$.

Precision is also referred as the **positive predicted value** (PPV). A classifier that has a high precision is likely to have most of its positive predictions correct. Precision is a useful measure for highly skewed test sets where the positive predictions, even though small in numbers, are required to be mostly correct. A measure that is closely related to precision is the **false discovery rate** (FDR), which can be defined as $1-p$.

FDR=FPTP+FP.

Although both FDR and FPR focus on FP, they are designed to capture different evaluation objectives and thus can take quite contrasting values, especially in the presence of class imbalance. To illustrate this, consider a classifier with the following confusion matrix.

		Predicted Class	
		+	-
Actual Class	+	100	0
	-	0	100



Since half of the positive predictions made by the classifier are incorrect, it has a FDR value of $100/(100+100)=0.5$. However, its FPR is equal to $100/(100+900)=0.1$, which is quite low. This example shows that in the presence of high skew (i.e., very small value of α), even a small FPR can result in high FDR. See [Section 10.6](#) for further discussion of this issue.

Note that the evaluation measures defined above provide an incomplete representation of performance, because they either only capture the effect of false positives (e.g., FPR and precision) or the effect of false negatives (e.g., TPR or recall), but not both. Hence, if we optimize only one of these evaluation measures, we may end up with a classifier that shows low FN but high FP, or vice-versa. For example, a classifier that declares every instance to be positive will have a perfect recall, but high FPR and very poor precision. On the other hand, a classifier that is very conservative in classifying an instance as positive (to reduce FP) may end up having high precision but very poor recall. We thus need evaluation measures that account for both types of misclassifications, FP and FN. Some examples of such evaluation measures are summarized by the following definitions.

Positive Likelihood Ratio= TP/FP .
F1 measure= $2rpr+p=2\times TP/2\times TP+FP+FN$.
G=($TP+FN$).

While some of these evaluation measures are invariant to the skew (e.g., the positive likelihood ratio), others (e.g., precision and the F1 measure) are sensitive to skew. Further, different evaluation measures capture the effects of different types of misclassification errors in various ways. For example, the F1 measure represents a harmonic mean between recall and precision, i.e.,

$$F1=\frac{2r}{r+p}+\frac{p}{r+p}$$

Because the harmonic mean of two numbers tends to be closer to the smaller of the two numbers, a high value of F1-measure ensures that both precision and recall are reasonably high. Similarly, the G measure represents the geometric mean between recall and precision. A comparison among harmonic, geometric, and arithmetic means is given in the next example.

Example 4.9.

Consider two positive numbers $a=1$ and $b=5$. Their arithmetic mean is $\mu_a = (a+b)/2=3$ and their geometric mean is $\mu_g=ab=2.236$. Their harmonic mean is $\mu_h=(2\times 1\times 5)/6=1.667$, which is closer to the smaller value between a and b than the arithmetic and geometric means.

A generic extension of the F1 measure is the $F\beta$ measure, which can be defined as follows.

$$F\beta=(\beta^2+1)rpr+\beta^2p=(\beta^2+1)\times TP(\beta^2+1)TP+\beta^2FP+FN. \quad (4.106)$$

Both precision and recall can be viewed as special cases of $F\beta$ by setting $\beta=0$ and $\beta=\infty$, respectively. Low values of β make $F\beta$ closer to precision, and high values make it closer to recall.

A more general measure that captures $F\beta$ as well as accuracy is the weighted accuracy measure, which is defined by the following equation:

$$\text{Weighted accuracy}=\frac{w_1TP+w_4TN}{w_1TP+w_2FP+w_3FN+w_4TN}. \quad (4.107)$$

The relationship between weighted accuracy and other performance measures is summarized in the following table:

Measure	w1	w2	w3	w4
---------	----	----	----	----

Recall	1	1	0	0
Precision	1	0	1	0
$F\beta$	$\beta^2 + 1$	β^2	1	0
Accuracy	1	1	1	1

4.11.3 Finding an Optimal Score Threshold

Given a suitably chosen evaluation measure E and a distribution of classification scores, $s(x)$, on a validation set, we can obtain the optimal score threshold s^* on the validation set using the following approach:

1. Sort the scores in increasing order of their values.
 2. For every unique value of score, s , consider the classification model that assigns an instance x as positive only if $s(x) > s$. Let $E(s)$ denote the performance of this model on the validation set.
 3. Find s^* that maximizes the evaluation measure, $E(s)$.
- $s^* = \text{argmax}_s E(s)$.

Note that s^* can be treated as a hyper-parameter of the classification algorithm that is learned during model selection. Using s^* , we can assign a positive label to a future test instance x only if $s(x) > s^*$. If the evaluation measure E is skew invariant (e.g., Positive Likelihood Ratio), then we can select s^* without knowing the skew of the test set, and the resultant classifier formed using s^* can be expected to show optimal performance on the test set.

(with respect to the evaluation measure E). On the other hand, if E is sensitive to the skew (e.g., precision or F1-measure), then we need to ensure that the skew of the validation set used for selecting s^* is similar to that of the test set, so that the classifier formed using s^* shows optimal test performance with respect to E . Alternatively, given an estimate of the skew of the test data, α , we can use it along with the TPR and TNR on the validation set to estimate all entries of the confusion matrix (see [Table 4.7](#)), and thus the estimate of any evaluation measure E on the test set. The score threshold s^* selected using this estimate of E can then be expected to produce optimal test performance with respect to E . Furthermore, the methodology of selecting s^* on the validation set can help in comparing the test performance of different classification algorithms, by using the optimal values of s^* for each algorithm.

4.11.4 Aggregate Evaluation of Performance

Although the above approach helps in finding a score threshold s^* that provides optimal performance with respect to a desired evaluation measure and a certain amount of skew, α , sometimes we are interested in evaluating the performance of a classifier on a number of possible score thresholds, each corresponding to a different choice of evaluation measure and skew value. Assessing the performance of a classifier over a range of score thresholds is called **aggregate evaluation** of performance. In this style of analysis, the emphasis is not on evaluating the performance of a single classifier corresponding to the optimal score threshold, but to assess the general quality of *ranking* produced by the classification scores on the test set. In general, this helps in obtaining robust estimates of classification performance that are not sensitive to specific choices of score thresholds.

ROC Curve

One of the widely-used tools for aggregate evaluation is the **receiver operating characteristic** (ROC) curve. An ROC curve is a graphical approach for displaying the trade-off between TPR and FPR of a classifier, over varying score thresholds. In an ROC curve, the TPR is plotted along the y-axis and the FPR is shown on the x-axis. Each point along the curve corresponds to a classification model generated by placing a threshold on the test scores produced by the classifier. The following procedure describes the generic approach for computing an ROC curve:

1. Sort the test instances in increasing order of their scores.
2. Select the lowest ranked test instance (i.e., the instance with lowest score). Assign the selected instance and those ranked above it to the positive class. This approach is equivalent to classifying all the test instances as positive class. Because all the positive examples are classified correctly and the negative examples are misclassified, $TPR=FPR=1$.
3. Select the next test instance from the sorted list. Classify the selected instance and those ranked above it as positive, while those ranked below it as negative. Update the counts of TP and FP by examining the actual class label of the selected instance. If this instance belongs to the positive class, the TP count is decremented and the FP count remains the same as before. If the instance belongs to the negative class, the FP count is decremented and TP count remains the same as before.
4. Repeat Step 3 and update the TP and FP counts accordingly until the highest ranked test instance is selected. At this final threshold, $TPR=FPR=0$, as all instances are labeled as negative.
5. Plot the TPR against FPR of the classifier.

Example 4.10. [Generating ROC Curve]

Figure 4.52 shows an example of how to compute the TPR and FPR values for every choice of score threshold. There are five positive examples and five negative examples in the test set. The class labels of the test instances are shown in the first row of the table, while the second row corresponds to the sorted score values for each instance. The next six rows contain the counts of TP , FP , TN , and FN , along with their corresponding TPR and FPR. The table is then filled from left to right. Initially, all the instances are predicted to be positive. Thus, $TP=FP=5$ and $TPR=FPR=1$. Next, we assign the test instance with the lowest score as the negative class. Because the selected instance is actually a positive example, the TP count decreases from 5 to 4 and the FP count is the same as before. The FPR and TPR are updated accordingly. This process is repeated until we reach the end of the list, where $TPR=0$ and $FPR=0$. The ROC curve for this example is shown in **Figure 4.53**.

Class	+	-	+	-	-	-	+	-	+	+	+	
	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00	
TP	5	4	4	3	3	3	3	2	2	1	0	0
FP	5	5	4	4	3	2	1	1	0	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0	0

Figure 4.52.

Computing the TPR and FPR at every score threshold.

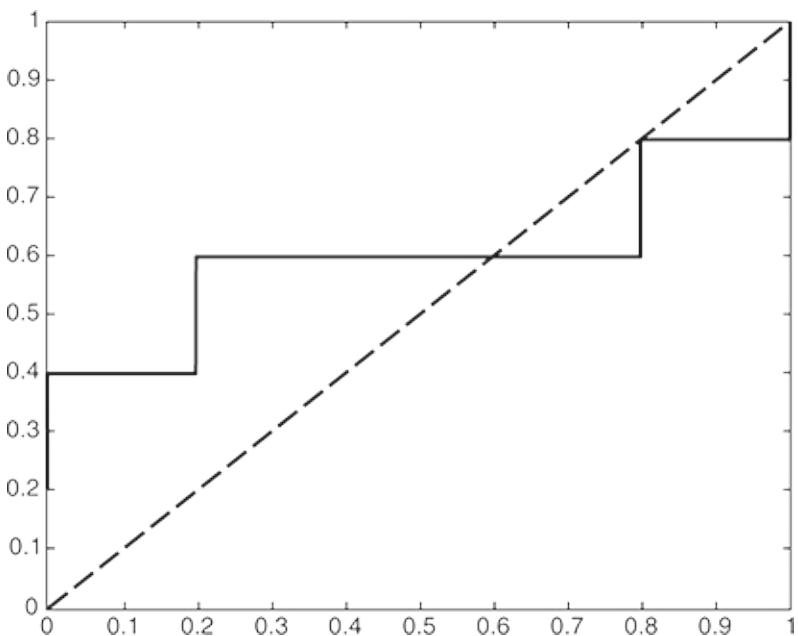


Figure 4.53.

ROC curve for the data shown in [Figure 4.52](#).

Note that in an ROC curve, the TPR monotonically increases with FPR, because the inclusion of a test instance in the set of predicted positives can either increase the TPR or the FPR. The ROC curve thus has a staircase pattern. Furthermore, there are several critical points along an ROC curve that have well-known interpretations:

(TPR=0, FPR=0): Model predicts every instance to be a negative class.

(TPR=1, FPR=1): Model predicts every instance to be a positive class.

(TPR=1, FPR=0): The *perfect* model with zero misclassifications.

A good classification model should be located as close as possible to the upper left corner of the diagram, while a model that makes random guesses should reside along the main diagonal, connecting the points (TPR=0, FPR=0) and (TPR=1, FPR=1). Random guessing means that an instance is classified as a positive class with a fixed probability p , irrespective of its attribute set.

For example, consider a data set that contains $n+$ positive instances and $n-$ negative instances. The random classifier is expected to correctly classify p_{n+} of the positive instances and to misclassify p_{n-} of the negative instances. Therefore, the TPR of the classifier is $(p_{n+})/n+=p$, while its FPR is $(p_{n-})/p=p$. Hence, this random classifier will reside at the point (p, p) in the ROC curve along the main diagonal.

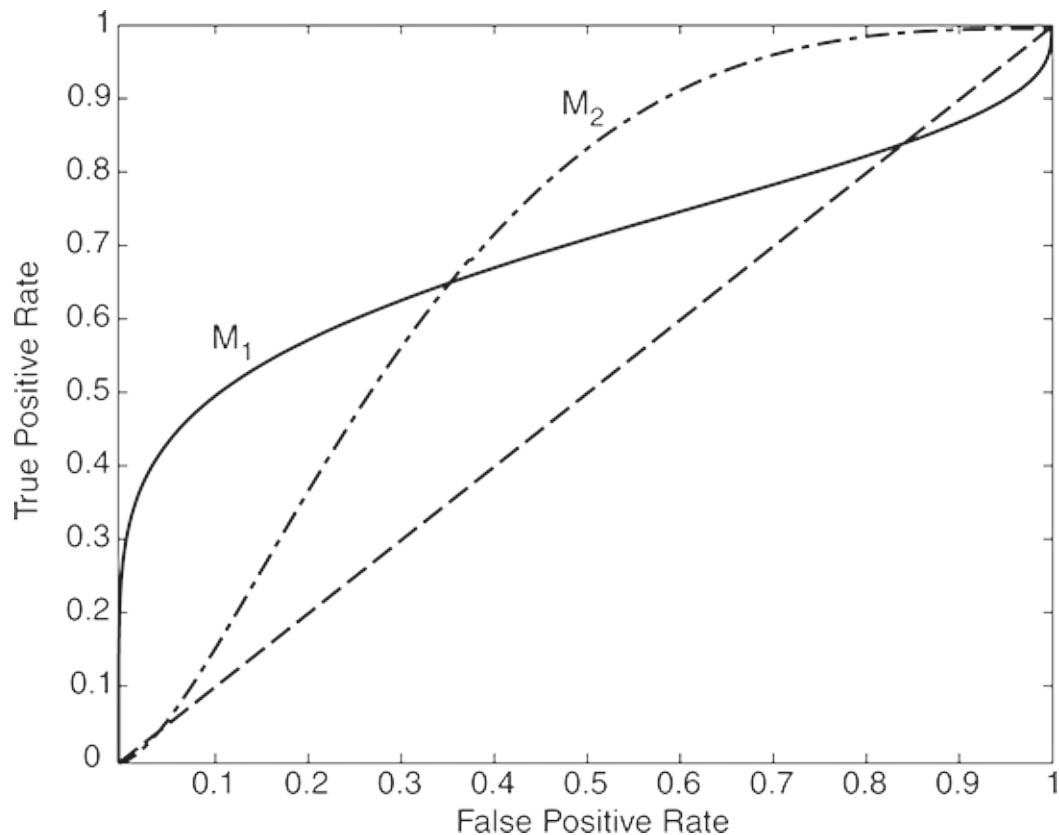


Figure 4.54.
ROC curves for two different classifiers.

Since every point on the ROC curve represents the performance of a classifier generated using a particular score threshold, they can be viewed as different **operating points** of the classifier. One may choose one of these operating points depending on the requirements of the application. Hence, an ROC curve facilitates the comparison of classifiers over a range of operating points. For example, [Figure 4.54](#) compares the ROC curves of two classifiers, M₁

and M2, generated by varying the score thresholds. We can see that M1 is better than M2 when FPR is less than 0.36, as M1 shows better TPR than M2 for this range of operating points. On the other hand, M2 is superior when FPR is greater than 0.36, since the TPR of M2 is higher than that of M1 for this range. Clearly, neither of the two classifiers **dominates** (is strictly better than) the other, i.e., shows higher values of TPR and lower values of FPR over all operating points.

To summarize the aggregate behavior across all operating points, one of the commonly used measures is the **area under the ROC curve** (AUC). If the classifier is perfect, then its area under the ROC curve will be equal 1. If the algorithm simply performs random guessing, then its area under the ROC curve will be equal to 0.5.

Although the AUC provides a useful summary of aggregate performance, there are certain caveats in using the AUC for comparing classifiers. First, even if the AUC of algorithm *A* is higher than the AUC of another algorithm *B*, this does not mean that algorithm *A* is always better than *B*, i.e., the ROC curve of *A* dominates that of *B* across all operating points. For example, even though M1 shows a slightly lower AUC than M2 in [Figure 4.54](#), we can see that both M1 and M2 are useful over different ranges of operating points and none of them are strictly better than the other across all possible operating points. Hence, we cannot use the AUC to determine which algorithm is better, unless we know that the ROC curve of one of the algorithms dominates the other.

Second, although the AUC summarizes the aggregate performance over all operating points, we are often interested in only a small range of operating points in most applications. For example, even though M1 shows slightly lower AUC than M2, it shows higher TPR values than M2 for small FPR values (smaller than 0.36). In the presence of class imbalance, the behavior of

an algorithm over small FPR values (also termed as **early retrieval**) is often more meaningful for comparison than the performance over all FPR values. This is because, in many applications, it is important to assess the TPR achieved by a classifier in the first few instances with highest scores, without incurring a large FPR. Hence, in [Figure 4.54](#), due to the high TPR values of M1 during early retrieval ($\text{FPR} < 0.36$), we may prefer M1 over M2 for imbalanced test sets, despite the lower AUC of M1. Hence, care must be taken while comparing the AUC values of different classifiers, usually by visualizing their ROC curves rather than just reporting their AUC.

A key characteristic of ROC curves is that they are agnostic to the skew in the test set, because both the evaluation measures used in constructing ROC curves (TPR and FPR) are invariant to class imbalance. Hence, ROC curves are not suitable for measuring the impact of skew on classification performance. In particular, we will obtain the same ROC curve for two test data sets that have very different skew.

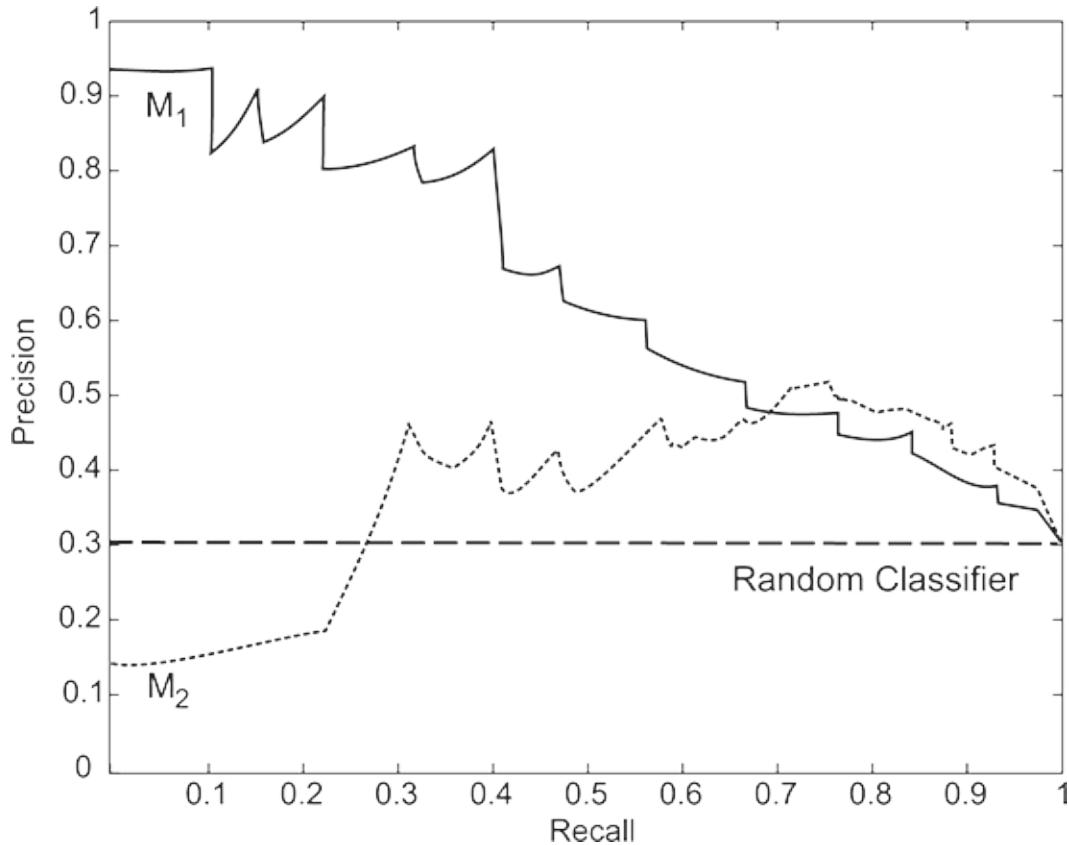


Figure 4.55.

PR curves for two different classifiers.

Precision-Recall Curve

An alternate tool for aggregate evaluation is the **precision recall curve** (PR curve). The PR curve plots the precision and recall values of a classifier on the y and x axes respectively, by varying the threshold on the test scores.

[Figure 4.55](#) shows an example of PR curves for two hypothetical classifiers, M1 and M2. The approach for generating a PR curve is similar to the approach described above for generating an ROC curve. However, there are some key distinguishing features in the PR curve:

1. PR curves are sensitive to the skew factor $\alpha = P/(P+N)$, and different PR curves are generated for different values of α .

2. When the score threshold is lowest (every instance is labeled as positive), the precision is equal to α while recall is 1. As we increase the score threshold, the number of predicted positives can stay the same or decrease. Hence, the recall monotonically declines as the score threshold increases. In general, the precision may increase or decrease for the same value of recall, upon addition of an instance into the set of predicted positives. For example, if the k^{th} ranked instance belongs to the negative class, then including it will result in a drop in the precision without affecting the recall. The precision may improve at the next step, which adds the $(k+1)^{\text{th}}$ ranked instance, if this instance belongs to the positive class. Hence, the PR curve is not a smooth, monotonically increasing curve like the ROC curve, and generally has a zigzag pattern. This pattern is more prominent in the left part of the curve, where even a small change in the number of false positives can cause a large change in precision.
3. As, as we increase the imbalance among the classes (reduce the value of α), the rightmost points of all PR curves will move downwards. At and near the leftmost point on the PR curve (corresponding to larger values of score threshold), the recall is close to zero, while the precision is equal to the fraction of positives in the top ranked instances of the algorithm. Hence, different classifiers can have different values of precision at the leftmost points of the PR curve. Also, if the classification score of an algorithm monotonically varies with the posterior probability of the positive class, we can expect the PR curve to gradually decrease from a high value of precision on the leftmost point to a constant value of α at the rightmost point, albeit with some ups and downs. This can be observed in the PR curve of algorithm M1 in [Figure 4.55](#), which starts from a higher value of precision on the left that gradually decreases as we move towards the right. On the other hand, the PR curve of algorithm M2 starts from a lower value of precision on the left and shows more drastic ups and downs as we

move right, suggesting that the classification score of M2 shows a weaker monotonic relationship with the posterior probability of the positive class.

4. A random classifier that assigns an instance to be positive with a fixed probability p has a precision of α and a recall of p . Hence, a classifier that performs random guessing has a horizontal PR curve with $y=\alpha$, as shown using a dashed line in [Figure 4.55](#). Note that the random baseline in PR curves depends on the skew in the test set, in contrast to the fixed main diagonal of ROC curves that represents random classifiers.
5. Note that the precision of an algorithm is impacted more strongly by false positives in the top ranked test instances than the FPR of the algorithm. For this reason, the PR curve generally helps to magnify the differences between classifiers in the left portion of the PR curve. Hence, in the presence of class imbalance in the test data, analyzing the PR curves generally provides a deeper insight into the performance of classifiers than the ROC curves, especially in the early retrieval range of operating points.
6. The classifier corresponding to (precision=1, recall=1) represents the perfect classifier. Similar to AUC, we can also compute the area under the PR curve of an algorithm, known as AUC-PR. The AUC-PR of a random classifier is equal to α , while that of a perfect algorithm is equal to 1. Note that AUC-PR varies with changing skew in the test set, in contrast to the area under the ROC curve, which is insensitive to the skew. The AUC-PR helps in accentuating the differences between classification algorithms in the early retrieval range of operating points. Hence, it is more suited for evaluating classification performance in the presence of class imbalance than the area under the ROC curve. However, similar to ROC curves, a higher value of AUC-PR does not guarantee the superiority of a classification algorithm over another. This is because the PR curves of two algorithms can easily cross each

other, such that they both show better performances in different ranges of operating points. Hence, it is important to visualize the PR curves before comparing their AUC-PR values, in order to ensure a meaningful evaluation.

4.12 Multiclass Problem

Some of the classification techniques described in this chapter are originally designed for binary classification problems. Yet there are many real-world problems, such as character recognition, face identification, and text classification, where the input data is divided into more than two categories. This section presents several approaches for extending the binary classifiers to handle multiclass problems. To illustrate these approaches, let $Y = \{y_1, y_2, \dots, y_K\}$ be the set of classes of the input data.

The first approach decomposes the multiclass problem into K binary problems. For each class $y_i \in Y$, a binary problem is created where all instances that belong to y_i are considered positive examples, while the remaining instances are considered negative examples. A binary classifier is then constructed to separate instances of class y_i from the rest of the classes. This is known as the one-against-rest (1-r) approach.

The second approach, which is known as the one-against-one (1-1) approach, constructs $K(K - 1)/2$ binary classifiers, where each classifier is used to distinguish between a pair of classes, (y_i, y_j) . Instances that do not belong to either y_i or y_j are ignored when constructing the binary classifier for (y_i, y_j) . In both 1-r and 1-1 approaches, a test instance is classified by combining the predictions made by the binary classifiers. A voting scheme is typically employed to combine the predictions, where the class that receives the highest number of votes is assigned to the test instance. In the 1-r approach, if an instance is classified as negative, then all classes except for the positive class receive a vote. This approach, however, may lead to ties among the different classes. Another possibility is to transform the outputs of the binary

classifiers into probability estimates and then assign the test instance to the class that has the highest probability.

Example 4.11.

Consider a multiclass problem where $Y=\{y_1, y_2, y_3, y_4\}$. Suppose a test instance is classified as $(+, -, -, -)$ according to the 1-r approach. In other words, it is classified as positive when y_1 is used as the positive class and negative when y_2, y_3 , and y_4 are used as the positive class. Using a simple majority vote, notice that y_1 receives the highest number of votes, which is four, while the remaining classes receive only three votes. The test instance is therefore classified as y_1 .

Example 4.12.

Suppose the test instance is classified using the 1-1 approach as follows:

Binary pair of classes	$+ : y_1 - : y_2$	$+ : y_1 - : y_3$	$+ : y_1 - : y_4$	$+ : y_2 - : y_3$	$+ : y_2 - : y_4$	$+ : y_3 - : y_4$
Classification	+	+	-	+	-	+

The first two rows in this table correspond to the pair of classes (y_i, y_j) chosen to build the classifier and the last row represents the predicted class for the test instance. After combining the predictions, y_1 and y_4 each receive two votes, while y_2 and y_3 each receives only one vote. The test instance is therefore classified as either y_1 or y_4 , depending on the tie-breaking procedure.

Error-Correcting Output Coding

A potential problem with the previous two approaches is that they may be sensitive to binary classification errors. For the 1-r approach given in Example 4.12, if at least one of the binary classifiers makes a mistake in its prediction, then the classifier may end up declaring a tie between classes or making a wrong prediction. For example, suppose the test instance is classified as $(+, -, +, -)$ due to misclassification by the third classifier. In this case, it will be difficult to tell whether the instance should be classified as y_1 or y_3 , unless the probability associated with each class prediction is taken into account.

The error-correcting output coding (ECOC) method provides a more robust way for handling multiclass problems. The method is inspired by an information-theoretic approach for sending messages across noisy channels.

The idea behind this approach is to add redundancy into the transmitted message by means of a codeword, so that the receiver may detect errors in the received message and perhaps recover the original message if the number of errors is small.

For multiclass learning, each class y_i is represented by a unique bit string of length n known as its codeword. We then train n binary classifiers to predict each bit of the codeword string. The predicted class of a test instance is given by the codeword whose Hamming distance is closest to the codeword produced by the binary classifiers. Recall that the Hamming distance between a pair of bit strings is given by the number of bits that differ.

Example 4.13.

Consider a multiclass problem where $Y=\{y_1, y_2, y_3, y_4\}$. Suppose we encode the classes using the following seven bit codewords:

Class	Codeword						
y1	1	1	1	1	1	1	1
y2	0	0	0	0	1	1	1
y3	0	0	1	1	0	0	1
y4	0	1	0	1	0	1	0

Each bit of the codeword is used to train a binary classifier. If a test instance is classified as (0,1,1,1,1,1,1) by the binary classifiers, then the Hamming distance between the codeword and y1 is 1, while the Hamming distance to the remaining classes is 3. The test instance is therefore classified as y1.

An interesting property of an error-correcting code is that if the minimum Hamming distance between any pair of codewords is d , then any $\lfloor (d-1)/2 \rfloor$ errors in the output code can be corrected using its nearest codeword. In [Example 4.13](#), because the minimum Hamming distance between any pair of codewords is 4, the classifier may tolerate errors made by one of the seven binary classifiers. If there is more than one classifier that makes a mistake, then the classifier may not be able to compensate for the error.

An important issue is how to design the appropriate set of codewords for different classes. From coding theory, a vast number of algorithms have been developed for generating n -bit codewords with bounded Hamming distance. However, the discussion of these algorithms is beyond the scope of this book. It is worthwhile mentioning that there is a significant difference between the design of error-correcting codes for communication tasks compared to those used for multiclass learning. For communication, the codewords should maximize the Hamming distance between the rows so that error correction

can be performed. Multiclass learning, however, requires that both the row-wise and column-wise distances of the codewords must be well separated. A larger column-wise distance ensures that the binary classifiers are mutually independent, which is an important requirement for ensemble learning methods.

4.13 Bibliographic Notes

Mitchell [278] provides excellent coverage on many classification techniques from a machine learning perspective. Extensive coverage on classification can also be found in Aggarwal [195], Duda et al. [229], Webb [307], Fukunaga [237], Bishop [204], Hastie et al. [249], Cherkassky and Mulier [215], Witten and Frank [310], Hand et al. [247], Han and Kamber [244], and Dunham [230].

Direct methods for rule-based classifiers typically employ the sequential covering scheme for inducing classification rules. Holte's 1R [255] is the simplest form of a rule-based classifier because its rule set contains only a single rule. Despite its simplicity, Holte found that for some data sets that exhibit a strong one-to-one relationship between the attributes and the class label, 1R performs just as well as other classifiers. Other examples of rule-based classifiers include IREP [234], RIPPER [218], CN2 [216, 217], AQ [276], RISE [224], and ITRULE [296]. **Table 4.8** shows a comparison of the characteristics of four of these classifiers.

Table 4.8. Comparison of various rule-based classifiers.

	RIPPER	CN2 (unordered)	CN2 (ordered)	AQR
Rule-growing strategy	General-to-specific	General-to-specific	General-to-specific	General-to-specific (seeded by a positive example)
Evaluation metric	FOIL's Info gain	Laplace	Entropy and likelihood ratio	Number of true positives

Stopping condition for rule-growing	All examples belong to the same class	No performance gain	No performance gain	Rules cover only positive class
Rule pruning	Reduced error pruning	None	None	None
Instance elimination	Positive and negative	Positive only	Positive only	Positive and negative
Stopping condition for adding rules	Error>50% or based on MDL	No performance gain	No performance gain	All positive examples are covered
Rule setp runing	Replace or modify rules	Statistical tests	None	None
Search strategy	Greedy	Beam search	Beam search	Beam search

For rule-based classifiers, the rule antecedent can be generalized to include any propositional or first-order logical expression (e.g., Horn clauses).

Readers who are interested in first-order logic rule-based classifiers may refer to references such as [278] or the vast literature on inductive logic programming [279]. Quinlan [287] proposed the C4.5rules algorithm for extracting classification rules from decision trees. An indirect method for extracting rules from artificial neural networks was given by Andrews et al. in [198].

Cover and Hart [220] presented an overview of the nearest neighbor classification method from a Bayesian perspective. Aha provided both theoretical and empirical evaluations for instance-based methods in [196]. PEBLS, which was developed by Cost and Salzberg [219], is a nearest neighbor classifier that can handle data sets containing nominal attributes.

Each training example in PEBLS is also assigned a weight factor that depends on the number of times the example helps make a correct prediction. Han et al. [243] developed a weight-adjusted nearest neighbor algorithm, in which the feature weights are learned using a greedy, hill-climbing optimization algorithm. A more recent survey of k -nearest neighbor classification is given by Steinbach and Tan [298].

Naïve Bayes classifiers have been investigated by many authors, including Langley et al. [267], Ramoni and Sebastiani [288], Lewis [270], and Domingos and Pazzani [227]. Although the independence assumption used in naïve Bayes classifiers may seem rather unrealistic, the method has worked surprisingly well for applications such as text classification. Bayesian networks provide a more flexible approach by allowing some of the attributes to be interdependent. An excellent tutorial on Bayesian networks is given by Heckerman in [252] and Jensen in [258]. Bayesian networks belong to a broader class of models known as probabilistic graphical models. A formal introduction to the relationships between graphs and probabilities was presented in Pearl [283]. Other great resources on probabilistic graphical models include books by Bishop [205], and Jordan [259]. Detailed discussions of concepts such as d -separation and Markov blankets are provided in Geiger et al. [238] and Russell and Norvig [291].

Generalized linear models (GLM) are a rich class of regression models that have been extensively studied in the statistical literature. They were formulated by Nelder and Wedderburn in 1972 [280] to unify a number of regression models such as linear regression, logistic regression, and Poisson regression, which share some similarities in their formulations. An extensive discussion of GLMs is provided in the book by McCullagh and Nelder [274].

Artificial neural networks (ANN) have witnessed a long and winding history of developments, involving multiple phases of stagnation and resurgence. The

idea of a mathematical model of a neural network was first introduced in 1943 by McCulloch and Pitts [275]. This led to a series of computational machines to simulate a neural network based on the theory of neural plasticity [289]. The perceptron, which is the simplest prototype of modern ANNs, was developed by Rosenblatt in 1958 [290]. The perceptron uses a single layer of processing units that can perform basic mathematical operations such as addition and multiplication. However, the perceptron can only learn linear decision boundaries and is guaranteed to converge only when the classes are linearly separable. Despite the interest in learning multi-layer networks to overcome the limitations of perceptron, progress in this area remain halted until the invention of the backpropagation algorithm by Werbos in 1974 [309], which allowed for the quick training of multi-layer ANNs using the gradient descent method. This led to an upsurge of interest in the artificial intelligence (AI) community to develop multi-layer ANN models, a trend that continued for more than a decade. Historically, ANNs mark a paradigm shift in AI from approaches based on expert systems (where knowledge is encoded using if-then rules) to machine learning approaches (where the knowledge is encoded in the parameters of a computational model). However, there were still a number of algorithmic and computational challenges in learning large ANN models, which remained unresolved for a long time. This hindered the development of ANN models to the scale necessary for solving real-world problems. Slowly, ANNs started getting outpaced by other classification models such as support vector machines, which provided better performance as well as theoretical guarantees of convergence and optimality. It is only recently that the challenges in learning *deep* neural networks have been circumvented, owing to better computational resources and a number of algorithmic improvements in ANNs since 2006. This re-emergence of ANN has been dubbed as “deep learning,” which has often outperformed existing classification models and gained wide-spread popularity.

Deep learning is a rapidly evolving area of research with a number of potentially impactful contributions being made every year. Some of the landmark advancements in deep learning include the use of large-scale restricted Boltzmann machines for learning generative models of data [201, 253], the use of autoencoders and its variants (denoising autoencoders) for learning robust feature representations [199, 305, 306], and sophisticated architectures to promote sharing of parameters across nodes such as convolutional neural networks for images [265, 268] and recurrent neural networks for sequences [241, 242, 277]. Other major improvements include the approach of unsupervised pretraining for initializing ANN models [232], the dropout technique for regularization [254, 297], batch normalization for fast learning of ANN parameters [256], and maxout networks for effective usage of the dropout technique [240]. Even though the discussions in this chapter on learning ANN models were centered around the gradient descent method, most of the modern ANN models involving a large number of hidden layers are trained using the stochastic gradient descent method since it is highly scalable [207]. An extensive survey of deep learning approaches has been presented in review articles by Bengio [200], LeCun et al. [269], and Schmidhuber [293]. An excellent summary of deep learning approaches can also be obtained from recent books by Goodfellow et al. [239] and Nielsen [281].

Vapnik [303, 304] has written two authoritative books on Support Vector Machines (SVM). Other useful resources on SVM and kernel methods include the books by Cristianini and Shawe-Taylor [221] and Schölkopf and Smola [294]. There are several survey articles on SVM, including those written by Burges [212], Bennet et al. [202], Hearst [251], and Mangasarian [272]. SVM can also be viewed as an L2 norm regularizer of the hinge loss function, as described in detail by Hastie et al. [249]. The L1 norm regularizer of the square loss function can be obtained using the least absolute shrinkage and selection operator (Lasso), which was introduced by Tibshirani in 1996 [301].

The Lasso has several interesting properties such as the ability to simultaneously perform feature selection as well as regularization, so that only a subset of features are selected in the final model. An excellent review of Lasso can be obtained from a book by Hastie et al. [250].

A survey of ensemble methods in machine learning was given by Diet-terich [222]. The bagging method was proposed by Breiman [209]. Freund and Schapire [236] developed the AdaBoost algorithm. Arcing, which stands for adaptive resampling and combining, is a variant of the boosting algorithm proposed by Breiman [210]. It uses the non-uniform weights assigned to training examples to resample the data for building an ensemble of training sets. Unlike AdaBoost, the votes of the base classifiers are not weighted when determining the class label of test examples. The random forest method was introduced by Breiman in [211]. The concept of bias-variance decomposition is explained in detail by Hastie et al. [249]. While the bias-variance decomposition was initially proposed for regression problems with squared loss function, a unified framework for classification problems involving 0–1 losses was introduced by Domingos [226].

Related work on mining rare and imbalanced data sets can be found in the survey papers written by Chawla et al. [214] and Weiss [308]. Sampling-based methods for mining imbalanced data sets have been investigated by many authors, such as Kubat and Matwin [266], Japkowitz [257], and Drummond and Holte [228]. Joshi et al. [261] discussed the limitations of boosting algorithms for rare class modeling. Other algorithms developed for mining rare classes include SMOTE [213], PNrule [260], and CREDOS [262].

Various alternative metrics that are well-suited for class imbalanced problems are available. The precision, recall, and F1-measure are widely-used metrics in information retrieval [302]. ROC analysis was originally used in signal detection theory for performing aggregate evaluation over a range of score

thresholds. A method for comparing classifier performance using the convex hull of ROC curves was suggested by Provost and Fawcett in [286]. Bradley [208] investigated the use of area under the ROC curve (AUC) as a performance metric for machine learning algorithms. Despite the vast body of literature on optimizing the AUC measure in machine learning models, it is well-known that AUC suffers from certain limitations. For example, the AUC can be used to compare the quality of two classifiers only if the ROC curve of one classifier strictly dominates the other. However, if the ROC curves of two classifiers intersect at any point, then it is difficult to assess the relative quality of classifiers using the AUC measure. An in-depth discussion of the pitfalls in using AUC as a performance measure can be obtained in works by Hand [245, 246], and Powers [284]. The AUC has also been considered to be an *incoherent* measure of performance, i.e., it uses different scales while comparing the performance of different classifiers, although a coherent interpretation of AUC has been provided by Ferri et al. [235]. Berrar and Flach [203] describe some of the common caveats in using the ROC curve for clinical microarray research. An alternate approach for measuring the aggregate performance of a classifier is the precision-recall (PR) curve, which is especially useful in the presence of class imbalance [292].

An excellent tutorial on cost-sensitive learning can be found in a review article by Ling and Sheng [271]. The properties of a cost matrix had been studied by Elkan in [231]. Margineantu and Dietterich [273] examined various methods for incorporating cost information into the C4.5 learning algorithm, including wrapper methods, class distribution-based methods, and loss-based methods. Other cost-sensitive learning methods that are algorithm-independent include AdaCost [233], MetaCost [225], and costing [312].

Extensive literature is also available on the subject of multiclass learning. This includes the works of Hastie and Tibshirani [248], Allwein et al. [197], Kong and Dietterich [264], and Tax and Duin [300]. The error-correcting output

coding (ECOC) method was proposed by Dietterich and Bakiri [223]. They had also investigated techniques for designing codes that are suitable for solving multiclass problems.

Apart from exploring algorithms for traditional classification settings where every instance has a single set of features with a unique categorical label, there has been a lot of recent interest in non-traditional classification paradigms, involving complex forms of inputs and outputs. For example, the paradigm of multi-label learning allows for an instance to be assigned multiple class labels rather than just one. This is useful in applications such as object recognition in images, where a photo image may include more than one classification object, such as, grass, sky, trees, and mountains. A survey on multi-label learning can be found in [313]. As another example, the paradigm of multi-instance learning considers the problem where the instances are available in the form of groups called *bags*, and training labels are available at the level of bags rather than individual instances. Multi-instance learning is useful in applications where an object can exist as multiple instances in different states (e.g., the different isomers of a chemical compound), and even if a single instance shows a specific characteristic, the entire bag of instances associated with the object needs to be assigned the relevant class. A survey on multi-instance learning is provided in [314].

In a number of real-world applications, it is often the case that the training labels are scarce in quantity, because of the high costs associated with obtaining gold-standard supervision. However, we almost always have abundant access to unlabeled test instances, which do not have supervised labels but contain valuable information about the structure or distribution of instances. Traditional learning algorithms, which only make use of the labeled instances in the training set for learning the decision boundary, are unable to exploit the information contained in unlabeled instances. In contrast, learning algorithms that make use of the structure in the unlabeled data for learning the

classification model are known as semi-supervised learning algorithms [315, 316]. The use of unlabeled data is also explored in the paradigm of multi-view learning [299, 311], where every object is observed in multiple views of the data, involving diverse sets of features. A common strategy used by multi-view learning algorithms is co-training [206], where a different model is learned for every view of the data, but the model predictions from every view are constrained to be identical to each other on the unlabeled test instances.

Another learning paradigm that is commonly explored in the paucity of training data is the framework of active learning, which attempts to seek the smallest set of label annotations to learn a reasonable classification model. Active learning expects the annotator to be involved in the process of model learning, so that the labels are requested incrementally over the most relevant set of instances, given a limited budget of label annotations. For example, it may be useful to obtain labels over instances closer to the decision boundary that can play a bigger role in fine-tuning the boundary. A review on active learning approaches can be found in [285, 295].

In some applications, it is important to simultaneously solve multiple learning tasks together, where some of the tasks may be similar to one another. For example, if we are interested in translating a passage written in English into different languages, the tasks involving lexically similar languages (such as Spanish and Portuguese) would require similar learning of models. The paradigm of multi-task learning helps in simultaneously learning across all tasks while sharing the learning among related tasks. This is especially useful when some of the tasks do not contain sufficiently many training samples, in which case borrowing the learning from other related tasks helps in the learning of robust models. A special case of multi-task learning is transfer learning, where the learning from a source task (with sufficient number of training samples) has to be *transferred* to a destination task (with paucity of

training data). An extensive survey of transfer learning approaches is provided by Pan et al. [282].

Most classifiers assume every data instance must belong to a class, which is not always true for some applications. For example, in malware detection, due to the ease in which new malwares are created, a classifier trained on existing classes may fail to detect new ones even if the features for the new malwares are considerably different than those for existing malwares. Another example is in critical applications such as medical diagnosis, where prediction errors are costly and can have severe consequences. In this situation, it would be better for the classifier to refrain from making any prediction on a data instance if it is unsure of its class. This approach, known as classifier with reject option, does not need to classify every data instance unless it determines the prediction is reliable (e.g., if the class probability exceeds a user-specified threshold). Instances that are unclassified can be presented to domain experts for further determination of their true class labels.

Classifiers can also be distinguished in terms of how the classification model is trained. A batch classifier assumes all the labeled instances are available during training. This strategy is applicable when the training set size is not too large and for stationary data, where the relationship between the attributes and classes does not vary over time. An online classifier, on the other hand, trains an initial model using a subset of the labeled data [263]. The model is then updated incrementally as more labeled instances become available. This strategy is effective when the training set is too large or when there is *concept drift* due to changes in the distribution of the data over time.

Bibliography

- [195] C. C. Aggarwal. *Data classification: algorithms and applications*. CRC Press, 2014.
- [196] D. W. Aha. *A study of instance-based algorithms for supervised learning tasks: mathematical, empirical, and psychological evaluations*. PhD thesis, University of California, Irvine, 1990.
- [197] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing Multiclass to Binary: A Unifying Approach to Margin Classifiers. *Journal of Machine Learning Research*, 1: 113–141, 2000.
- [198] R. Andrews, J. Diederich, and A. Tickle. A Survey and Critique of Techniques For Extracting Rules From Trained Artificial Neural Networks. *Knowledge Based Systems*, 8(6):373–389, 1995.
- [199] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. *ICML unsupervised and transfer learning*, 27(37-50):1, 2012.
- [200] Y. Bengio. Learning deep architectures for AI. *Foundations and trends R in Machine Learning*, 2(1):1–127, 2009.
- [201] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and*

machine intelligence, 35(8): 1798–1828, 2013.

[202] K. Bennett and C. Campbell. Support Vector Machines: Hype or Hallelujah. *SIGKDD Explorations*, 2(2):1–13, 2000.

[203] D. Berrar and P. Flach. Caveats and pitfalls of ROC analysis in clinical microarray research (and how to avoid them). *Briefings in bioinformatics*, page bbr008, 2011.

[204] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K., 1995.

[205] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[206] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[207] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[208] A. P. Bradley. The use of the area under the ROC curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, 30(7):1145–1149, 1997.

- [209] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
- [210] L. Breiman. Bias, Variance, and Arcing Classifiers. Technical Report 486, University of California, Berkeley, CA, 1996.
- [211] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.
- [212] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [213] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16: 321–357, 2002.
- [214] N. V. Chawla, N. Japkowicz, and A. Kolcz. Editorial: Special Issue on Learning from Imbalanced Data Sets. *SIGKDD Explorations*, 6(1):1–6, 2004.
- [215] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley Interscience, 1998.
- [216] P. Clark and R. Boswell. Rule Induction with CN2: Some Recent Improvements. In *Machine Learning: Proc. of the 5th European Conf. (EWSL-91)*, pages 151–163, 1991.

- [217] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3(4): 261–283, 1989.
- [218] W. W. Cohen. Fast Effective Rule Induction. In *Proc. of the 12th Intl. Conf. on Machine Learning*, pages 115–123, Tahoe City, CA, July 1995.
- [219] S. Cost and S. Salzberg. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning*, 10:57–78, 1993.
- [220] T. M. Cover and P. E. Hart. Nearest Neighbor Pattern Classification. *Knowledge Based Systems*, 8(6):373–389, 1995.
- [221] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
- [222] T. G. Dietterich. Ensemble Methods in Machine Learning. In *First Intl. Workshop on Multiple Classifier Systems*, Cagliari, Italy, 2000.
- [223] T. G. Dietterich and G. Bakiri. Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
- [224] P. Domingos. The RISE system: Conquering without separating. In *Proc. of the 6th IEEE Intl. Conf. on Tools with Artificial Intelligence*, pages 704–707, New Orleans, LA, 1994.

- [225] P. Domingos. MetaCost: A General Method for Making Classifiers Cost-Sensitive. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 155–164, San Diego, CA, August 1999.
- [226] P. Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning*, pages 231–238, 2000.
- [227] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(2-3):103–130, 1997.
- [228] C. Drummond and R. C. Holte. C4.5, Class imbalance, and Cost sensitivity: Why under-sampling beats over-sampling. In *ICML'2004 Workshop on Learning from Imbalanced Data Sets II*, Washington, DC, August 2003.
- [229] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.
- [230] M. H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2006.
- [231] C. Elkan. The Foundations of Cost-Sensitive Learning. In *Proc. of the 17th Intl. Joint Conf. on Artificial Intelligence*, pages 973–978, Seattle, WA, August 2001.

- [232] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.
- [233] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proc. of the 16th Intl. Conf. on Machine Learning*, pages 97–105, Bled, Slovenia, June 1999.
- [234] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In *Proc. of the 11th Intl. Conf. on Machine Learning*, pages 70–77, New Brunswick, NJ, July 1994.
- [235] C. Ferri, J. Hernández-Orallo, and P. A. Flach. A coherent interpretation of AUC as a measure of aggregated classification performance. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 657–664, 2011.
- [236] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1): 119–139, 1997.
- [237] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, 1990.
- [238] D. Geiger, T. S. Verma, and J. Pearl. d-separation: From theorems to algorithms. *arXiv preprint arXiv:1304.1505*, 2013.

- [239] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. Book in preparation for MIT Press, 2016.
- [240] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. *ICML* (3), 28:1319–1327, 2013.
- [241] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2009.
- [242] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.
- [243] E.-H. Han, G. Karypis, and V. Kumar. Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification. In *Proc. of the 5th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, Lyon, France, 2001.
- [244] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.
- [245] D. J. Hand. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine learning*, 77(1):103–123, 2009.
- [246] D. J. Hand. Evaluating diagnostic tests: the area under the ROC curve and the balance of errors. *Statistics in medicine*, 29(14):1502–1510, 2010.

- [247] D. J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.
- [248] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *Annals of Statistics*, 26(2):451–471, 1998.
- [249] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.
- [250] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC Press, 2015.
- [251] M. Hearst. Trends & Controversies: Support Vector Machines. *IEEE Intelligent Systems*, 13(4):18–28, 1998.
- [252] D. Heckerman. Bayesian Networks for Data Mining. *Data Mining and Knowledge Discovery*, 1(1):79–119, 1997.
- [253] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [254] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

- [255] R. C. Holte. Very Simple Classification Rules Perform Well on Most Commonly Used Data sets. *Machine Learning*, 11:63–91, 1993.
- [256] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [257] N. Japkowicz. The Class Imbalance Problem: Significance and Strategies. In *Proc. of the 2000 Intl. Conf. on Artificial Intelligence: Special Track on Inductive Learning*, volume 1, pages 111–117, Las Vegas, NV, June 2000.
- [258] F. V. Jensen. *An introduction to Bayesian networks*, volume 210. UCL press London, 1996.
- [259] M. I. Jordan. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998.
- [260] M. V. Joshi, R. C. Agarwal, and V. Kumar. Mining Needles in a Haystack: Classifying Rare Classes via Two-Phase Rule Induction. In *Proc. of 2001 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 91–102, Santa Barbara, CA, June 2001.
- [261] M. V. Joshi, R. C. Agarwal, and V. Kumar. Predicting rare classes: can boosting make any weak learner strong? In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 297–306, Edmonton, Canada, July 2002.

- [262] M. V. Joshi and V. Kumar. CREDOS: Classification Using Ripple Down Structure (A Case for Rare Classes). In *Proc. of the SIAM Intl. Conf. on Data Mining*, pages 321–332, Orlando, FL, April 2004.
- [263] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.
- [264] E. B. Kong and T. G. Dietterich. Error-Correcting Output Coding Corrects Bias and Variance. In *Proc. of the 12th Intl. Conf. on Machine Learning*, pages 313–321, Tahoe City, CA, July 1995.
- [265] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [266] M. Kubat and S. Matwin. Addressing the Curse of Imbalanced Training Sets: One Sided Selection. In *Proc. of the 14th Intl. Conf. on Machine Learning*, pages 179–186, Nashville, TN, July 1997.
- [267] P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *Proc. of the 10th National Conf. on Artificial Intelligence*, pages 223–228, 1992.
- [268] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

- [269] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [270] D. D. Lewis. Naive Bayes at Forty: The Independence Assumption in Information Retrieval. In *Proc. of the 10th European Conf. on Machine Learning (ECML 1998)*, pages 4–15, 1998.
- [271] C. X. Ling and V. S. Sheng. Cost-sensitive learning. In *Encyclopedia of Machine Learning*, pages 231–235. Springer, 2011.
- [272] O. Mangasarian. Data Mining via Support Vector Machines. Technical Report Technical Report 01-05, Data Mining Institute, May 2001.
- [273] D. D. Margineantu and T. G. Dietterich. Learning Decision Trees for Loss Minimization in Multi-Class Problems. Technical Report 99-30-03, Oregon State University, 1999.
- [274] P. McCullagh and J. A. Nelder. *Generalized linear models*, volume 37. CRC press, 1989.
- [275] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [276] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three

Medical Domains. In *Proc. of 5th National Conf. on Artificial Intelligence*, Orlando, August 1986.

- [277] T. Mikolov, M. Karafiát, L. Burget, J. Cernock`y, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.
- [278] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [279] S. Muggleton. *Foundations of Inductive Logic Programming*. Prentice Hall, Englewood Cliffs, NJ, 1995.
- [280] J. A. Nelder and R. J. Baker. Generalized linear models. *Encyclopedia of statistical sciences*, 1972.
- [281] M. A. Nielsen. Neural networks and deep learning. *Published online*: <http://neuralnetworksanddeeplearning.com/> .(visited: 10. 15. 2016) , 2015.
- [282] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.
- [283] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.
- [284] D. M. Powers. The problem of area under the curve. In *2012 IEEE International Conference on Information Science and Technology*, pages

567–573. IEEE, 2012.

- [285] M. Prince. Does active learning work? A review of the research. *Journal of engineering education*, 93(3):223–231, 2004.
- [286] F. J. Provost and T. Fawcett. Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 43–48, Newport Beach, CA, August 1997.
- [287] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann Publishers, San Mateo, CA, 1993.
- [288] M. Ramoni and P. Sebastiani. Robust Bayes classifiers. *Artificial Intelligence*, 125: 209–226, 2001.
- [289] N. Rochester, J. Holland, L. Haibt, and W. Duda. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on information Theory*, 2(3):80–93, 1956.
- [290] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [291] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

- [292] T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS one*, 10(3): e0118432, 2015.
- [293] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
- [294] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.
- [295] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52 (55-66):11, 2010.
- [296] P. Smyth and R. M. Goodman. An Information Theoretic Approach to Rule Induction from Databases. *IEEE Trans. on Knowledge and Data Engineering*, 4(4):301–316, 1992.
- [297] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [298] M. Steinbach and P.-N. Tan. kNN: k-Nearest Neighbors. In X. Wu and V. Kumar, editors, *The Top Ten Algorithms in Data Mining*. Chapman and Hall/CRC Reference, 1st edition, 2009.

- [299] S. Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7-8):2031–2038, 2013.
- [300] D. M. J. Tax and R. P. W. Duin. Using Two-Class Classifiers for Multiclass Classification. In *Proc. of the 16th Intl. Conf. on Pattern Recognition (ICPR 2002)*, pages 124–127, Quebec, Canada, August 2002.
- [301] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
- [302] C. J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, 1978.
- [303] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.
- [304] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.
- [305] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.
- [306] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a

- deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.
- [307] A. R. Webb. *Statistical Pattern Recognition*. John Wiley & Sons, 2nd edition, 2002.
- [308] G. M. Weiss. Mining with Rarity: A Unifying Framework. *SIGKDD Explorations*, 6 (1):7–19, 2004.
- [309] P. Werbos. Beyond regression: new tools for prediction and analysis in the behavioral sciences. *PhD thesis, Harvard University*, 1974.
- [310] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.
- [311] C. Xu, D. Tao, and C. Xu. A survey on multi-view learning. *arXiv preprint arXiv:1304.5634*, 2013.
- [312] B. Zadrozny, J. C. Langford, and N. Abe. Cost-Sensitive Learning by Cost-Proportionate Example Weighting. In *Proc. of the 2003 IEEE Intl. Conf. on Data Mining*, pages 435–442, Melbourne, FL, August 2003.
- [313] M.-L. Zhang and Z.-H. Zhou. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837, 2014.

- [314] Z.-H. Zhou. Multi-instance learning: A survey. *Department of Computer Science & Technology, Nanjing University, Tech. Rep*, 2004.
- [315] X. Zhu. Semi-supervised learning. In *Encyclopedia of machine learning*, pages 892–897. Springer, 2011.
- [316] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

4.14 Exercises

1. Consider a binary classification problem with the following set of attributes and attribute values:

- Air Conditioner={Working, Broken}
- Engine={Good, Bad}
- Mileage={High, Medium, Low}
- Rust={Yes, No}

Suppose a rule-based classifier produces the following rule set:

Mileage=High → Mileage=High
Mileage=Low → Value=High
Air Conditioner=Work → Rust=Yes

- a. Are the rules mutually exclusive?
- b. Is the rule set exhaustive?
- c. Is ordering needed for this set of rules?
- d. Do you need a default class for the rule set?

2. The RIPPER algorithm (by Cohen [218]) is an extension of an earlier algorithm called IREP (by Fürnkranz and Widmer [234]). Both algorithms apply the **reduced-error pruning** method to determine whether a rule needs to be pruned. The reduced error pruning method uses a validation set to estimate the generalization error of a classifier. Consider the following pair of rules:

R1:A → CR
R2:A ∧ B → C

R_2 is obtained by adding a new conjunct, B , to the left-hand side of R_1 . For this question, you will be asked to determine whether R_2 is preferred over R_1 from the perspectives of rule-growing and rule-pruning. To determine whether a rule should be pruned, IREP computes the following measure:

$$vIREP = p + (N - n)P + N,$$

where P is the total number of positive examples in the validation set, N is the total number of negative examples in the validation set, p is the number of positive examples in the validation set covered by the rule, and n is the number of negative examples in the validation set covered by the rule. $vIREP$ is actually similar to classification accuracy for the validation set. IREP favors rules that have higher values of $vIREP$. On the other hand, RIPPER applies the following measure to determine whether a rule should be pruned:

$$vRIPPER = p - nP + n.$$

- a. Suppose R_1 is covered by 350 positive examples and 150 negative examples, while R_2 is covered by 300 positive examples and 50 negative examples. Compute the FOIL's information gain for the rule R_2 with respect to R_1 .
- b. Consider a validation set that contains 500 positive examples and 500 negative examples. For R_1 , suppose the number of positive examples covered by the rule is 200, and the number of negative examples covered by the rule is 50. For R_2 , suppose the number of positive examples covered by the rule is 100 and the number of negative examples is 5. Compute $vIREP$ for both rules. Which rule does IREP prefer?
- c. Compute $vRIPPER$ for the previous problem. Which rule does RIPPER prefer?

3. C4.5rules is an implementation of an indirect method for generating rules from a decision tree. RIPPER is an implementation of a direct method for generating rules directly from data.

- a. Discuss the strengths and weaknesses of both methods.
- b. Consider a data set that has a large difference in the class size (i.e., some classes are much bigger than others). Which method (between C4.5rules and RIPPER) is better in terms of finding high accuracy rules for the small classes?

4. Consider a training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R1:A \rightarrow + (covers 4 positive and 1 negative examples), R2:B \rightarrow + (covers 30 positive and 10 negative examples), R3:C \rightarrow + (covers 100 positive and 90 negative examples),

determine which is the best and worst candidate rule according to:

- a. Rule accuracy.
- b. FOIL's information gain.
- c. The likelihood ratio statistic.
- d. The Laplace measure.
- e. The m-estimate measure (with k=2 and p+=0.2).

5. **Figure 4.3**  illustrates the coverage of the classification rules R1, R2, and R3. Determine which is the best and worst rule according to:

- a. The likelihood ratio statistic.
- b. The Laplace measure.

- c. The m-estimate measure (with $k=2$ and $p+=0.58$).
- d. The rule accuracy after R_1 has been discovered, where none of the examples covered by R_1 are discarded.
- e. The rule accuracy after R_1 has been discovered, where only the positive examples covered by R_1 are discarded.
- f. The rule accuracy after R_1 has been discovered, where both positive and negative examples covered by R_1 are discarded.

6.

- a. Suppose the fraction of undergraduate students who smoke is 15% and the fraction of graduate students who smoke is 23%. If one-fifth of the college students are graduate students and the rest are undergraduates, what is the probability that a student who smokes is a graduate student?
- b. Given the information in part (a), is a randomly chosen college student more likely to be a graduate or undergraduate student?
- c. Repeat part (b) assuming that the student is a smoker.
- d. Suppose 30% of the graduate students live in a dorm but only 10% of the undergraduate students live in a dorm. If a student smokes and lives in the dorm, is he or she more likely to be a graduate or undergraduate student? You can assume independence between students who live in a dorm and those who smoke.

7. Consider the data set shown in **Table 4.9** 

Table 4.9. Data set for Exercise 7.

Instance	A	B	C	Class
1	0	0	0	+

2	0	0	1	-
3	0	1	1	-
4	0	1	1	-
5	0	0	1	+
6	1	0	1	+
7	1	0	1	-
8	1	0	1	-
9	1	1	1	+
10	1	0	1	+

- a. Estimate the conditional probabilities for $P(A|+)$, $P(B|+)$, $P(C|+)$, $P(A|-)$, $P(B|-)$, and $P(C|-)$.
 - b. Use the estimate of conditional probabilities given in the previous question to predict the class label for a test sample ($A=0$, $B=1$, $C=0$) using the naïve Bayes approach.
 - c. Estimate the conditional probabilities using the m-estimate approach, with $p=1/2$ and $m=4$.
 - d. Repeat part (b) using the conditional probabilities given in part (c).
 - e. Compare the two methods for estimating probabilities. Which method is better and why?
8. Consider the data set shown in **Table 4.10**.

Table 4.10. Data set for Exercise 8.

--	--	--	--

Instance	A	B	C	Class
1	0	0	1	-
2	1	0	1	+
3	0	1	0	-
4	1	0	0	-
5	1	0	1	+
6	0	0	1	+
7	1	1	0	-
8	0	0	0	-
9	0	1	0	+
10	1	1	1	+

- Estimate the conditional probabilities for $P(A=1|+)$, $P(B=1|+)$, $P(C=1|+)$, $P(A=1|-)$, $P(B=1|-)$, and $P(C=1|-)$ using the same approach as in the previous problem.
- Use the conditional probabilities in part (a) to predict the class label for a test sample ($A=1$, $B=1$, $C=1$) using the naïve Bayes approach.
- Compare $P(A=1)$, $P(B=1)$, and $P(A=1, B=1)$. State the relationships between A and B .
- Repeat the analysis in part (c) using $P(A=1)$, $P(B=0)$, and $P(A=1, B=0)$.
- Compare $P(A=1, B=1|\text{Class}=+)$ against $P(A=1|\text{Class}=+)$ and $P(B=1|\text{Class}=+)$. Are the variables conditionally independent given the class?

9.

- a. Explain how naïve Bayes performs on the data set shown in [Figure 4.56](#).
- b. If each class is further divided such that there are four classes (A_1 , A_2 , B_1 , and B_2), will naïve Bayes perform better?
- c. How will a decision tree perform on this data set (for the two-class problem)? What if there are four classes?

10. [Figure 4.57](#) illustrates the Bayesian network for the data set shown in [Table 4.11](#). (Assume that all the attributes are binary).

- a. Draw the probability table for each node in the network.
- b. Use the Bayesian network to compute $P(\text{Engine}=\text{Bad}, \text{Air Conditioner}=\text{Broken})$.

11. Given the Bayesian network shown in [Figure 4.58](#), compute the following probabilities:

		Attributes		
		Distinguishing Attributes	Noise Attributes	
Records	A1			
	Class A			
Records	A2			
	Class B			
Records	B1			
	Class A			
Records	B2			
	Class B			

Figure 4.56.
Data set for Exercise 9.

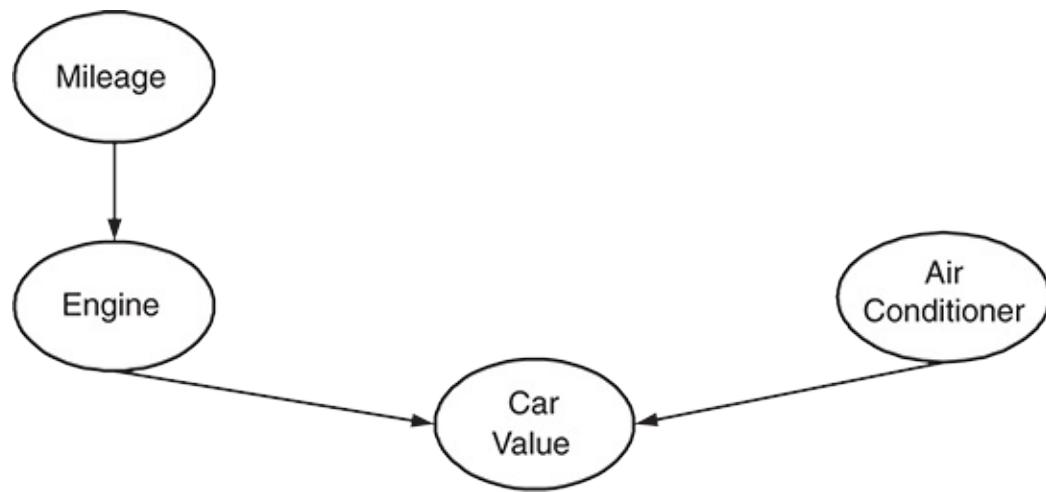


Figure 4.57.
Bayesian network.

a. $P(B=\text{good}, F=\text{empty}, G=\text{empty}, S=\text{yes})$.

- b. $P(B=\text{bad}, F=\text{empty}, G=\text{not empty}, S=\text{no})$.
- c. Given that the battery is bad, compute the probability that the car will start.
12. Consider the one-dimensional data set shown in [Table 4.12](#).
- Classify the data point $x=5.0$ according to its 1-, 3-, 5-, and 9-nearest neighbors (using majority vote).
 - Repeat the previous analysis using the distance-weighted voting approach described in [Section 4.3.1](#).

Table 4.11. Data set for Exercise 10.

Mileage	Engine	Air Conditioner	Number of Instances with Car Value=Hi	Number of Instances with Car Value=Lo
Hi	Good	Working	3	4
Hi	Good	Broken	1	2
Hi	Bad	Working	1	5
Hi	Bad	Broken	0	4
Lo	Good	Working	9	0
Lo	Good	Broken	5	1
Lo	Bad	Working	1	2
Lo	Bad	Broken	0	2

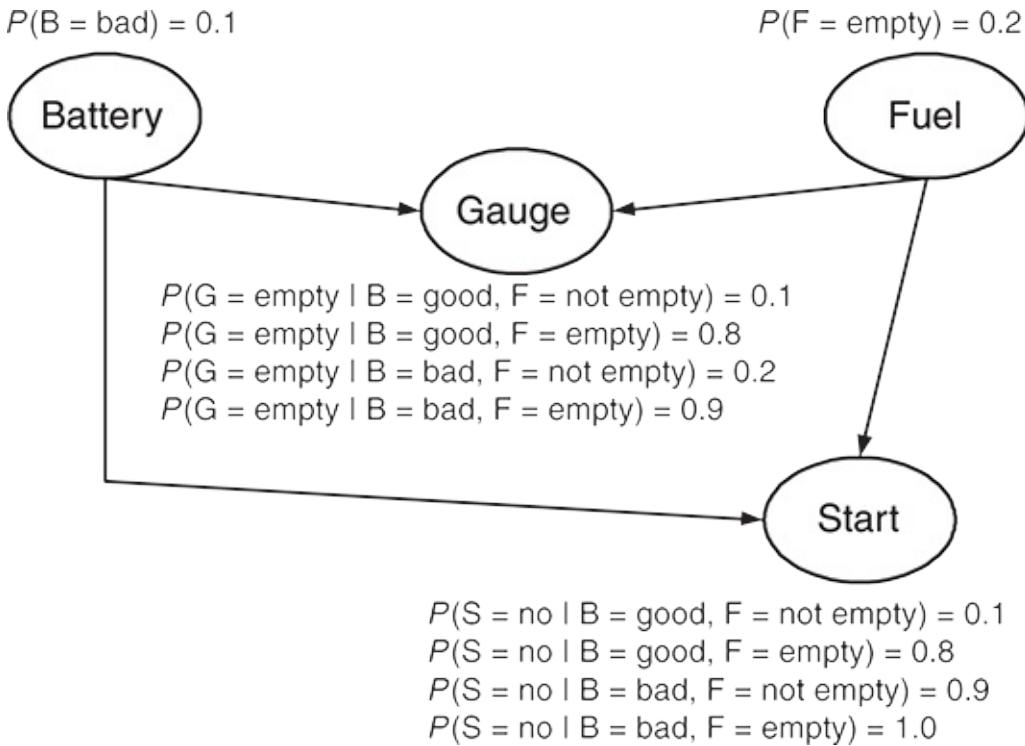


Figure 4.58.

Bayesian network for Exercise 11.

13. The nearest neighbor algorithm described in [Section 4.3](#) can be extended to handle nominal attributes. A variant of the algorithm called PEBLS (Parallel Exemplar-Based Learning System) by Cost and Salzberg [\[219\]](#) measures the distance between two values of a nominal attribute using the modified value difference metric (MVDM). Given a pair of nominal attribute values, V_1 and V_2 , the distance between them is defined as follows:

$$d(V_1, V_2) = \sum_{i=1}^k |n_{i1} - n_{i2}|, \quad (4.108)$$

where n_{ij} is the number of examples from class i with attribute value V_j and n_j is the number of examples with attribute value V_j .

Table 4.12. Data set for Exercise 12.

x	0.5	3.0	4.5	4.6	4.9	5.2	5.3	5.5	7.0	9.5
---	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

y	-	-	+	+	+	-	-	+	-	-
---	---	---	---	---	---	---	---	---	---	---

Consider the training set for the loan classification problem shown in [Figure 4.8](#). Use the MVDM measure to compute the distance between every pair of attribute values for the `Home Owner` and `Marital Status` attributes.

14. For each of the Boolean functions given below, state whether the problem is linearly separable.

- a. $A \text{ AND } B \text{ AND } C$
- b. $\text{NOT } A \text{ AND } B$
- c. $(A \text{ OR } B) \text{ AND } (A \text{ OR } C)$
- d. $(A \text{ XOR } B) \text{ AND } (A \text{ OR } B)$

15.

- a. Demonstrate how the perceptron model can be used to represent the AND and OR functions between a pair of Boolean variables.
- b. Comment on the disadvantage of using linear functions as activation functions for multi-layer neural networks.

16. You are asked to evaluate the performance of two classification models, M1 and M2. The test set you have chosen contains 26 binary attributes, labeled as A through Z. [Table 4.13](#) shows the posterior probabilities obtained by applying the models to the test set. (Only the posterior probabilities for the positive class are shown). As this is a two-class problem, $P(-)=1-P(+)$ and $P(-|A, \dots, Z)=1-P(+|A, \dots, Z)$. Assume that we are mostly interested in detecting instances from the positive class.

- a. Plot the ROC curve for both M1 and M2. (You should plot them on the same graph.) Which model do you think is better? Explain your reasons.

- b. For model M1, suppose you choose the cutoff threshold to be $t=0.5$. In other words, any test instances whose posterior probability is greater than t will be classified as a positive example. Compute the precision, recall, and F -measure for the model at this threshold value.
- c. Repeat the analysis for part (b) using the same cutoff threshold on model M2. Compare the F -measure results for both models. Which model is better? Are the results consistent with what you expect from the ROC curve?
- d. Repeat part (b) for model M1 using the threshold $t=0.1$. Which threshold do you prefer, $t=0.5$ or $t=0.1$? Are the results consistent with what you expect from the ROC curve?

Table 4.13. Posterior probabilities for Exercise 16.

Instance	True Class	$P(+ A, \dots, Z, M1)$	$P(+ A, \dots, Z, M2)$
1	+	0.73	0.61
2	+	0.69	0.03
3	-	0.44	0.68
4	-	0.55	0.31
5	+	0.67	0.45
6	+	0.47	0.09
7	-	0.08	0.38
8	-	0.15	0.05
9	+	0.45	0.01
10	-	0.35	0.04

17. Following is a data set that contains two attributes, X and Y , and two class labels, “+” and “-”. Each attribute can take three different values: 0, 1, or 2.

X	Y	Number of Instances	
		+	-
0	0	0	100
1	0	0	0
2	0	0	100
0	1	10	100
1	1	10	0
2	1	10	100
0	2	0	100
1	2	0	0
2	2	0	100

The concept for the “+” class is $Y=1$ and the concept for the “-” class is $X=0 \vee X=2$.

- Build a decision tree on the data set. Does the tree capture the “+” and “-” concepts?
- What are the accuracy, precision, recall, and F1-measure of the decision tree? (Note that precision, recall, and F1-measure are defined with

respect to the “+” class.)

- c. Build a new decision tree with the following cost function:

$$C(i, j) = \begin{cases} 0, & \text{if } i=j \\ 1, & \text{if } i=+, j=- \\ \text{Number of } - \text{ instances} & \text{Number of } + \text{ instances} \\ & \text{if } i=-, j=+ \end{cases}$$

(Hint: only the leaves of the old decision tree need to be changed.) Does the decision tree capture the “+” concept?

- d. What are the accuracy, precision, recall, and F1-measure of the new decision tree?

18. Consider the task of building a classifier from random data, where the attribute values are generated randomly irrespective of the class labels.

Assume the data set contains instances from two classes, “+” and “-.” Half of the data set is used for training while the remaining half is used for testing.

- a. Suppose there are an equal number of positive and negative instances in the data and the decision tree classifier predicts every test instance to be positive. What is the expected error rate of the classifier on the test data?
- b. Repeat the previous analysis assuming that the classifier predicts each test instance to be positive class with probability 0.8 and negative class with probability 0.2.
- c. Suppose two-thirds of the data belong to the positive class and the remaining one-third belong to the negative class. What is the expected error of a classifier that predicts every test instance to be positive?
- d. Repeat the previous analysis assuming that the classifier predicts each test instance to be positive class with probability 2/3 and negative class with probability 1/3.

19. Derive the dual Lagrangian for the linear SVM with non-separable data where the objective function is

$$f(w) = \|w\|^2 + C(\sum_{i=1}^N \xi_i)$$

20. Consider the XOR problem where there are four training points:

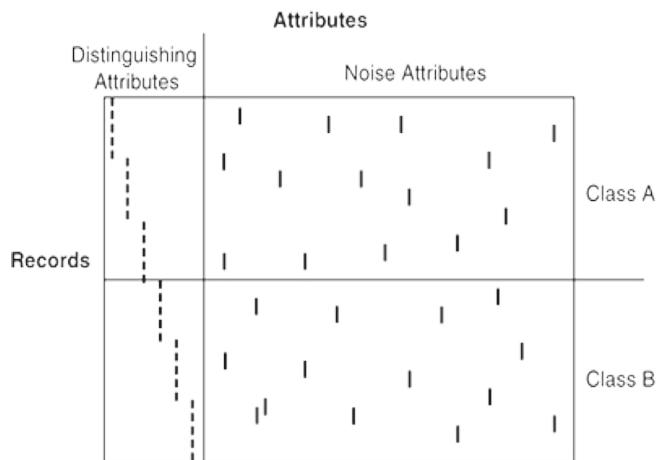
$$(1, 1, -), (1, 0, +), (0, 1, +), (0, 0, -).$$

Transform the data into the following feature space:

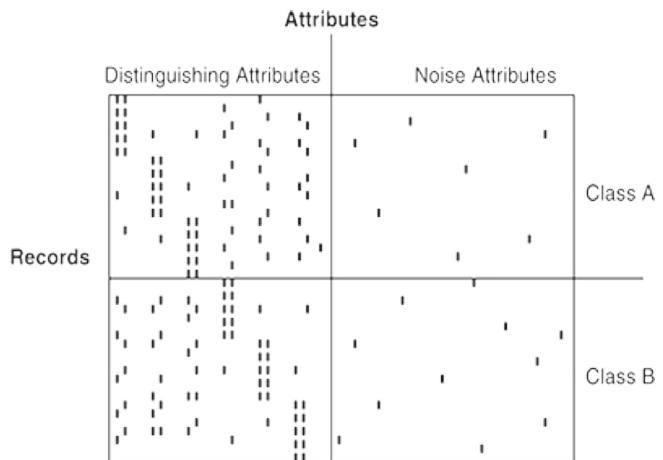
$$\phi = (1, 2x_1, 2x_2, 2x_1x_2, x_1^2, x_2^2).$$

Find the maximum margin linear decision boundary in the transformed space.

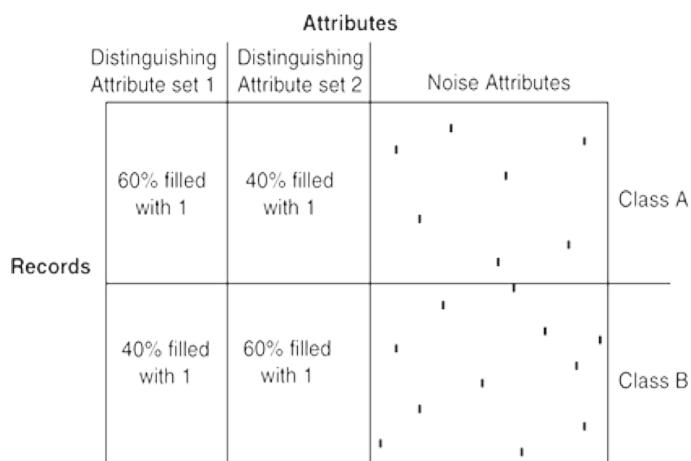
21. Given the data sets shown in [Figures 4.59](#), explain how the decision tree, naïve Bayes, and k -nearest neighbor classifiers would perform on these data sets.



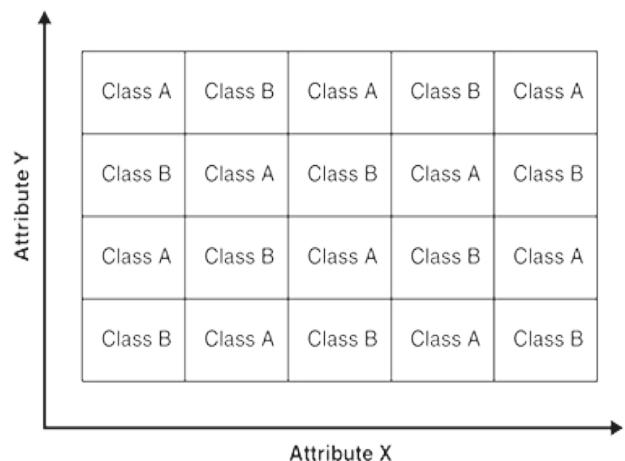
(a) Synthetic data set 1.



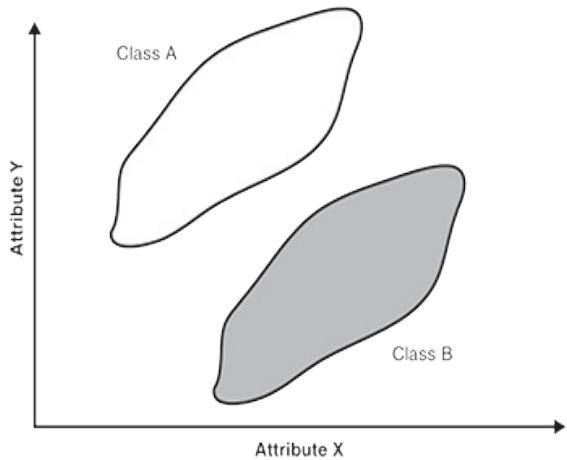
(b) Synthetic data set 2.



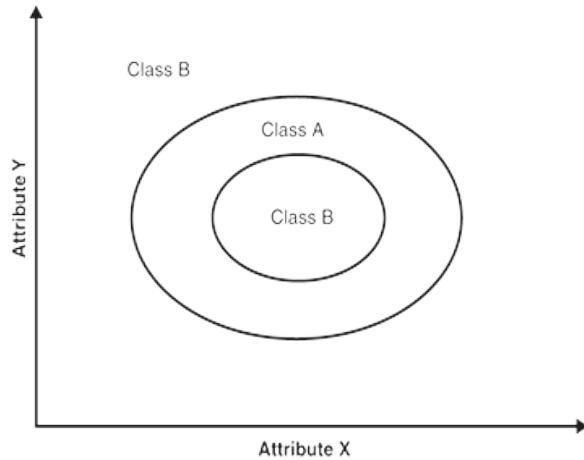
(c) Synthetic data set 3.



(d) Synthetic data set 4.



(e) Synthetic data set 5.



(f) Synthetic data set 6.

Figure 4.59.

Data set for Exercise 21.

5 Association Analysis: Basic Concepts and Algorithms

*Many business enterprises accumulate large quantities of data from their day-to-day operations. For example, huge amounts of customer purchase data are collected daily at the checkout counters of grocery stores. **Table 5.1** gives an example of such data, commonly known as **market basket transactions**. Each row in this table corresponds to a transaction, which contains a unique identifier labeled *TID* and a set of items bought by a given customer. Retailers are interested in analyzing the data to learn about the purchasing behavior of their customers. Such valuable information can be used to support a variety of business-related applications such as marketing promotions, inventory management, and customer relationship management.*

Table 5.1. An example of market basket transactions.

<i>TID</i>	Items

1	$\{Bread, Milk\}$
2	$\{Bread, Diapers, Beer, Eggs\}$
3	$\{Milk, Diapers, Beer, Cola\}$
4	$\{Bread, Milk, Diapers, Beer\}$
5	$\{Bread, Milk, Diapers, Cola\}$

This chapter presents a methodology known as **association analysis**, which is useful for discovering interesting relationships hidden in large data sets. The uncovered relationships can be represented in the form of sets of items present in many transactions, which are known as **frequent itemsets**, or **association rules**, that represent relationships between two itemsets. For example, the following rule can be extracted from the data set shown in [Table 5.1](#):

$\{Diapers\} \rightarrow \{Beer\}$.

The rule suggests a relationship between the sale of diapers and beer because many customers who buy diapers also buy beer. Retailers can use these types of rules to help them identify new opportunities for cross-selling their products to the customers.

Besides market basket data, association analysis is also applicable to data from other application domains such as bioinformatics, medical diagnosis, web mining, and scientific data analysis. In the analysis of Earth science data, for example, association patterns may reveal interesting connections among the ocean, land, and atmospheric processes. Such information may help Earth scientists develop a better understanding of how the different elements of the Earth system interact with each other. Even though the techniques presented here are generally applicable to a wider variety of data sets, for illustrative purposes, our discussion will focus mainly on market basket data.

There are two key issues that need to be addressed when applying association analysis to market basket data. First, discovering patterns from a large transaction data set can be computationally expensive. Second, some of the discovered patterns may be spurious (happen simply by chance) and even for non-spurious patterns, some are more interesting than others. The remainder of this chapter is organized around these two issues. The first part of the chapter is devoted to explaining the basic concepts of association analysis and the algorithms used to efficiently mine such patterns. The second part of the chapter deals with the issue of evaluating the discovered patterns in order to help prevent the generation of spurious results and to rank the patterns in terms of some interestingness measure.

5.1 Preliminaries

This section reviews the basic terminology used in association analysis and presents a formal description of the task.

Binary Representation

Market basket data can be represented in a binary format as shown in [Table 5.2](#), where each row corresponds to a transaction and each column corresponds to an item. An item can be treated as a binary variable whose value is one if the item is present in a transaction and zero otherwise. Because the presence of an item in a transaction is often considered more important than its absence, an item is an **asymmetric** binary variable. This representation is a simplistic view of real market basket data because it ignores important aspects of the data such as the quantity of items sold or the price paid to purchase them. Methods for handling such non-binary data will be explained in [Chapter 6](#).

Table 5.2. A binary 0/1 representation of market basket data.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Itemset and Support Count

Let $I=\{i_1, i_2, \dots, i_d\}$ be the set of all items in a market basket data and $T=\{t_1, t_2, \dots, t_N\}$ be the set of all transactions. Each transaction, t_i contains a subset of items chosen from I . In association analysis, a collection of zero or more items is termed an itemset. If an itemset contains k items, it is called a k -itemset. For instance, $\{\text{Beer}, \text{Diapers}, \text{Milk}\}$ is an example of a 3-itemset. The null (or empty) set is an itemset that does not contain any items.

A transaction t_j is said to contain an itemset X if X is a subset of t_j . For example, the second transaction shown in [Table 5.2](#) contains the itemset $\{\text{Bread}, \text{Diapers}\}$ but not $\{\text{Bread}, \text{Milk}\}$. An important property of an itemset is its support count, which refers to the number of transactions that contain a particular itemset. Mathematically, the support count, $\sigma(X)$, for an itemset X can be stated as follows:

$$\sigma(X)=|\{t_i|X\subseteq t_i, t_i\in T\}|,$$

where the symbol $|\cdot|$ denotes the number of elements in a set. In the data set shown in [Table 5.2](#), the support count for $\{\text{Beer}, \text{Diapers}, \text{Milk}\}$ is equal to two because there are only two transactions that contain all three items.

Often, the property of interest is the support, which is fraction of transactions in which an itemset occurs:

$$s(X)=\sigma(X)/N.$$

An itemset X is called frequent if $s(X)$ is greater than some user-defined threshold, $minsup$.

Association Rule

An association rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its **support** and **confidence**. Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in Y appear in transactions that contain X . The formal definitions of these metrics are

$$\text{Support}, s(X \rightarrow Y) = \sigma(X \cup Y)N; \quad (5.1)$$

$$\text{Confidence}, c(X \rightarrow Y) = \sigma(X \cup Y) \sigma(X). \quad (5.2)$$

Example 5.1.

Consider the rule $\{\text{Milk, Diapers}\} \rightarrow \{\text{Beer}\}$. Because the support count for $\{\text{Milk, Diapers, Beer}\}$ is 2 and the total number of transactions is 5, the rule's support is $2/5=0.4$. The rule's confidence is obtained by dividing the support count for $\{\text{Milk, Diapers, Beer}\}$ by the support count for $\{\text{Milk, Diapers}\}$. Since there are 3 transactions that contain milk and diapers, the confidence for this rule is $2/3=0.67$.

Why Use Support and Confidence?

Support is an important measure because a rule that has very low support might occur simply by chance. Also, from a business perspective a low support rule is unlikely to be interesting because it might not be profitable to promote items that customers seldom buy together (with the exception of the situation described in [Section 5.8](#)). For these reasons, we are interested in finding rules whose support is greater than some user-defined threshold. As

will be shown in [Section 5.2.1](#), support also has a desirable property that can be exploited for the efficient discovery of association rules.

Confidence, on the other hand, measures the reliability of the inference made by a rule. For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X . Confidence also provides an estimate of the conditional probability of Y given X .

Association analysis results should be interpreted with caution. The inference made by an association rule does not necessarily imply causality. Instead, it can sometimes suggest a strong co-occurrence relationship between items in the antecedent and consequent of the rule. Causality, on the other hand, requires knowledge about which attributes in the data capture cause and effect, and typically involves relationships occurring over time (e.g., greenhouse gas emissions lead to global warming). See [Section 5.7.1](#) for additional discussion.

Formulation of the Association Rule Mining Problem

The association rule mining problem can be formally stated as follows:

Definition 5.1. (Association Rule Discovery.)

Given a set of transactions T , find all the rules having support $\geq \text{minsup}$ and confidence $\geq \text{minconf}$, where minsup and minconf are the corresponding support and confidence thresholds.

A brute-force approach for mining association rules is to compute the support and confidence for every possible rule. This approach is prohibitively expensive because there are exponentially many rules that can be extracted from a data set. More specifically, assuming that neither the left nor the right-hand side of the rule is an empty set, the total number of possible rules, R , extracted from a data set that contains d items is

$$R=3^d-2^d+1. \quad (5.3)$$

The proof for this equation is left as an exercise to the readers (see Exercise 5 on page 440). Even for the small data set shown in [Table 5.1](#), this approach requires us to compute the support and confidence for $36-27+1=602$ rules. More than 80% of the rules are discarded after applying $\text{minsup}=20\%$ and $\text{minconf}=50\%$, thus wasting most of the computations. To avoid performing needless computations, it would be useful to prune the rules early without having to compute their support and confidence values.

An initial step toward improving the performance of association rule mining algorithms is to decouple the support and confidence requirements. From [Equation 5.1](#), notice that the support of a rule $X \rightarrow Y$ is the same as the support of its corresponding itemset, $X \cup Y$. For example, the following rules have identical support because they involve items from the same itemset,

{Beer, Diapers, Milk}:

{Beer, Diapers} \rightarrow {Milk}, {Beer, Milk} \rightarrow {Diapers}, {Diapers, Milk} \rightarrow {Beer}, {Beer} \rightarrow {Diapers, Milk}, {Milk} \rightarrow {Beer, Diapers}, {Diapers} \rightarrow {Beer, Milk}.

If the itemset is infrequent, then all six candidate rules can be pruned immediately without our having to compute their confidence values.

Therefore, a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1. **Frequent Itemset Generation**, whose objective is to find all the itemsets that satisfy the *minsup* threshold.
2. **Rule Generation**, whose objective is to extract all the high confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.

The computational requirements for frequent itemset generation are generally more expensive than those of rule generation. Efficient techniques for generating frequent itemsets and association rules are discussed in [Sections 5.2](#) and [5.3](#), respectively.

5.2 Frequent Itemset Generation

A lattice structure can be used to enumerate the list of all possible itemsets.

Figure 5.1 shows an itemset lattice for $I=\{a, b, c, d, e\}$. In general, a data set that contains k items can potentially generate up to $2^k - 1$ frequent itemsets, excluding the null set. Because k can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large.

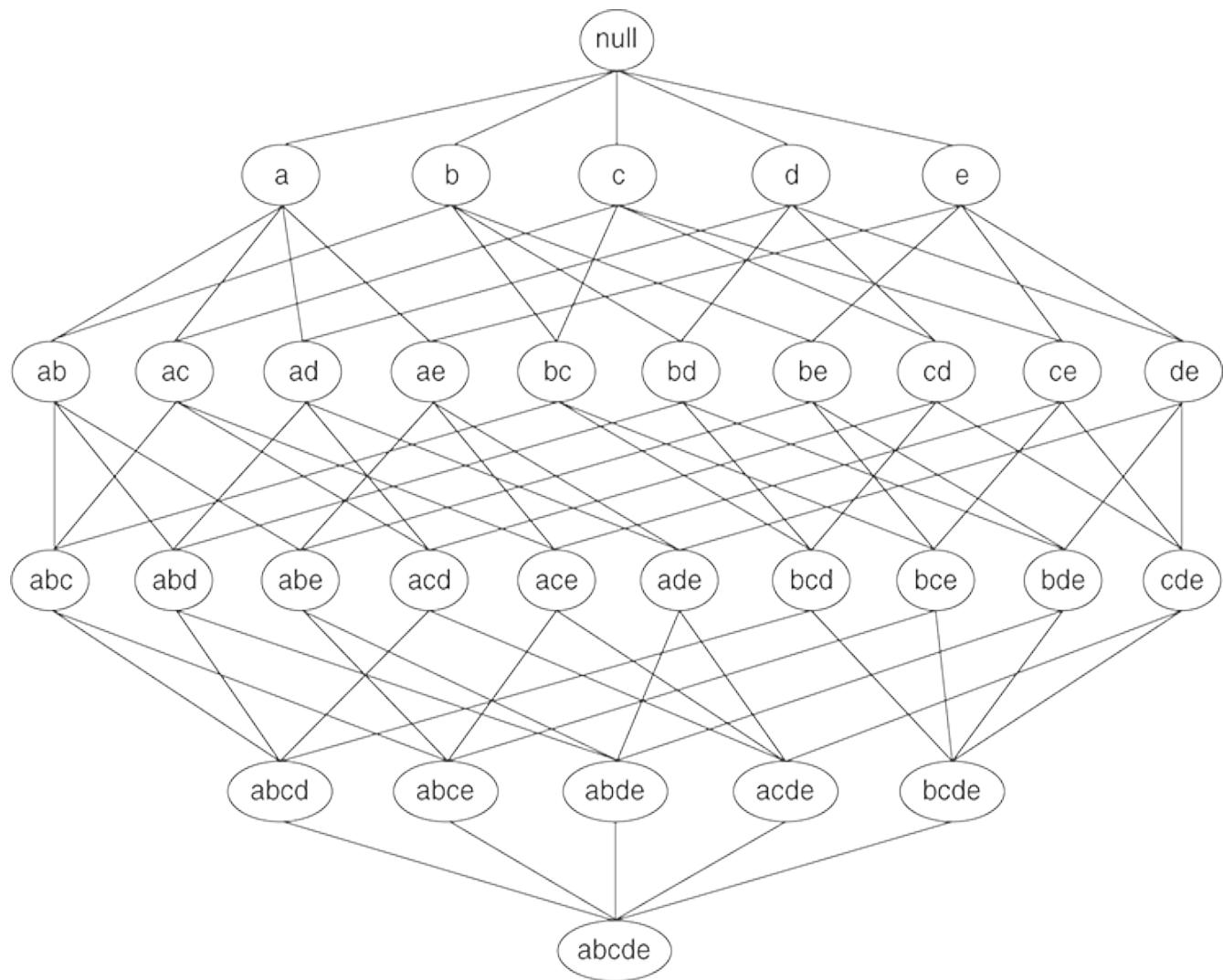


Figure 5.1.

An itemset lattice.

A brute-force approach for finding frequent itemsets is to determine the support count for every **candidate itemset** in the lattice structure. To do this, we need to compare each candidate against every transaction, an operation that is shown in [Figure 5.2](#). If the candidate is contained in a transaction, its support count will be incremented. For example, the support for $\{\text{Bread}, \text{Milk}\}$ is incremented three times because the itemset is contained in transactions 1, 4, and 5. Such an approach can be very expensive because it requires $O(NMw)$ comparisons, where N is the number of transactions, $M=2^k - 1$ is the number of candidate itemsets, and w is the maximum transaction width. **Transaction width** is the number of items present in a transaction.

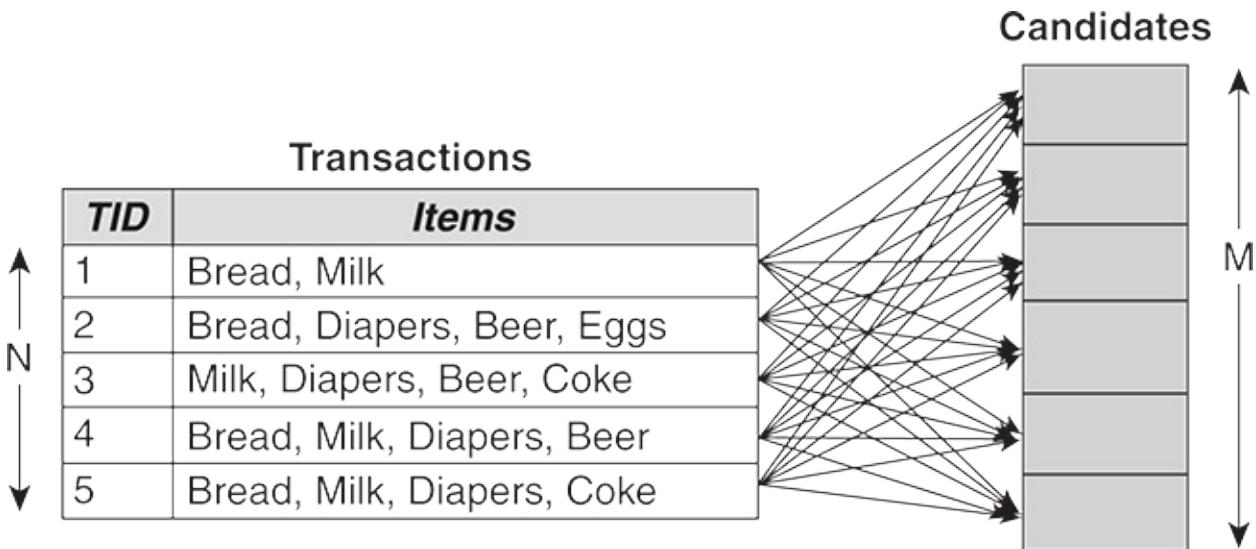


Figure 5.2.

Counting the support of candidate itemsets.

There are three main approaches for reducing the computational complexity of frequent itemset generation.

1. **Reduce the number of candidate itemsets (M)**. The *Apriori* principle, described in the next Section, is an effective way to eliminate some of

the candidate itemsets without counting their support values.

2. **Reduce the number of comparisons.** Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set. We will discuss these strategies in [Sections 5.2.4](#) and [5.6](#), respectively.
3. **Reduce the number of transactions (N).** As the size of candidate itemsets increases, fewer transactions will be supported by the itemsets. For instance, since the width of the first transaction in [Table 5.1](#) is 2, it would be advantageous to remove this transaction before searching for frequent itemsets of size 3 and larger. Algorithms that employ such a strategy are discussed in the Bibliographic Notes.

5.2.1 The *Apriori* Principle

This Section describes how the support measure can be used to reduce the number of candidate itemsets explored during frequent itemset generation. The use of support for pruning candidate itemsets is guided by the following principle.

Theorem 5.1 (*Apriori* Principle).

If an itemset is frequent, then all of its subsets must also be frequent.

To illustrate the idea behind the *Apriori* principle, consider the itemset lattice shown in [Figure 5.3](#). Suppose $\{c, d, e\}$ is a frequent itemset. Clearly, any transaction that contains $\{c, d, e\}$ must also contain its subsets, $\{c, d\}$, $\{c, e\}$, $\{d, e\}$, $\{c\}$, $\{d\}$, and $\{e\}$. As a result, if $\{c, d, e\}$ is frequent, then all subsets of $\{c, d, e\}$ (i.e., the shaded itemsets in this figure) must also be frequent.

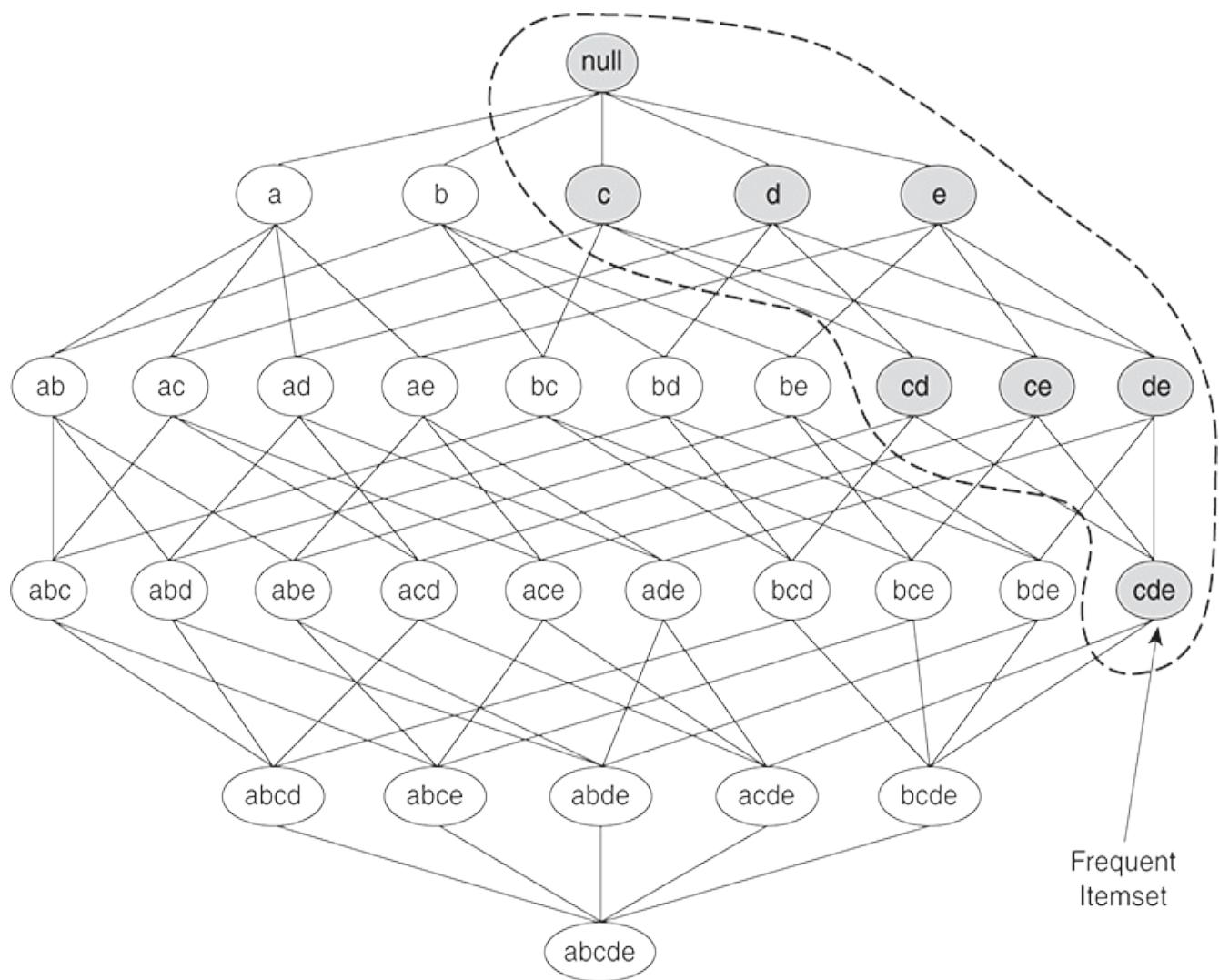


Figure 5.3.

An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Conversely, if an itemset such as $\{a, b\}$ is infrequent, then all of its supersets must be infrequent too. As illustrated in [Figure 5.4](#), the entire subgraph

containing the supersets of $\{a, b\}$ can be pruned immediately once $\{a, b\}$ is found to be infrequent. This strategy of trimming the exponential search space based on the support measure is known as **support-based pruning**. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property is also known as the **anti-monotone** property of the support measure.

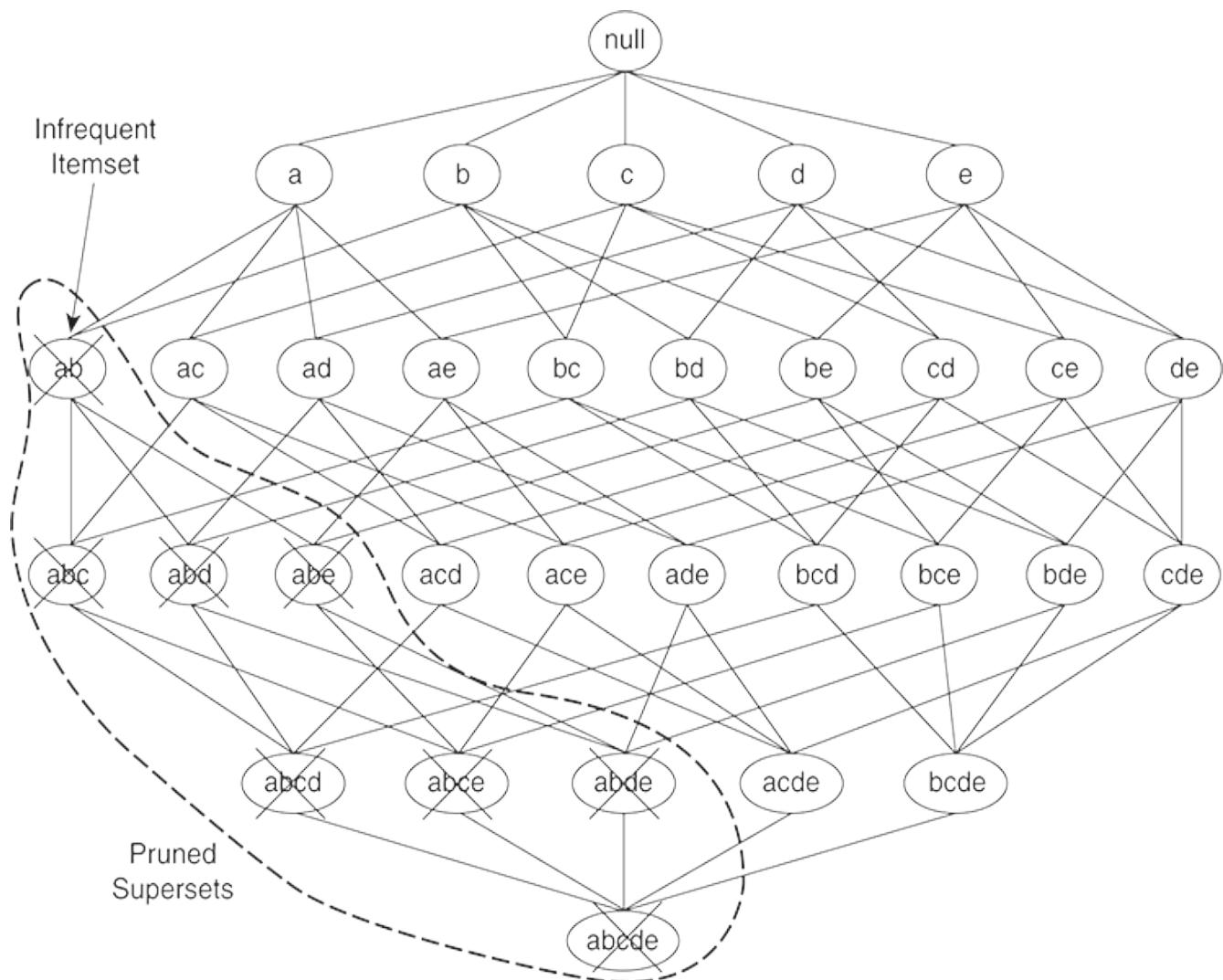


Figure 5.4.

An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

Definition 5.2. (Anti-monotone Property.)

A measure f possesses the anti-monotone property if for every itemset X that is a proper subset of itemset Y , i.e. $X \subset Y$, we have $f(Y) \leq f(X)$.

More generally, a large number of measures—see [Section 5.7.1](#)—can be applied to itemsets to evaluate various properties of itemsets. As will be shown in the next Section, any measure that has the anti-monotone property can be incorporated directly into an itemset mining algorithm to effectively prune the exponential search space of candidate itemsets.

5.2.2 Frequent Itemset Generation in the *Apriori* Algorithm

Apriori is the first association rule mining algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets. [Figure 5.5](#) provides a high-level illustration of the frequent itemset generation part of the *Apriori* algorithm for the transactions shown in [Table 5.1](#). We assume that the support threshold is 60%, which is equivalent to a minimum support count equal to 3.

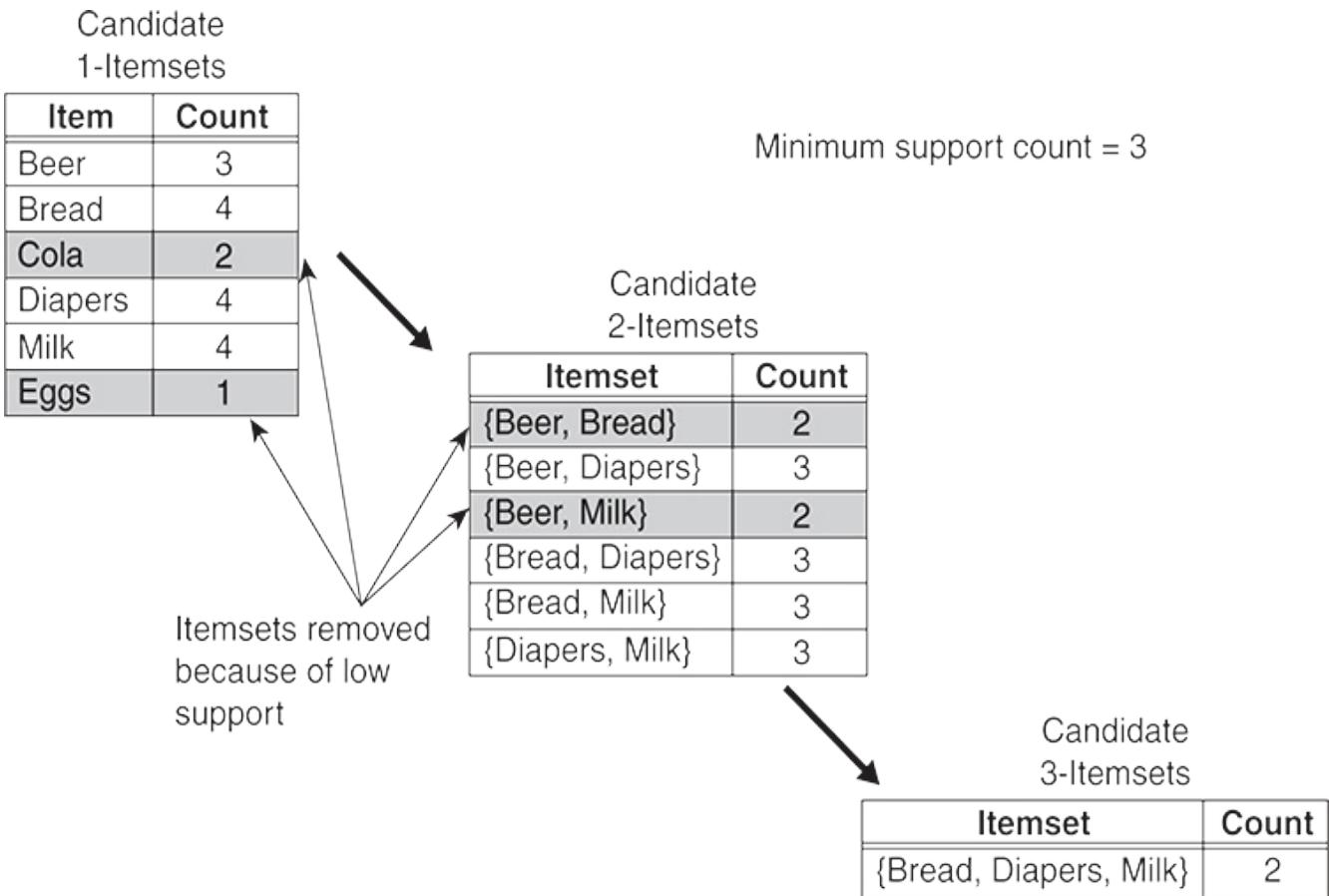


Figure 5.5.

Illustration of frequent itemset generation using the *Apriori* algorithm.

Initially, every item is considered as a candidate 1-itemset. After counting their supports, the candidate itemsets $\{\text{Cola}\}$ and $\{\text{Eggs}\}$ are discarded because they appear in fewer than three transactions. In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets because the *Apriori* principle ensures that all supersets of the infrequent 1-itemsets must be infrequent. Because there are only four frequent 1-itemsets, the number of candidate 2-itemsets generated by the algorithm is $(4^2)=16$. Two of these sixteen candidates, $\{\text{Beer}, \text{Bread}\}$ and $\{\text{Beer}, \text{Milk}\}$, are subsequently found to be infrequent after computing their support values. The remaining four candidates are frequent, and thus will be used to generate candidate 3-itemsets. Without support-based pruning, there are $(6^3)=20$ candidate 3-itemsets that can be formed using the six items given in this example. With

the *Apriori* principle, we only need to keep candidate 3-itemsets whose subsets are frequent. The only candidate that has this property is $\{\text{Bread}, \text{Diapers}, \text{Milk}\}$. However, even though the subsets of $\{\text{Bread}, \text{Diapers}, \text{Milk}\}$ are frequent, the itemset itself is not.

The effectiveness of the *Apriori* pruning strategy can be shown by counting the number of candidate itemsets generated. A brute-force strategy of enumerating all itemsets (up to size 3) as candidates will produce

$$(61)+(62)+(63)=6+15+20=41$$

candidates. With the *Apriori* principle, this number decreases to

$$(61)+(42)+1=6+6+1=13$$

candidates, which represents a 68% reduction in the number of candidate itemsets even in this simple example.

The pseudocode for the frequent itemset generation part of the *Apriori* algorithm is shown in [Algorithm 5.1](#). Let C_k denote the set of candidate k -itemsets and F_k denote the set of frequent k -itemsets:

- The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, F_1 , will be known (steps 1 and 2).
- Next, the algorithm will iteratively generate new candidate k -itemsets and prune unnecessary candidates that are guaranteed to be infrequent given the frequent $(k-1)$ -itemsets found in the previous iteration (steps 5 and 6). Candidate generation and pruning is implemented using the functions `candidate-gen` and `candidate-prune`, which are described in [Section 5.2.3](#).

- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 7–12). The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction t . The implementation of this function is described in [Section 5.2.4](#).
- After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than $N \times \text{minsup}$ (step 13).
- The algorithm terminates when there are no new frequent itemsets generated, i.e., $F_k = \emptyset$ (step 14).

The frequent itemset generation part of the *Apriori* algorithm has two important characteristics. First, it is a **level-wise** algorithm; i.e., it traverses the itemset lattice one level at a time, from frequent 1-itemsets to the maximum size of frequent itemsets. Second, it employs a **generate-and-test** strategy for finding frequent itemsets. At each iteration (level), new candidate itemsets are generated from the frequent itemsets found in the previous iteration. The support for each candidate is then counted and tested against the *minsup* threshold. The total number of iterations needed by the algorithm is $k_{\max} + 1$, where k_{\max} is the maximum size of the frequent itemsets.

5.2.3 Candidate Generation and Pruning

The candidate-gen and candidate-prune functions shown in Steps 5 and 6 of [Algorithm 5.1](#) generate candidate itemsets and prunes unnecessary ones by performing the following two operations, respectively:

1. **Candidate Generation.** This operation generates new candidate k -itemsets based on the frequent $(k-1)$ -itemsets found in the previous iteration.

Algorithm 5.1 Frequent itemset generation of the Apriori algorithm.

```

1:  $k = 1.$ 

2:  $F_k = \{i \mid i \in I \wedge \sigma(\{i\}) \geq N \times \text{minsup}\}.$  {Find all frequent
   1-itemsets}

3: repeat

4:    $k = k + 1.$ 

5:    $C_k = \text{candidate-gen}(F_{k-1}).$  {Generate candidate itemsets.}

6:    $C_k = \text{candidate-prune}(C_k, F_{k-1}).$  {Prune candidate
   itemsets.}

7:   for each transaction  $t \in T$  do

8:      $C_t = \text{subset}(C_k, t).$  {Identify all candidates that
   belong to  $t.$ }

9:     for each candidate itemset  $c \in C_t$  do

10:       $\sigma(c) = \sigma(c) + 1.$  {Increment support count.}

11:      end for

12:    end for

13:     $F_k = \{c \mid c \in C_k \wedge \sigma(c) \geq N \times \text{minsup}\}.$  {Extract the
   frequent  $k$ -itemsets.}

14: until  $F_k = \emptyset$ 

15:  $\text{Result} = \text{UF}_k.$ 

```

2. **Candidate Pruning.** This operation eliminates some of the candidate k -itemsets using support-based pruning, i.e. by removing k -itemsets whose subsets are known to be infrequent in previous iterations. Note

that this pruning is done without computing the actual support of these k -itemsets (which could have required comparing them against each transaction).

Candidate Generation

In principle, there are many ways to generate candidate itemsets. An effective candidate generation procedure must be complete and non-redundant. A candidate generation procedure is said to be *complete* if it does not omit any frequent itemsets. To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., $\forall k: F_k \subseteq C_k$. A candidate generation procedure is *non-redundant* if it does not generate the same candidate itemset more than once. For example, the candidate itemset $\{a, b, c, d\}$ can be generated in many ways—by merging $\{a, b, c\}$ with $\{d\}$, $\{b, d\}$ with $\{a, c\}$, $\{c\}$ with $\{a, b, d\}$, etc. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons. Also, an effective candidate generation procedure should avoid generating too many unnecessary candidates. A candidate itemset is unnecessary if at least one of its subsets is infrequent, and thus, eliminated in the candidate pruning step.

Next, we will briefly describe several candidate generation procedures, including the one used by the candidate-gen function.

Brute-Force Method

The brute-force method considers every k -itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates whose subsets are infrequent (see [Figure 5.6](#)). The number of candidate itemsets generated at level k is equal to $(d)_k$, where d is the total number of items. Although candidate generation is rather trivial, candidate

pruning becomes extremely expensive because a large number of itemsets must be examined.

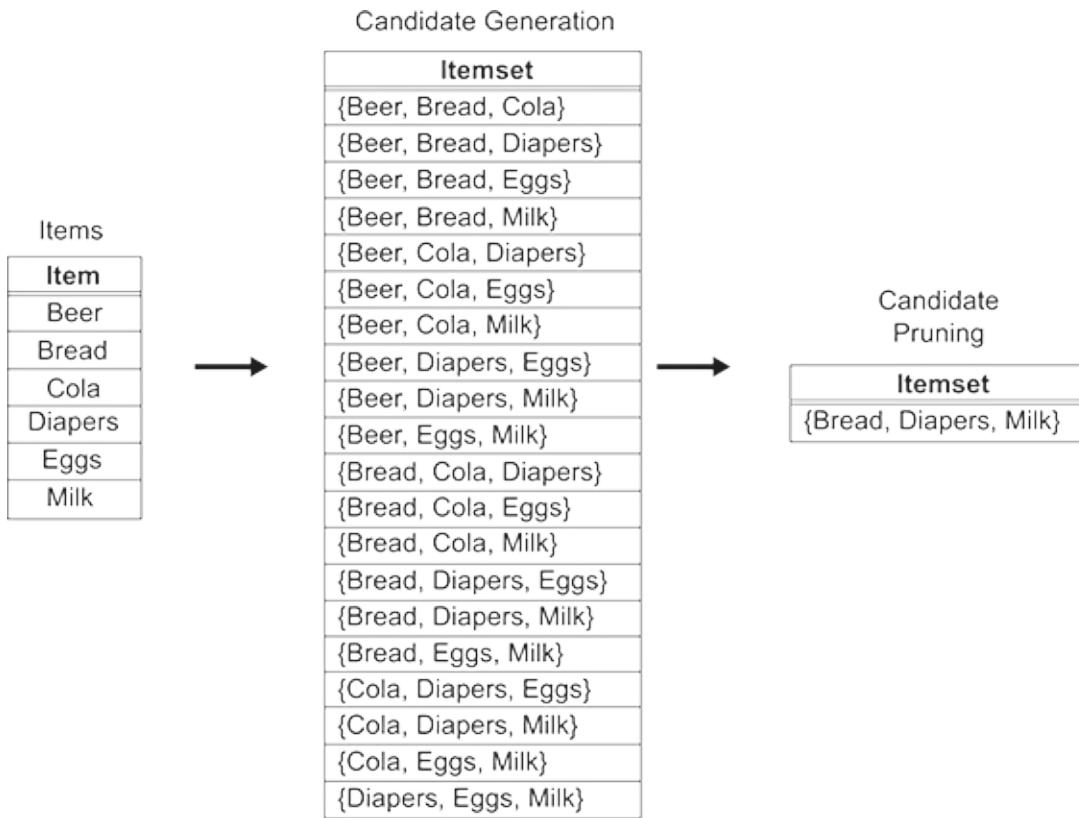


Figure 5.6.

A brute-force method for generating candidate 3-itemsets.

F_{k-1} × F₁ Method

An alternative method for candidate generation is to extend each frequent (k - 1)-itemset with frequent items that are not part of the (k - 1)-itemset. **Figure 5.7** illustrates how a frequent 2-itemset such as {Beer, Diapers} can be augmented with a frequent item such as Bread to produce a candidate 3-itemset {Beer, Diapers, Bread}.

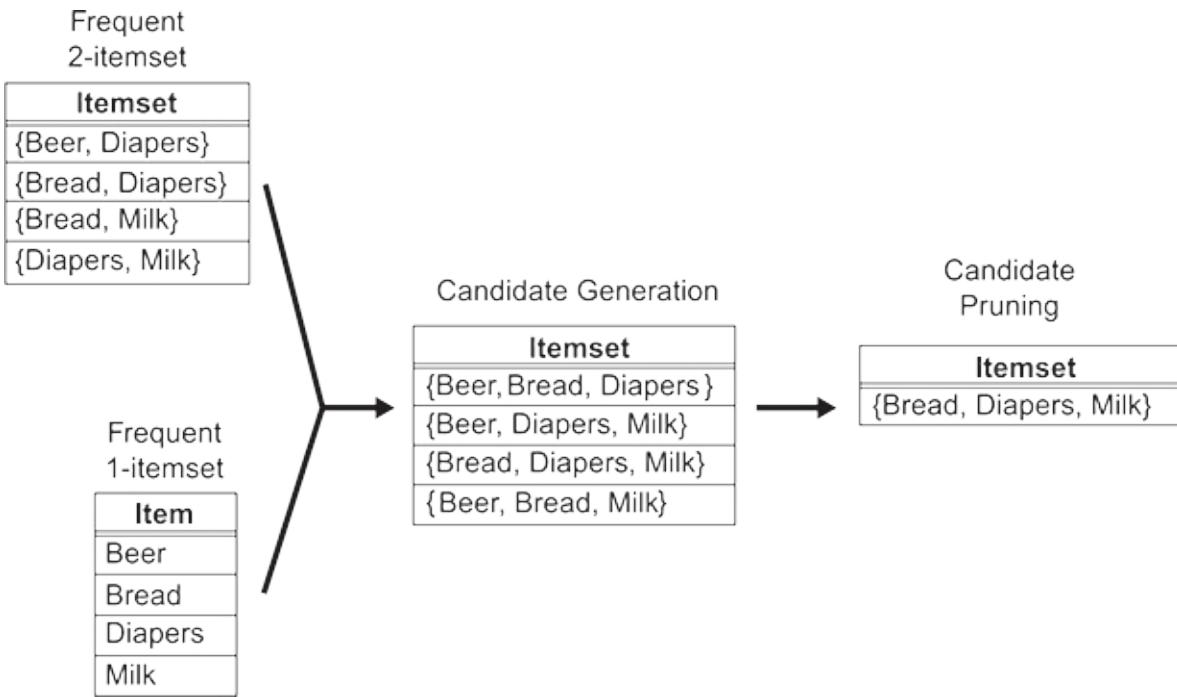


Figure 5.7.

Generating and pruning candidate k -itemsets by merging a frequent $(k-1)$ -itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

The procedure is complete because every frequent k -itemset is composed of a frequent $(k-1)$ -itemset and a frequent 1-itemset. Therefore, all frequent k -itemsets are part of the candidate k -itemsets generated by this procedure.

Figure 5.7 shows that the $F_{k-1} \times F_1$ candidate generation method only produces four candidate 3-itemsets, instead of the

$(63)=20$ itemsets produced by the brute-force method. The $F_{k-1} \times F_1$ method generates lower number of candidates because every candidate is guaranteed to contain at least one frequent $(k-1)$ -itemset. While this procedure is a substantial improvement over the brute-force method, it can still produce a large number of unnecessary candidates, as the remaining subsets of a candidate itemset can still be infrequent.

Note that the approach discussed above does not prevent the same candidate

itemset from being generated more than once. For instance, $\{\text{Bread}, \text{Diapers}, \text{Milk}\}$ can be generated by merging $\{\text{Bread}, \text{Diapers}\}$ with $\{\text{Milk}\}$, $\{\text{Bread}, \text{Milk}\}$ with $\{\text{Diapers}\}$, or $\{\text{Diapers}, \text{Milk}\}$ with $\{\text{Bread}\}$. One way to avoid generating duplicate candidates is by ensuring that the items in each frequent itemset are kept sorted in their lexicographic order. For example, itemsets such as $\{\text{Bread}, \text{Diapers}\}$, $\{\text{Bread}, \text{Diapers}, \text{Milk}\}$, and $\{\text{Diapers}, \text{Milk}\}$ follow lexicographic order as the items within every itemset are arranged alphabetically. Each frequent $(k-1)$ -itemset X is then extended with frequent items that are lexicographically larger than the items in X . For example, the itemset $\{\text{Bread}, \text{Diapers}\}$ can be augmented with $\{\text{Milk}\}$ because Milk is lexicographically larger than Bread and Diapers. However, we should not augment $\{\text{Diapers}, \text{Milk}\}$ with $\{\text{Bread}\}$ nor $\{\text{Bread}, \text{Milk}\}$ with $\{\text{Diapers}\}$ because they violate the lexicographic ordering condition. Every candidate k -itemset is thus generated exactly once, by merging the lexicographically largest item with the remaining $k-1$ items in the itemset. If the $F_{k-1} \times F_1$ method is used in conjunction with lexicographic ordering, then only two candidate 3-itemsets will be produced in the example illustrated in [Figure 5.7](#). $\{\text{Beer}, \text{Bread}, \text{Diapers}\}$ and $\{\text{Beer}, \text{Bread}, \text{Milk}\}$ will not be generated because $\{\text{Beer}, \text{Bread}\}$ is not a frequent 2-itemset.

F_{k-1} × F₁ Method

This candidate generation procedure, which is used in the candidate-gen function of the *Apriori* algorithm, merges a pair of frequent $(k-1)$ -itemsets only if their first $k-2$ items, arranged in lexicographic order, are identical. Let $A = \{a_1, a_2, \dots, a_{k-1}\}$ and $B = \{b_1, b_2, \dots, b_{k-1}\}$ be a pair of frequent $(k-1)$ -itemsets, arranged lexicographically. A and B are merged if they satisfy the following conditions:

$a_i = b_i$ (for $i=1, 2, \dots, k-2$).

Note that in this case, $a_{k-1} \neq b_{k-1}$ because A and B are two distinct itemsets. The candidate k -itemset generated by merging A and B consists of the first $k-2$ common items followed by a_{k-1} and b_{k-1} in lexicographic order. This candidate generation procedure is complete, because for every lexicographically ordered frequent k -itemset, there exists two lexicographically ordered frequent $(k-1)$ -itemsets that have identical items in the first $k-2$ positions.

In [Figure 5.8](#), the frequent itemsets $\{\text{Bread}, \text{Diapers}\}$ and $\{\text{Bread}, \text{Milk}\}$ are merged to form a candidate 3-itemset $\{\text{Bread}, \text{Diapers}, \text{Milk}\}$. The algorithm does not have to merge $\{\text{Beer}, \text{Diapers}\}$ with $\{\text{Diapers}, \text{Milk}\}$ because the first item in both itemsets is different. Indeed, if $\{\text{Beer}, \text{Diapers}, \text{Milk}\}$ is a viable candidate, it would have been obtained by merging $\{\text{Beer}, \text{Diapers}\}$ with $\{\text{Beer}, \text{Milk}\}$ instead. This example illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic ordering to prevent duplicate candidates. Also, if we order the frequent $(k-1)$ -itemsets according to their lexicographic rank, itemsets with identical first $k-2$ items would take consecutive ranks. As a result, the $F^{k-1} \times F^{k-1}$ candidate generation method would consider merging a frequent itemset only with ones that occupy the next few ranks in the sorted list, thus saving some computations.

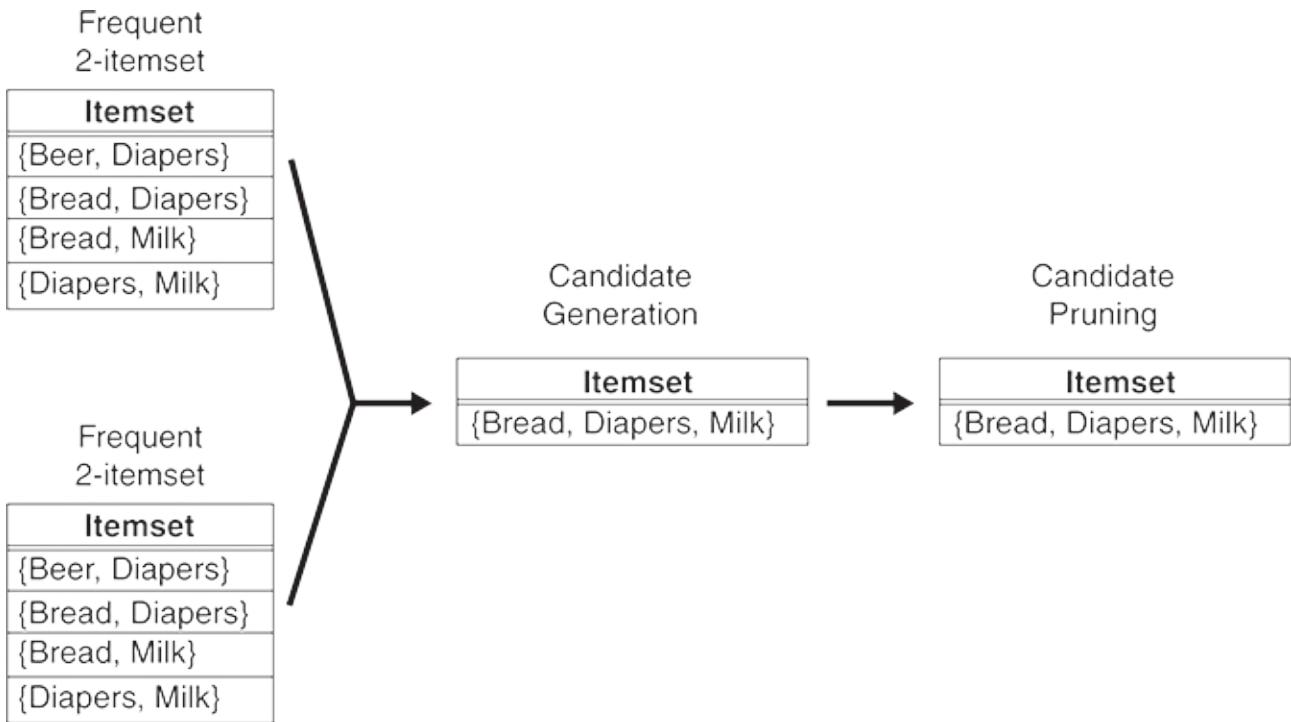


Figure 5.8.

Generating and pruning candidate k -itemsets by merging pairs of frequent $(k-1)$ -itemsets.

Figure 5.8 shows that the $F_{k-1} \times F_{k-1}$ candidate generation procedure results in only one candidate 3-itemset. This is a considerable reduction from the four candidate 3-itemsets generated by the $F_{k-1} \times F_1$ method. This is because the $F_{k-1} \times F_{k-1}$ method ensures that every candidate k -itemset contains at least two frequent $(k-1)$ -itemsets, thus greatly reducing the number of candidates that are generated in this step.

Note that there can be multiple ways of merging two frequent $(k-1)$ -itemsets in the $F_{k-1} \times F_{k-1}$ procedure, one of which is merging if their first $k-2$ items are identical. An alternate approach could be to merge two frequent $(k-1)$ -itemsets A and B if the last $k-2$ items of A are identical to the first $k-2$ items of B . For example, $\{\text{Bread}, \text{Diapers}\}$ and $\{\text{Diapers}, \text{Milk}\}$ could be merged using this approach to generate the candidate 3-itemset $\{\text{Bread}, \text{Diapers}, \text{Milk}\}$. As we will see later, this alternate $F_{k-1} \times F_{k-1}$ procedure is

useful in generating sequential patterns, which will be discussed in [Chapter 6](#).

Candidate Pruning

To illustrate the candidate pruning operation for a candidate k -itemset, $X = \{i_1, i_2, \dots, i_k\}$, consider its k proper subsets, $X - \{i_j\} (\forall j = 1, 2, \dots, k)$. If any of them are infrequent, then X is immediately pruned by using the *Apriori* principle. Note that we don't need to explicitly ensure that all subsets of X of size less than $k-1$ are frequent (see Exercise 7). This approach greatly reduces the number of candidate itemsets considered during support counting. For the brute-force candidate generation method, candidate pruning requires checking only k subsets of size $k-1$ for each candidate k -itemset. However, since the $F_{k-1} \times F_1$ candidate generation strategy ensures that at least one of the $(k-1)$ -size subsets of every candidate k -itemset is frequent, we only need to check for the remaining $k-1$ subsets. Likewise, the $F_{k-1} \times F_{k-1}$ strategy requires examining only $k-2$ subsets of every candidate k -itemset, since two of its $(k-1)$ -size subsets are already known to be frequent in the candidate generation step.

5.2.4 Support Counting

Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the candidate pruning step. Support counting is implemented in steps 6 through 11 of [Algorithm 5.1](#). A brute-force approach for doing this is to compare each transaction against every candidate itemset (see [Figure 5.2](#)) and to update the support counts of candidates contained in a transaction. This approach is computationally

expensive, especially when the numbers of transactions and candidate itemsets are large.

An alternative approach is to enumerate the itemsets contained in each transaction and use them to update the support counts of their respective candidate itemsets. To illustrate, consider a transaction t that contains five items, $\{1, 2, 3, 5, 6\}$. There are $(53)=10$ itemsets of size 3 contained in this transaction. Some of the itemsets may correspond to the candidate 3-itemsets under investigation, in which case, their support counts are incremented. Other subsets of t that do not correspond to any candidates can be ignored.

Figure 5.9 shows a systematic way for enumerating the 3-itemsets contained in t . Assuming that each itemset keeps its items in increasing lexicographic order, an itemset can be enumerated by specifying the smallest item first, followed by the larger items. For instance, given $t=\{1, 2, 3, 5, 6\}$, all the 3-itemsets contained in t must begin with item 1, 2, or 3. It is not possible to construct a 3-itemset that begins with items 5 or 6 because there are only two items in t whose labels are greater than or equal to 5. The number of ways to specify the first item of a 3-itemset contained in t is illustrated by the Level 1 prefix tree structure depicted in **Figure 5.9**. For instance, 1 2 3 5 6 represents a 3-itemset that begins with item 1, followed by two more items chosen from the set $\{2, 3, 5, 6\}$.

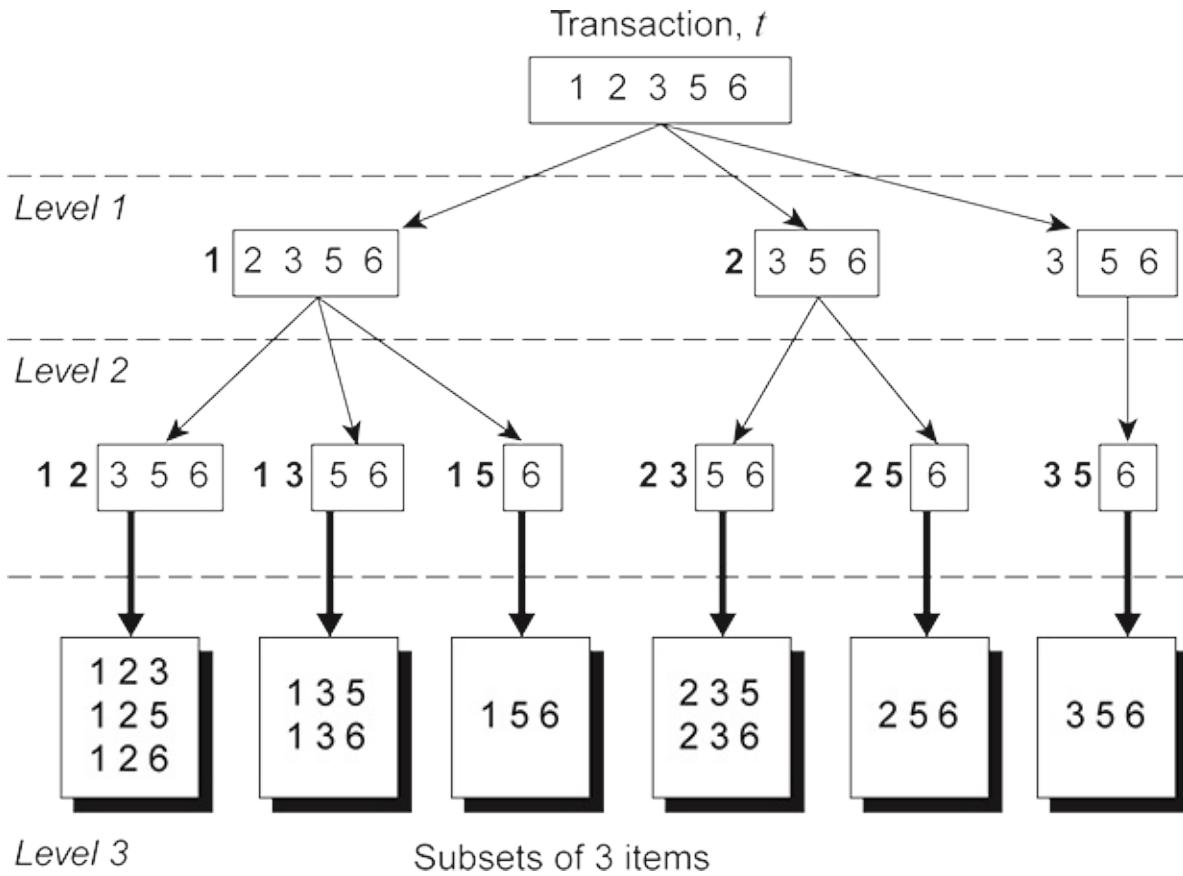


Figure 5.9.

Enumerating subsets of three items from a transaction t .

After fixing the first item, the prefix tree structure at Level 2 represents the number of ways to select the second item. For example, $1\ 2\ 3\ 5\ 6$ corresponds to itemsets that begin with the prefix $\{1\ 2\}$ and are followed by the items 3, 5, or 6. Finally, the prefix tree structure at Level 3 represents the complete set of 3-itemsets contained in t . For example, the 3-itemsets that begin with prefix $\{1\ 2\}$ are $\{1, 2, 3\}$, $\{1, 2, 5\}$, and $\{1, 2, 6\}$, while those that begin with prefix $\{2\ 3\}$ are $\{2, 3, 5\}$ and $\{2, 3, 6\}$.

The prefix tree structure shown in [Figure 5.9](#) demonstrates how itemsets contained in a transaction can be systematically enumerated, i.e., by specifying their items one by one, from the leftmost item to the rightmost item. We still have to determine whether each enumerated 3-itemset corresponds

to an existing candidate itemset. If it matches one of the candidates, then the support count of the corresponding candidate is incremented. In the next Section, we illustrate how this matching operation can be performed efficiently using a hash tree structure.

*Support Counting Using a Hash Tree**

In the *Apriori* algorithm, candidate itemsets are partitioned into different buckets and stored in a hash tree. During support counting, itemsets contained in each transaction are also hashed into their appropriate buckets. That way, instead of comparing each itemset in the transaction with every candidate itemset, it is matched only against candidate itemsets that belong to the same bucket, as shown in [Figure 5.10](#).

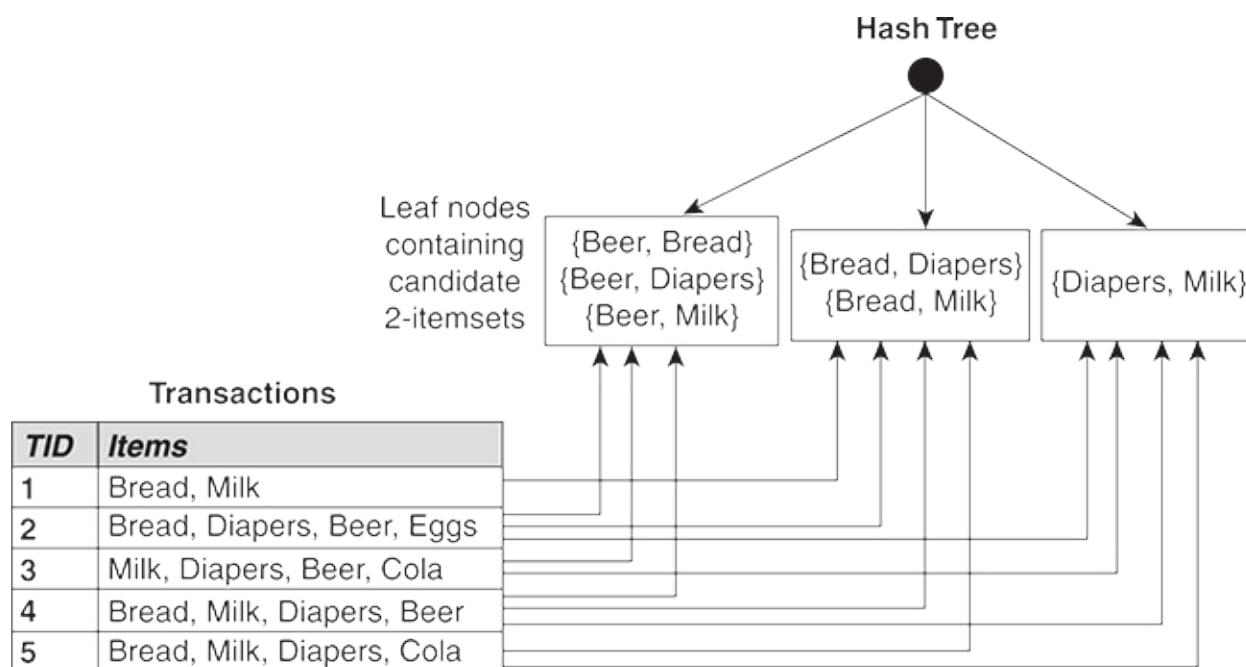


Figure 5.10.

Counting the support of itemsets using hash structure.

[Figure 5.11](#) shows an example of a hash tree structure. Each internal node of the tree uses the following hash function, $h(p) = (p-1) \bmod 3$, where mode

refers to the modulo (remainder) operator, to determine which branch of the current node should be followed next. For example, items 1, 4, and 7 are hashed to the same branch (i.e., the leftmost branch) because they have the same remainder after dividing the number by 3. All candidate itemsets are stored at the leaf nodes of the hash tree. The hash tree shown in [Figure 5.11](#) contains 15 candidate 3-itemsets, distributed across 9 leaf nodes.

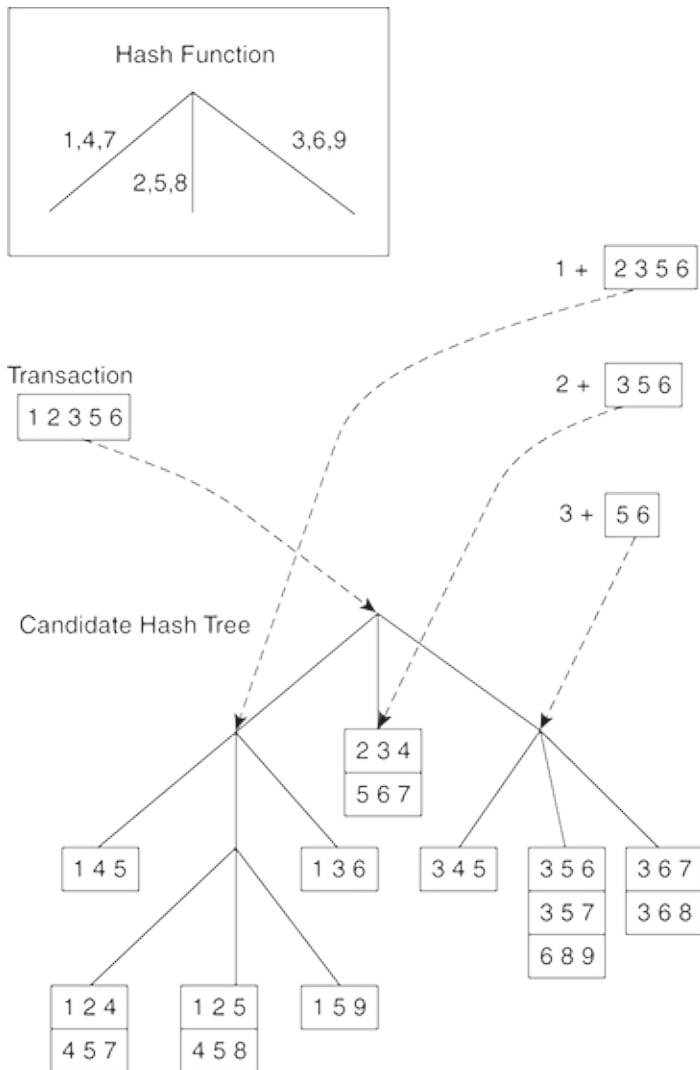


Figure 5.11.

Hashing a transaction at the root node of a hash tree.

Consider the transaction, $t = \{1, 2, 3, 4, 5, 6\}$. To update the support counts of the candidate itemsets, the hash tree must be traversed in such a way that all

the leaf nodes containing candidate 3-itemsets belonging to t must be visited at least once. Recall that the 3-itemsets contained in t must begin with items 1, 2, or 3, as indicated by the Level 1 prefix tree structure shown in [Figure 5.9](#). Therefore, at the root node of the hash tree, the items 1, 2, and 3 of the transaction are hashed separately. Item 1 is hashed to the left child of the root node, item 2 is hashed to the middle child, and item 3 is hashed to the right child. At the next level of the tree, the transaction is hashed on the second item listed in the Level 2 tree structure shown in [Figure 5.9](#). For example, after hashing on item 1 at the root node, items 2, 3, and 5 of the transaction are hashed. Based on the hash function, items 2 and 5 are hashed to the middle child, while item 3 is hashed to the right child, as shown in [Figure 5.12](#). This process continues until the leaf nodes of the hash tree are reached. The candidate itemsets stored at the visited leaf nodes are compared against the transaction. If a candidate is a subset of the transaction, its support count is incremented. Note that not all the leaf nodes are visited while traversing the hash tree, which helps in reducing the computational cost. In this example, 5 out of the 9 leaf nodes are visited and 9 out of the 15 itemsets are compared against the transaction.

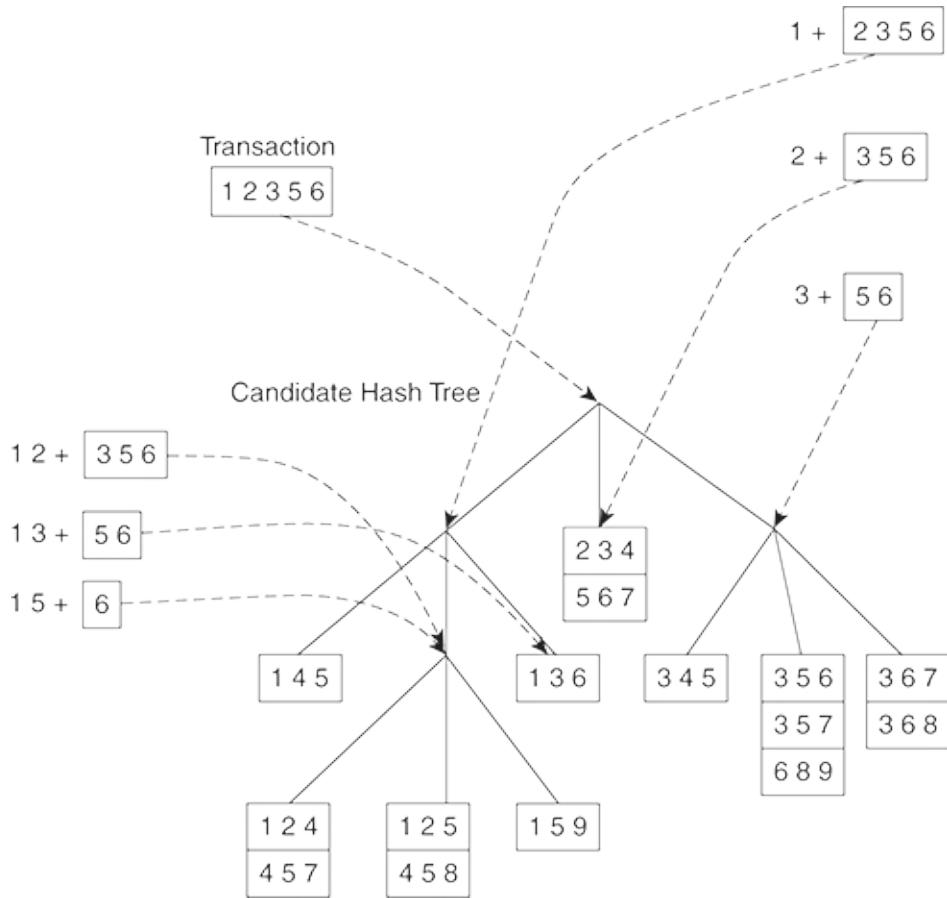


Figure 5.12.

Subset operation on the leftmost subtree of the root of a candidate hash tree.

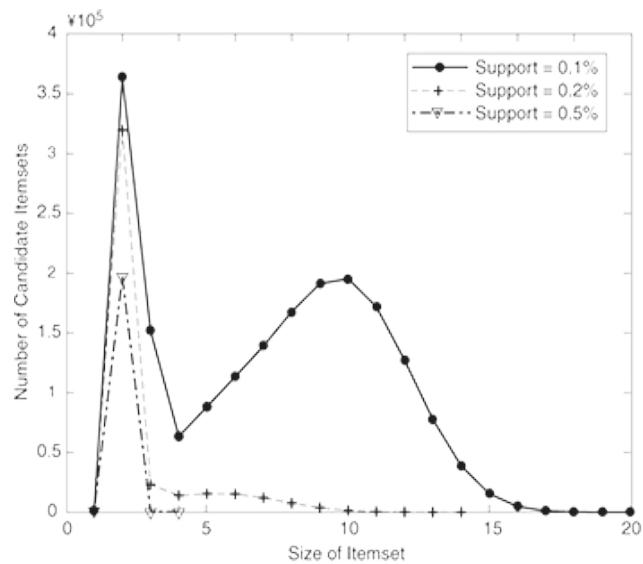
5.2.5 Computational Complexity

The computational complexity of the *Apriori* algorithm, which includes both its runtime and storage, can be affected by the following factors.

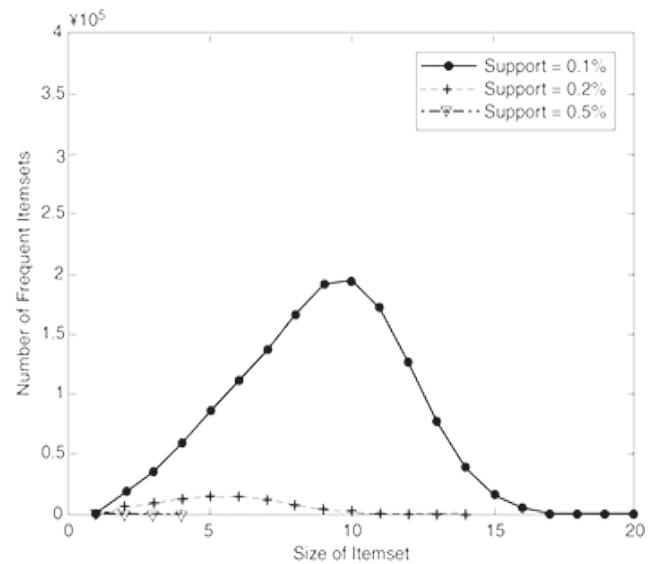
Support Threshold

Lowering the support threshold often results in more itemsets being declared as frequent. This has an adverse effect on the computational complexity of the algorithm because more candidate itemsets must be generated and counted

at every level, as shown in **Figure 5.13**. The maximum size of frequent itemsets also tends to increase with lower support thresholds. This increases the total number of iterations to be performed by the *Apriori* algorithm, further increasing the computational cost.



(a) Number of candidate itemsets.



(b) Number of frequent itemsets.

Figure 5.13.

Effect of support threshold on the number of candidate and frequent itemsets obtained from a benchmark data set.

Number of Items (Dimensionality)

As the number of items increases, more space will be needed to store the support counts of items. If the number of frequent items also grows with the dimensionality of the data, the runtime and storage requirements will increase because of the larger number of candidate itemsets generated by the algorithm.

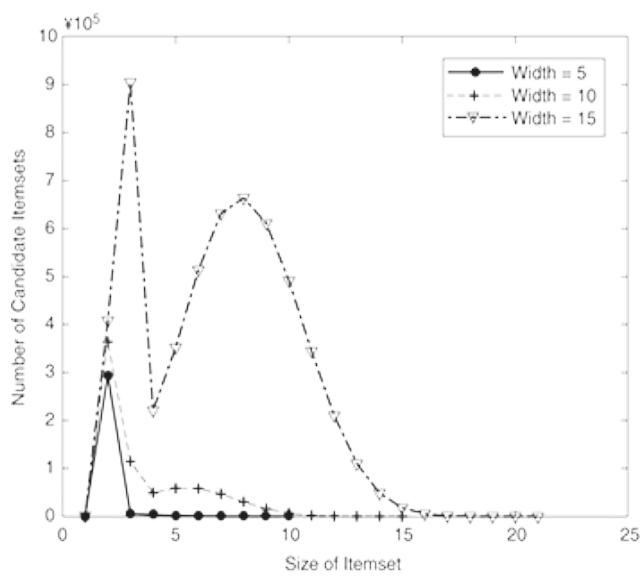
Number of Transactions

Because the *Apriori* algorithm makes repeated passes over the transaction data set, its run time increases with a larger number of transactions.

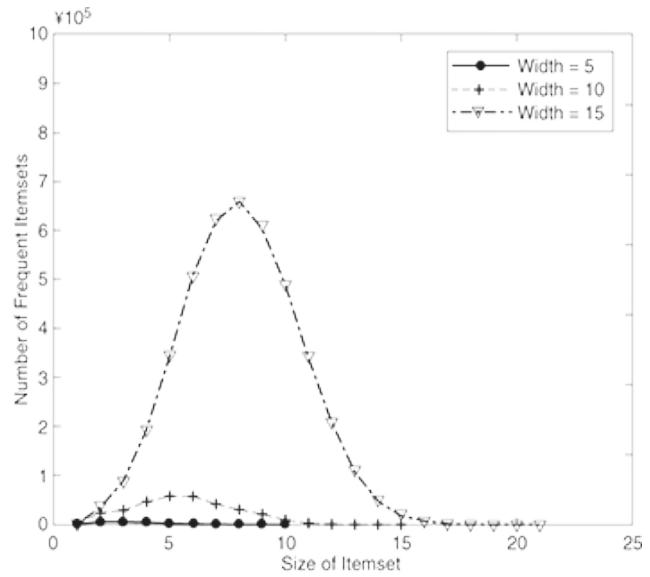
Average Transaction Width

For dense data sets, the average transaction width can be very large. This affects the complexity of the *Apriori* algorithm in two ways. First, the maximum size of frequent itemsets tends to increase as the average transaction width increases. As a result, more candidate itemsets must be examined during candidate generation and support counting, as illustrated in [Figure 5.14](#). Second, as the transaction width increases, more itemsets are contained in the transaction. This will increase the number of hash tree traversals performed during support counting.

A detailed analysis of the time complexity for the *Apriori* algorithm is presented next.



(a) Number of candidate itemsets.



(b) Number of Frequent Itemsets.

Figure 5.14.

Effect of average transaction width on the number of candidate and frequent itemsets obtained from a synthetic data set.

Generation of frequent 1-itemsets

For each transaction, we need to update the support count for every item present in the transaction. Assuming that w is the average transaction width, this operation requires $O(Nw)$ time, where N is the total number of transactions.

Candidate generation

To generate candidate k -itemsets, pairs of frequent $(k-1)$ -itemsets are merged to determine whether they have at least $k-2$ items in common. Each merging operation requires at most $k-2$ equality comparisons. Every merging step can produce at most one viable candidate k -itemset, while in the worst-case, the algorithm must try to merge every pair of frequent $(k-1)$ -itemsets found in the previous iteration. Therefore, the overall cost of merging frequent itemsets is

$$\sum_{k=2}^w (k-2) |C_k| < \text{Cost of merging} < \sum_{k=2}^w (k-2) |F_{k-1}|^2,$$

where w is the maximum transaction width. A hash tree is also constructed during candidate generation to store the candidate itemsets. Because the maximum depth of the tree is k , the cost for populating the hash tree with candidate itemsets is $O(\sum_{k=2}^w k |C_k|)$. During candidate pruning, we need to verify that the $k-2$ subsets of every candidate k -itemset are frequent. Since the cost for looking up a candidate in a hash tree is $O(k)$, the candidate pruning step requires $O(\sum_{k=2}^w k (k-2) |C_k|)$ time.

Support counting

Each transaction of width $|t|$ produces $(|t|k)$ itemsets of size k . This is also the effective number of hash tree traversals performed for each transaction. The cost for support counting is $O(N\sum k(wk)\alpha_k)$, where w is the maximum transaction width and α_k is the cost for updating the support count of a candidate k -itemset in the hash tree.

5.3 Rule Generation

This Section describes how to extract association rules efficiently from a given frequent itemset. Each frequent k -itemset, Y , can produce up to $2k-2$ association rules, ignoring rules that have empty antecedents or consequents ($\emptyset \rightarrow Y$ or $Y \rightarrow \emptyset$). An association rule can be extracted by partitioning the itemset Y into two non-empty subsets, X and $Y-X$, such that $X \rightarrow Y-X$ satisfies the confidence threshold. Note that all such rules must have already met the support threshold because they are generated from a frequent itemset.

Example 5.2.

Let $X=\{a, b, c\}$ be a frequent itemset. There are six candidate association rules that can be generated from X : $\{a, b\} \rightarrow \{c\}$, $\{a, c\} \rightarrow \{b\}$, $\{b, c\} \rightarrow \{a\}$, $\{a\} \rightarrow \{b, c\}$, $\{b\} \rightarrow \{a, c\}$, and $\{c\} \rightarrow \{a, b\}$. As each of their support is identical to the support for X , all the rules satisfy the support threshold.

Computing the confidence of an association rule does not require additional scans of the transaction data set. Consider the rule $\{1, 2\} \rightarrow \{3\}$, which is generated from the frequent itemset $X=\{1, 2, 3\}$. The confidence for this rule is $\sigma(\{1, 2, 3\})/\sigma(\{1, 2\})$. Because $\{1, 2, 3\}$ is frequent, the anti-monotone property of support ensures that $\{1, 2\}$ must be frequent, too. Since the support counts for both itemsets were already found during frequent itemset generation, there is no need to read the entire data set again.

5.3.1 Confidence-Based Pruning

Confidence does not show the anti-monotone property in the same way as the support measure. For example, the confidence for $X \rightarrow Y$ can be larger, smaller, or equal to the confidence for another rule $X' \rightarrow Y'$, where $X' \subseteq X$ and $Y' \subseteq Y$ (see Exercise 3 on page 439). Nevertheless, if we compare rules generated from the same frequent itemset Y , the following theorem holds for the confidence measure.

Theorem 5.2.

Let Y be an itemset and X is a subset of Y . If a rule $X \rightarrow Y - X$ does not satisfy the confidence threshold, then any rule $X' \rightarrow Y - X'$, where X' is a subset of X , must not satisfy the confidence threshold as well.

To prove this theorem, consider the following two rules: $X' \rightarrow Y - X'$ and $X \rightarrow Y - X$, where $X' \subseteq X$. The confidence of the rules are $\sigma(Y)/\sigma(X')$ and $\sigma(Y)/\sigma(X)$, respectively. Since X' is a subset of X , $\sigma(X')/\sigma(X)$. Therefore, the former rule cannot have a higher confidence than the latter rule.

5.3.2 Rule Generation in *Apriori* Algorithm

The *Apriori* algorithm uses a level-wise approach for generating association rules, where each level corresponds to the number of items that belong to the

rule consequent. Initially, all the high confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules. For example, if $\{acd\} \rightarrow \{b\}$ and $\{abd\} \rightarrow \{c\}$ are high confidence rules, then the candidate rule $\{ad\} \rightarrow \{bc\}$ is generated by merging the consequents of both rules. **Figure 5.15** shows a lattice structure for the association rules generated from the frequent itemset $\{a, b, c, d\}$. If any node in the lattice has low confidence, then according to **Theorem 5.2**, the entire subgraph spanned by the node can be pruned immediately. Suppose the confidence for $\{bcd\} \rightarrow \{a\}$ is low. All the rules containing item a in its consequent, including $\{cd\} \rightarrow \{ab\}$, $\{bd\} \rightarrow \{ac\}$, $\{bc\} \rightarrow \{ad\}$, and $\{d\} \rightarrow \{abc\}$ can be discarded.

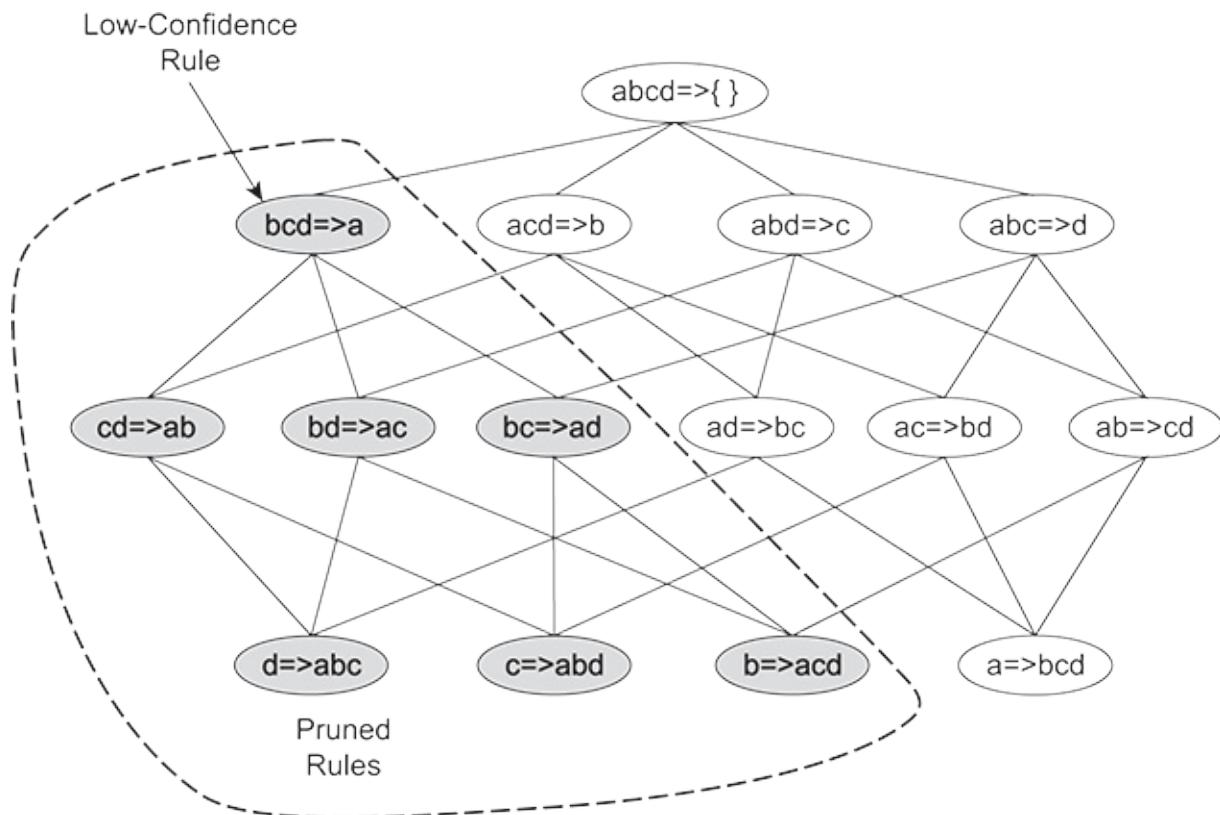


Figure 5.15.

Pruning of association rules using the confidence measure.

A pseudocode for the rule generation step is shown in [Algorithms 5.2](#) and [5.3](#). Note the similarity between the `ap-genrules` procedure given in [Algorithm 5.3](#) and the frequent itemset generation procedure given in [Algorithm 5.1](#). The only difference is that, in rule generation, we do not have to make additional passes over the data set to compute the confidence of the candidate rules. Instead, we determine the confidence of each rule by using the support counts computed during frequent itemset generation.

Algorithm 5.2 Rule generation of the Apriori algorithm.

```

1: for each frequent  $k$ -itemset  $f_k$ ,  $k \geq 2$  do
2:    $H_1 = \{i \mid i \in f_k\}$  {1-item consequents of the rule.}
3:   call ap-genrules( $f_k$ ,  $H_1$ .)
4: end for

```

Algorithm 5.3 Procedure ap-genrules (fk , Hm).

```

1:  $k = |f_k|$  {size of frequent itemset.}
2:  $m = |H_m|$  {size of rule consequent.}
3: if  $k > m + 1$  then
4:    $H_{m+1} = \text{candidate-gen}(H_m)$  .
5:    $H_{m+1} = \text{candidate-prune}(H_{m+1}, H_m)$  .
6:   for each  $h_{m+1} \in H_{m+1}$  do
7:      $\text{conf} = \sigma(f_k) / \sigma(f_k - h_{m+1})$  .
8:     if  $\text{conf} \geq \text{minconf}$  then
9:       output the rule  $(f_k - h_{m+1}) \rightarrow h_{m+1}$ .
10:    else

```

```

11:           delete  $h_{m+1}$  from  $H_{m+1}$ .
12:       end if
13:   end for
14:   call ap-genrules( $f_k$ ,  $H_{m+1}$ .)
15: end if

```

5.3.3 An Example: Congressional Voting Records

This Section demonstrates the results of applying association analysis to the voting records of members of the United States House of Representatives. The data is obtained from the 1984 Congressional Voting Records Database, which is available at the UCI machine learning data repository. Each transaction contains information about the party affiliation for a representative along with his or her voting record on 16 key issues. There are 435 transactions and 34 items in the data set. The set of items are listed in [Table 5.3](#).

Table 5.3. List of binary attributes from the 1984 United States Congressional Voting Records. Source: The UCI machine learning repository.

1. Republican
2. Democrat
3. handicapped-infants=yes
4. handicapped-infants=no
5. water project cost sharing=yes
6. water project cost sharing=no
7. budget-resolution=yes

8. budget-resolution=no
9. physician fee freeze=yes
10. physician fee freeze=no
11. aid to El Salvador=yes
12. aid to El Salvador=no
13. religious groups in schools=yes
14. religious groups in schools=no
15. anti-satellite test ban=yes
16. anti-satellite test ban=no
17. aid to Nicaragua=yes
18. aid to Nicaragua=no
19. MX-missile=yes
20. MX-missile=no
21. immigration=yes
22. immigration=no
23. synfuel corporation cutback=yes
24. synfuel corporation cutback=no
25. education spending=yes
26. education spending=no
27. right-to-sue=yes
28. right-to-sue=no
29. crime=yes
30. crime=no
31. duty-free-exports=yes
32. duty-free-exports=no
33. export administration act=yes
34. export administration act=no

The *Apriori* algorithm is then applied to the data set with $\text{minsup}=30\%$ and $\text{minconf}=90\%$. Some of the high confidence rules extracted by the algorithm are shown in [Table 5.4](#). The first two rules suggest that most of the members who voted yes for aid to El Salvador and no for budget resolution and MX missile are Republicans; while those who voted no for aid to El Salvador and yes for budget resolution and MX missile are Democrats. These

high confidence rules show the key issues that divide members from both political parties.

Table 5.4. Association rules extracted from the 1984 United States Congressional Voting Records.

Association Rule	Confidence
{budget resolution=no, MX-missile=no, aid to El Salvador=yes }→{Republican}	91.0%
{budget resolution=yes, MX-missile=yes, aid to El Salvador=no }→{Democrat}	97.5%
{crime=yes, right-to-sue=yes, physician fee freeze=yes }→{Republican}	93.5%
{crime=no, right-to-sue=no, physician fee freeze=no }→{Democrat}	100%

5.4 Compact Representation of Frequent Itemsets

In practice, the number of frequent itemsets produced from a transaction data set can be very large. It is useful to identify a small representative set of frequent itemsets from which all other frequent itemsets can be derived. Two such representations are presented in this Section in the form of maximal and closed frequent itemsets.

5.4.1 Maximal Frequent Itemsets

Definition 5.3. (Maximal Frequent Itemset.)

A frequent itemset is maximal if none of its immediate supersets are frequent.

To illustrate this concept, consider the itemset lattice shown in [Figure 5.16](#). The itemsets in the lattice are divided into two groups: those that are frequent and those that are infrequent. A frequent itemset border, which is represented by a dashed line, is also illustrated in the diagram. Every itemset located above the border is frequent, while those located below the border (the shaded nodes) are infrequent. Among the itemsets residing near the border,

$\{a, d\}$, $\{a, c, e\}$, and $\{b, c, d, e\}$ are maximal frequent itemsets because all of their immediate supersets are infrequent. For example, the itemset $\{a, d\}$ is maximal frequent because all of its immediate supersets, $\{a, b, d\}$, $\{a, c, d\}$, and $\{a, d, e\}$, are infrequent. In contrast, $\{a, c\}$ is non-maximal because one of its immediate supersets, $\{a, c, e\}$, is frequent.

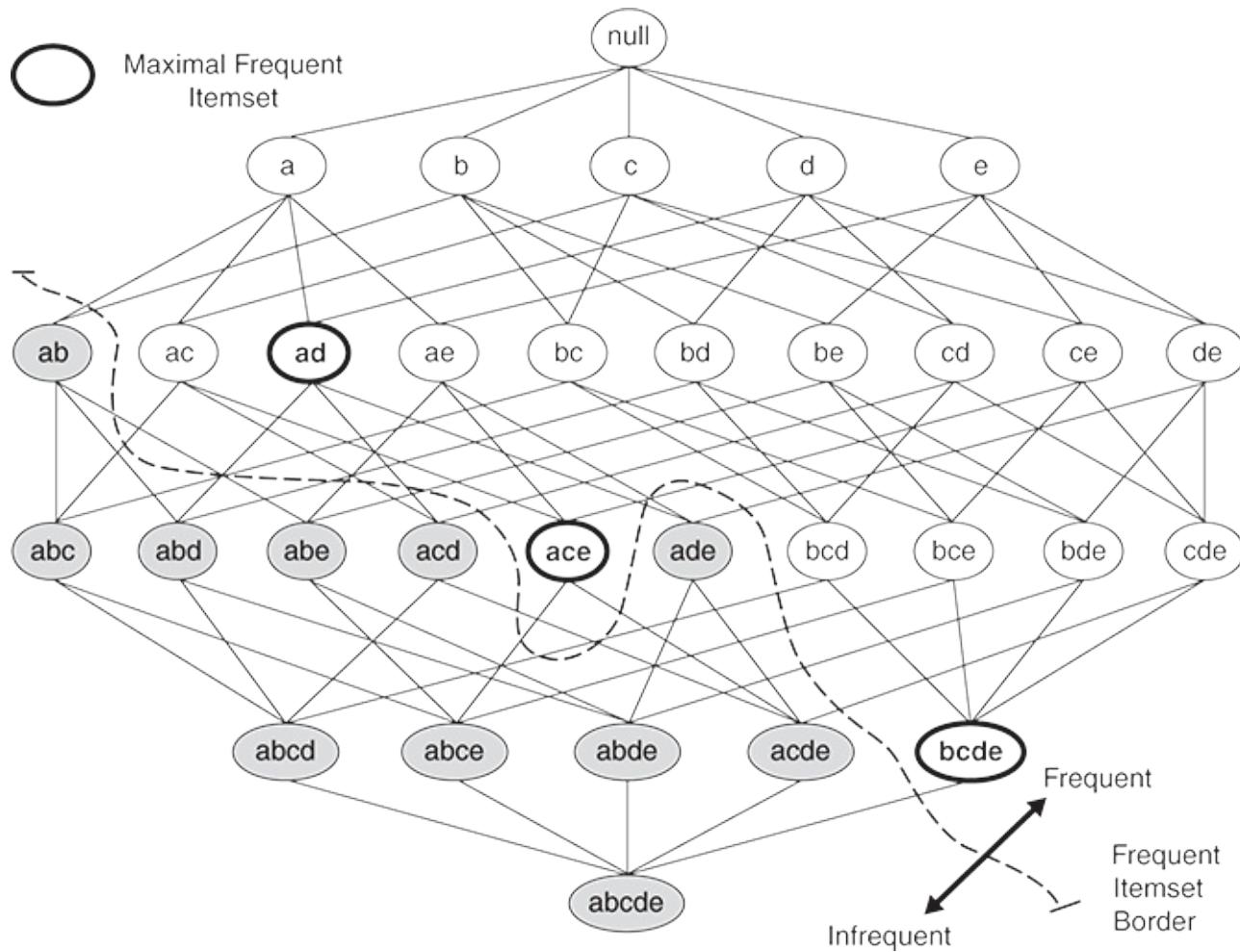


Figure 5.16.
Maximal frequent itemset.

Maximal frequent itemsets effectively provide a compact representation of frequent itemsets. In other words, they form the smallest set of itemsets from which all frequent itemsets can be derived. For example, every frequent itemset in [Figure 5.16](#) is a subset of one of the three maximal frequent

itemsets, $\{a, d\}$, $\{a, c, e\}$, and $\{b, c, d, e\}$. If an itemset is not a proper subset of any of the maximal frequent itemsets, then it is either infrequent (e.g., $\{a, d, e\}$) or maximal frequent itself (e.g., $\{b, c, d, e\}$). Hence, the maximal frequent itemsets $\{a, c, e\}$, $\{a, d\}$, and $\{b, c, d, e\}$ provide a compact representation of the frequent itemsets shown in [Figure 5.16](#). Enumerating all the subsets of maximal frequent itemsets generates the complete list of all frequent itemsets.

Maximal frequent itemsets provide a valuable representation for data sets that can produce very long, frequent itemsets, as there are exponentially many frequent itemsets in such data. Nevertheless, this approach is practical only if an efficient algorithm exists to explicitly find the maximal frequent itemsets. We briefly describe one such approach in [Section 5.5](#).

Despite providing a compact representation, maximal frequent itemsets do not contain the support information of their subsets. For example, the support of the maximal frequent itemsets $\{a, c, e\}$, $\{a, d\}$, and $\{b, c, d, e\}$ do not provide any information about the support of their subsets except that it meets the support threshold. An additional pass over the data set is therefore needed to determine the support counts of the non-maximal frequent itemsets. In some cases, it is desirable to have a minimal representation of itemsets that preserves the support information. We describe such a representation in the next Section.

5.4.2 Closed Itemsets

Closed itemsets provide a minimal representation of all itemsets without losing their support information. A formal definition of a closed itemset is presented below.

Definition 5.4. (Closed Itemset.)

An itemset X is closed if none of its immediate supersets has exactly the same support count as X .

Put another way, X is not closed if at least one of its immediate supersets has the same support count as X . Examples of closed itemsets are shown in

[Figure 5.17](#). To better illustrate the support count of each itemset, we have associated each node (itemset) in the lattice with a list of its corresponding transaction IDs. For example, since the node $\{b, c\}$ is associated with transaction IDs 1, 2, and 3, its support count is equal to three. From the transactions given in this diagram, notice that the support for $\{b\}$ is identical to $\{b, c\}$. This is because every transaction that contains b also contains c . Hence, $\{b\}$ is not a closed itemset. Similarly, since c occurs in every transaction that contains both a and d , the itemset $\{a, d\}$ is not closed as it has the same support as its superset $\{a, c, d\}$. On the other hand, $\{b, c\}$ is a closed itemset because it does not have the same support count as any of its supersets.

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

minsup = 40%

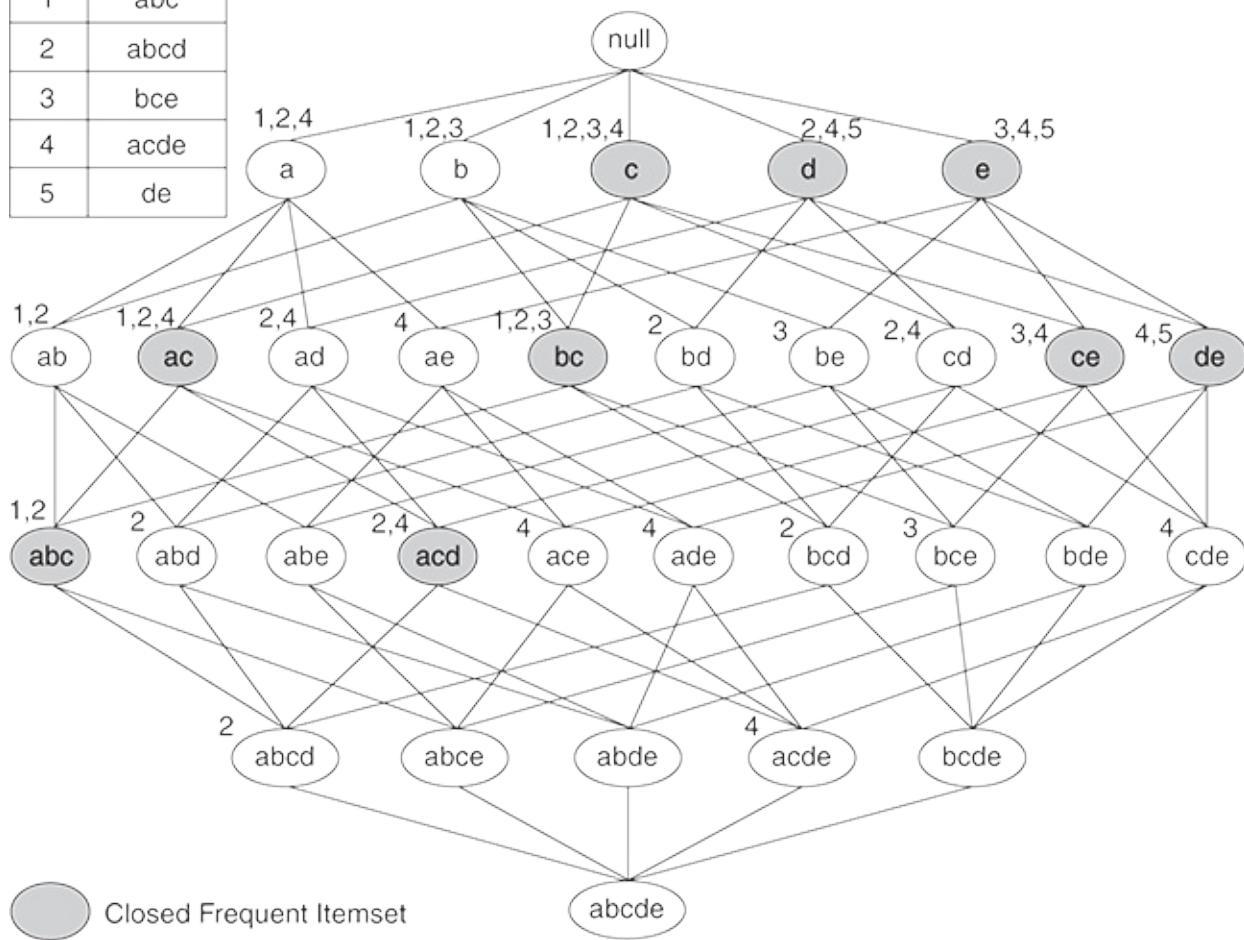


Figure 5.17.

An example of the closed frequent itemsets (with minimum support equal to 40%).

An interesting property of closed itemsets is that if we know their support counts, we can derive the support count of every other itemset in the itemset lattice without making additional passes over the data set. For example, consider the 2-itemset $\{b, e\}$ in [Figure 5.17](#). Since $\{b, e\}$ is not closed, its support must be equal to the support of one of its immediate supersets, $\{a, b, e\}$, $\{b, c, e\}$, and $\{b, d, e\}$. Further, none of the supersets of $\{b, e\}$ can have a support greater than the support of $\{b, e\}$, due to the anti-monotone nature of the support measure. Hence, the support of $\{b, e\}$ can be computed by examining the support counts of all of its immediate supersets of size three

and taking their maximum value. If an immediate superset is closed (e.g., $\{b, c, e\}$), we would know its support count. Otherwise, we can recursively compute its support by examining the supports of its immediate supersets of size four. In general, the support count of any non-closed $(k-1)$ -itemset can be determined as long as we know the support counts of all k -itemsets. Hence, one can devise an iterative algorithm that computes the support counts of itemsets at level $k-1$ using the support counts of itemsets at level k , starting from the level k_{\max} , where k_{\max} is the size of the largest closed itemset.

Even though closed itemsets provide a compact representation of the support counts of all itemsets, they can still be exponentially large in number. Moreover, for most practical applications, we only need to determine the support count of all frequent itemsets. In this regard, closed frequent item-sets provide a compact representation of the support counts of all frequent itemsets, which can be defined as follows.

Definition 5.5. (Closed Frequent Itemset.)

An itemset is a closed frequent itemset if it is closed and its support is greater than or equal to minsup .

In the previous example, assuming that the support threshold is 40%, $\{b, c\}$ is a closed frequent itemset because its support is 60%. In [Figure 5.17](#), the closed frequent itemsets are indicated by the shaded nodes.

Algorithms are available to explicitly extract closed frequent itemsets from a given data set. Interested readers may refer to the Bibliographic Notes at the

end of this chapter for further discussions of these algorithms. We can use closed frequent itemsets to determine the support counts for all non-closed frequent itemsets. For example, consider the frequent itemset $\{a, d\}$ shown in [Figure 5.17](#). Because this itemset is not closed, its support count must be equal to the maximum support count of its immediate supersets, $\{a, b, d\}$, $\{a, c, d\}$, and $\{a, d, e\}$. Also, since $\{a, d\}$ is frequent, we only need to consider the support of its frequent supersets. In general, the support count of every non-closed frequent k -itemset can be obtained by considering the support of all its frequent supersets of size $k+1$. For example, since the only frequent superset of $\{a, d\}$ is $\{a, c, d\}$, its support is equal to the support of $\{a, c, d\}$, which is 2. Using this methodology, an algorithm can be developed to compute the support for every frequent itemset. The pseudocode for this algorithm is shown in [Algorithm 5.4](#). The algorithm proceeds in a specific-to-general fashion, i.e., from the largest to the smallest frequent itemsets. This is because, in order to find the support for a non-closed frequent itemset, the support for all of its supersets must be known. Note that the set of all frequent itemsets can be easily computed by taking the union of all subsets of frequent closed itemsets.

Algorithm 5.4 Support counting using closed frequent itemsets.

```

1: Let  $C$  denote the set of closed frequent itemsets and  $F$  denote
the set of all frequent itemsets.

2: Let  $k_{\max}$  denote the maximum size of closed frequent itemsets

3:  $F_{k_{\max}} = \{f | f \in C, |f| = k_{\max}\}$  {Find all frequent itemsets of size
 $k_{\max}\}$ 

4: for  $k = k_{\max} - 1$  down to 1 do

5:    $F_k = \{f | f \in F, |f| = k\}$  {Find all frequent itemsets of size  $k\}$ 

```

```

6:   for each  $f \in F_k$  do
7:     if  $f \notin C$  then
8:        $f \cdot \text{support} = \max\{f' \cdot \text{support} \mid f' \in F_{k+1}, f \subset f'\}$ 
9:     end if
10:    end for
11:  end for

```

To illustrate the advantage of using closed frequent itemsets, consider the data set shown in **Table 5.5**, which contains ten transactions and fifteen items. The items can be divided into three groups: (1) Group *A*, which contains items *a*1 through *a*5; (2) Group *B*, which contains items *b*1 through *b*5; and (3) Group *C*, which contains items *c*1 through *c*5. Assuming that the support threshold is 20%, itemsets involving items from the same group are frequent, but itemsets involving items from different groups are infrequent. The total number of frequent itemsets is thus $3 \times (25 - 1) = 93$. However, there are only four closed frequent itemsets in the data:

($\{a_3, a_4\}$, $\{a_1, a_2, a_3, a_4, a_5\}$, $\{b_1, b_2, b_3, b_4, b_5\}$, and $\{c_1, c_2, c_3, c_4, c_5\}$). It is often sufficient to present only the closed frequent itemsets to the analysts instead of the entire set of frequent itemsets.

Table 5.5. A transaction data set for mining closed itemsets.

TID	a1	a2	a3	a4	a5	b1	b2	b3	b4	b5	c1	c2	c3	c4	c5
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0
5	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0

6	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Finally, note that all maximal frequent itemsets are closed because none of the maximal frequent itemsets can have the same support count as their immediate supersets. The relationships among frequent, closed, closed frequent, and maximal frequent itemsets are shown in [Figure 5.18](#).

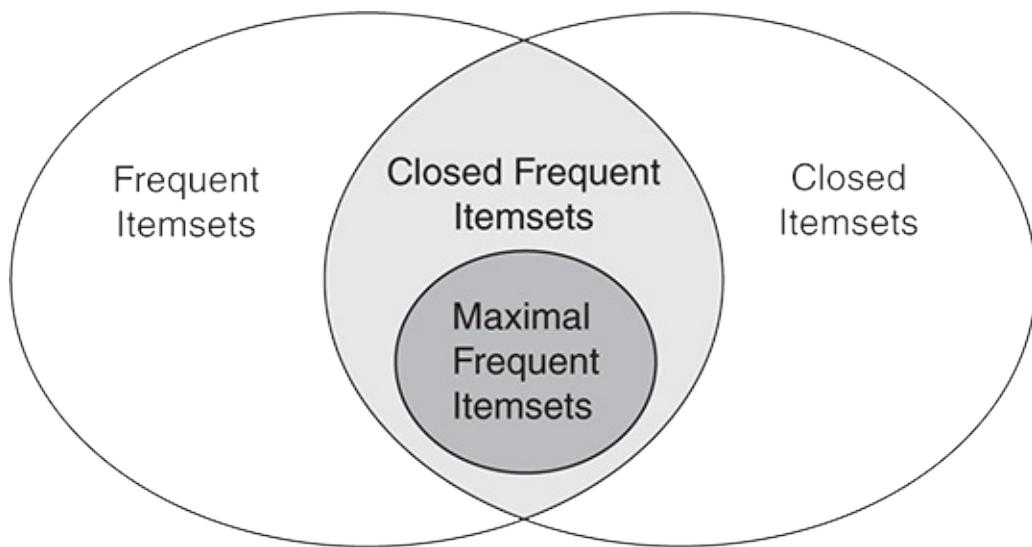


Figure 5.18.

Relationships among frequent, closed, closed frequent, and maximal frequent itemsets.

5.5 Alternative Methods for Generating Frequent Itemsets*

Apriori is one of the earliest algorithms to have successfully addressed the combinatorial explosion of frequent itemset generation. It achieves this by applying the *Apriori* principle to prune the exponential search space. Despite its significant performance improvement, the algorithm still incurs considerable I/O overhead since it requires making several passes over the transaction data set. In addition, as noted in [Section 5.2.5](#), the performance of the *Apriori* algorithm may degrade significantly for dense data sets because of the increasing width of transactions. Several alternative methods have been developed to overcome these limitations and improve upon the efficiency of the *Apriori* algorithm. The following is a high-level description of these methods.

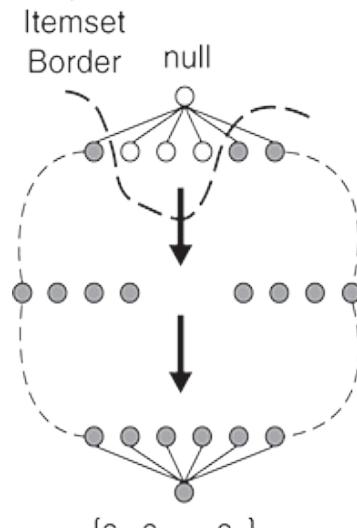
Traversal of Itemset Lattice

A search for frequent itemsets can be conceptually viewed as a traversal on the itemset lattice shown in [Figure 5.1](#). The search strategy employed by an algorithm dictates how the lattice structure is traversed during the frequent itemset generation process. Some search strategies are better than others, depending on the configuration of frequent itemsets in the lattice. An overview of these strategies is presented next.

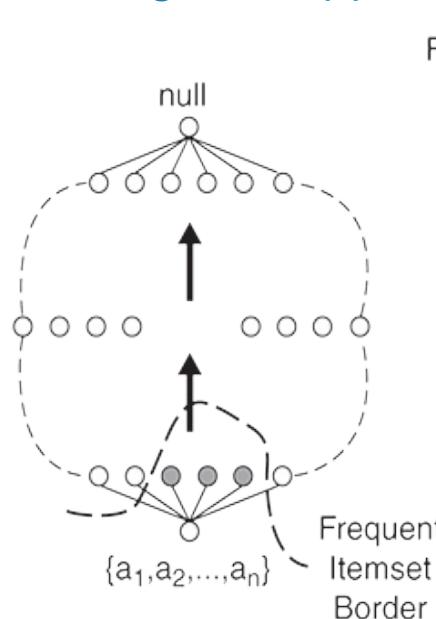
- **General-to-Specific versus Specific-to-General:** The *Apriori* algorithm uses a general-to-specific search strategy, where pairs of frequent $(k-1)$ -itemsets are merged to obtain candidate k -itemsets. This general-to-

specific search strategy is effective, provided the maximum length of a frequent itemset is not too long. The configuration of frequent itemsets that works best with this strategy is shown in [Figure 5.19\(a\)](#), where the darker nodes represent infrequent itemsets. Alternatively, a specific-to-general search strategy looks for more specific frequent itemsets first, before finding the more general frequent itemsets. This strategy is useful to discover maximal frequent itemsets in dense transactions, where the frequent itemset border is located near the bottom of the lattice, as shown in [Figure 5.19\(b\)](#). The *Apriori* principle can be applied to prune all subsets of maximal frequent itemsets. Specifically, if a candidate k -itemset is maximal frequent, we do not have to examine any of its subsets of size $k - 1$. However, if the candidate k -itemset is infrequent, we need to check all of its $k - 1$ subsets in the next iteration. Another approach is to combine both general-to-specific and specific-to-general search strategies. This bidirectional approach requires more space to store the candidate itemsets, but it can help to rapidly identify the frequent itemset border, given the configuration shown in [Figure 5.19\(c\)](#).

Frequent Itemset Border

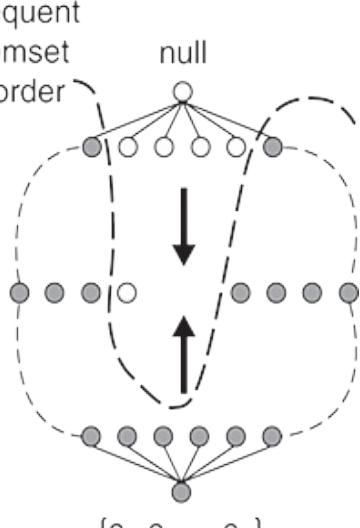


(a) General-to-specific



(b) Specific-to-general

Frequent Itemset Border



(c) Bidirectional

Figure 5.19.

General-to-specific, specific-to-general, and bidirectional search.

- **Equivalence Classes:** Another way to envision the traversal is to first partition the lattice into disjoint groups of nodes (or equivalence classes). A frequent itemset generation algorithm searches for frequent itemsets within a particular equivalence class first before moving to another equivalence class. As an example, the level-wise strategy used in the *Apriori* algorithm can be considered to be partitioning the lattice on the basis of itemset sizes; i.e., the algorithm discovers all frequent 1-itemsets first before proceeding to larger-sized itemsets. Equivalence classes can also be defined according to the prefix or suffix labels of an itemset. In this case, two itemsets belong to the same equivalence class if they share a common prefix or suffix of length k . In the prefix-based approach, the algorithm can search for frequent itemsets starting with the prefix a before looking for those starting with prefixes b , c , and so on. Both prefix-based and suffix-based equivalence classes can be demonstrated using the tree-like structure shown in [Figure 5.20](#).

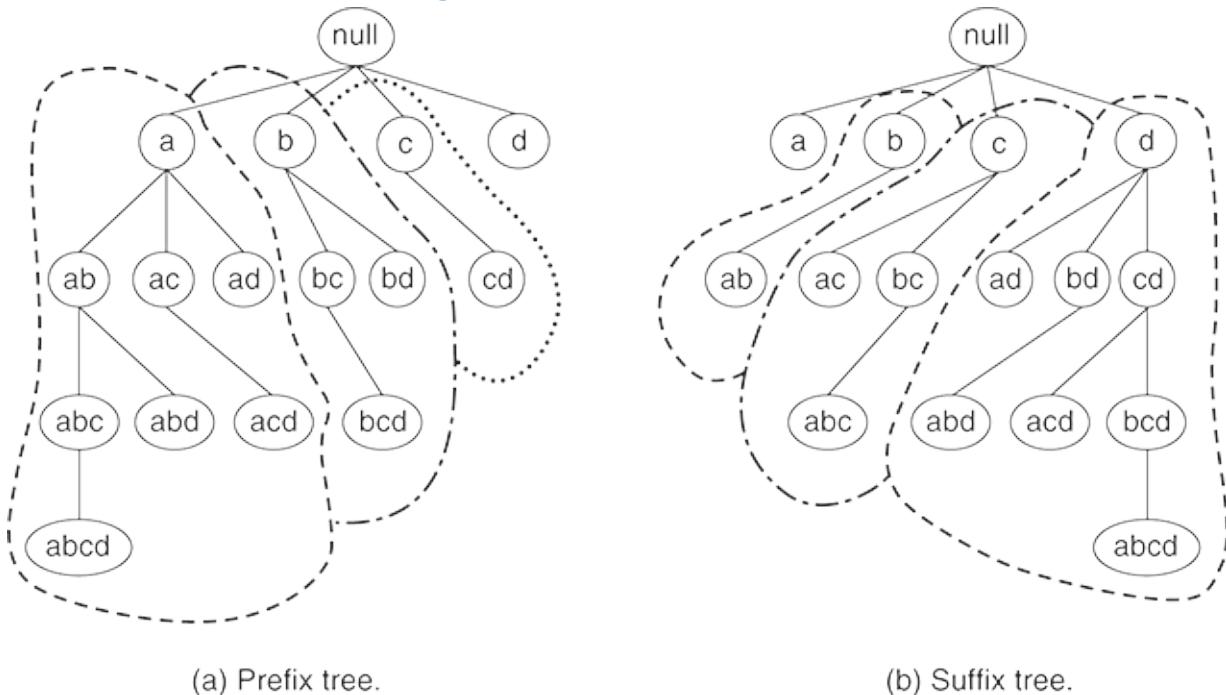


Figure 5.20.

Equivalence classes based on the prefix and suffix labels of itemsets.

- **Breadth-First versus Depth-First:** The *Apriori* algorithm traverses the lattice in a breadth-first manner, as shown in [Figure 5.21\(a\)](#). It first discovers all the frequent 1-itemsets, followed by the frequent 2-itemsets, and so on, until no new frequent itemsets are generated. The itemset lattice can also be traversed in a depth-first manner, as shown in [Figures 5.21\(b\)](#) and [5.22](#). The algorithm can start from, say, node *a* in [Figure 5.22](#), and count its support to determine whether it is frequent. If so, the algorithm progressively expands the next level of nodes, i.e., *ab*, *abc*, and so on, until an infrequent node is reached, say, *abcd*. It then backtracks to another branch, say, *abce*, and continues the search from there.

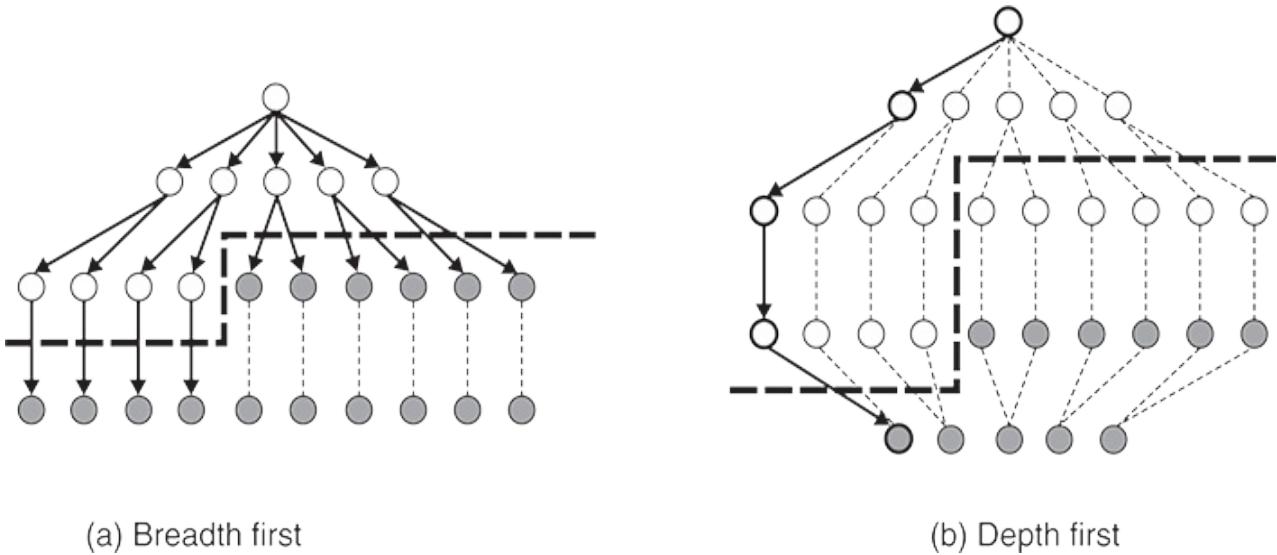


Figure 5.21.

Breadth-first and depth-first traversals.

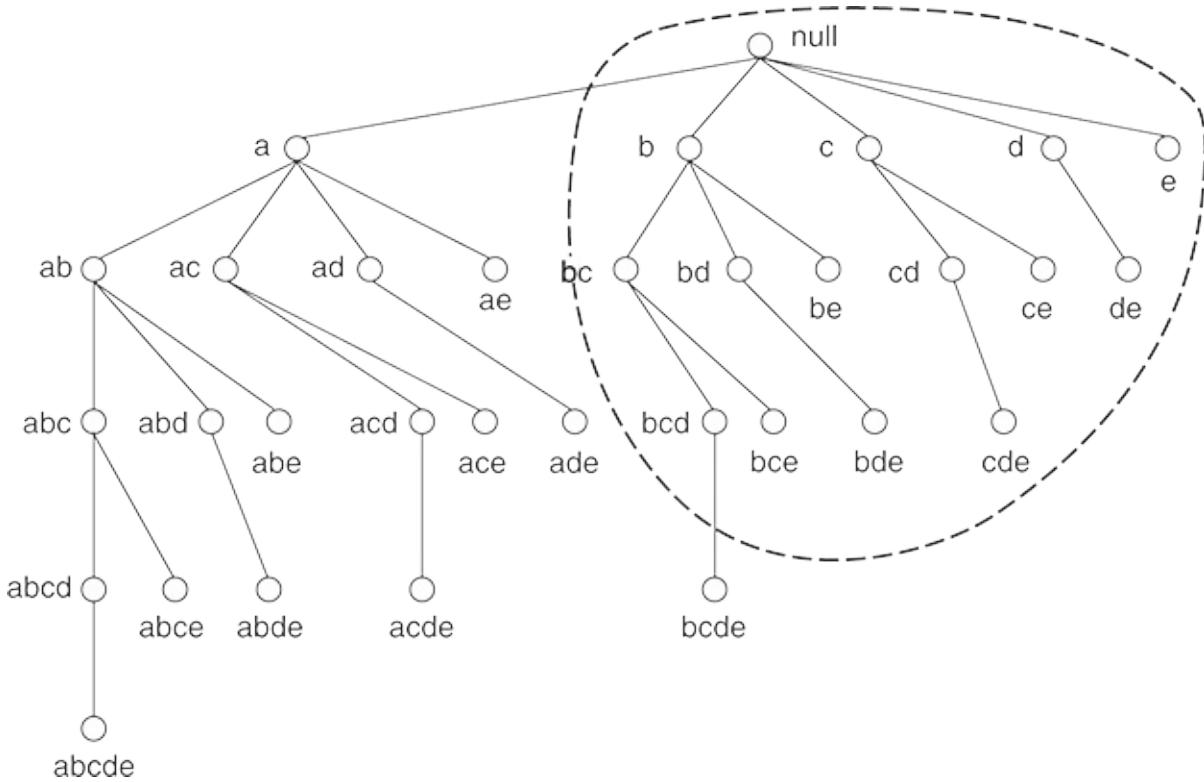


Figure 5.22.

Generating candidate itemsets using the depth-first approach.

The depth-first approach is often used by algorithms designed to find maximal frequent itemsets. This approach allows the frequent itemset border to be detected more quickly than using a breadth-first approach.

Once a maximal frequent itemset is found, substantial pruning can be performed on its subsets. For example, if the node *bcde* shown in [Figure 5.22](#) is maximal frequent, then the algorithm does not have to visit the subtrees rooted at *bd*, *be*, *c*, *d*, and *e* because they will not contain any maximal frequent itemsets. However, if *abc* is maximal frequent, only the nodes such as *ac* and *bc* are not maximal frequent (but the subtrees of *ac* and *bc* may still contain maximal frequent itemsets). The depth-first approach also allows a different kind of pruning based on the support of itemsets. For example, suppose the support for $\{a, b, c\}$ is identical to the support for $\{a, b\}$. The subtrees rooted at *abd* and *abe* can be skipped

because they are guaranteed not to have any maximal frequent itemsets. The proof of this is left as an exercise to the readers.

Representation of Transaction Data Set

There are many ways to represent a transaction data set. The choice of representation can affect the I/O costs incurred when computing the support of candidate itemsets. [Figure 5.23](#) shows two different ways of representing market basket transactions. The representation on the left is called a **horizontal** data layout, which is adopted by many association rule mining algorithms, including *Apriori*. Another possibility is to store the list of transaction identifiers (TID-list) associated with each item. Such a representation is known as the **vertical** data layout. The support for each candidate itemset is obtained by intersecting the TID-lists of its subset items. The length of the TID-lists shrinks as we progress to larger sized itemsets. However, one problem with this approach is that the initial set of TID-lists might be too large to fit into main memory, thus requiring more sophisticated techniques to compress the TID-lists. We describe another effective approach to represent the data in the next Section.

Horizontal
Data Layout

TID	Items
1	a,b,e
2	b,c,d
3	c,e
4	a,c,d
5	a,b,c,d
6	a,e
7	a,b
8	a,b,c
9	a,c,d
10	b

Vertical Data Layout

a	b	c	d	e
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

Figure 5.23.

Horizontal and vertical data format.

Horizontal Data Layout

5.6 FP-Growth Algorithm*

This Section presents an alternative algorithm called **FP-growth** that takes a radically different approach to discovering frequent itemsets. The algorithm does not subscribe to the generate-and-test paradigm of *Apriori*. Instead, it encodes the data set using a compact data structure called an **FP-tree** and extracts frequent itemsets directly from this structure. The details of this approach are presented next.

5.6.1 FP-Tree Representation

An FP-tree is a compressed representation of the input data. It is constructed by reading the data set one transaction at a time and mapping each transaction onto a path in the FP-tree. As different transactions can have several items in common, their paths might overlap. The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure. If the size of the FP-tree is small enough to fit into main memory, this will allow us to extract frequent itemsets directly from the structure in memory instead of making repeated passes over the data stored on disk.

Figure 5.24 shows a data set that contains ten transactions and five items. The structures of the FP-tree after reading the first three transactions are also depicted in the diagram. Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path. Initially, the FP-tree contains only the root node represented by the null symbol. The FP-tree is subsequently extended in the following way:

Transaction Data Set	
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

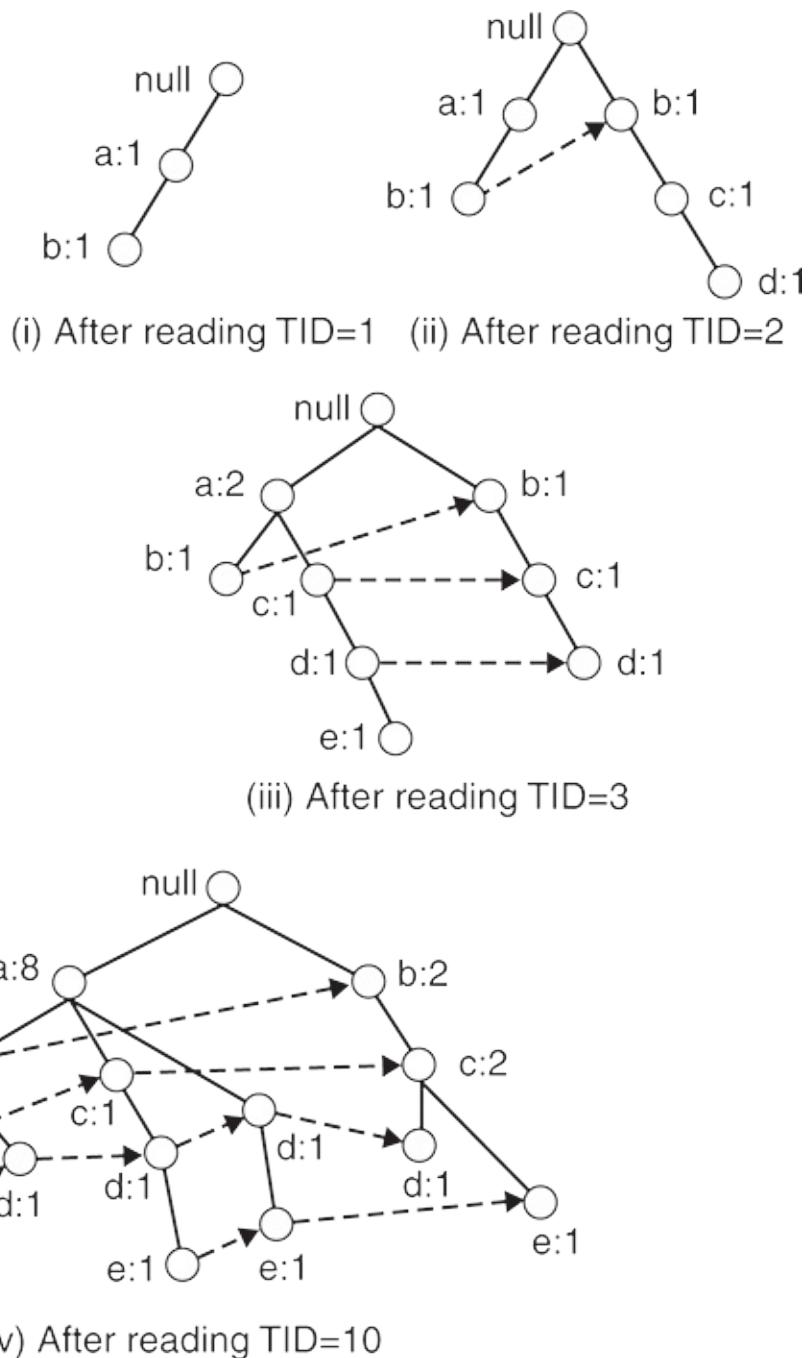


Figure 5.24.

Construction of an FP-tree.

1. The data set is scanned once to determine the support count of each item. Infrequent items are discarded, while the frequent items are sorted in decreasing support counts inside every transaction of the data

set. For the data set shown in [Figure 5.24](#), *a* is the most frequent item, followed by *b*, *c*, *d*, and *e*.

2. The algorithm makes a second pass over the data to construct the FP-tree. After reading the first transaction, {*a*, *b*}, the nodes labeled as *a* and *b* are created. A path is then formed from `null` →*a*→*b* to encode the transaction. Every node along the path has a frequency count of 1.
3. After reading the second transaction, {*b*, *c*, *d*}, a new set of nodes is created for items *b*, *c*, and *d*. A path is then formed to represent the transaction by connecting the nodes `null` →*b*→*c*→*d*. Every node along this path also has a frequency count equal to one. Although the first two transactions have an item in common, which is *b*, their paths are disjoint because the transactions do not share a common prefix.
4. The third transaction, {*a*, *c*, *d*, *e*}, shares a common prefix item (which is *a*) with the first transaction. As a result, the path for the third transaction, `null` →*a*→*c*→*d*→*e*, overlaps with the path for the first transaction, `null` →*a*→*b*. Because of their overlapping path, the frequency count for node *a* is incremented to two, while the frequency counts for the newly created nodes, *c*, *d*, and *e*, are equal to one.
5. This process continues until every transaction has been mapped onto one of the paths given in the FP-tree. The resulting FP-tree after reading all the transactions is shown at the bottom of [Figure 5.24](#).

The size of an FP-tree is typically smaller than the size of the uncompressed data because many transactions in market basket data often share a few items in common. In the best-case scenario, where all the transactions have the same set of items, the FP-tree contains only a single branch of nodes. The worst-case scenario happens when every transaction has a unique set of items. As none of the transactions have any items in common, the size of the FP-tree is effectively the same as the size of the original data. However, the physical storage requirement for the FP-tree is higher because it requires additional space to store pointers between nodes and counters for each item.

The size of an FP-tree also depends on how the items are ordered. The notion of ordering items in decreasing order of support counts relies on the possibility that the high support items occur more frequently across all paths and hence must be used as most commonly occurring prefixes. For example, if the ordering scheme in the preceding example is reversed, i.e., from lowest to highest support item, the resulting FP-tree is shown in [Figure 5.25](#). The tree appears to be denser because the branching factor at the root node has increased from 2 to 5 and the number of nodes containing the high support items such as *a* and *b* has increased from 3 to 12. Nevertheless, ordering by decreasing support counts does not always lead to the smallest tree, especially when the high support items do not occur frequently together with the other items. For example, suppose we augment the data set given in [Figure 5.24](#) with 100 transactions that contain {*e*}, 80 transactions that contain {*d*}, 60 transactions that contain {*c*}, and 40 transactions that contain {*b*}. Item *e* is now most frequent, followed by *d*, *c*, *b*, and *a*. With the augmented transactions, ordering by decreasing support counts will result in an FP-tree similar to [Figure 5.25](#), while a scheme based on increasing support counts produces a smaller FP-tree similar to [Figure 5.24\(iv\)](#).

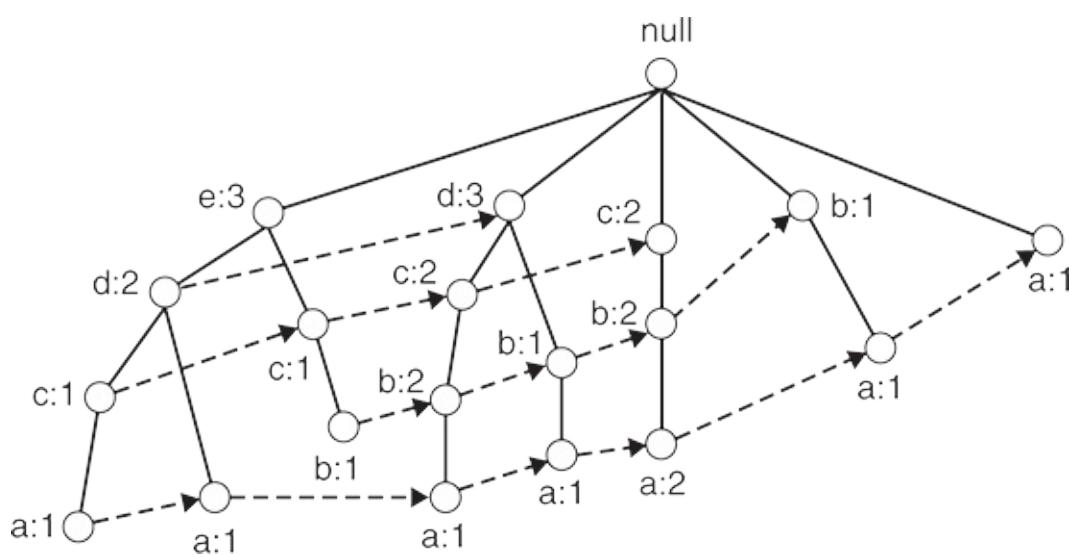


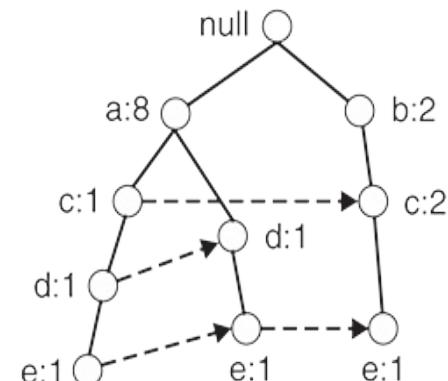
Figure 5.25.

An FP-tree representation for the data set shown in [Figure 5.24](#) with a different item ordering scheme.

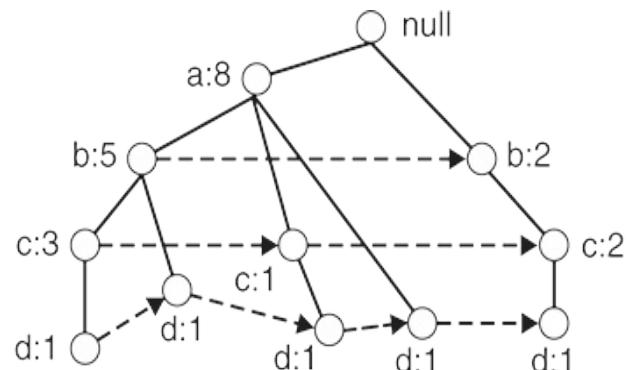
An FP-tree also contains a list of pointers connecting nodes that have the same items. These pointers, represented as dashed lines in [Figures 5.24](#) and [5.25](#), help to facilitate the rapid access of individual items in the tree. We explain how to use the FP-tree and its corresponding pointers for frequent itemset generation in the next Section.

5.6.2 Frequent Itemset Generation in FP-Growth Algorithm

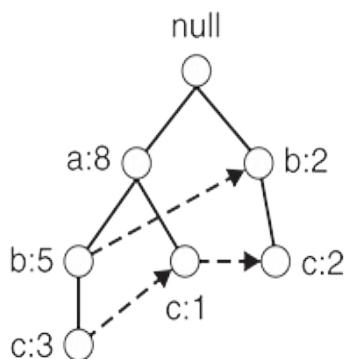
FP-growth is an algorithm that generates frequent itemsets from an FP-tree by exploring the tree in a bottom-up fashion. Given the example tree shown in [Figure 5.24](#), the algorithm looks for frequent itemsets ending in *e* first, followed by *d*, *c*, *b*, and finally, *a*. This bottom-up strategy for finding frequent itemsets ending with a particular item is equivalent to the suffix-based approach described in [Section 5.5](#). Since every transaction is mapped onto a path in the FP-tree, we can derive the frequent itemsets ending with a particular item, say, *e*, by examining only the paths containing node *e*. These paths can be accessed rapidly using the pointers associated with node *e*. The extracted paths are shown in [Figure 5.26 \(a\)](#). Similar paths for itemsets ending in *d*, *c*, *b*, and *a* are shown in [Figures 5.26 \(b\)](#), [\(c\)](#), [\(d\)](#), and [\(e\)](#), respectively.



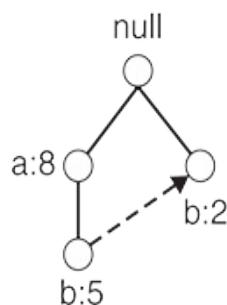
(a) Paths containing node e



(b) Paths containing node d



(c) Paths containing node c



(d) Paths containing node b



(e) Paths containing node a

Figure 5.26.

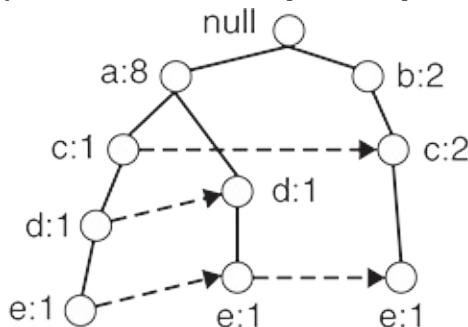
Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e , d , c , b , and a .

FP-growth finds all the frequent itemsets ending with a particular suffix by employing a divide-and-conquer strategy to split the problem into smaller subproblems. For example, suppose we are interested in finding all frequent itemsets ending in e . To do this, we must first check whether the itemset $\{e\}$ itself is frequent. If it is frequent, we consider the subproblem of finding frequent itemsets ending in de , followed by ce , be , and ae . In turn, each of these subproblems are further decomposed into smaller subproblems. By merging the solutions obtained from the subproblems, all the frequent itemsets ending in e can be found. Finally, the set of all frequent itemsets can be generated by merging the solutions to the subproblems of finding frequent

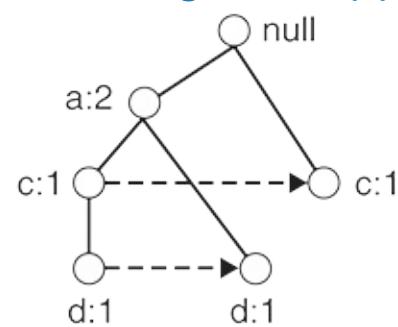
itemsets ending in e , d , c , b , and a . This divide-and-conquer approach is the key strategy employed by the FP-growth algorithm.

For a more concrete example on how to solve the subproblems, consider the task of finding frequent itemsets ending with e .

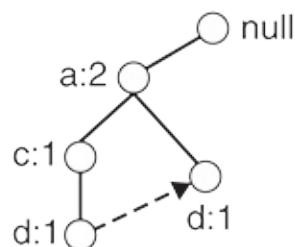
1. The first step is to gather all the paths containing node e . These initial paths are called **prefix paths** and are shown in [Figure 5.27\(a\)](#).



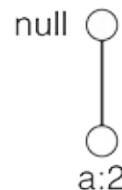
(a) Prefix paths ending in e



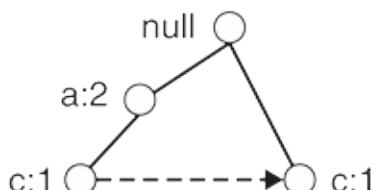
(b) Conditional FP-tree for e



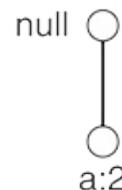
(c) Prefix paths ending in de



(d) Conditional FP-tree for de



(e) Prefix paths ending in ce



(f) Prefix paths ending in ae

Figure 5.27.

Example of applying the FP-growth algorithm to find frequent itemsets ending in e .

2. From the prefix paths shown in [Figure 5.27\(a\)](#), the support count for e is obtained by adding the support counts associated with node e. Assuming that the minimum support count is 2, {e} is declared a frequent itemset because its support count is 3.
3. Because {e} is frequent, the algorithm has to solve the subproblems of finding frequent itemsets ending in de, ce, be, and ae. Before solving these subproblems, it must first convert the prefix paths into a **conditional FP-tree**, which is structurally similar to an FP-tree, except it is used to find frequent itemsets ending with a particular suffix. A conditional FP-tree is obtained in the following way:
 - a. First, the support counts along the prefix paths must be updated because some of the counts include transactions that do not contain item e. For example, the rightmost path shown in [Figure 5.27\(a\)](#), `null → b:2 → c:2 → e:1`, includes a transaction {b, c} that does not contain item e. The counts along the prefix path must therefore be adjusted to 1 to reflect the actual number of transactions containing {b, c, e}.
 - b. The prefix paths are truncated by removing the nodes for e. These nodes can be removed because the support counts along the prefix paths have been updated to reflect only transactions that contain e and the subproblems of finding frequent itemsets ending in de, ce, be, and ae no longer need information about node e.
 - c. After updating the support counts along the prefix paths, some of the items may no longer be frequent. For example, the node b appears only once and has a support count equal to 1, which means that there is only one transaction that contains both b and e. Item b can be safely ignored from subsequent analysis because all itemsets ending in be must be infrequent.

The conditional FP-tree for e is shown in [Figure 5.27\(b\)](#). The tree looks different than the original prefix paths because the frequency counts have been updated and the nodes b and e have been eliminated.

4. FP-growth uses the conditional FP-tree for e to solve the subproblems of finding frequent itemsets ending in de , ce , and ae . To find the frequent itemsets ending in de , the prefix paths for d are gathered from the conditional FP-tree for e ([Figure 5.27\(c\)](#)). By adding the frequency counts associated with node d , we obtain the support count for $\{d, e\}$. Since the support count is equal to 2, $\{d, e\}$ is declared a frequent itemset. Next, the algorithm constructs the conditional FP-tree for de using the approach described in step 3. After updating the support counts and removing the infrequent item c , the conditional FP-tree for de is shown in [Figure 5.27\(d\)](#). Since the conditional FP-tree contains only one item, a , whose support is equal to $minsup$, the algorithm extracts the frequent itemset $\{a, d, e\}$ and moves on to the next subproblem, which is to generate frequent itemsets ending in ce . After processing the prefix paths for c , $\{c, e\}$ is found to be frequent. However, the conditional FP-tree for ce will have no frequent items and thus will be eliminated. The algorithm proceeds to solve the next subproblem and finds $\{a, e\}$ to be the only frequent itemset remaining.

This example illustrates the divide-and-conquer approach used in the FP-growth algorithm. At each recursive step, a conditional FP-tree is constructed by updating the frequency counts along the prefix paths and removing all infrequent items. Because the subproblems are disjoint, FP-growth will not generate any duplicate itemsets. In addition, the counts associated with the nodes allow the algorithm to perform support counting while generating the common suffix itemsets.

FP-growth is an interesting algorithm because it illustrates how a compact representation of the transaction data set helps to efficiently generate frequent itemsets. In addition, for certain transaction data sets, FP-growth outperforms the standard *Apriori* algorithm by several orders of magnitude. The run-time performance of FP-growth depends on the **compaction factor** of the data set. If the resulting conditional FP-trees are very bushy (in the worst case, a full prefix tree), then the performance of the algorithm degrades significantly because it has to generate a large number of subproblems and merge the results returned by each subproblem.

5.7 Evaluation of Association Patterns

Although the *Apriori* principle significantly reduces the exponential search space of candidate itemsets, association analysis algorithms still have the potential to generate a large number of patterns. For example, although the data set shown in [Table 5.1](#) contains only six items, it can produce hundreds of association rules at particular support and confidence thresholds. As the size and dimensionality of real commercial databases can be very large, we can easily end up with thousands or even millions of patterns, many of which might not be interesting. Identifying the most interesting patterns from the multitude of all possible ones is not a trivial task because “one person’s trash might be another person’s treasure.” It is therefore important to establish a set of well-accepted criteria for evaluating the quality of association patterns.

The first set of criteria can be established through a data-driven approach to define **objective interestingness measures**. These measures can be used to rank patterns—itemsets or rules—and thus provide a straightforward way of dealing with the enormous number of patterns that are found in a data set. Some of the measures can also provide statistical information, e.g., itemsets that involve a set of unrelated items or cover very few transactions are considered uninteresting because they may capture spurious relationships in the data and should be eliminated. Examples of objective interestingness measures include support, confidence, and correlation.

The second set of criteria can be established through subjective arguments. A pattern is considered subjectively uninteresting unless it reveals unexpected information about the data or provides useful knowledge that can lead to profitable actions. For example, the rule $\{\text{Butter}\} \rightarrow \{\text{Bread}\}$ may not be interesting, despite having high support and confidence values, because the

relationship represented by the rule might seem rather obvious. On the other hand, the rule {Diapers}→{Beer} is interesting because the relationship is quite unexpected and may suggest a new cross-selling opportunity for retailers. Incorporating subjective knowledge into pattern evaluation is a difficult task because it requires a considerable amount of prior information from domain experts. Readers interested in subjective interestingness measures may refer to resources listed in the bibliography at the end of this chapter.

5.7.1 Objective Measures of Interestingness

An objective measure is a data-driven approach for evaluating the quality of association patterns. It is domain-independent and requires only that the user specifies a threshold for filtering low-quality patterns. An objective measure is usually computed based on the frequency counts tabulated in a **contingency table**. [Table 5.6](#) shows an example of a contingency table for a pair of binary variables, A and B . We use the notation $A^-(B^-)$ to indicate that A (B) is absent from a transaction. Each entry f_{ij} in this 2×2 table denotes a frequency count. For example, f_{11} is the number of times A and B appear together in the same transaction, while f_{01} is the number of transactions that contain B but not A . The row sum f_{1+} represents the support count for A , while the column sum f_{+1} represents the support count for B . Finally, even though our discussion focuses mainly on asymmetric binary variables, note that contingency tables are also applicable to other attribute types such as symmetric binary, nominal, and ordinal variables.

Table 5.6. A 2-way contingency table for variables A and B .

	B	B^-	
A	f_{11}	f_{10}	f_{1+}
A^-	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

Limitations of the Support-Confidence Framework

The classical association rule mining formulation relies on the support and confidence measures to eliminate uninteresting patterns. The drawback of support, which is described more fully in [Section 5.8](#), is that many potentially interesting patterns involving low support items might be eliminated by the support threshold. The drawback of confidence is more subtle and is best demonstrated with the following example.

Example 5.3.

Suppose we are interested in analyzing the relationship between people who drink tea and coffee. We may gather information about the beverage preferences among a group of people and summarize their responses into a contingency table such as the one shown in [Table 5.7](#).

Table 5.7. Beverage preferences among a group of 1000 people.

	Coffee	Coffee $^-$	
Tea	150	50	200
Tea $^-$	650	150	800
	800	200	1000

The information given in this table can be used to evaluate the association rule $\{\text{Tea}\} \rightarrow \{\text{Coffee}\}$. At first glance, it may appear that people who drink tea also tend to drink coffee because the rule's support (15%) and confidence (75%) values are reasonably high. This argument would have been acceptable except that the fraction of people who drink coffee, regardless of whether they drink tea, is 80%, while the fraction of tea drinkers who drink coffee is only 75%. Thus knowing that a person is a tea drinker actually decreases her probability of being a coffee drinker from 80% to 75%! The rule $\{\text{Tea}\} \rightarrow \{\text{Coffee}\}$ is therefore misleading despite its high confidence value.

Now consider a similar problem where we are interested in analyzing the relationship between people who drink tea and people who use honey in their beverage. **Table 5.8**  summarizes the information gathered over the same group of people about their preferences for drinking tea and using honey. If we evaluate the association rule $\{\text{Tea}\} \rightarrow \{\text{Honey}\}$ using this information, we will find that the confidence value of this rule is merely 50%, which might be easily rejected using a reasonable threshold on the confidence value, say 70%. One thus might consider that the preference of a person for drinking tea has no influence on her preference for using honey. However, the fraction of people who use honey, regardless of whether they drink tea, is only 12%. Hence, knowing that a person drinks tea significantly increases her probability of using honey from 12% to 50%. Further, the fraction of people who do not drink tea but use honey is only 2.5%! This suggests that there is definitely some information in the preference of a person of using honey given that she drinks tea. The rule $\{\text{Tea}\} \rightarrow \{\text{Honey}\}$ may therefore be falsely rejected if confidence is used as the evaluation measure.

Table 5.8. Information about people who drink tea and people who use honey in their beverage.

	<i>Honey</i>	<i>Honey</i> ⁻	
<i>Tea</i>	100	100	200
<i>Tea</i> ⁻	20	780	800
	120	880	1000

Note that if we take the support of coffee drinkers into account, we would not be surprised to find that many of the people who drink tea also drink coffee, since the overall number of coffee drinkers is quite large by itself. What is more surprising is that the fraction of tea drinkers who drink coffee is actually less than the overall fraction of people who drink coffee, which points to an inverse relationship between tea drinkers and coffee drinkers. Similarly, if we account for the fact that the support of using honey is inherently small, it is easy to understand that the fraction of tea drinkers who use honey will naturally be small. Instead, what is important to measure is the change in the fraction of honey users, given the information that they drink tea.

The limitations of the confidence measure are well-known and can be understood from a statistical perspective as follows. The support of a variable measures the probability of its occurrence, while the support $s(A, B)$ of a pair of variables A and B measures the probability of the two variables occurring together. Hence, the joint probability $P(A, B)$ can be written as

$$P(A, B) = s(A, B) = f_{11}N.$$

If we assume A and B are statistically independent, i.e. there is no relationship between the occurrences of A and B , then $P(A, B) = P(A) \times P(B)$. Hence, under the assumption of statistical independence between A and B , the support $s_{\text{indep}}(A, B)$ of A and B can be written as

$$\text{sindep}(A, B) = s(A) \times s(B) \text{ or equivalently, } \text{sindep}(A, B) = f_1 + N \times f + 1/N. \quad (5.4)$$

If the support between two variables, $s(A, B)$ is equal to $\text{sindep}(A, B)$, then A and B can be considered to be unrelated to each other. However, if $s(A, B)$ is widely different from $\text{sindep}(A, B)$, then A and B are most likely dependent. Hence, any deviation of $s(A, B)$ from $s(A) \times s(B)$ can be seen as an indication of a statistical relationship between A and B . Since the confidence measure only considers the deviance of $s(A, B)$ from $s(A)$ and not from $s(A) \times s(B)$, it fails to account for the support of the consequent, namely $s(B)$. This results in the detection of spurious patterns (e.g., $\{\text{Tea}\} \rightarrow \{\text{Coffee}\}$) and the rejection of truly interesting patterns (e.g., $\{\text{Tea}\} \rightarrow \{\text{Honey}\}$), as illustrated in the previous example.

Various objective measures have been used to capture the deviance of $s(A, B)$ from $\text{sindep}(A, B)$, that are not susceptible to the limitations of the confidence measure. Below, we provide a brief description of some of these measures and discuss some of their properties.

Interest Factor

The interest factor, which is also called as the “lift,” can be defined as follows:

$$I(A, B) = s(A, B) / s(A) \times s(B) = Nf_1 / (f_1 + f + 1). \quad (5.5)$$

Notice that $s(A) \times s(B) = \text{sindep}(A, B)$. Hence, the interest factor measures the ratio of the support of a pattern $s(A, B)$ against its baseline support $\text{sindep}(A, B)$ computed under the statistical independence assumption. Using

Equations 5.5  and **5.4** , we can interpret the measure as follows:

$$I(A, B) = \begin{cases} = 1, & \text{if } A \text{ and } B \text{ are independent;} \\ > 1, & \text{if } A \text{ and } B \text{ are positively related;} \\ < 1, & \text{if } A \text{ and } B \text{ are negatively related.} \end{cases} \quad (5.6)$$

For the tea-coffee example shown in [Table 5.7](#), $I=0.150.2\times0.8=0.9375$, thus suggesting a slight negative relationship between tea drinkers and coffee drinkers. Also, for the tea-honey example shown in [Table 5.8](#), $I=0.10.12\times0.2=4.1667$, suggesting a strong positive relationship between people who drink tea and people who use honey in their beverage. We can thus see that the interest factor is able to detect meaningful patterns in the tea-coffee and tea-honey examples. Indeed, the interest factor has a number of statistical advantages over the confidence measure that make it a suitable measure for analyzing statistical independence between variables.

Piatesky-Shapiro (PS) Measure

Instead of computing the ratio between $s(A, B)$ and $\text{sindep}(A, B)=s(A)\times s(B)$, the *PS* measure considers the difference between $s(A, B)$ and $s(A)\times s(B)$ as follows.

$$PS=s(A, B)-s(A)\times s(B)=f_{11}N-f_1+f_1N_2 \quad (5.7)$$

The PS value is 0 when A and B are mutually independent of each other. Otherwise, $PS>0$ when there is a positive relationship between the two variables, and $PS<0$ when there is a negative relationship.

Correlation Analysis

Correlation analysis is one of the most popular techniques for analyzing relationships between a pair of variables. For continuous variables, correlation is defined using Pearson's correlation coefficient (see [Equation 2.10](#) on page [83](#)). For binary variables, correlation can be measured using the ϕ -coefficient, which is defined as

$$\phi=f_{11}f_{00}-f_{01}f_{10}f_1+f_1f_0+f_0. \quad (5.8)$$

If we rearrange the terms in 5.8, we can show that the ϕ -coefficient can be rewritten in terms of the support measures of A , B , and $\{A, B\}$ as follows:

$$\phi = s(A, B) - s(A) \times s(B) s(A) \times (1 - s(A)) \times s(B) \times (1 - s(B)). \quad (5.9)$$

Note that the numerator in the above equation is identical to the PS measure. Hence, the ϕ -coefficient can be understood as a normalized version of the PS measure, where the value of the ϕ -coefficient ranges from -1 to $+1$. From a statistical viewpoint, the correlation captures the normalized difference between $s(A, B)$ and $\text{sindep}(A, B)$. A correlation value of 0 means no relationship, while a value of $+1$ suggests a perfect positive relationship and a value of -1 suggests a perfect negative relationship. The correlation measure has a statistical meaning and hence is widely used to evaluate the strength of statistical independence among variables. For instance, the correlation between tea and coffee drinkers in [Table 5.7](#) is -0.0625 which is slightly less than 0 . On the other hand, the correlation between people who drink tea and people who use honey in [Table 5.8](#) is 0.5847 , suggesting a positive relationship.

/S Measure

/S is an alternative measure for capturing the relationship between $s(A, B)$ and $s(A) \times s(B)$. The */S* measure is defined as follows:

$$IS(A, B) = I(A, B) \times s(A, B) = s(A, B) s(A) s(B) = f_{11} f_1 + f + 1. \quad (5.10)$$

Although the definition of */S* looks quite similar to the interest factor, they share some interesting differences. Since */S* is the geometric mean between the interest factor and the support of a pattern, */S* is large when both the interest factor and support are large. Hence, if the interest factor of two patterns are identical, the */S* has a preference of selecting the pattern with higher support. It is also possible to show that */S* is mathematically equivalent

to the cosine measure for binary variables (see [Equation 2.6](#) on page [81](#)). The value of IS thus varies from 0 to 1, where an IS value of 0 corresponds to no co-occurrence of the two variables, while an IS value of 1 denotes perfect relationship, since they occur in exactly the same transactions. For the tea-coffee example shown in [Table 5.7](#), the value of IS is equal to 0.375, while the value of IS for the tea-honey example in [Table 5.8](#) is 0.6455. The IS measure thus gives a higher value for the $\{\text{Tea}\} \rightarrow \{\text{Honey}\}$ rule than the $\{\text{Tea}\} \rightarrow \{\text{Coffee}\}$ rule, which is consistent with our understanding of the two rules.

Alternative Objective Interestingness Measures

Note that all of the measures defined in the previous Section use different techniques to capture the deviance between $s(A, B)$ and $\sindep = s(A) \times s(B)$. Some measures use the ratio between $s(A, B)$ and $\sindep(A, B)$, e.g., the interest factor and IS , while some other measures consider the difference between the two, e.g., the PS and the ϕ -coefficient. Some measures are bounded in a particular range, e.g., the IS and the ϕ -coefficient, while others are unbounded and do not have a defined maximum or minimum value, e.g., the Interest Factor. Because of such differences, these measures behave differently when applied to different types of patterns. Indeed, the measures defined above are not exhaustive and there exist many alternative measures for capturing different properties of relationships between pairs of binary variables. [Table 5.9](#) provides the definitions for some of these measures in terms of the frequency counts of a 2×2 contingency table.

Table 5.9. Examples of objective measures for the itemset $\{A, B\}$.

Measure (Symbol)	Definition
Correlation (ϕ)	$Nf_{11} - f_1 + f_+ + f_1 + f_+ + f_0 + f_+$

Odds ratio (α)	$(f_{11}f_{00})/(f_{10}f_{01})$
Kappa (κ)	$Nf_{11}+Nf_{00}-f_{1+f+1-f_0+f_0}N_2-f_{1+f+1-f_0+f_0}$
Interest (I)	$(Nf_{11})/(f_{1+f+1})$
Cosine (I_S)	$(f_{11})/(f_{1+f+1})$
Piatetsky-Shapiro (PS)	$f_{11}N-f_{1+f+1}N_2$
Collective strength (S)	$f_{11}+f_{00}f_{1+f+1+f_0+f_0}\times N-f_{1+f+1-f_0+f_0}N-f_{11-f_00}$
Jaccard (ζ)	$f_{11}/(f_{1++f+1-f_{11}})$
All-confidence (h)	$\min[f_{11}f_{1+}, f_{11}f_{+1}]$

Consistency among Objective Measures

Given the wide variety of measures available, it is reasonable to question whether the measures can produce similar ordering results when applied to a set of association patterns. If the measures are consistent, then we can choose any one of them as our evaluation metric. Otherwise, it is important to understand what their differences are in order to determine which measure is more suitable for analyzing certain types of patterns.

Suppose the measures defined in [Table 5.9](#) are applied to rank the ten contingency tables shown in [Table 5.10](#). These contingency tables are chosen to illustrate the differences among the existing measures. The ordering produced by these measures is shown in [Table 5.11](#) (with 1 as the most interesting and 10 as the least interesting table). Although some of the measures appear to be consistent with each other, others produce quite different ordering results. For example, the rankings given by the ϕ -coefficient agrees mostly with those provided by κ and collective strength, but are quite

different than the rankings produced by interest factor. Furthermore, a contingency table such as E10 is ranked lowest according to the ϕ -coefficient, but highest according to interest factor.

Table 5.10. Example of contingency tables.

Example	f11	f10	f01	f00
E1	8123	83	424	1370
E2	8330	2	622	1046
E3	3954	3080	5	2961
E4	2886	1363	1320	4431
E5	1500	2000	500	6000
E6	4000	2000	1000	3000
E7	9481	298	127	94
E8	4000	2000	2000	2000
E9	7450	2483	4	63
E10	61	2483	4	7452

Table 5.11. Rankings of contingency tables using the measures given in [Table 5.9](#).

E3	3	2	4	4	5	1	3	6	8
E4	4	8	3	3	7	3	4	7	5
E5	5	7	6	2	9	6	6	9	9
E6	6	9	5	5	6	4	5	5	7
E7	7	6	7	9	1	8	7	1	1
E8	8	10	8	8	8	7	8	8	7
E9	9	4	9	10	4	9	9	4	4
E10	10	5	10	1	10	10	10	10	10

Properties of Objective Measures

The results shown in [Table 5.11](#) suggest that the measures greatly differ from each other and can provide conflicting information about the quality of a pattern. In fact, no measure is universally best for all applications. In the following, we describe some properties of the measures that play an important role in determining if they are suited for a certain application.

Inversion Property

Consider the binary vectors shown in [Figure 5.28](#). The 0/1 value in each column vector indicates whether a transaction (row) contains a particular item (column). For example, the vector A indicates that the item appears in the first and last transactions, whereas the vector B indicates that the item is contained only in the fifth transaction. The vectors A^- and B^- are the inverted versions of A and B , i.e., their 1 values have been changed to 0 values (absence to presence) and vice versa. Applying this transformation to a binary

vector is called **inversion**. If a measure is invariant under the inversion operation, then its value for the vector pair $\{A^-, B^-\}$ should be identical to its value for $\{A, B\}$. The inversion property of a measure can be tested as follows.

A	B	\bar{A}	\bar{B}
1	0	0	1
0	0	1	1
0	0	1	1
0	0	1	1
0	1	1	0
0	0	1	1
0	0	1	1
0	0	1	1
1	0	0	1

(a)

(b)

Figure 5.28.

Effect of the inversion operation. The vectors A^- and E^- are inversions of vectors A and B , respectively.

Definition 5.6. (Inversion Property.)

An objective measure M is invariant under the inversion operation if its value remains the same when exchanging the frequency counts f_{11} with f_{00} and f_{10} with f_{01} .

Measures that are invariant to the inversion property include the correlation (ϕ -coefficient), odds ratio, κ , and collective strength. These measures are especially useful in scenarios where the presence (1's) of a variable is as

important as its absence (0's). For example, if we compare two sets of answers to a series of true/false questions where 0's (true) and 1's (false) are equally meaningful, we should use a measure that gives equal importance to occurrences of 0–0's and 1–1's. For the vectors shown in [Figure 5.28](#), the ϕ -coefficient is equal to -0.1667 regardless of whether we consider the pair {A, B} or pair {A[−], B[−]}. Similarly, the odds ratio for both pairs of vectors is equal to a constant value of 0. Note that even though the ϕ -coefficient and the odds ratio are invariant to inversion, they can still show different results, as will be shown later.

Measures that do not remain invariant under the inversion operation include the interest factor and the *IS* measure. For example, the *IS* value for the pair {A[−], B[−]} in [Figure 5.28](#) is 0.825, which reflects the fact that the 1's in A[−] and B[−] occur frequently together. However, the *IS* value of its inverted pair {A, B} is equal to 0, since A and B do not have any co-occurrence of 1's. For asymmetric binary variables, e.g., the occurrence of words in documents, this is indeed the desired behavior. A desired similarity measure between asymmetric variables should not be invariant to inversion, since for these variables, it is more meaningful to capture relationships based on the presence of a variable rather than its absence. On the other hand, if we are dealing with symmetric binary variables where the relationships between 0's and 1's are equally meaningful, care should be taken to ensure that the chosen measure is invariant to inversion.

Although the values of the interest factor and *IS* change with the inversion operation, they can still be inconsistent with each other. To illustrate this, consider [Table 5.12](#), which shows the contingency tables for two pairs of variables, {p, q} and {r, s}. Note that r and s are inverted transformations of p and q, respectively, where the roles of 0's and 1's have just been reversed. The interest factor for {p, q} is 1.02 and for {r, s} is 4.08, which means that the interest factor finds the inverted pair {r, s} more related than the original pair

$\{p, q\}$. On the contrary, the I/S value decreases upon inversion from 0.9346 for $\{p, q\}$ to 0.286 for $\{r, s\}$, suggesting quite an opposite trend to that of the interest factor. Even though these measures conflict with each other for this example, they may be the right choice of measure in different applications.

Table 5.12. Contingency tables for the pairs $\{p,q\}$ and $\{r,s\}$.

	p	p^-	
q	880	50	930
q^-	50	20	70
	930	70	1000
	r	r^-	
s	20	50	70
s^-	50	880	930
	70	930	1000

Scaling Property

Table 5.13 shows two contingency tables for gender and the grades achieved by students enrolled in a particular course. These tables can be used to study the relationship between gender and performance in the course. The second contingency table has data from the same population but has twice as many males and three times as many females. The actual number of males or females can depend upon the samples available for study, but the relationship between gender and grade should not change just because of differences in sample sizes. Similarly, if the number of students with high and low grades are changed in a new study, a measure of association between

gender and grades should remain unchanged. Hence, we need a measure that is invariant to scaling of rows or columns. The process of multiplying a row or column of a contingency table by a constant value is called a row or column scaling operation. A measure that is invariant to scaling does not change its value after any row or column scaling operation.

Table 5.13. The grade-gender example. (a) Sample data of size 100.

	Male	Female	
High	30	20	50
Low	40	10	50
	70	30	100

(b) Sample data of size 230.

	Male	Female	
High	60	60	120
Low	80	30	110
	140	90	230

Definition 5.7. (Scaling Invariance Property.)

Let T be a contingency table with frequency counts $[f_{11}; f_{10}; f_{01}; f_{00}]$. Let T' be the transformed a contingency table

with scaled frequency counts
[$k_1 k_3 f_{11}; k_2 k_3 f_{10}; k_1 k_4 f_{01}; k_2 k_4 f_{00}$], where k_1, k_2, k_3, k_4 are positive constants used to scale the two rows and the two columns of T . An objective measure M is invariant under the row/column scaling operation if $M(T)=M(T')$.

Note that the use of the term ‘scaling’ here should not be confused with the scaling operation for continuous variables introduced in [Chapter 2](#) on page 23, where all the values of a variable were being multiplied by a constant factor, instead of scaling a row or column of a contingency table.

Scaling of rows and columns in contingency tables occurs in multiple ways in different applications. For example, if we are measuring the effect of a particular medical procedure on two sets of subjects, *healthy* and *diseased*, the ratio of healthy and diseased subjects can widely vary across different studies involving different groups of participants. Further, the fraction of healthy and diseased subjects chosen for a controlled study can be quite different from the true fraction observed in the complete population. These differences can result in a row or column scaling in the contingency tables for different populations of subjects. In general, the frequencies of items in a contingency table closely depends on the sample of transactions used to generate the table. Any change in the sampling procedure may affect a row or column scaling transformation. A measure that is expected to be invariant to differences in the sampling procedure must not change with row or column scaling.

Of all the measures introduced in [Table 5.9](#), only the odds ratio (α) is invariant to row and column scaling operations. For example, the value of odds ratio for both the tables in [Table 5.13](#) is equal to 0.375. All other

measures such as the ϕ -coefficient, κ , I/S , interest factor, and collective strength (S) change their values when the rows and columns of the contingency table are rescaled. Indeed, the odds ratio is a preferred choice of measure in the medical domain, where it is important to find relationships that do not change with differences in the population sample chosen for a study.

Null Addition Property

Suppose we are interested in analyzing the relationship between a pair of words, such as `data` and `mining`, in a set of documents. If a collection of articles about ice fishing is added to the data set, should the association between `data` and `mining` be affected? This process of adding unrelated data (in this case, documents) to a given data set is known as the **null addition** operation.

Definition 5.8. (Null Addition Property.)

An objective measure M is invariant under the null addition operation if it is not affected by increasing f_{00} , while all other frequencies in the contingency table stay the same.

For applications such as document analysis or market basket analysis, we would like to use a measure that remains invariant under the null addition operation. Otherwise, the relationship between words can be made to change simply by adding enough documents that do not contain both words! Examples of measures that satisfy this property include cosine (I/S) and

Jaccard (ξ) measures, while those that violate this property include interest factor, PS , odds ratio, and the ϕ -coefficient.

To demonstrate the effect of null addition, consider the two contingency tables T1 and T2 shown in [Table 5.14](#). Table T2 has been obtained from T1 by adding 1000 extra transactions with both A and B absent. This operation only affects the f_{00} entry of Table T2, which has increased from 100 to 1100, whereas all the other frequencies in the table (f_{11} , f_{10} , and f_{01}) remain the same. Since IS is invariant to null addition, it gives a constant value of 0.875 to both the tables. However, the addition of 1000 extra transactions with occurrences of 0–0's changes the value of interest factor from 0.972 for T1 (denoting a slightly negative correlation) to 1.944 for T2 (positive correlation). Similarly, the value of odds ratio increases from 7 for T1 to 77 for T2. Hence, when the interest factor or odds ratio are used as the association measure, the relationships between variables changes by the addition of null transactions where both the variables are absent. In contrast, the IS measure is invariant to null addition, since it considers two variables to be related only if they frequently occur together. Indeed, the IS measure (cosine measure) is widely used to measure similarity among documents, which is expected to depend only on the joint occurrences (1's) of words in documents, but not their absences (0's).

Table 5.14. An example demonstrating the effect of null addition.

(a) Table T1.

	B	B^-	
A	700	100	800
A^-	100	100	200
	800	200	1000

(b) Table T2.

	B	B^-	
A	700	100	800
A^-	10	1100	1200
	800	1200	2000

Table 5.15 provides a summary of properties for the measures defined in **Table 5.9**. Even though this list of properties is not exhaustive, it can serve as a useful guide for selecting the right choice of measure for an application. Ideally, if we know the specific requirements of a certain application, we can ensure that the selected measure shows properties that adhere to those requirements. For example, if we are dealing with asymmetric variables, we would prefer to use a measure that is not invariant to null addition or inversion. On the other hand, if we require the measure to remain invariant to changes in the sample size, we would like to use a measure that does not change with scaling.

Table 5.15. Properties of symmetric measures.

Symbol	Measure	Inversion	Null Addition	Scaling
ϕ	ϕ -coefficient	Yes	No	No
α	odds ratio	Yes	No	Yes
κ	Cohen's	Yes	No	No
I	Interest	No	No	No
IS	Cosine	No	Yes	No
PS	Piatetsky-Shapiro's	Yes	No	No

s	Collective strength	Yes	No	No
ζ	Jaccard	No	Yes	No
h	All-confidence	No	Yes	No
s	Support	No	No	No

Asymmetric Interestingness Measures

Note that in the discussion so far, we have only considered measures that do not change their value when the order of the variables are reversed. More specifically, if M is a measure and A and B are two variables, then $M(A, B)$ is equal to $M(B, A)$ if the order of the variables does not matter. Such measures are called **symmetric**. On the other hand, measures that depend on the order of variables ($M(A, B) \neq M(B, A)$) are called **asymmetric** measures. For example, the interest factor is a symmetric measure because its value is identical for the rules $A \rightarrow B$ and $B \rightarrow A$. In contrast, confidence is an asymmetric measure since the confidence for $A \rightarrow B$ and $B \rightarrow A$ may not be the same. Note that the use of the term ‘asymmetric’ to describe a particular type of measure of relationship—one in which the order of the variables is important—should not be confused with the use of ‘asymmetric’ to describe a binary variable for which only 1's are important. Asymmetric measures are more suitable for analyzing association rules, since the items in a rule do have a specific order. Even though we only considered symmetric measures to discuss the different properties of association measures, the above discussion is also relevant for the asymmetric measures. See Bibliographic Notes for more information about different kinds of asymmetric measures and their properties.

5.7.2 Measures beyond Pairs of Binary Variables

The measures shown in [Table 5.9](#) are defined for pairs of binary variables (e.g., 2-itemsets or association rules). However, many of them, such as support and all-confidence, are also applicable to larger-sized itemsets. Other measures, such as interest factor, IS , PS , and Jaccard coefficient, can be extended to more than two variables using the frequency tables tabulated in a multidimensional contingency table. An example of a three-dimensional contingency table for a , b , and c is shown in [Table 5.16](#). Each entry f_{ijk} in this table represents the number of transactions that contain a particular combination of items a , b , and c . For example, f_{101} is the number of transactions that contain a and c , but not b . On the other hand, a marginal frequency such as f_{1+1} is the number of transactions that contain a and c , irrespective of whether b is present in the transaction.

Table 5.16. Example of a three-dimensional contingency table.

c	b	b^-	
a	f_{111}	f_{101}	f_{1+1}
a^-	f_{011}	f_{001}	f_{0+1}
	f_{+11}	f_{+01}	f_{++1}
c	b	b^-	
a	f_{110}	f_{100}	f_{1+0}
a^-	f_{010}	f_{000}	f_{0+0}



Given a k -itemset $\{i_1, i_2, \dots, i_k\}$, the condition for statistical independence can be stated as follows:

$$f_{i1}i_2\dots i_k = f_{i1} + \dots + f_{i2} + \dots + \dots + f_{ik} - N_k - 1. \quad (5.11)$$

With this definition, we can extend objective measures such as interest factor and PS , which are based on deviations from statistical independence, to more than two variables:

$$I = N_k - 1 \times f_{i1}i_2\dots i_k f_{i1} + \dots + f_{i2} + \dots + \dots + f_{ik} PS = f_{i1}i_2\dots i_k N_k - f_{i1} + \dots + f_{i2} + \dots + \dots + f_{ik} N_k$$

Another approach is to define the objective measure as the maximum, minimum, or average value for the associations between pairs of items in a pattern. For example, given a k -itemset $X = \{i_1, i_2, \dots, i_k\}$, we may define the ϕ -coefficient for X as the average ϕ -coefficient between every pair of items (i_p, i_q) in X . However, because the measure considers only pairwise associations, it may not capture all the underlying relationships within a pattern. Also, care should be taken in using such alternate measures for more than two variables, since they may not always show the anti-monotone property in the same way as the support measure, making them unsuitable for mining patterns using the *Apriori* principle.

Analysis of multidimensional contingency tables is more complicated because of the presence of partial associations in the data. For example, some associations may appear or disappear when conditioned upon the value of certain variables. This problem is known as **Simpson's paradox** and is described in [Section 5.7.3](#). More sophisticated statistical techniques are

available to analyze such relationships, e.g., loglinear models, but these techniques are beyond the scope of this book.

5.7.3 Simpson's Paradox

It is important to exercise caution when interpreting the association between variables because the observed relationship may be influenced by the presence of other confounding factors, i.e., hidden variables that are not included in the analysis. In some cases, the hidden variables may cause the observed relationship between a pair of variables to disappear or reverse its direction, a phenomenon that is known as Simpson's paradox. We illustrate the nature of this paradox with the following example.

Consider the relationship between the sale of high-definition televisions (HDTV) and exercise machines, as shown in [Table 5.17](#). The rule $\{\text{HDTV}=\text{Yes}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}$ has a confidence of $99/180=55\%$ and the rule $\{\text{HDTV}=\text{No}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}$ has a confidence of $54/120=45\%$. Together, these rules suggest that customers who buy high-definition televisions are more likely to buy exercise machines than those who do not buy high-definition televisions.

Table 5.17. A two-way contingency table between the sale of high-definition television and exercise machine.

Buy HDTV	Buy Exercise Machine		
	Yes	No	
Yes	99	81	180
No	54	66	120

However, a deeper analysis reveals that the sales of these items depend on whether the customer is a college student or a working adult. **Table 5.18** summarizes the relationship between the sale of HDTVs and exercise machines among college students and working adults. Notice that the support counts given in the table for college students and working adults sum up to the frequencies shown in **Table 5.17**. Furthermore, there are more working adults than college students who buy these items. For college students:

Table 5.18. Example of a three-way contingency table.

Customer Group	Buy HDTV	Buy Exercise Machine		Total
		Yes	No	
College Students	Yes	1	9	10
	No	4	30	34
Working Adult	Yes	98	72	170
	No	50	36	86

$c(\{HDTV=Yes\} \rightarrow \{\text{Exercise machine}=Yes\}) = 1/10 = 10\%$, $c(\{HDTV=No\} \rightarrow \{\text{Exercise machine}=Yes\}) = 4/34 = 11.8\%$.

while for working adults:

$c(\{HDTV=Yes\} \rightarrow \{\text{Exercise machine}=Yes\}) = 98/170 = 57.7\%$, $c(\{HDTV=No\} \rightarrow \{\text{Exercise machine}=Yes\}) = 50/86 = 58.1\%$.

The rules suggest that, for each group, customers who do not buy high-definition televisions are more likely to buy exercise machines, which contradicts the previous conclusion when data from the two customer groups are pooled together. Even if alternative measures such as correlation, odds ratio, or interest are applied, we still find that the sale of HDTV and exercise machine is positively related in the combined data but is negatively related in the stratified data (see Exercise 21 on page 449). The reversal in the direction of association is known as Simpson's paradox.

The paradox can be explained in the following way. First, notice that most customers who buy HDTVs are working adults. This is reflected in the high confidence of the rule $\{\text{HDTV}=\text{Yes}\} \rightarrow \{\text{Working Adult}\}$ ($170/180=94.4\%$). Second, the high confidence of the rule $\{\text{Exercise machine}=\text{Yes}\} \rightarrow \{\text{Working Adult}\}$ ($148/153=96.7\%$) suggests that most customers who buy exercise machines are also working adults. Since working adults form the largest fraction of customers for both HDTVs and exercise machines, they both look related and the rule $\{\text{HDTV}=\text{Yes}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}$ turns out to be stronger in the combined data than what it would have been if the data is stratified. Hence, customer group acts as a *hidden* variable that affects both the fraction of customers who buy HDTVs and those who buy exercise machines. If we factor out the effect of the hidden variable by stratifying the data, we see that the relationship between buying HDTVs and buying exercise machines is not direct, but shows up as an indirect consequence of the effect of the hidden variable.

The Simpson's paradox can also be illustrated mathematically as follows.

Suppose

$$a/b < c/d \text{ and } p/q < r/s,$$

where a/b and p/q may represent the confidence of the rule $A \rightarrow B$ in two different strata, while c/d and r/s may represent the confidence of the rule $A^- \rightarrow B$ in the two strata. When the data is pooled together, the confidence values of the rules in the combined data are $(a+p)/(b+q)$ and $(c+r)/(d+s)$, respectively. Simpson's paradox occurs when

$$a+pb+q > c+rd+s,$$

thus leading to the wrong conclusion about the relationship between the variables. The lesson here is that proper stratification is needed to avoid generating spurious patterns resulting from Simpson's paradox. For example, market basket data from a major supermarket chain should be stratified according to store locations, while medical records from various patients should be stratified according to confounding factors such as age and gender.

5.8 Effect of Skewed Support Distribution

The performances of many association analysis algorithms are influenced by properties of their input data. For example, the computational complexity of the *Apriori* algorithm depends on properties such as the number of items in the data, the average transaction width, and the support threshold used. This Section examines another important property that has significant influence on the performance of association analysis algorithms as well as the quality of extracted patterns. More specifically, we focus on data sets with skewed support distributions, where most of the items have relatively low to moderate frequencies, but a small number of them have very high frequencies.

p	q	r
0	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Figure 5.29.

A transaction data set containing three items, p , q , and r , where p is a high support item and q and r are low support items.

Figure 5.29 shows an illustrative example of a data set that has a skewed support distribution of its items. While p has a high support of 83.3% in the data, q and r are low-support items with a support of 16.7%. Despite their low support, q and r always occur together in the limited number of transactions that they appear and hence are strongly related. A pattern mining algorithm therefore should report $\{q, r\}$ as interesting.

However, note that choosing the right support threshold for mining item-sets such as $\{q, r\}$ can be quite tricky. If we set the threshold too high (e.g., 20%),

then we may miss many interesting patterns involving low-support items such as $\{q, r\}$. Conversely, setting the support threshold too low can be detrimental to the pattern mining process for the following reasons. First, the computational and memory requirements of existing association analysis algorithms increase considerably with low support thresholds. Second, the number of extracted patterns also increases substantially with low support thresholds, which makes their analysis and interpretation difficult. In particular, we may extract many spurious patterns that relate a high-frequency item such as p to a low-frequency item such as q . Such patterns, which are called **cross-support** patterns, are likely to be spurious because the association between p and q is largely influenced by the frequent occurrence of p instead of the joint occurrence of p and q together. Because the support of $\{p, q\}$ is quite close to the support of $\{q, r\}$, we may easily select $\{p, q\}$ if we set the support threshold low enough to include $\{q, r\}$.

An example of a real data set that exhibits a skewed support distribution is shown in [Figure 5.30](#). The data, taken from the PUMS (Public Use Microdata Sample) census data, contains 49,046 records and 2113 asymmetric binary variables. We shall treat the asymmetric binary variables as items and records as transactions. While more than 80% of the items have support less than 1%, a handful of them have support greater than 90%. To understand the effect of skewed support distribution on frequent itemset mining, we divide the items into three groups, G1, G2, and G3, according to their support levels, as shown in [Table 5.19](#). We can see that more than 82% of items belong to G1 and have a support less than 1%. In market basket analysis, such low support items may correspond to expensive products (such as jewelry) that are seldom bought by customers, but whose patterns are still interesting to retailers. Patterns involving such low-support items, though meaningful, can easily be rejected by a frequent pattern mining algorithm with a high support threshold. On the other hand, setting a low support threshold may result in the extraction of spurious patterns that relate a high-frequency

item in G3 to a low-frequency item in G1. For example, at a support threshold equal to 0.05%, there are 18,847 frequent pairs involving items from G1 and G3. Out of these, 93% of them are cross-support patterns; i.e., the patterns contain items from both G1 and G3.

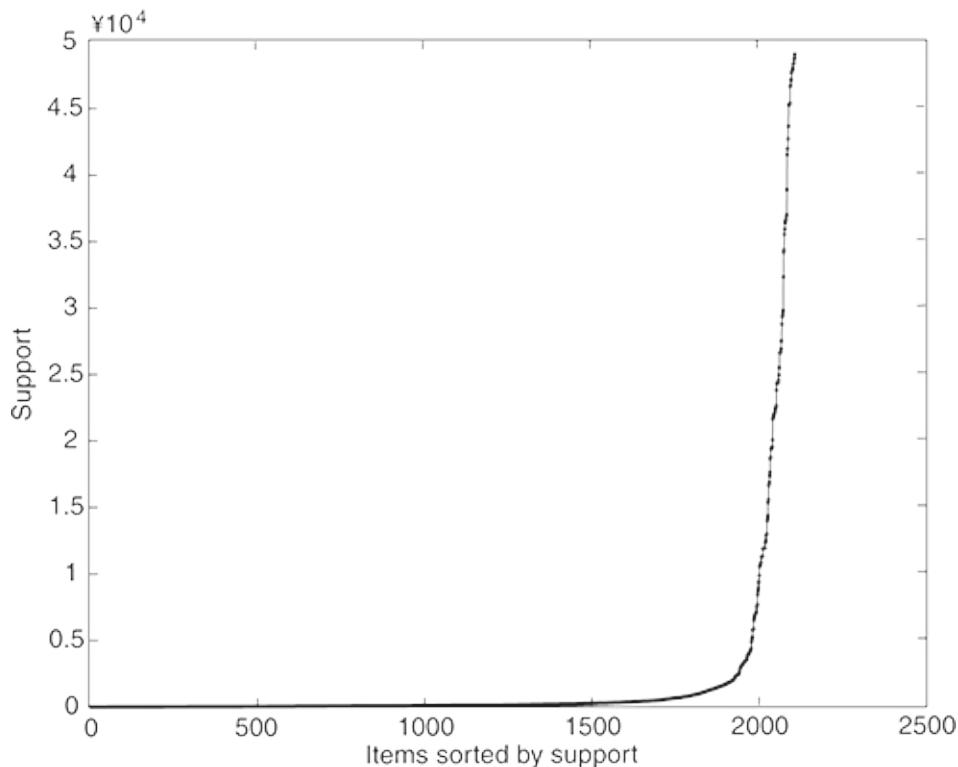


Figure 5.30.

Support distribution of items in the census data set.

Table 5.19. Grouping the items in the census data set based on their support values.

Group	G1	G2	G3
Support	<1%	1%–90%	>90%
Number of Items	1735	358	20

This example shows that a large number of weakly related cross-support patterns can be generated when the support threshold is sufficiently low. Note that finding interesting patterns in data sets with skewed support distributions is not just a challenge for the support measure, but similar statements can be made about many other objective measures discussed in the previous Sections. Before presenting a methodology for finding interesting patterns and pruning spurious ones, we formally define the concept of cross-support patterns.

Definition 5.9. (Cross-support Pattern.)

Let us define the support ratio, $r(X)$, of an itemset $X = \{i_1, i_2, \dots, i_k\}$ as

$$r(X) = \frac{\min[s(i_1), s(i_2), \dots, s(i_k)]}{\max[s(i_1), s(i_2), \dots, s(i_k)]} \quad (5.12)$$

Given a user-specified threshold hc , an itemset X is a cross-support pattern if $r(X) < hc$.

Example 5.4.

Suppose the support for milk is 70%, while the support for sugar is 10% and caviar is 0.04%. Given $hc=0.01$, the frequent itemset {milk, sugar, caviar} is a cross-support pattern because its support ratio is

$$r = \frac{\min[0.7, 0.1, 0.0004]}{\max[0.7, 0.1, 0.0004]} = \frac{0.0004}{0.7} = 0.00058 < 0.01.$$

Existing measures such as support and confidence may not be sufficient to eliminate cross-support patterns. For example, if we assume $hc=0.3$ for the data set presented in [Figure 5.29](#), the itemsets $\{p, q\}$, $\{p, r\}$, and $\{p, q, r\}$ are cross-support patterns because their support ratios, which are equal to 0.2, are less than the threshold hc . However, their supports are comparable to that of $\{q, r\}$, making it difficult to eliminate cross-support patterns without losing interesting ones using a support-based pruning strategy. Confidence pruning also does not help because the confidence of the rules extracted from cross-support patterns can be very high. For example, the confidence for $\{q\} \rightarrow \{p\}$ is 80% even though $\{p, q\}$ is a cross-support pattern. The fact that the cross-support pattern can produce a high confidence rule should not come as a surprise because one of its items (p) appears very frequently in the data. Therefore, p is expected to appear in many of the transactions that contain q . Meanwhile, the rule $\{q\} \rightarrow \{r\}$ also has high confidence even though $\{q, r\}$ is not a cross-support pattern. This example demonstrates the difficulty of using the confidence measure to distinguish between rules extracted from cross-support patterns and interesting patterns involving strongly connected but low-support items.

Even though the rule $\{q\} \rightarrow \{p\}$ has very high confidence, notice that the rule $\{p\} \rightarrow \{q\}$ has very low confidence because most of the transactions that contain p do not contain q . In contrast, the rule $\{r\} \rightarrow \{q\}$, which is derived from $\{q, r\}$, has very high confidence. This observation suggests that cross-support patterns can be detected by examining the lowest confidence rule that can be extracted from a given itemset. An approach for finding the rule with the lowest confidence given an itemset can be described as follows.

1. Recall the following anti-monotone property of confidence:

$$\text{conf}(\{i_1 i_2\} \rightarrow \{i_3, i_4, \dots, i_k\}) \leq \text{conf}(\{i_1 i_2 i_3\} \rightarrow \{i_4, i_5, \dots, i_k\}).$$

This property suggests that confidence never increases as we shift more items from the left- to the right-hand side of an association rule. Because of this property, the lowest confidence rule extracted from a frequent itemset contains only one item on its left-hand side. We denote the set of all rules with only one item on its left-hand side as R1.

2. Given a frequent itemset $\{i_1, i_2, \dots, i_k\}$, the rule

$$\{i_j\} \rightarrow \{i_1, i_2, \dots, i_{j-1}, i_{j+1}, \dots, i_k\}$$

has the lowest confidence in R1 if $s(i_j) = \max[s(i_1), s(i_2), \dots, s(i_k)]$. This follows directly from the definition of confidence as the ratio between the rule's support and the support of the rule antecedent. Hence, the confidence of a rule will be lowest when the support of the antecedent is highest.

3. Summarizing the previous points, the lowest confidence attainable from a frequent itemset $\{i_1, i_2, \dots, i_k\}$ is

$$s(\{i_1, i_2, \dots, i_k\}) \max[s(i_1), s(i_2), \dots, s(i_k)].$$

This expression is also known as the **h-confidence** or **all-confidence** measure. Because of the anti-monotone property of support, the numerator of the h-confidence measure is bounded by the minimum support of any item that appears in the frequent itemset. In other words, the h-confidence of an itemset $X = \{i_1, i_2, \dots, i_k\}$ must not exceed the following expression:

$$h\text{-confidence}(X) \leq \min[s(i_1), s(i_2), \dots, s(i_k)] \max[s(i_1), s(i_2), \dots, s(i_k)].$$

Note that the upper bound of h-confidence in the above equation is exactly same as support ratio (r) given in [Equation 5.12](#). Because the support ratio for a cross-support pattern is always less than hc , the h-confidence of the pattern is also guaranteed to be less than hc . Therefore, cross-support patterns can be eliminated by ensuring that the h-confidence values for the patterns exceed hc . As a final note, the advantages of using h-confidence go

beyond eliminating cross-support patterns. The measure is also anti-monotone, i.e.,

$$h\text{-confidence}(\{i_1, i_2, \dots, i_k\}) \geq h\text{-confidence}(\{i_1, i_2, \dots, i_{k+1}\}),$$

and thus can be incorporated directly into the mining algorithm. Furthermore, h -confidence ensures that the items contained in an itemset are strongly associated with each other. For example, suppose the h -confidence of an itemset X is 80%. If one of the items in X is present in a transaction, there is at least an 80% chance that the rest of the items in X also belong to the same transaction. Such strongly associated patterns involving low-support items are called **hyperclique patterns**.

Definition 5.10. (Hyperclique Pattern.)

An itemset X is a hyperclique pattern if $h\text{-confidence}(X) > hc$, where hc is a user-specified threshold.

5.9 Bibliographic Notes

The association rule mining task was first introduced by Agrawal et al. [324, 325] to discover interesting relationships among items in market basket transactions. Since its inception, extensive research has been conducted to address the various issues in association rule mining, from its fundamental concepts to its implementation and applications. **Figure 5.31** shows a taxonomy of the various research directions in this area, which is generally known as *association analysis*. As much of the research focuses on finding patterns that appear significantly often in the data, the area is also known as *frequent pattern mining*. A detailed review on some of the research topics in this area can be found in [362] and in [319].

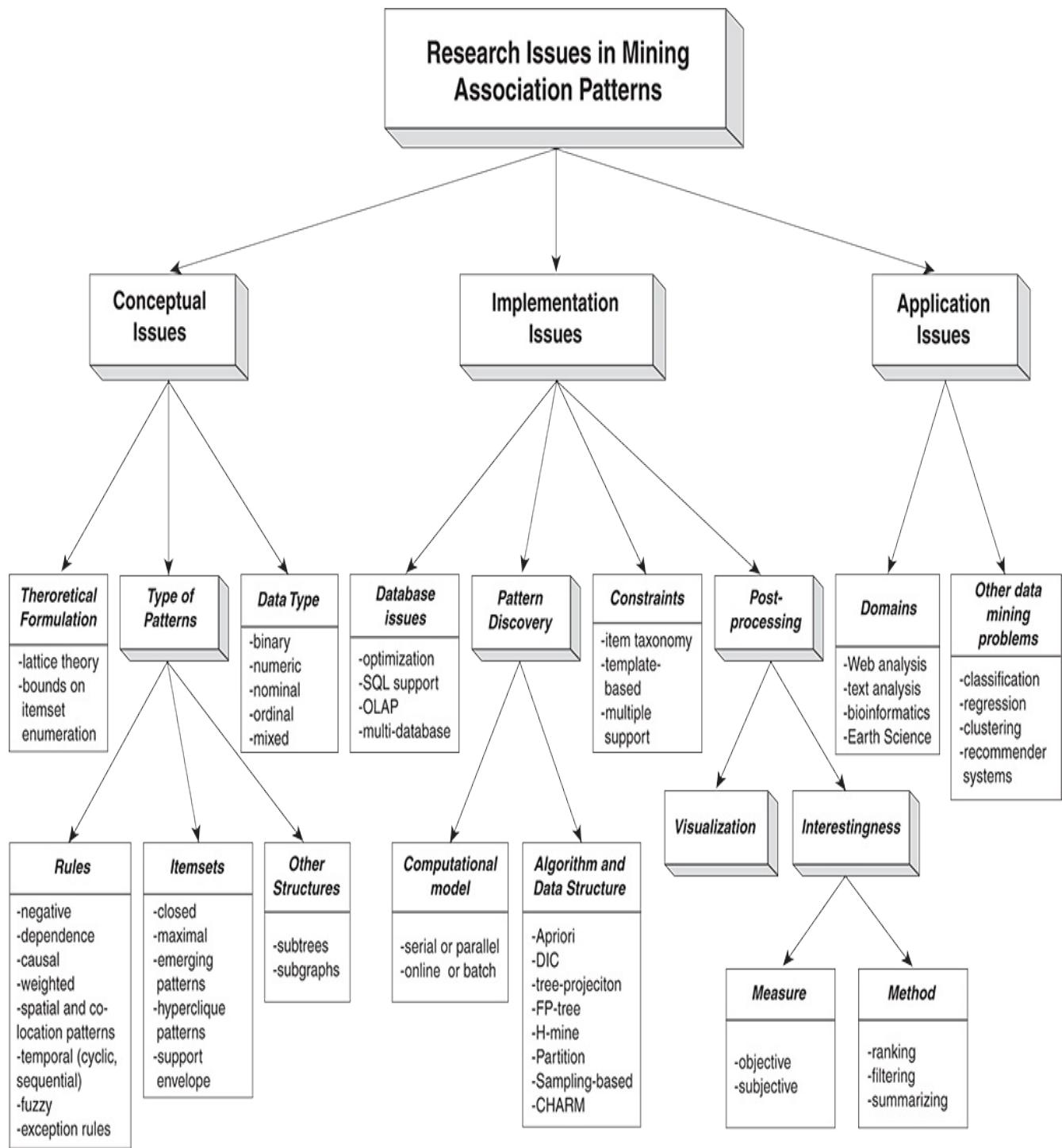


Figure 5.31.

An overview of the various research directions in association analysis.

Conceptual Issues

Research on the conceptual issues of association analysis has focused on developing a theoretical formulation of association analysis and extending the formulation to new types of patterns and going beyond asymmetric binary attributes.

Following the pioneering work by Agrawal et al. [324, 325], there has been a vast amount of research on developing a theoretical formulation for the association analysis problem. In [357], Gunopoulos et al. showed the connection between finding maximal frequent itemsets and the hypergraph transversal problem. An upper bound on the complexity of the association analysis task was also derived. Zaki et al. [454, 456] and Pasquier et al. [407] have applied formal concept analysis to study the frequent itemset generation problem. More importantly, such research has led to the development of a class of patterns known as closed frequent itemsets [456]. Friedman et al. [355] have studied the association analysis problem in the context of **bump hunting** in multidimensional space. Specifically, they consider frequent itemset generation as the task of finding high density regions in multidimensional space. Formalizing association analysis in a statistical learning framework is another active research direction [414, 435, 444] as it can help address issues related to identifying statistically significant patterns and dealing with uncertain data [320, 333, 343].

Over the years, the association rule mining formulation has been expanded to encompass other rule-based patterns, such as, profile association rules [321], cyclic association rules [403], fuzzy association rules [379], exception rules [431], negative association rules [336, 418], weighted association rules [338, 413], dependence rules [422], peculiar rules[462], inter-transaction association rules [353, 440], and partial classification rules [327, 397]. Additionally, the concept of frequent itemset has been extended to other types of patterns including closed itemsets [407, 456], maximal itemsets [330], hyperclique patterns [449], support envelopes [428], emerging patterns [347],

contrast sets [329], high-utility itemsets [340, 390], approximate or error-tolerant item-sets [358, 389, 451], and discriminative patterns [352, 401, 430]. Association analysis techniques have also been successfully applied to sequential [326, 426], spatial [371], and graph-based [374, 380, 406, 450, 455] data.

Substantial research has been conducted to extend the original association rule formulation to nominal [425], ordinal [392], interval [395], and ratio [356, 359, 425, 443, 461] attributes. One of the key issues is how to define the support measure for these attributes. A methodology was proposed by Steinbach et al. [429] to extend the traditional notion of support to more general patterns and attribute types.

Implementation Issues

Research activities in this area revolve around (1) integrating the mining capability into existing database technology, (2) developing efficient and scalable mining algorithms, (3) handling user-specified or domain-specific constraints, and (4) post-processing the extracted patterns.

There are several advantages to integrating association analysis into existing database technology. First, it can make use of the indexing and query processing capabilities of the database system. Second, it can also exploit the DBMS support for scalability, check-pointing, and parallelization [415]. The SETM algorithm developed by Houtsma et al. [370] was one of the earliest algorithms to support association rule discovery via SQL queries. Since then, numerous methods have been developed to provide capabilities for mining association rules in database systems. For example, the DMQL [363] and M-SQL [373] query languages extend the basic SQL with new operators for mining association rules. The Mine Rule operator [394] is an expressive SQL operator that can handle both clustered attributes and item hierarchies. Tsur et

al. [439] developed a generate-and-test approach called **query flocks** for mining association rules. A distributed OLAP-based infrastructure was developed by Chen et al. [341] for mining multilevel association rules.

Despite its popularity, the *Apriori* algorithm is computationally expensive because it requires making multiple passes over the transaction database. Its runtime and storage complexities were investigated by Dunkel and Soparkar [349]. The FP-growth algorithm was developed by Han et al. in [364]. Other algorithms for mining frequent itemsets include the DHP (dynamic hashing and pruning) algorithm proposed by Park et al. [405] and the Partition algorithm developed by Savasere et al [417]. A sampling-based frequent itemset generation algorithm was proposed by Toivonen [436]. The algorithm requires only a single pass over the data, but it can produce more candidate item-sets than necessary. The Dynamic Itemset Counting (DIC) algorithm [337] makes only 1.5 passes over the data and generates less candidate itemsets than the sampling-based algorithm. Other notable algorithms include the tree-projection algorithm [317] and H-Mine [408]. Survey articles on frequent itemset generation algorithms can be found in [322, 367]. A repository of benchmark data sets and software implementation of association rule mining algorithms is available at the Frequent Itemset Mining Implementations (FIMI) repository (<http://fimi.cs.helsinki.fi>).

Parallel algorithms have been developed to scale up association rule mining for handling big data [318, 360, 399, 420, 457]. A survey of such algorithms can be found in [453]. Online and incremental association rule mining algorithms have also been proposed by Hidber [365] and Cheung et al. [342]. More recently, new algorithms have been developed to speed up frequent itemset mining by exploiting the processing power of GPUs [459] and the MapReduce/Hadoop distributed computing framework [382, 384, 396]. For example, an implementation of frequent itemset mining for the Hadoop framework is available in the Apache Mahout software¹.

¹ <http://mahout.apache.org>

Srikant et al. [427] have considered the problem of mining association rules in the presence of Boolean constraints such as the following:

(Cookies \wedge Milk) \vee (descendants(Cookies) \wedge \neg ancestors(Wheat Bread))

Given such a constraint, the algorithm looks for rules that contain both cookies and milk, or rules that contain the descendent items of cookies but not ancestor items of wheat bread. Singh et al. [424] and Ng et al. [400] had also developed alternative techniques for constrained-based association rule mining. Constraints can also be imposed on the support for different itemsets. This problem was investigated by Wang et al. [442], Liu et al. in [387], and Seno et al. [419]. In addition, constraints arising from privacy concerns of mining sensitive data have led to the development of privacy-preserving frequent pattern mining techniques [334, 350, 441, 458].

One potential problem with association analysis is the large number of patterns that can be generated by current algorithms. To overcome this problem, methods to rank, summarize, and filter patterns have been developed. Toivonen et al. [437] proposed the idea of eliminating redundant rules using **structural rule covers** and grouping the remaining rules using clustering. Liu et al. [388] applied the statistical chi-square test to prune spurious patterns and summarized the remaining patterns using a subset of the patterns called **direction setting rules**. The use of objective measures to filter patterns has been investigated by many authors, including Brin et al. [336], Bayardo and Agrawal [331], Aggarwal and Yu [323], and DuMouchel and Pregibon[348]. The properties for many of these measures were analyzed by Piatetsky-Shapiro [410], Kamber and Singhal [376], Hilderman and Hamilton [366], and Tan et al. [433]. The grade-gender example used to highlight the importance of the row and column scaling invariance property

was heavily influenced by the discussion given in [398] by Mosteller. Meanwhile, the tea-coffee example illustrating the limitation of confidence was motivated by an example given in [336] by Brin et al. Because of the limitation of confidence, Brin et al. [336] had proposed the idea of using interest factor as a measure of interestingness. The all-confidence measure was proposed by Omiecinski [402]. Xiong et al. [449] introduced the cross-support property and showed that the all-confidence measure can be used to eliminate cross-support patterns. A key difficulty in using alternative objective measures besides support is their lack of a monotonicity property, which makes it difficult to incorporate the measures directly into the mining algorithms. Xiong et al. [447] have proposed an efficient method for mining correlations by introducing an upper bound function to the ϕ -coefficient. Although the measure is non-monotone, it has an upper bound expression that can be exploited for the efficient mining of strongly correlated item pairs.

Fabris and Freitas [351] have proposed a method for discovering interesting associations by detecting the occurrences of Simpson's paradox [423]. Megiddo and Srikant [393] described an approach for validating the extracted patterns using hypothesis testing methods. A resampling-based technique was also developed to avoid generating spurious patterns because of the multiple comparison problem. Bolton et al. [335] have applied the Benjamini-Hochberg [332] and Bonferroni correction methods to adjust the p-values of discovered patterns in market basket data. Alternative methods for handling the multiple comparison problem were suggested by Webb [445], Zhang et al. [460], and Llinares-Lopez et al. [391].

Application of subjective measures to association analysis has been investigated by many authors. Silberschatz and Tuzhilin [421] presented two principles in which a rule can be considered interesting from a subjective point of view. The concept of unexpected condition rules was introduced by Liu et al. in [385]. Cooley et al. [344] analyzed the idea of combining soft belief sets

using the Dempster-Shafer theory and applied this approach to identify contradictory and novel association patterns in web data. Alternative approaches include using Bayesian networks [375] and neighborhood-based information [346] to identify subjectively interesting patterns.

Visualization also helps the user to quickly grasp the underlying structure of the discovered patterns. Many commercial data mining tools display the complete set of rules (which satisfy both support and confidence threshold criteria) as a two-dimensional plot, with each axis corresponding to the antecedent or consequent itemsets of the rule. Hofmann et al. [368] proposed using Mosaic plots and Double Decker plots to visualize association rules. This approach can visualize not only a particular rule, but also the overall contingency table between itemsets in the antecedent and consequent parts of the rule. Nevertheless, this technique assumes that the rule consequent consists of only a single attribute.

Application Issues

Association analysis has been applied to a variety of application domains such as web mining [409, 432], document analysis [369], telecommunication alarm diagnosis [377], network intrusion detection [328, 345, 381], and bioinformatics [416, 446]. Applications of association and correlation pattern analysis to Earth Science studies have been investigated in [411, 412, 434]. Trajectory pattern mining [339, 372, 438] is another application of spatio-temporal association analysis to identify frequently traversed paths of moving objects.

Association patterns have also been applied to other learning problems such as classification [383, 386], regression [404], and clustering [361, 448, 452]. A comparison between classification and association rule mining was made by Freitas in his position paper [354]. The use of association patterns for

clustering has been studied by many authors including Han et al.[361], Kosters et al. [378], Yang et al. [452] and Xiong et al. [448].

Bibliography

- [317] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, 61(3):350–371, 2001.
- [318] R. C. Agarwal and J. C. Shafer. Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, March 1998.
- [319] C. Aggarwal and J. Han. *Frequent Pattern Mining*. Springer, 2014.
- [320] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent pattern mining with uncertain data. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 29–38, Paris, France, 2009.
- [321] C. C. Aggarwal, Z. Sun, and P. S. Yu. Online Generation of Profile Association Rules. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 129— 133, New York, NY, August 1996.
- [322] C. C. Aggarwal and P. S. Yu. Mining Large Itemsets for Association Rules. *Data Engineering Bulletin*, 21(1):23–31, March 1998.

- [323] C. C. Aggarwal and P. S. Yu. Mining Associations with the Collective Strength Approach. *IEEE Trans. on Knowledge and Data Engineering*, 13(6):863–873, January/February 2001.
- [324] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5:914–925, 1993.
- [325] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Intl. Conf. Management of Data*, pages 207–216, Washington, DC, 1993.
- [326] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of Intl. Conf. on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995.
- [327] K. Ali, S. Manganaris, and R. Srikant. Partial Classification using Association Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 115— 118, Newport Beach, CA, August 1997.
- [328] D. Barbará, J. Couto, S. Jajodia, and N. Wu. ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection. *SIGMOD Record*, 30(4):15–24, 2001.
- [329] S. D. Bay and M. Pazzani. Detecting Group Differences: Mining Contrast Sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001.

- [330] R. Bayardo. Efficiently Mining Long Patterns from Databases. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 85–93, Seattle, WA, June 1998.
- [331] R. Bayardo and R. Agrawal. Mining the Most Interesting Rules. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 145–153, San Diego, CA, August 1999.
- [332] Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal Royal Statistical Society B*, 57 (1):289–300, 1995.
- [333] T. Bernecker, H. Kriegel, M. Renz, F. Verhein, and A. Züle. Probabilistic frequent itemset mining in uncertain databases. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 119–128, Paris, France, 2009.
- [334] R. Bhaskar, S. Laxman, A. D. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 503–512, Washington, DC, 2010.
- [335] R. J. Bolton, D. J. Hand, and N. M. Adams. Determining Hit Rate in Pattern Search. In *Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining*, pages 36–48, London, UK, September 2002.

- [336] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proc. ACM SIGMOD Intl. Conf. Management of Data*, pages 265–276, Tucson, AZ, 1997.
- [337] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for market basket data. In *Proc. of 1997 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 255–264, Tucson, AZ, June 1997.
- [338] C. H. Cai, A. Fu, C. H. Cheng, and W. W. Kwong. Mining Association Rules with Weighted Items. In *Proc. of IEEE Intl. Database Engineering and Applications Symp.*, pages 68–77, Cardiff, Wales, 1998.
- [339] H. Cao, N. Mamoulis, and D. W. Cheung. Mining Frequent Spatio-Temporal Sequential Patterns. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 82–89, Houston, TX, 2005.
- [340] R. Chan, Q. Yang, and Y. Shen. Mining High Utility Itemsets. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 19–26, Melbourne, FL, 2003.
- [341] Q. Chen, U. Dayal, and M. Hsu. A Distributed OLAP infrastructure for E-Commerce. In *Proc. of the 4th IFCIS Intl. Conf. on Cooperative Information Systems*, pages 209— 220, Edinburgh, Scotland, 1999.
- [342] D. C. Cheung, S. D. Lee, and B. Kao. A General Incremental Technique for Maintaining Discovered Association Rules. In *Proc. of the 5th Intl. Conf.*

on Database Systems for Advanced Applications, pages 185–194, Melbourne, Australia, 1997.

- [343] C. K. Chui, B. Kao, and E. Hung. Mining Frequent Itemsets from Uncertain Data. In *Proceedings of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 47–58, Nanjing, China, 2007.
- [344] R. Cooley, P. N. Tan, and J. Srivastava. Discovery of Interesting Usage Patterns from Web Data. In M. Spiliopoulou and B. Masand, editors, *Advances in Web Usage Analysis and User Profiling*, volume 1836, pages 163–182. Lecture Notes in Computer Science, 2000.
- [345] P. Dokas, L. Ertöz, V. Kumar, A. Lazarevic, J. Srivastava, and P. N. Tan. Data Mining for Network Intrusion Detection. In *Proc. NSF Workshop on Next Generation Data Mining*, Baltimore, MD, 2002.
- [346] G. Dong and J. Li. Interestingness of discovered association rules in terms of neighborhood-based unexpectedness. In *Proc. of the 2nd Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 72–86, Melbourne, Australia, April 1998.
- [347] G. Dong and J. Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 43–52, San Diego, CA, August 1999.
- [348] W. DuMouchel and D. Pregibon. Empirical Bayes Screening for Multi-Item Associations. In *Proc. of the 7th Intl. Conf. on Knowledge Discovery*

and Data Mining, pages 67–76, San Francisco, CA, August 2001.

[349] B. Dunkel and N. Soparkar. Data Organization and Access for Efficient Data Mining. In *Proc. of the 15th Intl. Conf. on Data Engineering*, pages 522–529, Sydney, Australia, March 1999.

[350] A. V. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228, Edmonton, Canada, 2002.

[351] C. C. Fabris and A. A. Freitas. Discovering surprising patterns by detecting occurrences of Simpson's paradox. In *Proc. of the 19th SGES Intl. Conf. on Knowledge-Based Systems and Applied Artificial Intelligence*, pages 148–160, Cambridge, UK, December 1999.

[352] G. Fang, G. Pandey, W. Wang, M. Gupta, M. Steinbach, and V. Kumar. Mining Low-Support Discriminative Patterns from Dense and High-Dimensional Data. *IEEE Trans. Knowl. Data Eng.*, 24(2):279–294, 2012.

[353] L. Feng, H. J. Lu, J. X. Yu, and J. Han. Mining inter-transaction associations with templates. In *Proc. of the 8th Intl. Conf. on Information and Knowledge Management*, pages 225–233, Kansas City, Missouri, Nov 1999.

[354] A. A. Freitas. Understanding the crucial differences between classification and discovery of association rules—a position paper. *SIGKDD Explorations*, 2(1):65–69, 2000.

- [355] J. H. Friedman and N. I. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, April 1999.
- [356] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining Optimized Association Rules for Numeric Attributes. In *Proc. of the 15th Symp. on Principles of Database Systems*, pages 182–191, Montreal, Canada, June 1996.
- [357] D. Gunopulos, R. Kharden, H. Mannila, and H. Toivonen. Data Mining, Hypergraph Transversals, and Machine Learning. In *Proc. of the 16th Symp. on Principles of Database Systems*, pages 209–216, Tucson, AZ, May 1997.
- [358] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 301–309, Las Vegas, NV, 2008.
- [359] E. Han, G. Karypis, and V. Kumar. Min-apriori: An algorithm for finding association rules in data with continuous attributes. *Department of Computer Science and Engineering, University of Minnesota, Tech. Rep*, 1997.
- [360] E.-H. Han, G. Karypis, and V. Kumar. Scalable Parallel Data Mining for Association Rules. In *Proc. of 1997 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 277–288, Tucson, AZ, May 1997.

- [361] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering Based on Association Rule Hypergraphs. In *Proc. of the 1997 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Tucson, AZ, 1997.
- [362] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.
- [363] J. Han, Y. Fu, K. Koperski, W. Wang, and O. R. Zaïane. DMQL: A data mining query language for relational databases. In *Proc. of the 1996 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Montreal, Canada, June 1996.
- [364] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.
- [365] C. Hidber. Online Association Rule Mining. In *Proc. of 1999 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 145–156, Philadelphia PA, 1999.
- [366] R. J. Hilderman and H. J. Hamilton. *Knowledge Discovery and Measures of Interest*. Kluwer Academic Publishers, 2001.
- [367] J. Hipp, U. Guntzer, and G. Nakhaeizadeh. Algorithms for Association Rule Mining— A General Survey. *SigKDD Explorations*, 2(1):58–64, June

2000.

- [368] H. Hofmann, A. P. J. M. Siebes, and A. F. X. Wilhelm. Visualizing Association Rules with Interactive Mosaic Plots. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 227–235, Boston, MA, August 2000.
- [369] J. D. Holt and S. M. Chung. Efficient Mining of Association Rules in Text Databases. In *Proc. of the 8th Intl. Conf. on Information and Knowledge Management*, pages 234–242, Kansas City, Missouri, 1999.
- [370] M. Houtsma and A. Swami. Set-oriented Mining for Association Rules in Relational Databases. In *Proc. of the 11th Intl. Conf. on Data Engineering*, pages 25–33, Taipei, Taiwan, 1995.
- [371] Y. Huang, S. Shekhar, and H. Xiong. Discovering Co-location Patterns from SpatialDatasets: A General Approach. *IEEE Trans. on Knowledge and Data Engineering*, 16 (12):1472–1485, December 2004.
- [372] S. Hwang, Y. Liu, J. Chiu, and E. Lim. Mining Mobile Group Patterns: A Trajectory-Based Approach. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 713–718, Hanoi, Vietnam, 2005.
- [373] T. Imielinski, A. Virmani, and A. Abdulghani. DataMine: Application Programming Interface and Query Language for Database Mining. In *Proc. of the 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 256–262, Portland, Oregon, 1996.

- [374] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data. In *Proc. of the 4th European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 13–23, Lyon, France, 2000.
- [375] S. Jaroszewicz and D. Simovici. Interestingness of Frequent Itemsets Using Bayesian Networks as Background Knowledge. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 178–186, Seattle, WA, August 2004.
- [376] M. Kamber and R. Shinghal. Evaluating the Interestingness of Characteristic Rules. In *Proc. of the 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 263–266, Portland, Oregon, 1996.
- [377] M. Klemettinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases*. PhD thesis, University of Helsinki, 1999.
- [378] W. A. Kosters, E. Marchiori, and A. Oerlemans. Mining Clusters with Association Rules. In *The 3rd Symp. on Intelligent Data Analysis (IDA99)*, pages 39–50, Amsterdam, August 1999.
- [379] C. M. Kuok, A. Fu, and M. H. Wong. Mining Fuzzy Association Rules in Databases. *ACM SIGMOD Record*, 27(1):41–46, March 1998.
- [380] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 313–320, San Jose, CA,

November 2001.

- [381] W. Lee, S. J. Stolfo, and K. W. Mok. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review*, 14(6):533–567, 2000.
- [382] N. Li, L. Zeng, Q. He, and Z. Shi. Parallel Implementation of Apriori Algorithm Based on MapReduce. In *Proceedings of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 236–241, Kyoto, Japan, 2012.
- [383] W. Li, J. Han, and J. Pei. CMAR: Accurate and Efficient Classification Based on Multiple Class-association Rules. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 369–376, San Jose, CA, 2001.
- [384] M. Lin, P. Lee, and S. Hsueh. Apriori-based frequent itemset mining algorithms on MapReduce. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, pages 26–30, Kuala Lumpur, Malaysia, 2012.
- [385] B. Liu, W. Hsu, and S. Chen. Using General Impressions to Analyze Discovered Classification Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 31–36, Newport Beach, CA, August 1997.
- [386] B. Liu, W. Hsu, and Y. Ma. Integrating Classification and Association Rule Mining. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 80–86, New York, NY, August 1998.

- [387] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 125— 134, San Diego, CA, August 1999.
- [388] B. Liu, W. Hsu, and Y. Ma. Pruning and Summarizing the Discovered Associations. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 125–134, San Diego, CA, August 1999.
- [389] J. Liu, S. Paulsen, W. Wang, A. B. Nobel, and J. Prins. Mining Approximate Frequent Itemsets from Noisy Data. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 721–724, Houston, TX, 2005.
- [390] Y. Liu, W.-K. Liao, and A. Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 689–695, Hanoi, Vietnam, 2005.
- [391] F. Llinares-López, M. Sugiyama, L. Papaxanthos, and K. M. Borgwardt. Fast and Memory-Efficient Significant Pattern Mining via Permutation Testing. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 725–734, Sydney, Australia, 2015.
- [392] A. Marcus, J. I. Maletic, and K.-I. Lin. Ordinal association rules for error identification in data sets. In *Proc. of the 10th Intl. Conf. on Information and Knowledge Management*, pages 589–591, Atlanta, GA, October 2001.

- [393] N. Megiddo and R. Srikant. Discovering Predictive Association Rules. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 274–278, New York, August 1998.
- [394] R. Meo, G. Psaila, and S. Ceri. A New SQL-like Operator for Mining Association Rules. In *Proc. of the 22nd VLDB Conf.*, pages 122–133, Bombay, India, 1996.
- [395] R. J. Miller and Y. Yang. Association Rules over Interval Data. In *Proc. of 1997 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 452–461, Tucson, AZ, May 1997.
- [396] S. Moens, E. Aksehirli, and B. Goethals. Frequent Itemset Mining for Big Data. In *Proceedings of the 2013 IEEE International Conference on Big Data*, pages 111–118, Santa Clara, CA, 2013.
- [397] Y. Morimoto, T. Fukuda, H. Matsuzawa, T. Tokuyama, and K. Yoda. Algorithms for mining association rules for binary segmentations of huge categorical databases. In *Proc. of the 24th VLDB Conf.*, pages 380–391, New York, August 1998.
- [398] F. Mosteller. Association and Estimation in Contingency Tables. *JASA*, 63:1–28, 1968.
- [399] A. Mueller. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, August 1995.

- [400] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory Mining and Pruning Optimizations of Constrained Association Rules. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 13–24, Seattle, WA, June 1998.
- [401] P. K. Novak, N. Lavrač, and G. I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10(Feb):377–403, 2009.
- [402] E. Omiecinski. Alternative Interest Measures for Mining Associations in Databases. *IEEE Trans. on Knowledge and Data Engineering*, 15(1):57–69, January/February 2003.
- [403] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic Association Rules. In *Proc. of the 14th Intl. Conf. on Data Eng.*, pages 412–421, Orlando, FL, February 1998.
- [404] A. Ozgur, P. N. Tan, and V. Kumar. RBA: An Integrated Framework for Regression based on Association Rules. In *Proc. of the SIAM Intl. Conf. on Data Mining*, pages 210–221, Orlando, FL, April 2004.
- [405] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD Record*, 25(2):175–186, 1995.
- [406] S. Parthasarathy and M. Coatney. Efficient Discovery of Common Substructures in Macromolecules. In *Proc. of the 2002 IEEE Intl. Conf. on*

Data Mining, pages 362— 369, Maebashi City, Japan, December 2002.

- [407] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of the 7th Intl. Conf. on Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, January 1999.
- [408] J. Pei, J. Han, H. J. Lu, S. Nishio, and S. Tang. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 441–448, San Jose, CA, November 2001.
- [409] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining Access Patterns Efficiently from Web Logs. In *Proc. of the 4th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 396–407, Kyoto, Japan, April 2000.
- [410] G. Piatetsky-Shapiro. Discovery, Analysis and Presentation of Strong Rules. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. MIT Press, Cambridge, MA, 1991.
- [411] C. Potter, S. Klooster, M. Steinbach, P. N. Tan, V. Kumar, S. Shekhar, and C. Carvalho. Understanding Global Teleconnections of Climate to Regional Model Estimates of Amazon Ecosystem Carbon Fluxes. *Global Change Biology*, 10(5):693— 703, 2004.

- [412] C. Potter, S. Klooster, M. Steinbach, P. N. Tan, V. Kumar, S. Shekhar, R. Myneni, and R. Nemani. Global Teleconnections of Ocean Climate to Terrestrial Carbon Flux. *Journal of Geophysical Research*, 108(D17), 2003.
- [413] G. D. Ramkumar, S. Ranka, and S. Tsur. Weighted association rules: Model and algorithm. In *Proc. ACM SIGKDD*, 1998.
- [414] M. Riondato and F. Vandin. Finding the True Frequent Itemsets. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 497–505, Philadelphia, PA, 2014.
- [415] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating Mining with Relational Database Systems: Alternatives and Implications. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 343–354, Seattle, WA, 1998.
- [416] K. Satou, G. Shibayama, T. Ono, Y. Yamamura, E. Furuichi, S. Kuhara, and T. Takagi. Finding Association Rules on Heterogeneous Genome Data. In *Proc. of the Pacific Symp. on Biocomputing*, pages 397–408, Hawaii, January 1997.
- [417] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. of the 21st Int. Conf. on Very Large Databases (VLDB'95)*, pages 432–444, Zurich, Switzerland, September 1995.

- [418] A. Savasere, E. Omiecinski, and S. Navathe. Mining for Strong Negative Associations in a Large Database of Customer Transactions. In *Proc. of the 14th Intl. Conf. on Data Engineering*, pages 494–502, Orlando, Florida, February 1998.
- [419] M. Seno and G. Karypis. LPMiner: An Algorithm for Finding Frequent Itemsets Using Length-Decreasing Support Constraint. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 505–512, San Jose, CA, November 2001.
- [420] T. Shintani and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proc of the 4th Intl. Conf. on Parallel and Distributed Info. Systems*, pages 19–30, Miami Beach, FL, December 1996.
- [421] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowledge and Data Engineering*, 8(6):970–974, 1996.
- [422] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1): 39–68, 1998.
- [423] E.-H. Simpson. The Interpretation of Interaction in Contingency Tables. *Journal of the Royal Statistical Society, B*(13):238–241, 1951.
- [424] L. Singh, B. Chen, R. Haight, and P. Scheuermann. An Algorithm for Constrained Association Rule Mining in Semi-structured Data. In *Proc. of*

the 3rd Pacific-Asia Conf. on Knowledge Discovery and Data Mining, pages 148–158, Beijing, China, April 1999.

- [425] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of 1996 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 1–12, Montreal, Canada, 1996.
- [426] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the 5th Intl Conf. on Extending Database Technology (EDBT'96)*, pages 18–32, Avignon, France, 1996.
- [427] R. Srikant, Q. Vu, and R. Agrawal. Mining Association Rules with Item Constraints. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 67–73, Newport Beach, CA, August 1997.
- [428] M. Steinbach, P. N. Tan, and V. Kumar. Support Envelopes: A Technique for Exploring the Structure of Association Patterns. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 296–305, Seattle, WA, August 2004.
- [429] M. Steinbach, P. N. Tan, H. Xiong, and V. Kumar. Extending the Notion of Support. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 689–694, Seattle, WA, August 2004.
- [430] M. Steinbach, H. Yu, G. Fang, and V. Kumar. Using constraints to generate and explore higher order discriminative patterns. *Advances in Knowledge Discovery and Data Mining*, pages 338–350, 2011.

- [431] E. Suzuki. Autonomous Discovery of Reliable Exception Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 259–262, Newport Beach, CA, August 1997.
- [432] P. N. Tan and V. Kumar. Mining Association Patterns in Web Usage Data. In *Proc. of the Intl. Conf. on Advances in Infrastructure for e-Business, e-Education, e-Science and e-Medicine on the Internet*, L'Aquila, Italy, January 2002.
- [433] P. N. Tan, V. Kumar, and J. Srivastava. Selecting the Right Interestingness Measure for Association Patterns. In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 32–41, Edmonton, Canada, July 2002.
- [434] P. N. Tan, M. Steinbach, V. Kumar, S. Klooster, C. Potter, and A. Torregrosa. Finding Spatio-Temporal Patterns in Earth Science Data. In *KDD 2001 Workshop on Temporal Data Mining*, San Francisco, CA, 2001.
- [435] N. Tatti. Probably the best itemsets. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 293–302, Washington, DC, 2010.
- [436] H. Toivonen. Sampling Large Databases for Association Rules. In *Proc. of the 22nd VLDB Conf.*, pages 134–145, Bombay, India, 1996.
- [437] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hatonen, and H. Mannila. Pruning and Grouping Discovered Association Rules. In *ECML-95*

Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, pages 47–52, Heraklion, Greece, April 1995.

- [438] I. Tsoukatos and D. Gunopulos. Efficient mining of spatiotemporal patterns. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 425–442, 2001.
- [439] S. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query Flocks: A Generalization of Association Rule Mining. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 1–12, Seattle, WA, June 1998.
- [440] A. Tung, H. J. Lu, J. Han, and L. Feng. Breaking the Barrier of Transactions: Mining Inter-Transaction Association Rules. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 297–301, San Diego, CA, August 1999.
- [441] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, Edmonton, Canada, 2002.
- [442] K. Wang, Y. He, and J. Han. Mining Frequent Itemsets Using Support Constraints. In *Proc. of the 26th VLDB Conf.*, pages 43–52, Cairo, Egypt, September 2000.
- [443] K. Wang, S. H. Tay, and B. Liu. Interestingness-Based Interval Merger for Numeric Association Rules. In *Proc. of the 4th Intl. Conf. on Knowledge*

Discovery and Data Mining, pages 121–128, New York, NY, August 1998.

- [444] L. Wang, R. Cheng, S. D. Lee, and D. W. Cheung. Accelerating probabilistic frequent itemset mining: a model-based approach. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management*, pages 429–438, 2010.
- [445] G. I. Webb. Preliminary investigations into statistically valid exploratory rule discovery. In *Proc. of the Australasian Data Mining Workshop (AusDM03)*, Canberra, Australia, December 2003.
- [446] H. Xiong, X. He, C. Ding, Y. Zhang, V. Kumar, and S. R. Holbrook. Identification of Functional Modules in Protein Complexes via Hyperclique Pattern Discovery. In *Proc. of the Pacific Symposium on Biocomputing, (PSB 2005)*, Maui, January 2005.
- [447] H. Xiong, S. Shekhar, P. N. Tan, and V. Kumar. Exploiting a Support-based Upper Bound of Pearson's Correlation Coefficient for Efficiently Identifying Strongly Correlated Pairs. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 334–343, Seattle, WA, August 2004.
- [448] H. Xiong, M. Steinbach, P. N. Tan, and V. Kumar. HICAP: Hierarchical Clustering with Pattern Preservation. In *Proc. of the SIAM Intl. Conf. on Data Mining*, pages 279–290, Orlando, FL, April 2004.
- [449] H. Xiong, P. N. Tan, and V. Kumar. Mining Strong Affinity Association Patterns in Data Sets with Skewed Support Distribution. In *Proc. of the*

2003 IEEE Intl. Conf. on Data Mining, pages 387–394, Melbourne, FL, 2003.

- [450] X. Yan and J. Han. gSpan: Graph-based Substructure Pattern Mining. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 721–724, Maebashi City, Japan, December 2002.
- [451] C. Yang, U. M. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 194–203, , San Francisco, CA, 2001.
- [452] C. Yang, U. M. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. of the 7th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 194–203, San Francisco, CA, August 2001.
- [453] M. J. Zaki. Parallel and Distributed Association Mining: A Survey. *IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining*, 7(4):14–25, December 1999.
- [454] M. J. Zaki. Generating Non-Redundant Association Rules. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 34–43, Boston, MA, August 2000.
- [455] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 71–80, Edmonton, Canada, July 2002.

- [456] M. J. Zaki and M. Orihara. Theoretical foundations of association rules. In *Proc. of the 1998 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Seattle, WA, June 1998.
- [457] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 283–286, Newport Beach, CA, August 1997.
- [458] C. Zeng, J. F. Naughton, and J. Cai. On differentially private frequent itemset mining. *Proceedings of the VLDB Endowment*, 6(1):25–36, 2012.
- [459] F. Zhang, Y. Zhang, and J. Bakos. GPAPriori: GPU-Accelerated Frequent Itemset Mining. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, pages 590–594, Austin, TX, 2011.
- [460] H. Zhang, B. Padmanabhan, and A. Tuzhilin. On the Discovery of Significant Statistical Quantitative Rules. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 374–383, Seattle, WA, August 2004.
- [461] Z. Zhang, Y. Lu, and B. Zhang. An Effective Partitioning-Combining Algorithm for Discovering Quantitative Association Rules. In *Proc. of the 1st Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, Singapore, 1997.

[462] N. Zhong, Y. Y. Yao, and S. Ohsuga. Peculiarity Oriented Multi-database Mining. In *Proc. of the 3rd European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 136–146, Prague, Czech Republic, 1999.

5.10 Exercises

1. For each of the following questions, provide an example of an association rule from the market basket domain that satisfies the following conditions. Also, describe whether such rules are subjectively interesting.
 - a. A rule that has high support and high confidence.
 - b. A rule that has reasonably high support but low confidence.
 - c. A rule that has low support and low confidence.
 - d. A rule that has low support and high confidence.

2. Consider the data set shown in **Table 5.20**.

Table 5.20. Example of market basket transactions.

Customer ID	Transaction ID	Items Bought
1	0001	{a, d, e}
1	0024	{a, b, c, e}
2	0012	{a, b, d, e}
2	0031	{a, c, d, e}
3	0015	{b, c e}
3	0022	{b, d, e}
4	0029	{c d}
4	0040	{a, b, c}

5	0033	$\{a, d, e\}$
5	0038	$\{a, b, e\}$

- a. Compute the support for itemsets $\{e\}$, $\{b, d\}$, and $\{b, d, e\}$ by treating each transaction ID as a market basket.
- b. Use the results in part (a) to compute the confidence for the association rules $\{b, d\} \rightarrow \{e\}$ and $\{e\} \rightarrow \{b, d\}$. Is confidence a symmetric measure?
- c. Repeat part (a) by treating each customer ID as a market basket. Each item should be treated as a binary variable (1 if an item appears in at least one transaction bought by the customer, and 0 otherwise).
- d. Use the results in part (c) to compute the confidence for the association rules $\{b, d\} \rightarrow \{e\}$ and $\{e\} \rightarrow \{b, d\}$.
- e. Suppose s_1 and c_1 are the support and confidence values of an association rule r when treating each transaction ID as a market basket. Also, let s_2 and c_2 be the support and confidence values of r when treating each customer ID as a market basket. Discuss whether there are any relationships between s_1 and s_2 or c_1 and c_2 .

3.

- a. What is the confidence for the rules $\emptyset \rightarrow A$ and $A \rightarrow \emptyset$?
- b. Let c_1 , c_2 , and c_3 be the confidence values of the rules $\{p\} \rightarrow \{q\}$, $\{p\} \rightarrow \{q, r\}$, and $\{p, r\} \rightarrow \{q\}$, respectively. If we assume that c_1 , c_2 , and c_3 have different values, what are the possible relationships that may exist among c_1 , c_2 , and c_3 ? Which rule has the lowest confidence?
- c. Repeat the analysis in part (b) assuming that the rules have identical support. Which rule has the highest confidence?

- d. Transitivity: Suppose the confidence of the rules $A \rightarrow B$ and $B \rightarrow C$ are larger than some threshold, $minconf$. Is it possible that $A \rightarrow C$ has a confidence less than $minconf$?
4. For each of the following measures, determine whether it is monotone, anti-monotone, or non-monotone (i.e., neither monotone nor anti-monotone).

Example: Support, $s=\sigma(x)|T|$ is anti-monotone because $s(X) \geq s(Y)$ whenever $X \subset Y$.

- a. A characteristic rule is a rule of the form $\{p\} \rightarrow \{q_1, q_2, \dots, q_n\}$, where the rule antecedent contains only a single item. An itemset of size k can produce up to k characteristic rules. Let ζ be the minimum confidence of all characteristic rules generated from a given itemset:

$$\zeta(\{p_1, p_2, \dots, p_k\}) = \min[c(\{p_1\} \rightarrow \{p_2, p_3, \dots, p_k\}), \dots, c(\{p_k\} \rightarrow \{p_1, p_2, \dots, p_{k-1}\})]$$

Is ζ monotone, anti-monotone, or non-monotone?

- b. A discriminant rule is a rule of the form $\{p_1, p_2, \dots, p_n\} \rightarrow \{q\}$, where the rule consequent contains only a single item. An itemset of size k can produce up to k discriminant rules. Let η be the minimum confidence of all discriminant rules generated from a given itemset:

$$\eta(\{p_1, p_2, \dots, p_k\}) = \min[c(\{p_2, p_3, \dots, p_k\} \rightarrow \{p_1\}), \dots, c(\{p_1, p_2, \dots, p_{k-1}\} \rightarrow \{p_k\})]$$

Is η monotone, anti-monotone, or non-monotone?

- c. Repeat the analysis in parts (a) and (b) by replacing the min function with a max function.

5. Prove [Equation 5.3](#). (Hint: First, count the number of ways to create an itemset that forms the left-hand side of the rule. Next, for each size k itemset selected for the left-hand side, count the number of ways to choose the remaining $d-k$ items to form the right-hand side of the rule.) Assume that neither of the itemsets of a rule are empty.

6. Consider the market basket transactions shown in [Table 5.21](#).

- What is the maximum number of association rules that can be extracted from this data (including rules that have zero support)?
- What is the maximum size of frequent itemsets that can be extracted (assuming $\text{minsup} > 0$)?

Table 5.21. Market basket transactions.

Transaction ID	Items Bought
1	{Milk, Beer, Diapers}
2	{Bread, Butter, Milk}
3	{Milk, Diapers, Cookies}
4	{Bread, Butter, Cookies}
5	{Beer, Cookies, Diapers}
6	{Milk, Diapers, Bread, Butter}
7	{Bread, Butter, Diapers}
8	{Beer, Diapers}
9	{Milk, Diapers, Bread, Butter}
10	{Beer, Cookies}

- c. Write an expression for the maximum number of size-3 itemsets that can be derived from this data set.
 - d. Find an itemset (of size 2 or larger) that has the largest support.
 - e. Find a pair of items, a and b , such that the rules $\{a\} \rightarrow \{b\}$ and $\{b\} \rightarrow \{a\}$ have the same confidence.
7. Show that if a candidate k -itemset X has a subset of size less than $k-1$ that is infrequent, then at least one of the $(k-1)$ -size subsets of X is necessarily infrequent.
8. Consider the following set of frequent 3-itemsets:
- $\{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 5\}, \{1, 3, 4\}, \{1, 3, 5\}, \{2, 3, 4\}, \{2, 3, 5\}, \{3, 4, 5\}$.
- Assume that there are only five items in the data set.
- a. List all candidate 4-itemsets obtained by a candidate generation procedure using the $F_{k-1} \times F_1$ merging strategy.
 - b. List all candidate 4-itemsets obtained by the candidate generation procedure in *Apriori*.
 - c. List all candidate 4-itemsets that survive the candidate pruning step of the *Apriori* algorithm.
9. The *Apriori* algorithm uses a generate-and-count strategy for deriving frequent itemsets. Candidate itemsets of size $k+1$ are created by joining a pair of frequent itemsets of size k (this is known as the candidate generation step). A candidate is discarded if any one of its subsets is found to be infrequent during the candidate pruning step. Suppose the *Apriori* algorithm is applied to the data set shown in [Table 5.22](#) with $\text{minsup}=30\%$, i.e., any itemset occurring in less than 3 transactions is considered to be infrequent.

Table 5.22. Example of market basket transactions.

Transaction ID	Items Bought
1	{a, b, d, e}
2	{b, c, d}
3	{a, b, d, e}
4	{a, c, d, e}
5	{b, c, d, e}
6	{b, d, e}
7	{c, d}
8	{a, b, c}
9	{a, d, e}
10	{b, d}

- a. Draw an itemset lattice representing the data set given in [Table 5.22](#). Label each node in the lattice with the following letter(s):

- **N:** If the itemset is not considered to be a candidate itemset by the *Apriori* algorithm. There are two reasons for an itemset not to be considered as a candidate itemset: (1) it is not generated at all during the candidate generation step, or (2) it is generated during the candidate generation step but is subsequently removed during the candidate pruning step because one of its subsets is found to be infrequent.

- **F:** If the candidate itemset is found to be frequent by the *Apriori* algorithm.
 - **I:** If the candidate itemset is found to be infrequent after support counting.
- b. What is the percentage of frequent itemsets (with respect to all itemsets in the lattice)?
- c. What is the pruning ratio of the *Apriori* algorithm on this data set? (Pruning ratio is defined as the percentage of itemsets not considered to be a candidate because (1) they are not generated during candidate generation or (2) they are pruned during the candidate pruning step.)
- d. What is the false alarm rate (i.e., percentage of candidate itemsets that are found to be infrequent after performing support counting)?
10. The *Apriori* algorithm uses a hash tree data structure to efficiently count the support of candidate itemsets. Consider the hash tree for candidate 3-itemsets shown in [Figure 5.32](#).

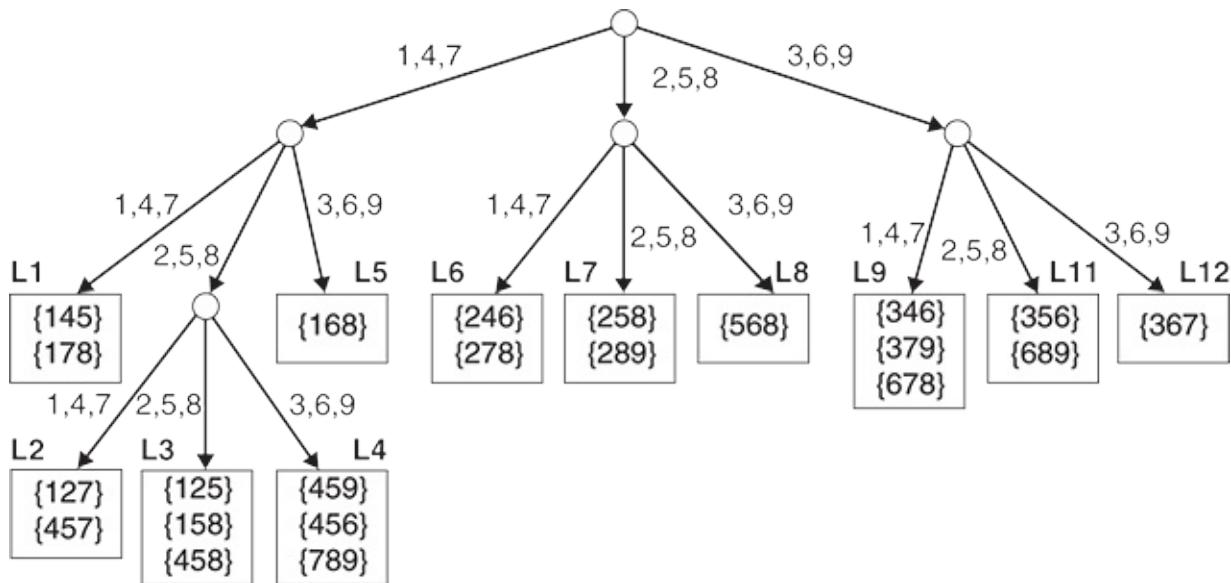


Figure 5.32.

An example of a hash tree structure.

- a. Given a transaction that contains items $\{1, 3, 4, 5, 8\}$, which of the hash tree leaf nodes will be visited when finding the candidates of the transaction?
 - b. Use the visited leaf nodes in part (a) to determine the candidate itemsets that are contained in the transaction $\{1, 3, 4, 5, 8\}$.
11. Consider the following set of candidate 3-itemsets:

$\{1, 2, 3\}, \{1, 2, 6\}, \{1, 3, 4\}, \{2, 3, 4\}, \{2, 4, 5\}, \{3, 4, 6\}, \{4, 5, 6\}$

- a. Construct a hash tree for the above candidate 3-itemsets. Assume the tree uses a hash function where all odd-numbered items are hashed to the left child of a node, while the even-numbered items are hashed to the right child. A candidate k -itemset is inserted into the tree by hashing on each successive item in the candidate and then following the appropriate branch of the tree according to the hash value. Once a leaf node is reached, the candidate is inserted based on one of the following conditions:

Condition 1: If the depth of the leaf node is equal to k (the root is assumed to be at depth 0), then the candidate is inserted regardless of the number of itemsets already stored at the node.

Condition 2: If the depth of the leaf node is less than k , then the candidate can be inserted as long as the number of itemsets stored at the node is less than *maxsize*. Assume *maxsize*=2 for this question.

Condition 3: If the depth of the leaf node is less than k and the number of itemsets stored at the node is equal to *maxsize*, then the leaf node is converted into an internal node. New leaf nodes are created as children

of the old leaf node. Candidate itemsets previously stored in the old leaf node are distributed to the children based on their hash values. The new candidate is also hashed to its appropriate leaf node.

- b. How many leaf nodes are there in the candidate hash tree? How many internal nodes are there?
 - c. Consider a transaction that contains the following items: {1, 2, 3, 5, 6}. Using the hash tree constructed in part (a), which leaf nodes will be checked against the transaction? What are the candidate 3-itemsets contained in the transaction?
12. Given the lattice structure shown in [Figure 5.33](#) and the transactions given in [Table 5.22](#), label each node with the following letter(s):

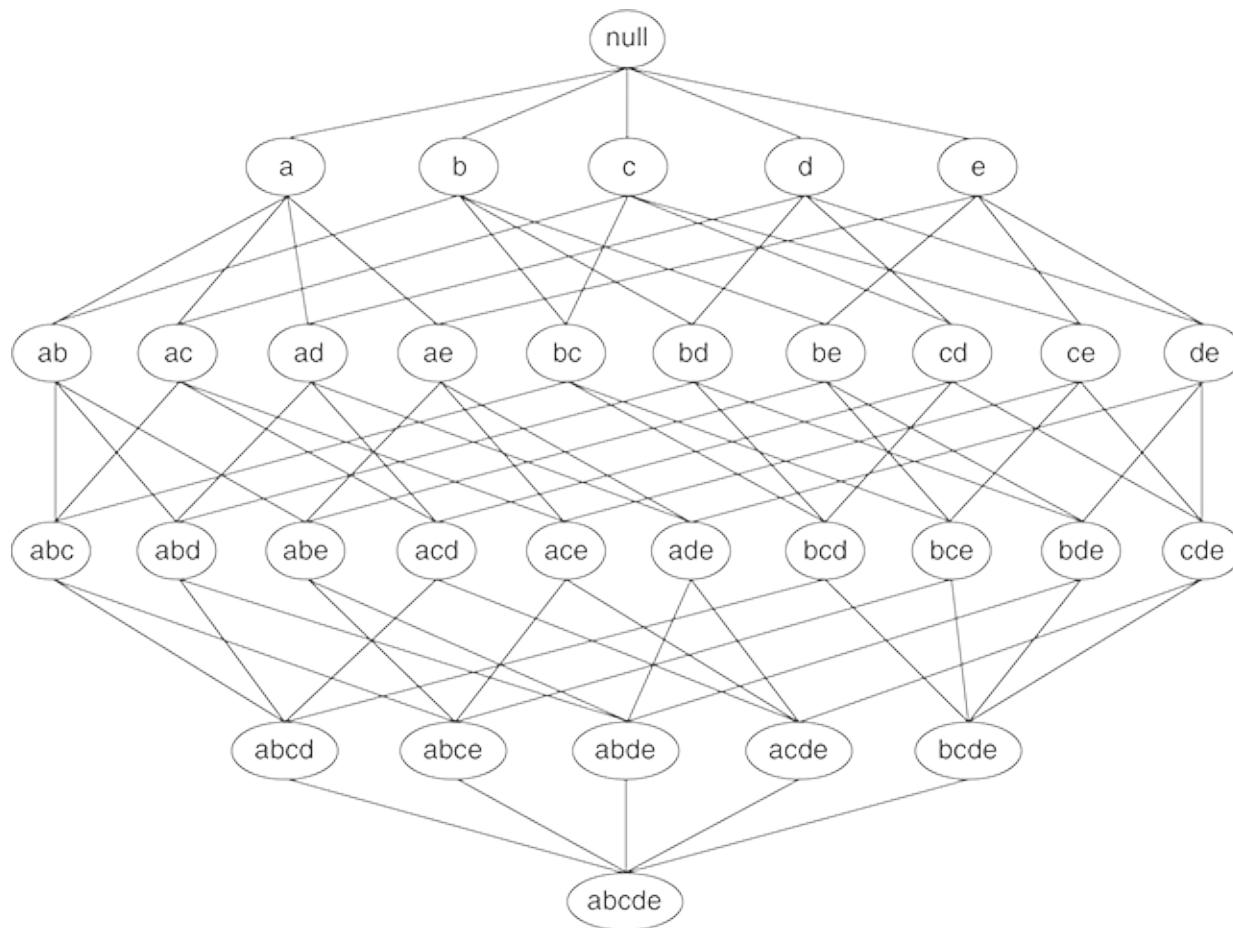


Figure 5.33.

An itemset lattice

- M if the node is a maximal frequent itemset,
- C if it is a closed frequent itemset,
- N if it is frequent but neither maximal nor closed, and
- I if it is infrequent

Assume that the support threshold is equal to 30%.

13. The original association rule mining formulation uses the support and confidence measures to prune uninteresting rules.

- a. Draw a contingency table for each of the following rules using the transactions shown in [Table 5.23](#).

Table 5.23. Example of market basket transactions.

Transaction ID	Items Bought
1	{a, b, d, e}
2	{b, c, d}
3	{a, b, d, e}
4	{a, c, d, e}
5	{b, c, d, e}
6	{b, d, e}
7	{c, d}
8	{a, b, c}

9	$\{a, d, e\}$
10	$\{b, d\}$

Rules: $\{b\} \rightarrow \{c\}$, $\{a\} \rightarrow \{d\}$, $\{b\} \rightarrow \{d\}$, $\{e\} \rightarrow \{c\}$, $\{c\} \rightarrow \{a\}$.

- b. Use the contingency tables in part (a) to compute and rank the rules in decreasing order according to the following measures.

- i. Support.
- ii. Confidence.
- iii. Interest $(X \rightarrow Y) = P(X, Y)P(X)P(Y)$.
- iv. IS($X \rightarrow Y$) = $P(X, Y)P(X)P(Y)$.
- v. Klosgen($X \rightarrow Y$) = $P(X, Y) \times \max(P(Y|X)) - P(Y)$, $P(X|Y) - P(X)$, where $P(Y|X) = P(X, Y)P(X)$.
- vi. Odds ratio($X \rightarrow Y$) = $P(X, Y)P(X^-, Y^-)P(X, Y^-)P(X^-, Y)$.

14. Given the rankings you had obtained in Exercise 13, compute the correlation between the rankings of confidence and the other five measures. Which measure is most highly correlated with confidence? Which measure is least correlated with confidence?

15. Answer the following questions using the data sets shown in **Figure 5.34**. Note that each data set contains 1000 items and 10,000 transactions. Dark cells indicate the presence of items and white cells indicate the absence of items. We will apply the *Apriori* algorithm to extract frequent itemsets with $\text{minsup}=10\%$ (i.e., itemsets must be contained in at least 1000 transactions).

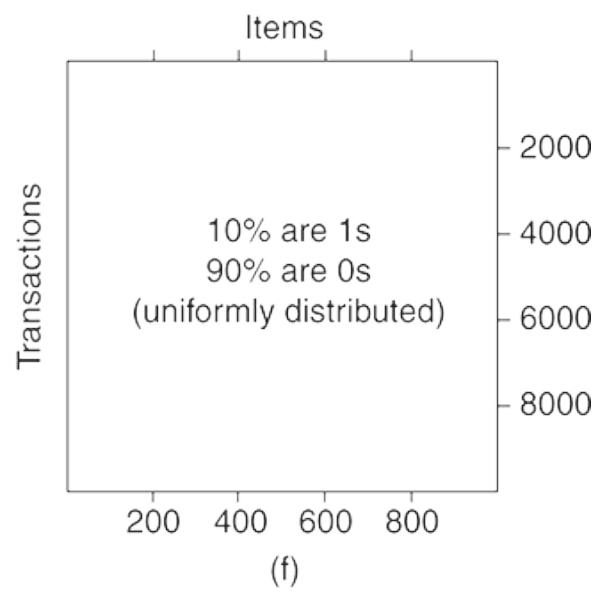
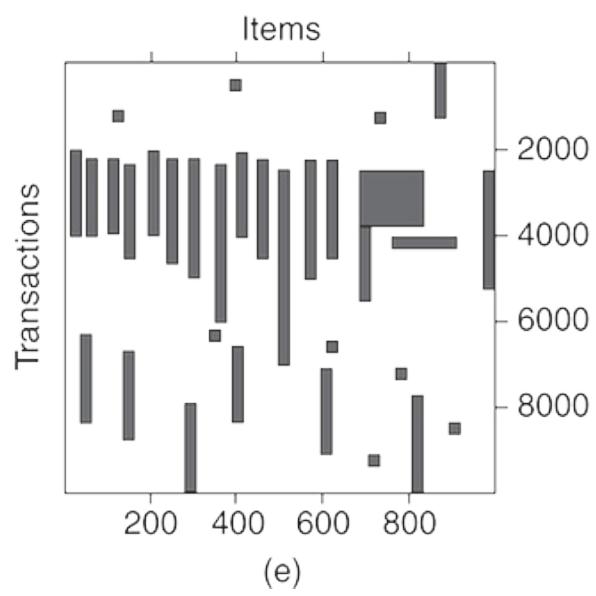
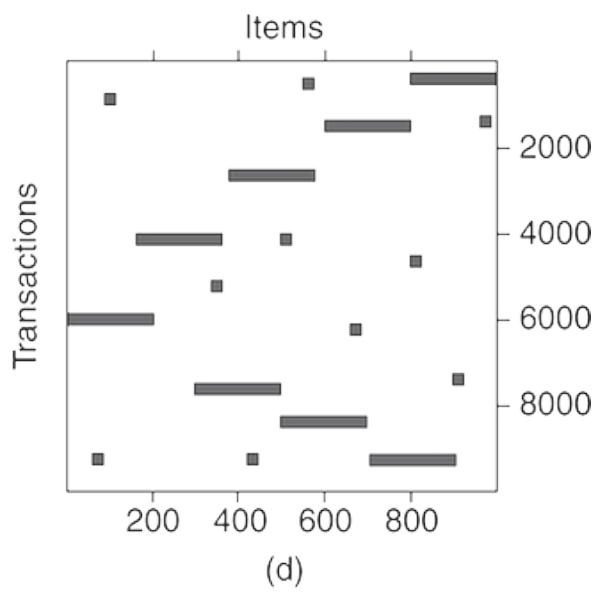
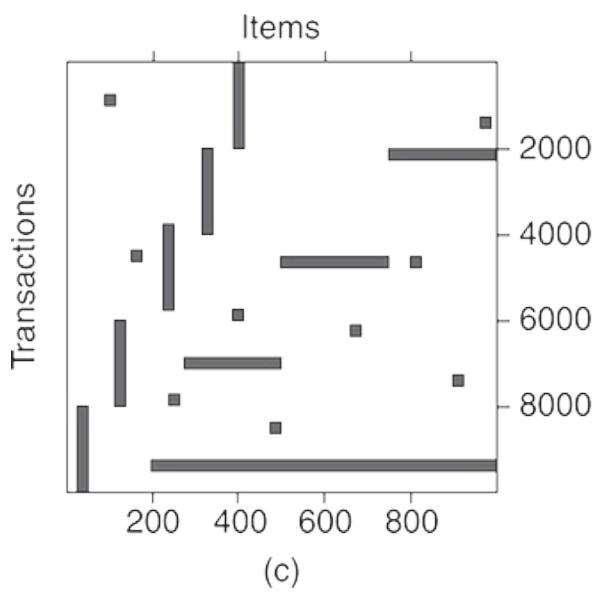
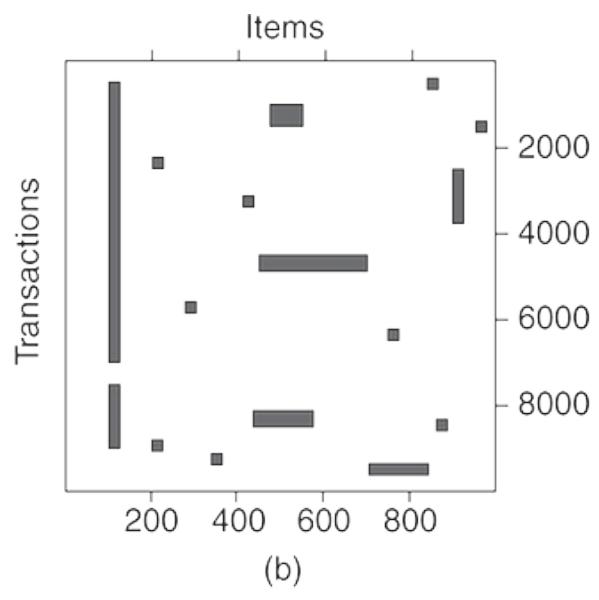
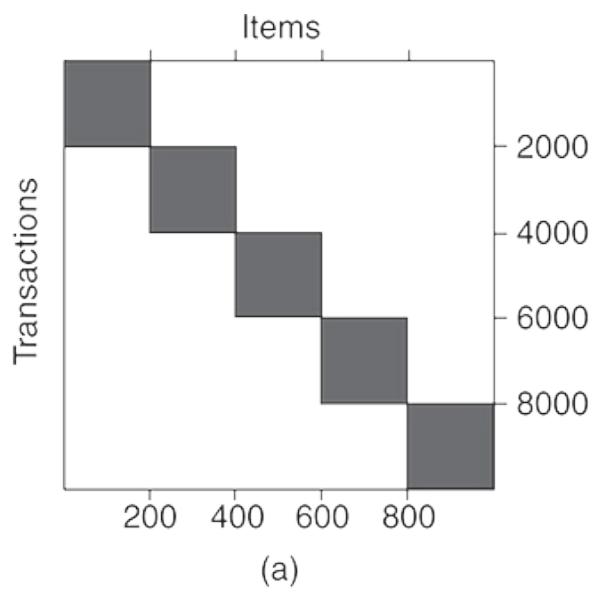


Figure 5.34.

Figures for Exercise 15.

- a. Which data set(s) will produce the most number of frequent itemsets?
- b. Which data set(s) will produce the fewest number of frequent itemsets?
- c. Which data set(s) will produce the longest frequent itemset?
- d. Which data set(s) will produce frequent itemsets with highest maximum support?
- e. Which data set(s) will produce frequent itemsets containing items with wide-varying support levels (i.e., items with mixed support, ranging from less than 20% to more than 70%)?

16.

- a. Prove that the ϕ coefficient is equal to 1 if and only if $f_{11}=f_{1+}=f_{+1}$.

- b. Show that if A and B are independent, then

$$P(A, B) \times P(A^-, B^-) = P(A, B^-) \times P(A^-, B).$$

- c. Show that Yule's Q and Y coefficients

$$Q = [f_{11}f_{00} - f_{10}f_{01}f_{11}f_{00} + f_{10}f_{01}] \quad Y = [f_{11}f_{00} - f_{10}f_{01}f_{11}f_{00} + f_{10}f_{01}]$$

are normalized versions of the odds ratio.

- d. Write a simplified expression for the value of each measure shown in **Table 5.9** when the variables are statistically independent.

17. Consider the interestingness measure, $M = P(B|A) - P(B)1 - P(B)$, for an association rule $A \rightarrow B$.

- a. What is the range of this measure? When does the measure attain its maximum and minimum values?
 - b. How does M behave when $P(A, B)$ is increased while $P(A)$ and $P(B)$ remain unchanged?
 - c. How does M behave when $P(A)$ is increased while $P(A, B)$ and $P(B)$ remain unchanged?
 - d. How does M behave when $P(B)$ is increased while $P(A, B)$ and $P(A)$ remain unchanged?
 - e. Is the measure symmetric under variable permutation?
 - f. What is the value of the measure when A and B are statistically independent?
 - g. Is the measure null-invariant?
 - h. Does the measure remain invariant under row or column scaling operations?
 - i. How does the measure behave under the inversion operation?
18. Suppose we have market basket data consisting of 100 transactions and 20 items. Assume the support for item a is 25%, the support for item b is 90% and the support for itemset $\{a, b\}$ is 20%. Let the support and confidence thresholds be 10% and 60%, respectively.
- a. Compute the confidence of the association rule $\{a\} \rightarrow \{b\}$. Is the rule interesting according to the confidence measure?
 - b. Compute the interest measure for the association pattern $\{a, b\}$. Describe the nature of the relationship between item a and item b in terms of the interest measure.

- c. What conclusions can you draw from the results of parts (a) and (b)?
- d. Prove that if the confidence of the rule $\{a\} \rightarrow \{b\}$ is less than the support of $\{b\}$, then:
- $c(\{\bar{a}\} \rightarrow \{b\}) > c(\{a\} \rightarrow \{b\})$,
 - $c(\{\bar{a}\} \rightarrow \{b\}) > s(\{b\})$,

where $c(\cdot)$ denote the rule confidence and $s(\cdot)$ denote the support of an itemset.

19. **Table 5.24** shows a $2 \times 2 \times 2$ contingency table for the binary variables A and B at different values of the control variable C .

Table 5.24. A Contingency Table.

		A		
		1	0	
C=0	B	1	0	15
		0	15	30
C=1	B	1	5	0
		0	0	15

- a. Compute the ϕ coefficient for A and B when $C=0$, $C=1$, and $C=0$ or 1 . Note that $\phi = P(A, B) - P(A)P(B)P(A)P(B)(1-P(A))(1-P(B))$.
- b. What conclusions can you draw from the above result?

20. Consider the contingency tables shown in **Table 5.25**.

- a. For table I, compute support, the interest measure, and the ϕ correlation coefficient for the association pattern $\{A, B\}$. Also, compute the confidence of rules $A \rightarrow B$ and $B \rightarrow A$.
- b. For table II, compute support, the interest measure, and the ϕ correlation coefficient for the association pattern $\{A, B\}$. Also, compute the confidence of rules $A \rightarrow B$ and $B \rightarrow A$.

Table 5.25. Contingency tables for Exercise 20.

(a) Table I.

	B	B^-
A	9	1
A^-	1	89

(b) Table II.

	B	B^-
A	89	1
A^-	1	9

- c. What conclusions can you draw from the results of (a) and (b)?
21. Consider the relationship between customers who buy high-definition televisions and exercise machines as shown in [Tables 5.17](#) and [5.18](#).
- a. Compute the odds ratios for both tables.
 - b. Compute the ϕ -coefficient for both tables.
 - c. Compute the interest factor for both tables.

For each of the measures given above, describe how the direction of association changes when data is pooled together instead of being stratified.

6 Association Analysis: Advanced Concepts

The association rule mining formulation described in the previous chapter assumes that the input data consists of binary attributes called items. The presence of an item in a transaction is also assumed to be more important than its absence. As a result, an item is treated as an asymmetric binary attribute and only frequent patterns are considered interesting.

This chapter extends the formulation to data sets with symmetric binary, categorical, and continuous attributes. The formulation will also be extended to incorporate more complex entities such as sequences and graphs. Although the overall structure of association analysis algorithms remains unchanged, certain aspects of the algorithms must be modified to handle the non-traditional entities.

6.1 Handling Categorical Attributes

There are many applications that contain symmetric binary and nominal attributes. The Internet survey data shown in **Table 6.1** contains symmetric binary attributes such as `Gender`, `Computer at Home`, `Chat Online`, `Shop Online`, and `Privacy Concerns`; as well as nominal attributes such as `Level of Education` and `State`. Using association analysis, we may uncover interesting information about the characteristics of Internet users such as

$$\{\text{Shop Online} = \text{Yes}\} \rightarrow \{\text{Privacy Concerns} = \text{Yes}\}.$$

Table 6.1. Internet survey data with categorical attributes.

Gender	Level of Education	State	Computer at Home	Chat Online	Shop Online	Privacy Concerns
Female	Graduate	Illinois	Yes	Yes	Yes	Yes
Male	College	California	No	No	No	No
Male	Graduate	Michigan	Yes	Yes	Yes	Yes
Female	College	Virginia	No	No	Yes	Yes
Female	Graduate	California	Yes	No	No	Yes
Male	College	Minnesota	Yes	Yes	Yes	Yes
Male	College	Alaska	Yes	Yes	Yes	No
Male	High School	Oregon	Yes	No	No	No

Female	Graduate	Texas	No	Yes	No	No
...

This rule suggests that most Internet users who shop online are concerned about their personal privacy.

To extract such patterns, the categorical and symmetric binary attributes are transformed into “items” first, so that existing association rule mining algorithms can be applied. This type of transformation can be performed by creating a new item for each distinct attribute-value pair. For example, the nominal attribute `Level of Education` can be replaced by three binary items:

`Education = College`, `Education = Graduate`, and `Education = High School`.

Similarly, symmetric binary attributes such as `Gender` can be converted into a pair of binary items, `Male` and `Female`. **Table 6.2** shows the result of binarizing the Internet survey data.

Table 6.2. Internet survey data after binarizing categorical and symmetric binary attributes.

Male	Female	<code>Education = Graduate</code>	<code>Education = College</code>	...	<code>Privacy = Yes</code>	<code>Privacy = No</code>
0	1	1	0	...	1	0
1	0	0	1	...	0	1
1	0	1	0	...	1	0
0	1	0	1	...	1	0
0	1	1	0	...	1	0
1	0	0	1	...	1	0

1	0	0	1	...	0	1
1	0	0	0	...	0	1
0	1	1	0	...	0	1
...

There are several issues to consider when applying association analysis to the binarized data:

1. Some attribute values may not be frequent enough to be part of a frequent pattern. This problem is more evident for nominal attributes that have many possible values, e.g., state names. Lowering the support threshold does not help because it exponentially increases the number of frequent patterns found (many of which may be spurious) and makes the computation more expensive. A more practical solution is to group related attribute values into a small number of categories. For example, each state name can be replaced by its corresponding geographical region, such as `Midwest`, `Pacific Northwest`, `Southwest`, and `East Coast`. Another possibility is to aggregate the less frequent attribute values into a single category called `Others`, as shown in **Figure 6.1**.

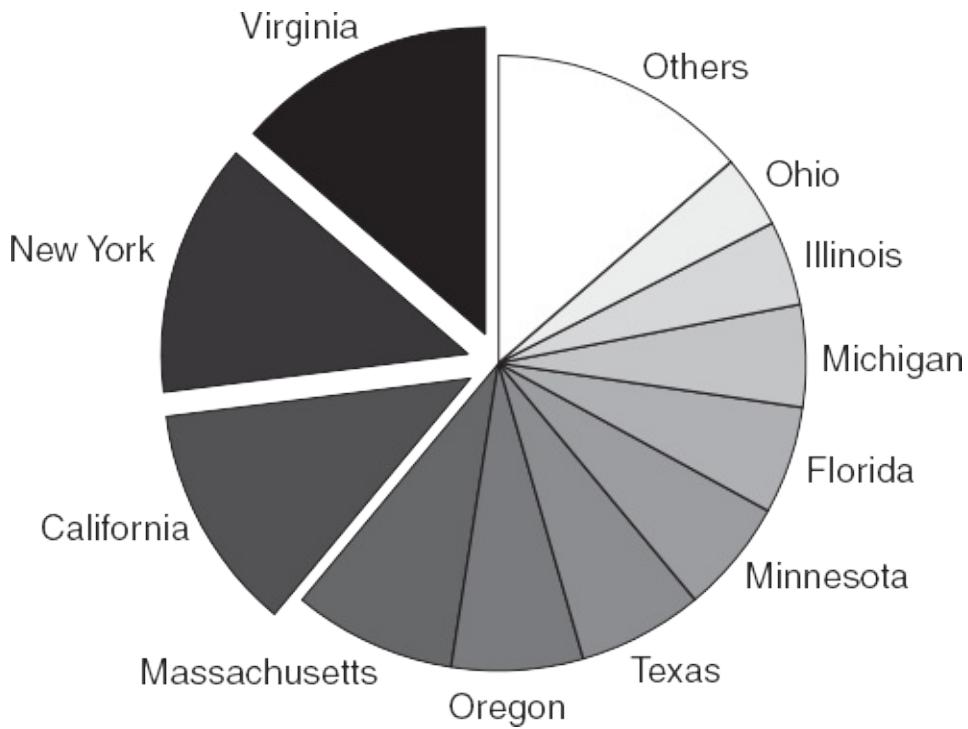


Figure 6.1.

A pie chart with a merged category called `Others`.

2. Some attribute values may have considerably higher frequencies than others. For example, suppose 85% of the survey participants own a home computer. By creating a binary item for each attribute value that appears frequently in the data, we may potentially generate many redundant patterns, as illustrated by the following example:

```
{Computer at home = Yes, Shop Online = Yes}
→ {Privacy Concerns = Yes}.
```

The rule is redundant because it is subsumed by the more general rule given at the beginning of this section. Because the high-frequency items correspond to the typical values of an attribute, they seldom carry any new information that can help us to better understand the pattern. It may therefore be useful to remove such items before applying

standard association analysis algorithms. Another possibility is to apply the techniques presented in [Section 5.8](#) for handling data sets with a wide range of support values.

3. Although the width of every transaction is the same as the number of attributes in the original data, the computation time may increase especially when many of the newly created items become frequent. This is because more time is needed to deal with the additional candidate itemsets generated by these items (see [Exercise 1](#) on [page 510](#)). One way to reduce the computation time is to avoid generating candidate itemsets that contain more than one item from the same attribute. For example, we do not have to generate a candidate itemset such as `{State = X, State = Y, ...}` because the support count of the itemset is zero.

6.2 Handling Continuous Attributes

The Internet survey data described in the previous section may also contain continuous attributes such as the ones shown in [Table 6.3](#). Mining the continuous attributes may reveal useful insights about the data such as “users whose annual income is more than \$120K belong to the 45–60 age group” or “users who have more than 3 email accounts and spend more than 15 hours online per week are often concerned about their personal privacy.” Association rules that contain continuous attributes are commonly known as **quantitative association rules**.

Table 6.3. Internet survey data with continuous attributes.

Gender	...	Age	Annual Income	No. of Hours Spent Online per Week	No. of Email Accounts	Privacy Concern
Female	...	26	90K	20	4	Yes
Male	...	51	135K	10	2	No
Male	...	29	80K	10	3	Yes
Female	...	45	120K	15	3	Yes
Female	...	31	95K	20	5	Yes
Male	...	25	55K	25	5	Yes
Male	...	37	100K	10	1	No
Male	...	41	65K	8	2	No
Female	...	26	85K	12	1	No

...
-----	-----	-----	-----	-----	-----	-----	-----

This section describes the various methodologies for applying association analysis to continuous data. We will specifically discuss three types of methods: (1) discretization-based methods, (2) statistics-based methods, and (3) nondiscretization methods. The quantitative association rules derived using these methods are quite different in nature.

6.2.1 Discretization-Based Methods

Discretization is the most common approach for handling continuous attributes. This approach groups the adjacent values of a continuous attribute into a finite number of intervals. For example, the `Age` attribute can be divided into the following intervals: $\text{Age} \in [12, 16)$, $\text{Age} \in [16, 20)$, $\text{Age} \in [20, 24)$, ..., $\text{Age} \in [56, 60)$, where $[a, b)$ represents an interval that includes a but not b .

Discretization can be performed using any of the techniques described in [Section 2.3.6](#) (equal interval width, equal frequency, entropy-based, or clustering). The discrete intervals are then mapped into asymmetric binary attributes so that existing association analysis algorithms can be applied.

Table 6.4 shows the Internet survey data after discretization and binarization.

Table 6.4. Internet survey data after binarizing categorical and continuous attributes.

Male	Female	...	Age < 13	Age ∈ [13, 21)	Age ∈ [21, 30)	...	Privacy = Yes	Privacy = No
0	1	...	0	0	1	...	1	0

1	0	...	0	0	0	...	0	1
1	0	...	0	0	1	...	1	0
0	1	...	0	0	0	...	1	0
0	1	...	0	0	0	...	1	0
1	0	...	0	0	1	...	1	0
1	0	...	0	0	0	...	0	1
1	0	...	0	0	0	...	0	1
0	1	...	0	0	1	...	0	1
...

A key parameter in attribute discretization is the number of intervals used to partition each attribute. This parameter is typically provided by the users and can be expressed in terms of the interval width (for the equal interval width approach), the average number of transactions per interval (for the equal frequency approach), or the number of desired clusters (for the clustering-based approach). The difficulty in determining the right number of intervals can be illustrated using the data set shown in **Table 6.5**, which summarizes the responses of 250 users who participated in the survey. There are two strong rules embedded in the data:

R1:Age \in [16,24) \rightarrow Chat Online=Yes (s=8.8%, c=81.5%). R2:Age \in [44,60) \rightarrow Chat

Table 6.5. A breakdown of Internet users who participated in online chat according to their age group.

Age Group	Chat Online = Yes	Chat Online = No

[12, 16)	12	13
[16, 20)	11	2
[20, 24)	11	3
[24, 28)	12	13
[28, 32)	14	12
[32, 36)	15	12
[36, 40)	16	14
[40, 44)	16	14
[44, 48)	4	10
[48, 52)	5	11
[52, 56)	5	10
[56, 60)	4	11

These rules suggest that most of the users from the age group of 16–24 often participate in online chatting, while those from the age group of 44–60 are less likely to chat online. In this example, we consider a rule to be interesting only if its support (*s*) exceeds 5% and its confidence (*c*) exceeds 65%. One of the problems encountered when discretizing the `Age` attribute is how to determine the interval width.

1. If the interval is too wide, then we may lose some patterns because of their lack of confidence. For example, when the interval width is 24 years, R1 and R2 are replaced by the following rules:
 $R'1: \text{Age} \in [12, 36] \rightarrow \text{Chat Online} = \text{Yes} \ (\text{s}=30\%, \text{c}=57.7\%). R$

'2:Age∈[36,60)→Chat Online=No (s=28%, c=58.3%).

Despite their higher supports, the wider intervals have caused the confidence for both rules to drop below the minimum confidence threshold. As a result, both patterns are lost after discretization.

2. If the interval is too narrow, then we may lose some patterns because of their lack of support. For example, if the interval width is 4 years, then R1 is broken up into the following two subrules:

R11(4):Age∈[16,20)→Chat Online=Yes (s=4.4%, c=84.6%).R12(4):Age∈[20,24)

Since the supports for the subrules are less than the minimum support threshold, R1 is lost after discretization. Similarly, the rule R2, which is broken up into four subrules, will also be lost because the support of each subrule is less than the minimum support threshold.

3. If the interval width is 8 years, then the rule R2 is broken up into the following two subrules:

R21(8):Age∈[44,52)→Chat Online=No (s=8.4%, c=70%).R22(8):Age∈[52,60)

Since R21(8) and R22(8) have sufficient support and confidence, R2 can be recovered by aggregating both subrules. Meanwhile, R1 is broken up into the following two subrules:

R11(8):Age∈[12,20)→Chat Online=Yes (s=9.2%, c=60.5%).R12(8):Age∈[20,24)

Unlike R2, we cannot recover the rule R1 by aggregating the subrules because both subrules fail the confidence threshold.

One way to address these issues is to consider every possible grouping of adjacent intervals. For example, we can start with an interval width of 4 years and then merge the adjacent intervals into wider intervals: Age ∈ [12, 16), Age ∈ [12, 20), ..., Age ∈ [12, 60), Age ∈ [16, 20), Age ∈ [16, 24), etc. This

approach enables the detection of both R1 and R2 as strong rules. However, it also leads to the following computational issues:

1. **The computation becomes extremely expensive.** If the range is initially divided into k intervals, then $k(k-1)/2$ binary items must be generated to represent all possible intervals. Furthermore, if an item corresponding to the interval $[a,b)$ is frequent, then all other items corresponding to intervals that subsume $[a,b)$ must be frequent too. This approach can therefore generate far too many candidate and frequent itemsets. To address these problems, a maximum support threshold can be applied to prevent the creation of items corresponding to very wide intervals and to reduce the number of itemsets.
2. **Many redundant rules are extracted.** For example, consider the following pair of rules:

R3:{Age \in [16,20), Gender=Male} \rightarrow {Chat Online=Yes}, R4:
{Age \in [16,24), Gender=Male} \rightarrow {Chat Online=Yes}.

R4 is a generalization of R3 (and R3 is a specialization of R4) because R4 has a wider interval for the **Age** attribute. If the confidence values for both rules are the same, then R4 should be more interesting because it covers more examples—including those for R3. R3 is therefore a redundant rule.

6.2.2 Statistics-Based Methods

Quantitative association rules can be used to infer the statistical properties of a population. For example, suppose we are interested in finding the average age of certain groups of Internet users based on the data provided in **Tables**

6.1 and **6.3**. Using the statistics-based method described in this section, quantitative association rules such as the following can be extracted:

{Annual Income>\$100K, Shop Online=Yes}→Age: Mean=38.

The rule states that the average age of Internet users whose annual income exceeds \$100K and who shop online regularly is 38 years old.

Rule Generation

To generate the statistics-based quantitative association rules, the target attribute used to characterize interesting segments of the population must be specified. By withholding the target attribute, the remaining categorical and continuous attributes in the data are binarized using the methods described in the previous section. Existing algorithms such as *Apriori* or FP-growth are then applied to extract frequent itemsets from the binarized data. Each frequent itemset identifies an interesting segment of the population. The distribution of the target attribute in each segment can be summarized using descriptive statistics such as mean, median, variance, or absolute deviation. For example, the preceding rule is obtained by averaging the age of Internet users who support the frequent itemset {Annual Income > \$100K, Shop Online = Yes}.

The number of quantitative association rules discovered using this method is the same as the number of extracted frequent itemsets. Because of the way the quantitative association rules are defined, the notion of confidence is not applicable to such rules. An alternative method for validating the quantitative association rules is presented next.

Rule Validation

A quantitative association rule is interesting only if the statistics computed from transactions covered by the rule are different than those computed from transactions not covered by the rule. For example, the rule given at the beginning of this section is interesting only if the average age of Internet users who do not support the frequent itemset $\{\text{Annual Income} > 100K, \text{Shop Online} = \text{Yes}\}$ is significantly higher or lower than 38 years old. To determine whether the difference in their average ages is statistically significant, statistical hypothesis testing methods should be applied.

Consider the quantitative association rule, $A \rightarrow t : \mu$, where A is a frequent itemset, t is the continuous target attribute, and μ is the average value of t among transactions covered by A . Furthermore, let μ' denote the average value of t among transactions not covered by A . The goal is to test whether the difference between μ and μ' is greater than some user-specified threshold, Δ . In statistical hypothesis testing, two opposite propositions, known as the null hypothesis and the alternative hypothesis, are given. A hypothesis test is performed to determine which of these two hypotheses should be accepted, based on evidence gathered from the data (see Appendix C).

In this case, assuming that $\mu < \mu'$, the null hypothesis is $H_0: \mu' = \mu + \Delta$, while the alternative hypothesis is $H_1: \mu' > \mu + \Delta$. To determine which hypothesis should be accepted, the following Z-statistic is computed:

$$Z = \frac{\mu' - \mu - \Delta}{\sqrt{s_1^2 n_1 + s_2^2 n_2}}, \quad (6.1)$$

where n_1 is the number of transactions supporting A , n_2 is the number of transactions not supporting A , s_1 is the standard deviation for t among transactions that support A , and s_2 is the standard deviation for t among transactions that do not support A . Under the null hypothesis, Z has a standard normal distribution with mean 0 and variance 1. The value of Z computed using [Equation 6.1](#) is then compared against a critical value, Z_α ,

which is a threshold that depends on the desired confidence level. If $Z > Z_\alpha$, then the null hypothesis is rejected and we may conclude that the quantitative association rule is interesting. Otherwise, there is not enough evidence in the data to show that the difference in mean is statistically significant.

Example 6.1. Consider the quantitative association rule

$\{\text{Income} > 100K, \text{Shop Online} = \text{Yes}\} \rightarrow \text{Age}: \mu = 38$.

Suppose there are 50 Internet users who supported the rule antecedent. The standard deviation of their ages is 3.5. On the other hand, the average age of the 200 users who do not support the rule antecedent is 30 and their standard deviation is 6.5. Assume that a quantitative association rule is considered interesting only if the difference between μ and μ' is more than 5 years. Using [Equation 6.1](#) we obtain

$$Z = 38 - 30 - \frac{53.5250 + 6.52200}{\sqrt{50}} = 4.4414.$$

For a one-sided hypothesis test at a 95% confidence level, the critical value for rejecting the null hypothesis is 1.64. Since $Z > 1.64$, the null hypothesis can be rejected. We therefore conclude that the quantitative association rule is interesting because the difference between the average ages of users who support and do not support the rule antecedent is more than 5 years.

6.2.3 Non-discretization Methods

There are certain applications in which analysts are more interested in finding associations among the continuous attributes, rather than associations among discrete intervals of the continuous attributes. For example, consider the problem of finding word associations in text documents. **Table 6.6** shows a document-word matrix where every entry represents the number of times a word appears in a given document. Given such a data matrix, we are interested in finding associations between words (e.g., `data` and `mining`) instead of associations between ranges of word counts (e.g., `data` $\in [1, 4]$ and `mining` $\in [2, 3]$). One way to do this is to transform the data into a 0/1 matrix, where the entry is 1 if the count exceeds some threshold t , and 0 otherwise. While this approach allows analysts to apply existing frequent itemset generation algorithms to the binarized data set, finding the right threshold for binarization can be quite tricky. If the threshold is set too high, it is possible to miss some interesting associations. Conversely, if the threshold is set too low, there is a potential for generating a large number of spurious associations.

Table 6.6. Document-word matrix.

Document	word1	word2	word3	word4	word5	word6
d1	3	6	0	0	0	2
d2	1	2	0	0	0	2
d3	4	2	7	0	0	2
d4	2	0	3	0	0	1
d5	0	0	0	1	1	0

This section presents another methodology for finding associations among continuous attributes, known as the min-*Apriori* approach. Analogous to

traditional association analysis, an itemset is considered to be a collection of continuous attributes, while its support measures the degree of association among the attributes, across multiple rows of the data matrix. For example, a collection of words in [Table 6.6](#) can be referred to as an itemset, whose support is determined by the co-occurrence of words across documents. In min-*Apriori*, the association among attributes in a given row of the data matrix is obtained by taking the minimum value of the attributes. For example, the association between words, word1 and word2, in the document d1 is given by $\min(3,6)=3$. The support of an itemset is then computed by aggregating its association over all the documents.

$$s(\{word1, word2\}) = \min(3,6) + \min(1,2) + \min(4,2) + \min(2,0) = 6.$$

The support measure defined in min-*Apriori* has the following desired properties, which makes it suitable for finding word associations in documents:

1. Support increases monotonically as the number of occurrences of a word increases.
2. Support increases monotonically as the number of documents that contain the word increases.
3. Support has an anti-monotone property. For example, consider a pair of itemsets $\{A,B\}$ and $\{A,B,C\}$. Since $\min(\{A,B\}) \geq \min(\{A,B,C\})$, $s(\{A,B\}) \geq s(\{A,B,C\})$. Therefore, support decreases monotonically as the number of words in an itemset increases.

The standard *Apriori* algorithm can be modified to find associations among words using the new support definition.

6.3 Handling a Concept Hierarchy

A concept hierarchy is a multilevel organization of the various entities or concepts defined in a particular domain. For example, in market basket analysis, a concept hierarchy has the form of an item taxonomy describing the “is-a” relationships among items sold at a grocery store—e.g., milk is a kind of food and DVD is a kind of home electronics equipment (see [Figure 6.2](#)). Concept hierarchies are often defined according to domain knowledge or based on a standard classification scheme defined by certain organizations (e.g., the Library of Congress classification scheme is used to organize library materials based on their subject categories).

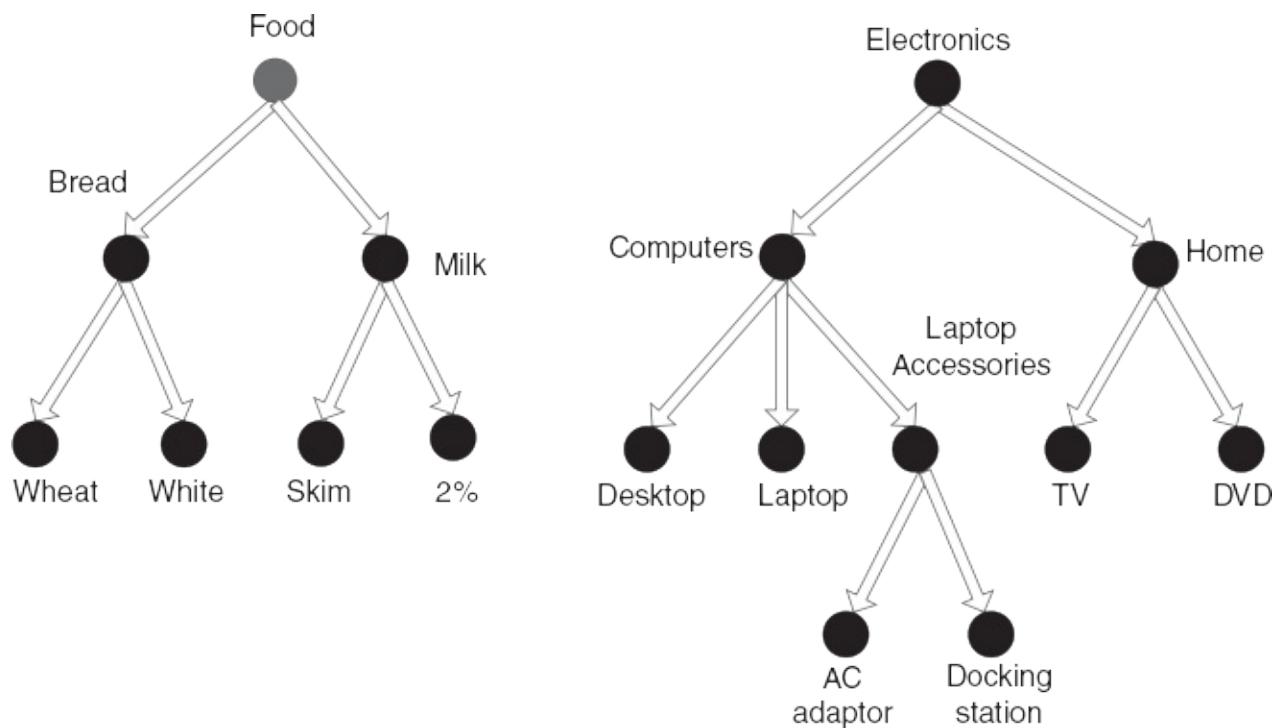


Figure 6.2.
Example of an item taxonomy.

A concept hierarchy can be represented using a **directed acyclic graph**, as shown in [Figure 6.2](#). If there is an edge in the graph from a node p to another node c , we call p the **parent** of c and c the **child** of p . For example, `milk` is the parent of `skim milk` because there is a directed edge from the node `milk` to the node `skim milk`. X^{\wedge} is called an **ancestor** of X (and X a **descendent** of X^{\wedge}) if there is a path from node X^{\wedge} to node X in the directed acyclic graph. In the diagram shown in [Figure 6.2](#), `food` is an ancestor of `skim milk` and `AC adaptor` is a descendent of `electronics`.

The main advantages of incorporating concept hierarchies into association analysis are as follows:

1. Items at the lower levels of a hierarchy may not have enough support to appear in any frequent itemset. For example, although the sale of AC adaptors and docking stations may be low, the sale of laptop accessories, which is their parent node in the concept hierarchy, may be high. Also, rules involving high-level categories may have lower confidence than the ones generated using low-level categories. Unless the concept hierarchy is used, there is a potential to miss interesting patterns at different levels of categories.
2. Rules found at the lower levels of a concept hierarchy tend to be overly specific and may not be as interesting as rules at the higher levels. For example, staple items such as milk and bread tend to produce many low-level rules such as `skim milk → wheat bread`, `2% milk → wheat bread`, and `skim milk → white bread`. Using a concept hierarchy, they can be summarized into a single rule, `milk → bread`. Considering only items residing at the top level of their hierarchies also may not be good enough, because such rules may not be of any practical use. For example, although the rule `electronics → food` may satisfy the support and confidence thresholds, it is not informative because the

combination of electronics and food items that are frequently purchased by customers are unknown. If milk and batteries are the only items sold together frequently, then the pattern $\{\text{food, electronics}\}$ may have overgeneralized the situation.

Standard association analysis can be extended to incorporate concept hierarchies in the following way. Each transaction t is initially replaced with its **extended transaction** t' , which contains all the items in t along with their corresponding ancestors. For example, the transaction $\{\text{DVD, wheat bread}\}$ can be extended to $\{\text{DVD, wheat bread, home electronics, electronics, bread, food}\}$, where home electronics and electronics are the ancestors of DVD , while bread and food are the ancestors of wheat bread .

We can then apply existing algorithms, such as *Apriori*, to the database of extended transactions. Although such an approach would find rules that span different levels of the concept hierarchy, it would suffer from several obvious limitations as described below:

1. Items residing at the higher levels tend to have higher support counts than those residing at the lower levels of a concept hierarchy. As a result, if the support threshold is set too high, then only patterns involving the high-level items are extracted. On the other hand, if the threshold is set too low, then the algorithm generates far too many patterns (most of which may be spurious) and becomes computationally inefficient.
2. Introduction of a concept hierarchy tends to increase the computation time of association analysis algorithms because of the larger number of items and wider transactions. The number of candidate patterns and frequent patterns generated by these algorithms may also grow exponentially with wider transactions.

3. Introduction of a concept hierarchy may produce redundant rules. A rule $X \rightarrow Y$ is redundant if there exists a more general rule $X^{\wedge} \rightarrow Y^{\wedge}$, where X^{\wedge} is an ancestor of X , Y^{\wedge} is an ancestor of Y , and both rules have very similar confidence. For example, suppose $\{\text{bread}\} \rightarrow \{\text{milk}\}$, $\{\text{white bread}\} \rightarrow \{2\% \text{ milk}\}$, $\{\text{wheat bread}\} \rightarrow \{2\% \text{ milk}\}$, $\{\text{white bread}\} \rightarrow \{\text{skim milk}\}$, and $\{\text{wheat bread}\} \rightarrow \{\text{skim milk}\}$ have very similar confidence. The rules involving items from the lower level of the hierarchy are considered redundant because they can be summarized by a rule involving the ancestor items. An itemset such as $\{\text{skim milk, milk, food}\}$ is also redundant because food and milk are ancestors of skim milk . Fortunately, it is easy to eliminate such redundant itemsets during frequent itemset generation, given the knowledge of the hierarchy.

6.4 Sequential Patterns

Market basket data often contains temporal information about when an item was purchased by customers. Such information can be used to piece together the sequence of transactions made by a customer over a certain period of time. Similarly, event-based data collected from scientific experiments or the monitoring of physical systems, such as telecommunications networks, computer networks, and wireless sensor networks, have an inherent sequential nature to them. This means that an ordinal relation, usually based on temporal precedence, exists among events occurring in such data. However, the concepts of association patterns discussed so far emphasize only “co-occurrence” relationships and disregard the sequential information of the data. The latter information may be valuable for identifying recurring features of a dynamic system or predicting future occurrences of certain events. This section presents the basic concept of sequential patterns and the algorithms developed to discover them.

6.4.1 Preliminaries

The input to the problem of discovering sequential patterns is a sequence data set, an example of which is shown on the left-hand side of [Figure 6.3](#). Each row records the occurrences of events associated with a particular object at a given time. For example, the first row contains the set of events occurring at timestamp $t=10$ for object A. Note that if we only consider the last column of this sequence data set, it would look similar to a market basket data where every row represents a transaction containing a set of events (items). The traditional concept of association patterns in this data would correspond

to common co-occurrences of events across transactions. However, a sequence data set also contains information about the object and the timestamp of a transaction of events in the first two columns. These columns add context to every transaction, which enables a different style of association analysis for sequence data sets. The right-hand side of [Figure 6.3](#) shows a different representation of the sequence data set where the events associated with every object appear together, sorted in increasing order of their timestamps. In a sequence data set, we can look for association patterns of events that commonly occur in a sequential order across objects. For example, in the sequence data shown in [Figure 6.3](#), event 6 is followed by event 1 in all of the sequences. Note that such a pattern cannot be inferred if we treat this as a market basket data by ignoring information about the object and timestamp.

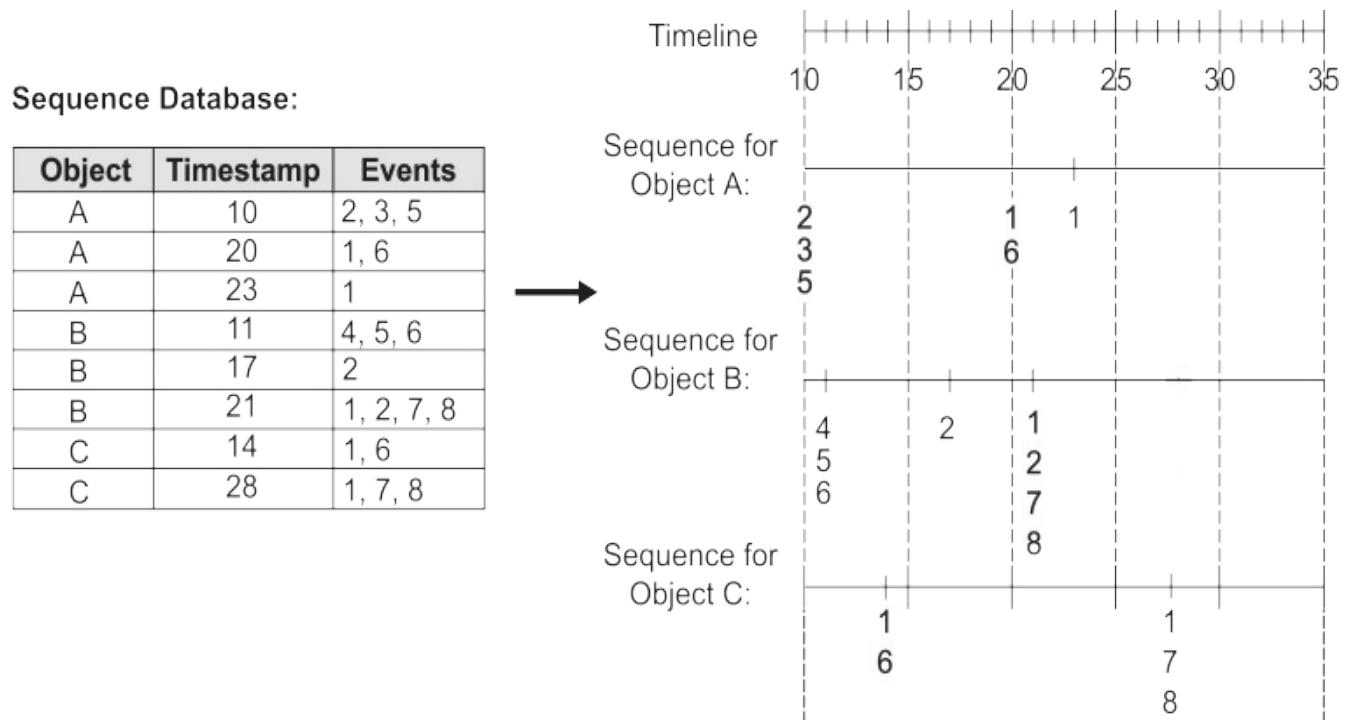


Figure 6.3.
Example of a sequence database.

Before presenting a methodology for finding sequential patterns, we provide a brief description of sequences and subsequences.

Sequences

Generally speaking, a sequence is an ordered list of **elements** (transactions). A sequence can be denoted as $s = \langle e_1 e_2 e_3 \dots e_n \rangle$, where each element e_j is a collection of one or more events (items), i.e., $e_j = \{i_1, i_2, \dots, i_k\}$. The following is a list of examples of sequences:

- Sequence of web pages viewed by a web site visitor:
 $\langle \{\text{Homepage}\} \{\text{Electronics}\} \{\text{Cameras and Camcorders}\} \{\text{Digital Cameras}\} \{\text{Shopping Cart}\} \{\text{Order Confirmation}\} \{\text{Return to Shopping}\} \rangle$
- Sequence of events leading to the nuclear accident at Three-Mile Island:
 $\langle \{\text{clogged resin}\} \{\text{outlet valve closure}\} \{\text{loss of feedwater}\} \{\text{condenser polisher outlet valve shut}\} \{\text{booster pumps trip}\} \{\text{main waterpump trips}\} \{\text{main turbine trips}\} \{\text{reactor pressure increases}\} \rangle$
- Sequence of classes taken by a computer science major student in different semesters:
 $\langle \{\text{Algorithms and Data Structures, Introduction to Operating Systems}\} \{\text{Database Systems, Computer Architecture}\} \{\text{Computer Networks, Software Engineering}\} \{\text{Computer Graphics, Parallel Programming}\} \rangle$

A sequence can be characterized by its length and the number of occurring events. The length of a sequence corresponds to the number of elements present in the sequence, while we refer to a sequence that contains k events as a k -sequence. The web sequence in the previous example contains 7 elements and 7 events; the event sequence at Three-Mile Island contains 8

elements and 8 events; and the class sequence contains 4 elements and 8 events.

Figure 6.4 provides examples of sequences, elements, and events defined for a variety of application domains. Except for the last row, the ordinal attribute associated with each of the first three domains corresponds to calendar time. For the last row, the ordinal attribute corresponds to the location of the bases (A, C, G, T) in the gene sequence. Although the discussion on sequential patterns is primarily focused on temporal events, it can be extended to the case where the events have non-temporal ordering, such as spatial ordering.

Sequence Database	Sequence	Element (Transaction)	Event (Item)
Customer	Purchase history of a given customer	A set of items bought by a customer at time t	Books, diary products, CDs, etc
Web Data	Browsing activity of a particular Web visitor	The collection of files viewed by a Web visitor after a single mouse click	Home page, index page, contact info, etc
Event data	History of events generated by a given sensor	Events triggered by a sensor at time t	Types of alarms generated by sensors
Genome sequences	DNA sequence of a particular species	An element of the DNA sequence	Bases A,T,G,C

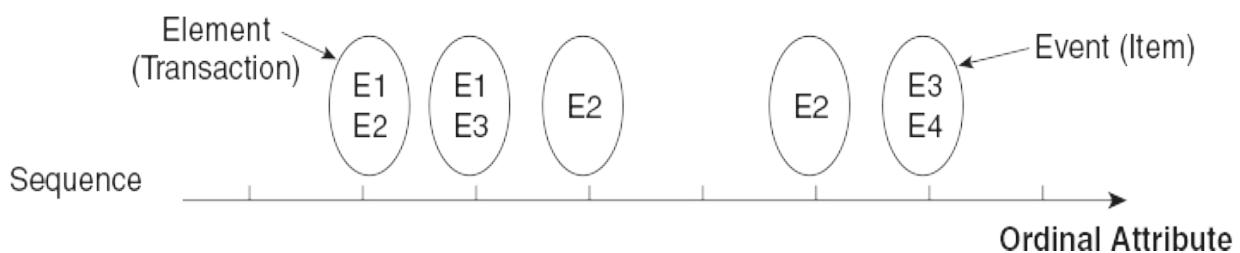


Figure 6.4.

Examples of elements and events in sequence data sets.

Subsequences

A sequence t is a **subsequence** of another sequence s if it is possible to derive t from s by simply deleting some events from elements in s or even deleting some elements in s completely. Formally, the sequence $t=\langle t_1t_2\dots t_m \rangle$ is a subsequence of $s=\langle s_1s_2\dots s_n \rangle$ if there exist integers $1 \leq j_1 < j_2 < \dots < j_m \leq n$ such that $t_1 \subseteq s_{j_1}, t_2 \subseteq s_{j_2}, \dots, t_m \subseteq s_{j_m}$. If t is a subsequence of s , then we say that t is **contained** in s . **Table 6.7** gives examples illustrating the idea of subsequences for various sequences.

Table 6.7. Examples illustrating the concept of a subsequence.

Sequence, s	Sequence, t	Is t a subsequence of s ?
$\langle \{2,4\} \{3,5,6\} \{8\} \rangle$	$\langle \{2\} \{3,6\} \{8\} \rangle$	Yes
$\langle \{2,4\} \{3,5,6\} \{8\} \rangle$	$\langle \{2\} \{8\} \rangle$	Yes
$\langle \{1,2\} \{3,4\} \rangle$	$\langle \{1\} \{2\} \rangle$	No
$\langle \{2,4\} \{2,4\} \{2,5\} \rangle$	$\langle \{2\} \{4\} \rangle$	Yes

6.4.2 Sequential Pattern Discovery

Let D be a data set that contains one or more **data sequences**. The term data sequence refers to an ordered list of elements associated with a single data object. For example, the data set shown in **Figure 6.5** contains five data sequences, one for each object A, B, C, D, and E.

Object	Timestamp	Events
A	1	1, 2, 4
A	2	2, 3
A	3	5
B	1	1, 2
B	2	2, 3, 4
C	1	1, 2
C	2	2, 3, 4
C	3	2, 4, 5
D	1	2
D	2	3, 4
D	3	4, 5
E	1	1, 3
E	2	2, 4, 5

$Minsup = 50\%$

Examples of Sequential Patterns:

$\langle \{1,2\} \rangle$	$s=60\%$
$\langle \{2,3\} \rangle$	$s=60\%$
$\langle \{2,4\} \rangle$	$s=80\%$
$\langle \{3\} \{5\} \rangle$	$s=80\%$
$\langle \{1\} \{2\} \rangle$	$s=80\%$
$\langle \{2\} \{2\} \rangle$	$s=60\%$
$\langle \{1\} \{2,3\} \rangle$	$s=60\%$
$\langle \{2\} \{2,3\} \rangle$	$s=60\%$
$\langle \{1,2\} \{2,3\} \rangle$	$s=60\%$

Figure 6.5.

Sequential patterns derived from a data set that contains five data sequences.

The support of a sequence s is the fraction of all data sequences that contain s . If the support for s is greater than or equal to a user-specified threshold $minsup$, then s is declared to be a sequential pattern (or frequent sequence).

Definition 6.1 (Sequential Pattern Discovery).

Given a sequence data set D and a user-specified minimum support threshold $minsup$, the task of sequential pattern discovery is to find all sequences with support $\geq minsup$.

In [Figure 6.5](#), the support for the sequence $\langle \{1\}\{2\} \rangle$ is equal to 80% because it is contained in four of the five data sequences (every object except for D). Assuming that the minimum support threshold is 50%, any sequence that is contained in at least three data sequences is considered to be a sequential pattern. Examples of sequential patterns extracted from the given data set include $\langle \{1\}\{2\} \rangle, \langle \{1,2\} \rangle, \langle \{2,3\} \rangle, \langle \{1,2\}\{2,3\} \rangle$, etc.

Sequential pattern discovery is a computationally challenging task because the set of all possible sequences that can be generated from a collection of events is exponentially large and difficult to enumerate. For example, a collection of n events can result in the following examples of 1-sequences, 2-sequences, and 3-sequences:

1- sequences:	$\langle i_1 \rangle, \langle i_2 \rangle, \dots, \langle i_n \rangle$
2- sequences:	$\langle \{i_1, i_2\} \rangle, \langle \{i_1, i_3\} \rangle, \dots, \langle \{i_{n-1}, i_n\} \rangle, \dots, \langle \{i_1\}\{i_1\} \rangle, \langle \{i_1\}\{i_2\} \rangle, \dots, \langle \{i_n\}\{i_n\} \rangle$
3- sequences:	$\langle \{i_1, i_2, i_3\} \rangle, \langle \{i_1, i_2, i_4\} \rangle, \dots, \langle \{i_{n-2}, i_{n-1}, i_n\} \rangle, \dots, \langle \{i_1\}\{i_1, i_2\} \rangle, \langle \{i_1\}\{i_1, i_3\} \rangle, \dots, \langle \{i_{n-1}\}\{i_{n-1}, i_n\} \rangle, \dots, \langle \{i_1, i_2\}\{i_2\} \rangle, \langle \{i_1, i_2\}\{i_3\} \rangle, \dots, \langle \{i_{n-1}, i_n\}\{i_n\} \rangle, \dots, \langle \{i_1\}\{i_1\}\{i_1\} \rangle, \langle \{i_1\}\{i_1\}\{i_2\} \rangle, \dots, \langle \{i_n\}\{i_n\}\{i_n\} \rangle$

The above enumeration is similar in some ways to the itemset lattice introduced in [Chapter 5](#) for market basket data. However, note that the above enumeration is not exhaustive; it only shows some sequences and omits a large number of remaining ones by the use of ellipses (...). This is because the number of candidate sequences is substantially larger than the number of candidate itemsets, which makes their enumeration difficult. There are three reasons for the additional number of candidates sequences:

1. An item can appear at most once in an itemset, but an event can appear more than once in a sequence, in different elements of the

sequence. For example, given a pair of items, i_1 and i_2 , only one candidate 2-itemset, $\{i_1, i_2\}$, can be generated. In contrast, there are many candidate 2-sequences that can be generated using only two events: $\langle \{i_1\} \{i_1\} \rangle, \langle \{i_1\} \{i_2\} \rangle, \langle \{i_2\} \{i_1\} \rangle, \langle \{i_2\} \{i_2\} \rangle$, and $\langle \{i_1, i_2\} \rangle$.

2. Order matters in sequences, but not for itemsets. For example, $\{i_1, i_2\}$ and $\{i_2, i_1\}$ refers to the same itemset, whereas $\langle \{i_1\} \{i_2\} \rangle, \langle \{i_2\} \{i_1\} \rangle$, and $\langle \{i_1, i_2\} \rangle$ correspond to different sequences, and thus must be generated separately.
3. For market basket data, the number of distinct items n puts an upper bound on the number of candidate itemsets ($2^n - 1$), whereas for sequence data, even two events a and b can lead to infinitely many candidate sequences (see [Figure 6.6](#) for an illustration).

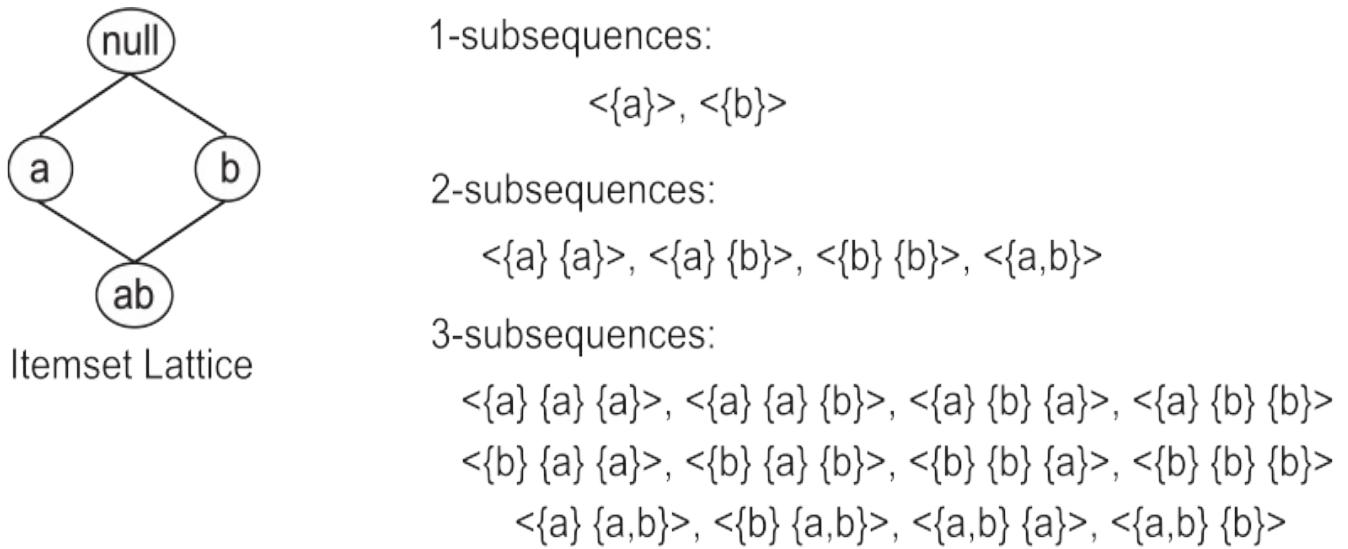


Figure 6.6.

Comparing the number of itemsets with the number of sequences generated using two events (items). We only show 1-sequences, 2-sequences, and 3-sequences for illustration.

Because of the above reasons, it is challenging to create a sequence lattice that enumerates all possible sequences even when the number of events in the data is small. It is thus difficult to use a brute-force approach for

generating sequential patterns that enumerates all possible sequences by traversing the sequence lattice. Despite these challenges, the *Apriori* principle still holds for sequential data because any data sequence that contains a particular k -sequence must also contain all of its $(k-1)$ -subsequences. As we will see later, even though it is challenging to construct the sequence lattice, it is possible to generate candidate k -sequences from the frequent $(k-1)$ -sequences using the *Apriori* principle. This allows us to extract sequential patterns from a sequence data set using an *Apriori*-like algorithm. The basic structure of this algorithm is shown in [Algorithm 6.1](#).

Algorithm 6.1 *Apriori*-like algorithm for sequential pattern discovery.

```
1: k=1.  
2: Fk={i|i∈I ∧ σ({i})N≥minsup}. {Find all frequent 1-  
subsequences.}  
3: repeat  
4: k=k+1.  
5: Ck=candidate-gen(Fk-1). {Generate candidate  $k$ -  
subsequences.}  
6: Ck=candidate-prune(Ck,Fk-1). {Prune candidate  $k$ -  
subsequences.}  
7: for each data sequence t∈T do  
8: Ct=subsequence(Ck,t). {Identify all candidates contained in  
t.}  
9: for each candidate  $k$ -subsequence c∈Ct do  
10: σ(c)=σ(c)+1. {Increment the support count.}  
11: end for  
12: end for
```

- 13: $F_k = \{c | c \in C_k \wedge \sigma(c) \geq \text{minsup}\}$. {Extract the frequent k -subsequences.}
- 14: **until** $F_k = \emptyset$
- 15: Answer = $\bigcup F_k$.

Notice that the structure of the algorithm is almost identical to *Apriori* algorithm for frequent itemset discovery, presented in the previous chapter. The algorithm would iteratively generate new candidate k -sequences, prune candidates whose $(k-1)$ -sequences are infrequent, and then count the supports of the remaining candidates to identify the sequential patterns. The detailed aspects of these steps are given next.

Candidate Generation

We generate candidate k -sequences by merging a pair of frequent $(k-1)$ -sequences. Although this approach is similar to the $F_{k-1} \times F_{k-1}$ strategy introduced in [Chapter 5](#) for generating candidate item-sets, there are certain differences. First, in the case of generating sequences, we can merge a $(k-1)$ -sequence with itself to produce a k -sequence, since. For example, we can merge the 1-sequence $\langle a \rangle$ with itself to produce a candidate 2-sequence, $\langle \{a\}{a} \rangle$. Second, recall that in order to avoid generating duplicate candidates, the traditional *Apriori* algorithm merges a pair of frequent k -itemsets only if their first $k-1$ items, arranged in lexicographic order, are identical. In the case of generating sequences, we still use the lexicographic order for arranging events within an element. However, the arrangement of elements in a sequence may not follow the lexicographic order. For example, $\langle \{b,c\}{a\{d\}} \rangle$ is a viable representation of a 4-sequence, even though the elements in the sequence are not arranged according to their lexicographic ranks. On the other hand, $\langle \{c,b\}{a\{d\}} \rangle$ is not a viable representation of the same 4-sequence, since the events in the first element violate the lexicographic order.

Given a sequence $s = \langle e_1e_2e_3\dots e_n \rangle$, where the events in every element are arranged lexicographically, we can refer to first event of e_1 as the first event of s and the last event of e_n as the last event of s . The criteria for merging sequences can then be stated in the form of the following procedure.

Sequence Merging Procedure

A sequence $s(1)$ is merged with another sequence $s(2)$ only if the subsequence obtained by dropping the first event in $s(1)$ is identical to the subsequence obtained by dropping the last event in $s(2)$. The resulting candidate is given by extending the sequence $s(1)$ as follows:

1. If the last element of $s(2)$ has only one event, append the last element of $s(2)$ to the end of $s(1)$ and obtain the merged sequence.
2. If the last element of $s(2)$ has more than one event, append the last event from the last element of $s(2)$ (that is absent in the last element of $s(1)$) to the last element of $s(1)$ and obtain the merged sequence.

Figure 6.7 illustrates examples of candidate 4-sequences obtained by merging pairs of frequent 3-sequences. The first candidate, $\langle \{1\}\{2\}\{3\}\{4\} \rangle$, is obtained by merging $\langle \{1\}\{2\}\{3\} \rangle$ with $\langle \{2\}\{3\}\{4\} \rangle$. Since the last element of the second sequence ($\{4\}$) contains only one element, it is simply appended to the first sequence to generate the merged sequence. On the other hand, merging $\langle \{1\}\{5\}\{3\} \rangle$ with $\langle \{5\}\{3,4\} \rangle$ produces the candidate 4-sequence $\langle \{1\}\{5\}\{3,4\} \rangle$. In this case, the last element of the second sequence ($\{3,4\}$)

contains two events. Hence, the last event in this element (4) is added to the last element of the first sequence ($\{3\}$) to obtain the merged sequence.

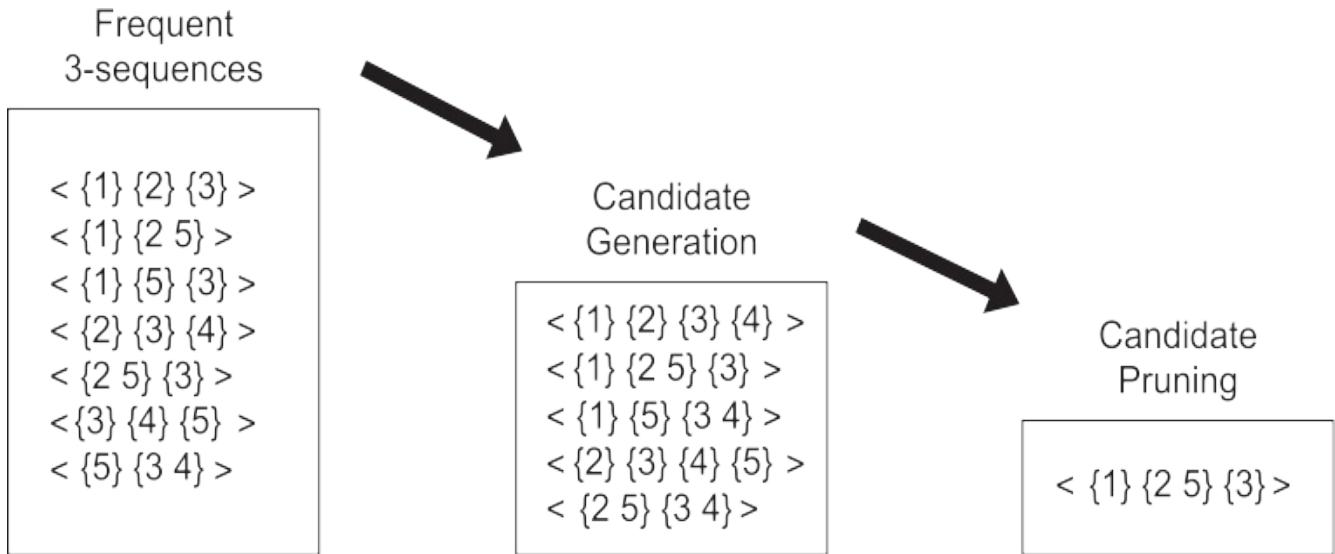


Figure 6.7.

Example of the candidate generation and pruning steps of a sequential pattern mining algorithm.

It can be shown that the sequence merging procedure is complete, i.e., it generates every frequent k -subsequence. This is because every frequent k -subsequence s includes a frequent $(k-1)$ -sequence s_1 , that does not contain the first event of s , and a frequent $(k-1)$ -sequence s_2 , that does not contain the last event of s . Since s_1 and s_2 are frequent and follow the criteria for merging sequences, they will be merged to produce every frequent k -subsequence s as one of the candidates. Also, the sequence merging procedure ensures that there is a unique way of generating s only by merging s_1 and s_2 . For example, in [Figure 6.7](#), the sequences $\langle \{1\}\{2\}\{3\} \rangle$ and $\langle \{1\}\{2,5\} \rangle$ do not have to be merged because removing the first event from the first sequence does not give the same subsequence as removing the last event from the second sequence. Although $\langle \{1\}\{2,5\}\{3\} \rangle$ is a viable candidate, it is generated by merging a different pair of sequences, $\langle \{1\}\{2,5\}$

\rangle and $\langle \{2,5\}\{3\} \rangle$. This example illustrates that the sequence merging procedure does not generate duplicate candidate sequences.

Candidate Pruning

A candidate k -sequence is pruned if at least one of its $(k-1)$ -sequences is infrequent. For example, consider the candidate 4-sequence $\langle \{1\}\{2\}\{3\}\{4\} \rangle$. We need to check if any of the 3-sequences contained in this 4-sequence is infrequent. Since the sequence $\langle \{1\}\{2\}\{4\} \rangle$ is contained in this sequence and is infrequent, the candidate $\langle \{1\}\{2\}\{3\}\{4\} \rangle$ can be eliminated. Readers should be able to verify that the only candidate 4-sequence that survives the candidate pruning step in [Figure 6.7](#) is $\langle \{1\}\{2\}\{5\}\{3\} \rangle$.

Support Counting

During support counting, the algorithm identifies all candidate k -sequences belonging to a particular data sequence and increments their support counts. After performing this step for each data sequence, the algorithm identifies the frequent k -sequences and discards all candidate sequences whose support values are less than the $minsup$ threshold.

6.4.3 Timing Constraints*

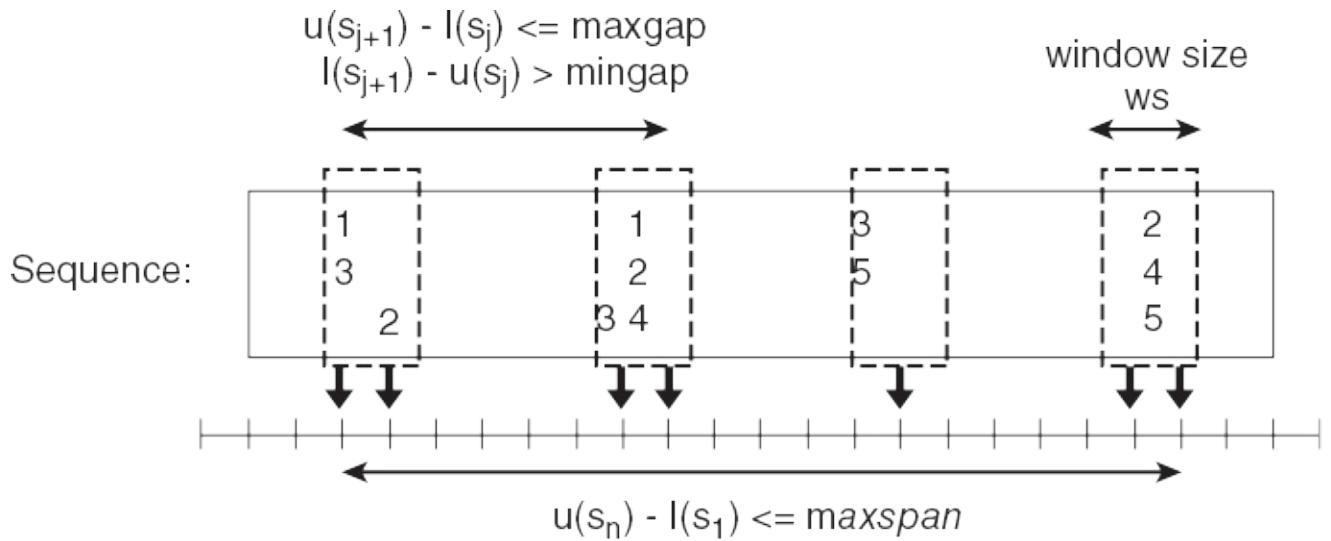
This section presents a sequential pattern formulation where timing constraints are imposed on the events and elements of a pattern. To motivate the need for timing constraints, consider the following sequence of courses taken by two students who enrolled in a data mining class:

Student A: $\langle \{\text{Statistics}\} \{\text{Database Systems}\} \{\text{Data Mining}\} \rangle$. Student B: \langle

{Database Systems} {Statistics} {Data Mining} >.

The sequential pattern of interest is < {Statistics, Database Systems} {Data Mining} >, which means that students who are enrolled in the data mining class must have previously taken a course in statistics and database systems. Clearly, the pattern is supported by both students even though they do not take statistics and database systems at the same time. In contrast, a student who took a statistics course ten years earlier should not be considered as supporting the pattern because the time gap between the courses is too long. Because the formulation presented in the previous section does not incorporate these timing constraints, a new sequential pattern definition is needed.

Figure 6.8  illustrates some of the timing constraints that can be imposed on a pattern. The definition of these constraints and the impact they have on sequential pattern discovery algorithms will be discussed in the following sections. Note that each element of the sequential pattern is associated with a time window $[l, u]$, where l is the earliest occurrence of an event within the time window and u is the latest occurrence of an event within the time window. Note that in this discussion, we allow events within an element to occur at different times. Hence, the actual timing of the event occurrence may not be the same as the lexicographic ordering.



Time window (w) for each element is characterized by $[l, u]$
 where l : earliest time of occurrence of an event in w
 u : latest time of occurrence of an event in w

Figure 6.8.

Timing constraints of a sequential pattern.

The maxspan Constraint

The *maxspan* constraint specifies the maximum allowed time difference between the latest and the earliest occurrences of events in the entire sequence. For example, suppose the following data sequences contain elements that occur at consecutive time stamps (1,2,3,...), i.e., the i th element in the sequence occurs at the i th timestamp. Assuming that *maxspan* = 3, the following table contains sequential patterns that are supported and not supported by a given data sequence.

Data Sequence, s	Sequential Pattern, t	Does s support t ?
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{3\} \{4\} \rangle$	Yes
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{3\} \{6\} \rangle$	Yes

$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{1,3\} \{6\} \rangle$	No
---	---------------------------------	----

In general, the longer the *maxspan*, the more likely it is to detect a pattern in a data sequence. However, a longer *maxspan* can also capture spurious patterns because it increases the chance for two unrelated events to be temporally related. In addition, the pattern may involve events that are already obsolete.

The *maxspan* constraint affects the support counting step of sequential pattern discovery algorithms. As shown in the preceding examples, some data sequences no longer support a candidate pattern when the *maxspan* constraint is imposed. If we simply apply [Algorithm 6.1](#), the support counts for some patterns may be overestimated. To avoid this problem, the algorithm must be modified to ignore cases where the interval between the first and last occurrences of events in a given pattern is greater than *maxspan*.

The mingap and maxgap Constraints

Timing constraints can also be specified to restrict the time difference between two consecutive elements of a sequence. If the maximum time difference (*maxgap*) is one week, then events in one element must occur within a week's time of the events occurring in the previous element. If the minimum time difference (*mingap*) is zero, then events in one element must occur after the events occurring in the previous element. (See [Figure 6.8](#).) The following table shows examples of patterns that pass or fail the *maxgap* and *mingap* constraints, assuming that $maxgap = 3$ and $mingap = 1$. These examples assume each element occurs at consecutive time steps.

Data Sequence, s	Sequential Pattern, t	<i>maxgap</i>	<i>mingap</i>
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{3\} \{6\} \rangle$	Pass	Pass

$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{6\} \{8\} \rangle$	Pass	Fail
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{1,3\} \{6\} \rangle$	Fail	Pass
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{1\} \{3\} \{8\} \rangle$	Fail	Fail

As with *maxspan*, these constraints will affect the support counting step of sequential pattern discovery algorithms because some data sequences no longer support a candidate pattern when *mingap* and *maxgap* constraints are present. These algorithms must be modified to ensure that the timing constraints are not violated when counting the support of a pattern. Otherwise, some infrequent sequences may mistakenly be declared as frequent patterns.

A side effect of using the *maxgap* constraint is that the *Apriori* principle might be violated. To illustrate this, consider the data set shown in [Figure 6.5](#). Without *mingap* or *maxgap* constraints, the support for $\langle \{2\}\{5\} \rangle$ and $\langle \{2\}\{3\}\{5\} \rangle$ are both equal to 60%. However, if *mingap* = 0 and *maxgap* = 1, then the support for $\langle \{2\}\{5\} \rangle$ reduces to 40%, while the support for $\langle \{2\}\{3\}\{5\} \rangle$ is still 60%. In other words, support has increased when the number of events in a sequence increases—which contradicts the *Apriori* principle. The violation occurs because the object D does not support the pattern $\langle \{2\}\{5\} \rangle$ since the time gap between events 2 and 5 is greater than *maxgap*. This problem can be avoided by using the concept of a contiguous subsequence.

Definition 6.2 (Contiguous Subsequence).

A sequence s is a contiguous subsequence of $w=\langle e_1e_2\dots e_k \rangle$ if any one of the following conditions hold:

1. s is obtained from w after deleting an event from either e_1 or e_k ,
2. s is obtained from w after deleting an event from any element $e_i \in w$ that contains at least two events, or
3. s is a contiguous subsequence of t and t is a contiguous subsequence of w .

The following examples illustrate the concept of a contiguous subsequence:

Data Sequence, s	Sequential Pattern, t	Is t a contiguous subsequence of s ?
$\langle \{1\} \{2,3\} \rangle$	$\langle \{1\} \{2\} \rangle$	Yes
$\langle \{1,2\} \{2\} \{3\} \rangle$	$\langle \{1\} \{2\} \rangle$	Yes
$\langle \{3,4\} \{1,2\} \{2,3\} \{4\} \rangle$	$\langle \{1\} \{2\} \rangle$	Yes
$\langle \{1\} \{3\} \{2\} \rangle$	$\langle \{1\} \{2\} \rangle$	No
$\langle \{1,2\} \{1\} \{3\} \{2\} \rangle$	$\langle \{1\} \{2\} \rangle$	No

Using the concept of contiguous subsequences, the *Apriori* principle can be modified to handle *maxgap* constraints in the following way.

Definition 6.3 (Modified Apriori Principle).

If a k -sequence is frequent, then all of its contiguous $k-1$ -subsequences must also be frequent.

The modified *Apriori* principle can be applied to the sequential pattern discovery algorithm with minor modifications. During candidate pruning, not all k -sequences need to be verified since some of them may violate the *maxgap* constraint. For example, if $\text{maxgap} = 1$, it is not necessary to check whether the subsequence $\langle \{1\}\{2,3\}\{5\} \rangle$ of the candidate $\langle \{1\}\{2,3\}\{4\}\{5\} \rangle$ is frequent since the time difference between elements $\langle \{2,3\} \rangle$ and $\{5\}$ is greater than one time unit. Instead, only the contiguous subsequences of $\langle \{1\}\{2,3\}\{4\}\{5\} \rangle$ need to be examined. These subsequences include $\langle \{1\}\{2,3\}\{4\} \rangle$, $\langle \{2,3\}\{4\}\{5\} \rangle$, $\langle \{1\}\{2\}\{4\}\{5\} \rangle$, and $\langle \{1\}\{3\}\{4\}\{5\} \rangle$.

The Window Size Constraint

Finally, events within an element s_j do not have to occur at the same time. A **window size** threshold (ws) can be defined to specify the maximum allowed time difference between the latest and earliest occurrences of events in any element of a sequential pattern. A window size of 0 means all events in the same element of a pattern must occur simultaneously.

The following example uses $ws=2$ to determine whether a data sequence supports a given sequence (assuming $\text{mingap}=0$, $\text{maxgap}=3$, and $\text{maxspan}=\infty$).

Data Sequence, s	Sequential Pattern, t	Does s support t ?
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{3,4\} \{5\} \rangle$	Yes
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{4,6\}\{8\} \rangle$	Yes
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{3,4,6\}\{8\} \rangle$	No
$\langle \{1,3\} \{3,4\} \{4\} \{5\} \{6,7\} \{8\} \rangle$	$\langle \{1,3,4\}\{6,7,8\} \rangle$	No

In the last example, although the pattern $\langle \{1,3,4\} \{6,7,8\} \rangle$ satisfies the window size constraint, it violates the *maxgap* constraint because the maximum time difference between events in the two elements is 5 units. The window size constraint also affects the support counting step of sequential pattern discovery algorithms. If [Algorithm 6.1](#) is applied without imposing the window size constraint, the support counts for some of the candidate patterns might be underestimated, and thus some interesting patterns may be lost.

6.4.4 Alternative Counting Schemes*

There are multiple ways of defining a sequence given a data sequence. For example, if our database involves long sequences of events, we might be interested in finding subsequences that occur multiple times in the same data sequence. Hence, instead of counting the support of a subsequence as the number of data sequences it is contained in, we can also take into account the *number of times* a subsequence is contained in a data sequence. This viewpoint gives rise to several different formulations for counting the support of a candidate k -sequence from a database of sequences. For illustrative purposes, consider the problem of counting the support for sequence $\langle \{p\}\{q\} \rangle$, as shown in [Figure 6.9](#). Assume that ws=0, mingap=0, maxgap=2, and maxspan=2.

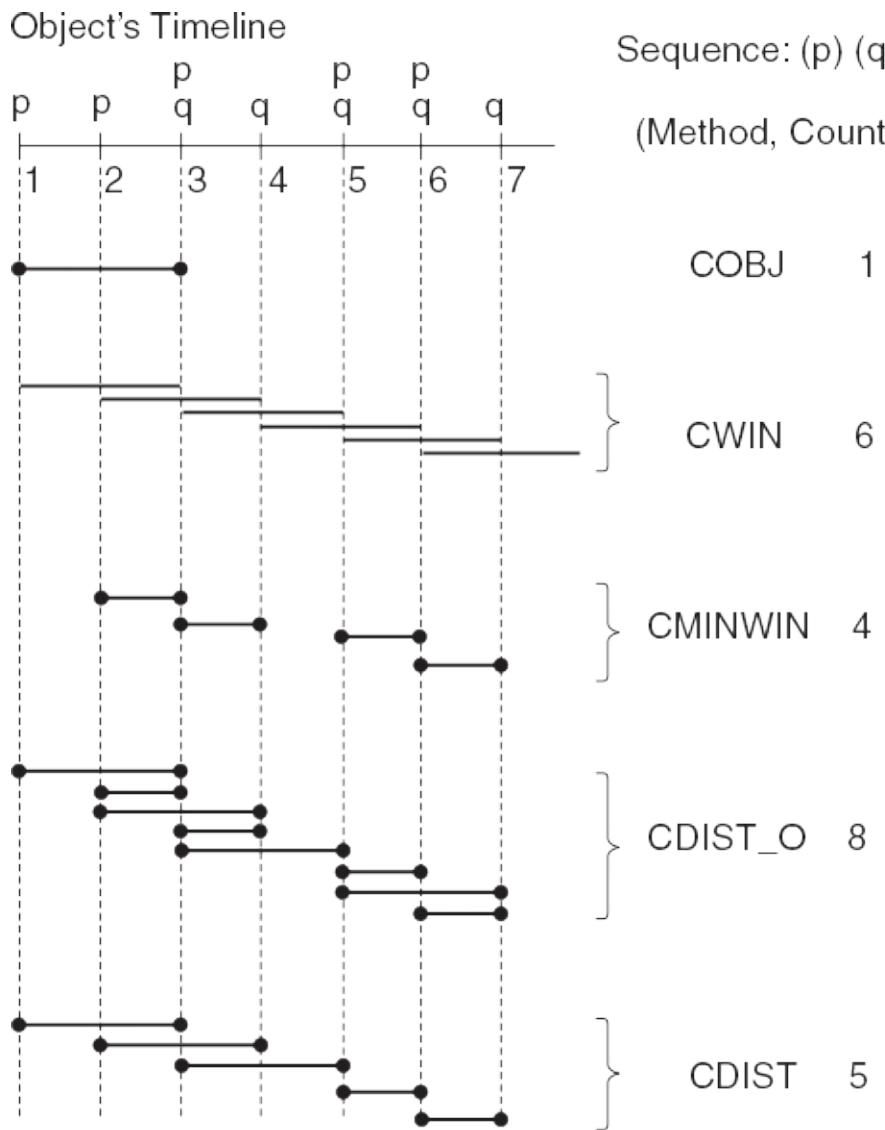


Figure 6.9.

Comparing different support counting methods.

- **COBJ:** One occurrence per object.

This method looks for at least one occurrence of a given sequence in an object's timeline. In [Figure 6.9](#), even though the sequence $\langle (p)(q) \rangle$ appears several times in the object's timeline, it is counted only once—with p occurring at $t=1$ and q occurring at $t=3$.

- **CWIN:** One occurrence per sliding window.

In this approach, a sliding time window of fixed length (*maxspan*) is moved across an object's timeline, one unit at a time. The support count is incremented each time the sequence is encountered in the sliding window.

In [Figure 6.9](#), the sequence $\langle \{p\}\{q\} \rangle$ is observed six times using this method.

- **CMINWIN:** Number of minimal windows of occurrence.

A minimal window of occurrence is the smallest window in which the sequence occurs given the timing constraints. In other words, a minimal window is the time interval such that the sequence occurs in that time interval, but it does not occur in any of the proper subintervals of it. This definition can be considered as a restrictive version of CWIN, because its effect is to shrink and collapse some of the windows that are counted by CWIN. For example, sequence $\langle \{p\}\{q\} \rangle$ has four minimal window occurrences: (1) the pair $(p:t=2,q:t=3)$, (2) the pair $(p:t=3,q:t=4)$, (3) the pair $(p:t=5,q:t=6)$, and (4) the pair $(p:t=6,q:t=7)$. The occurrence of event p at $t=1$ and event q at $t=3$ is not a minimal window occurrence because it contains a smaller window with $(p:t=2,q:t=3)$, which is indeed a minimal window of occurrence.

- **CDIST_O:** Distinct occurrences with possibility of event-timestamp overlap.

A distinct occurrence of a sequence is defined to be the set of event-timestamp pairs such that there has to be at least one new event-timestamp pair that is different from a previously counted occurrence.

Counting all such distinct occurrences results in the CDIST_O method. If the occurrence time of events p and q is denoted as a tuple $(t(p),t(q))$, then this method yields eight distinct occurrences of sequence $\langle \{p\}\{q\} \rangle$ at times $(1,3)$, $(2,3)$, $(2,4)$, $(3,4)$, $(3,5)$, $(5,6)$, $(5,7)$, and $(6,7)$.

- **CDIST:** Distinct occurrences with no event-timestamp overlap allowed.

In CDIST_O above, two occurrences of a sequence were allowed to have overlapping event-timestamp pairs, e.g., the overlap between (1,3) and (2,3). In the CDIST method, no overlap is allowed. Effectively, when an event-timestamp pair is considered for counting, it is marked as used and is never used again for subsequent counting of the same sequence. As an example, there are five distinct, non-overlapping occurrences of the sequence $\langle \{p\}\{q\} \rangle$ in the diagram shown in [Figure 6.9](#). These occurrences happen at times (1,3), (2,4), (3,5), (5,6), and (6,7). Observe that these occurrences are subsets of the occurrences observed in CDIST_O.

One final point regarding the counting methods is the need to determine the baseline for computing the support measure. For frequent itemset mining, the baseline is given by the total number of transactions. For sequential pattern mining, the baseline depends on the counting method used. For the COBJ method, the total number of objects in the input data can be used as the baseline. For the CWIN and CMINWIN methods, the baseline is given by the sum of the number of time windows possible in all objects. For methods such as CDIST and CDIST_O, the baseline is given by the sum of the number of distinct timestamps present in the input data of each object.

6.5 Subgraph Patterns

This section describes the application of association analysis methods to graphs, which are more complex entities than itemsets and sequences. A number of entities such as chemical compounds, 3-D protein structures, computer networks, and tree structured XML documents can be modeled using a graph representation, as shown in [Table 6.8](#).

Table 6.8. Graph representation of entities in various application domains.

Application	Graphs	Vertices	Edges
Web mining	Collection of inter-linked Web pages	Web pages	Hyperlink between pages
Computational chemistry	Chemical compounds	Atoms or ions	Bond between atoms or ions
Computer security	Computer networks	Computers and servers	Interconnection between machines
Semantic Web	XML documents	XML elements	Parent-child relationship between elements
Bioinformatics	3-D Protein structures	Amino acids	Contact residue

A useful data mining task to perform on this type of data is to derive a set of frequently occurring substructures in a collection of graphs. Such a task is known as **frequent subgraph mining**. A potential application of frequent subgraph mining can be seen in the context of computational chemistry. Each

year, new chemical compounds are designed for the development of pharmaceutical drugs, pesticides, fertilizers, etc. Although the structure of a compound is known to play a major role in determining its chemical properties, it is difficult to establish their exact relationship. Frequent subgraph mining can aid this undertaking by identifying the substructures commonly associated with certain properties of known compounds. Such information can help scientists to develop new chemical compounds that have certain desired properties.

This section presents a methodology for applying association analysis to graph-based data. The section begins with a review of some of the basic graph-related concepts and definitions. The frequent subgraph mining problem is then introduced, followed by a description of how the traditional *Apriori* algorithm can be extended to discover such patterns.

6.5.1 Preliminaries

Graphs

A graph is a data structure that can be used to represent relationships among a set of entities. Mathematically, a graph $G=(V,E)$ is composed of a vertex set V and a set of edges E connecting pairs of vertices. Each edge is denoted by a vertex pair (v_i, v_j) , where $v_i, v_j \in V$. A label $l(v_i)$ can be assigned to each vertex v_i representing the name of an entity. Similarly each edge (v_i, v_j) can also be associated with a label $l(v_i, v_j)$ describing the relationship between a pair of entities. **Table 6.8** shows the vertices and edges associated with different types of graphs. For example, in a web graph, the vertices correspond to web pages and the edges represent the hyperlinks between web pages.

Although the size of a graph can generally be represented either by the number of its vertices or its edges, in this chapter, we will consider the size of a graph as its number of edges. Further, we will denote a graph with k edges as a k -graph.

Graph Isomorphism

A basic primitive that is needed to work with graphs is to decide if two graphs with the same number of vertices and edges are *equivalent* to each other, i.e., represent the same structure of relationships among entities. Graph isomorphism provides a formal definition of graph equivalence that serves as a building block for computing similarities among graphs.

Definition 6.4 (Graph Isomorphism).

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic to each other (denoted as $G_1 \simeq G_2$) if there exists functions, $f_v : V_1 \rightarrow V_2$ and $f_e : E_1 \rightarrow E_2$, that map every vertex and edge, respectively, from G_1 to G_2 , such that the following properties are satisfied:

1. **Edge-preserving property:** Two vertices v_a and v_b in G_1 form an edge in G_1 if and only if the vertices $f_v(v_a)$ and $f_v(v_b)$ form an edge in G_2
2. **Label-preserving property:** The labels of two vertices v_a and v_b in G_1 are equal if and only if the labels of $f_v(v_a)$ and $f_v(v_b)$ in G_2 are equal. Similarly, the labels of two edges (v_a, v_b) and (v_c, v_d) in G_1 are equal if and only if the labels of $f_e(v_a, v_b)$ and $f_e(v_c, v_d)$ are equal.

The mapping functions (f_v, f_e) constitute the isomorphism between the graphs G_1 and G_2 . This is denoted as $(f_v, f_e) : G_1 \rightarrow G_2$. An automorphism is a type of isomorphism where a graph is mapped unto itself, i.e., $V_1 = V_2$ and $E_1 = E_2$. **Figure 6.10** shows an example of a graph automorphism where the set of vertex labels in both graphs is $\{A, B\}$. Even though both graphs look different, they are actually isomorphic to each other because there is a one-to-one mapping between the vertices and edges of both graphs. Since the same graph can be depicted in multiple forms, detecting graph automorphism is a non-trivial problem. A common approach to solving this problem is to assign a *canonical label* to every graph, such that every automorphism of a graph shares the same canonical label. Canonical labels can also help in arranging graphs in a particular (canonical) order and checking for duplicates. Techniques for constructing canonical labels are not covered in this chapter, but interested readers may consult the Bibliographic Notes at the end of this chapter for more details.

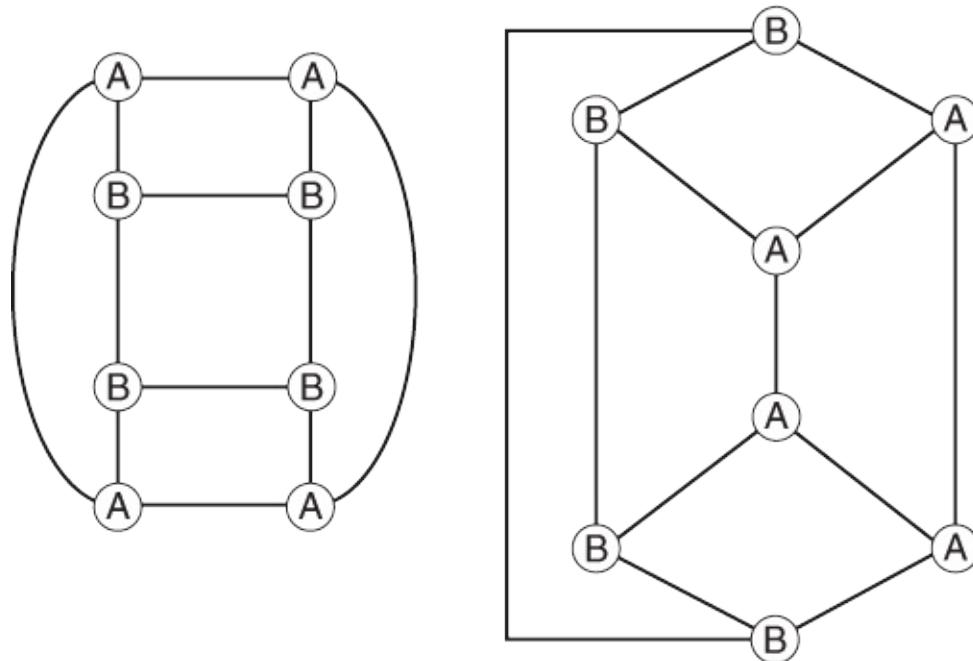


Figure 6.10.

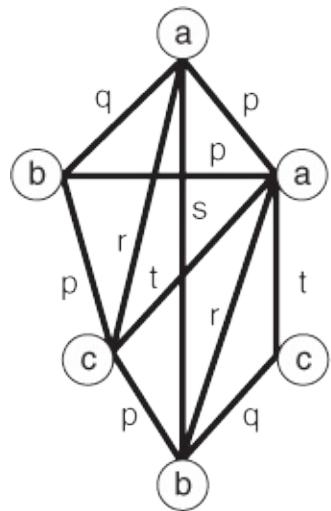
Subgraphs

Definition 6.5 (Subgraph).

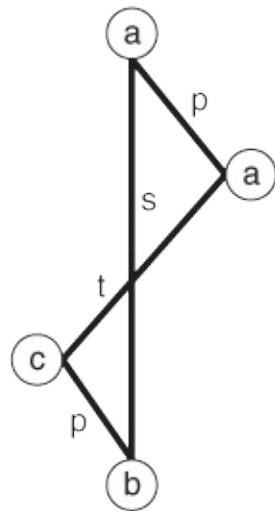
A graph $G'=(V',E')$ is a subgraph of another graph $G=(V,E)$ if its vertex set V' is a subset of V and its edge set E' is a subset of E , such that the endpoints of every edge in E' is contained in V' . The subgraph relationship is denoted as $G' \subseteq S G$.

Example 6.2.

Figure 6.11 shows a graph that contains 6 vertices and 11 edges along with one of its possible subgraphs. The subgraph, which is shown in **Figure 6.11(b)**, contains only 4 of the 6 vertices and 4 of the 11 edges in the original graph.



(a) Labeled graph.



(b) Subgraph.

Figure 6.11.

Example of a subgraph.

Definition 6.6 (Support).

Given a collection of graphs G , the support for a subgraph g is defined as the fraction of all graphs that contain g as its subgraph, i.e.,

$$s(g) = \frac{|\{G \in \mathcal{S} \mid g \subseteq G\}|}{|\mathcal{G}|}. \quad (6.2)$$

Example 6.3.

Consider the five graphs, G_1 through G_5 , shown in [Figure 6.12](#), where the set of vertex labels ranges from a to e but all the edges in the graphs have the same label. The graph g_1 shown on the top right-hand diagram is a subgraph of G_1 , G_3 , G_4 , and G_5 . Therefore, $s(g_1) = 4/5 = 80\%$.

Similarly, we can show that $s(g_2) = 60\%$ because g_2 is a subgraph of G_1 , G_2 , and G_3 , while $s(g_3) = 40\%$ because g_3 is a subgraph of G_1 and G_2 .

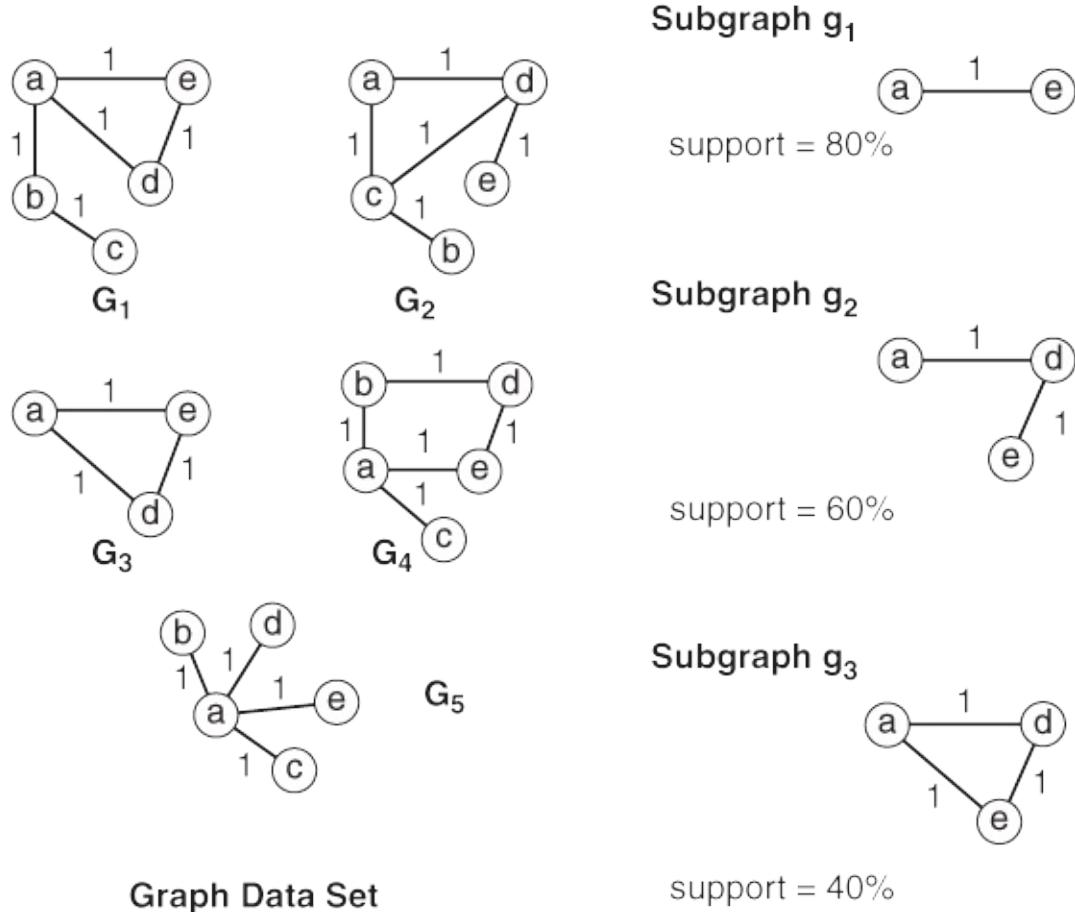


Figure 6.12.

Computing the support of a subgraph from a set of graphs.

6.5.2 Frequent Subgraph Mining

This section presents a formal definition of the frequent subgraph mining problem and illustrates the complexity of this task.

Definition 6.7 (Frequent Subgraph Mining).

Given a set of graphs G and a support threshold, minsup , the goal of frequent subgraph mining is to find all subgraphs g such that $s(g) \geq \text{minsup}$.

While this formulation is generally applicable to any type of graph, the discussion presented in this chapter focuses primarily on **undirected**, **connected** graphs. The definitions of these graphs are given below:

1. A graph is undirected if it contains only undirected edges. An edge (v_i, v_j) is undirected if it is indistinguishable from (v_j, v_i) .
2. A graph is connected if there exists a path between every pair of vertices in the graph, in which a path is a sequence of vertices $\langle v_1 v_2 \dots v_k \rangle$ such that there is an edge connecting every pair of adjacent vertices (v_i, v_{i+1}) in the sequence.

Methods for handling other types of subgraphs (directed or disconnected) are left as an exercise to the readers (see [Exercise 15](#) on page 519).

Mining frequent subgraphs is a computationally expensive task that is much more challenging than mining frequent itemsets or frequent subsequences. The additional complexity in frequent subgraph mining arises due to two major reasons. First, computing the support of a subgraph g given a collection of graphs G is not as straightforward as for itemsets or sequences. This is because it is a non-trivial problem to check if a subgraph g is contained in a graph $g' \in G$, since the same graph g can be present in a different form in g' due to graph isomorphism. The problem of verifying if a graph is a subgraph

of another graph is known as the subgraph isomorphism problem, which is proven to be NP-complete, i.e., there is no known algorithm for this problem that runs in polynomial time.

Second, the number of candidate subgraphs that can be generated from a given set of vertex and edge labels is far larger than the number of candidate itemsets generated using traditional market basket data sets. This is because of the following reasons:

1. A collection of items forms a unique itemset but the same set of edge labels can be arranged in exponential number of ways in a graph, with multiple choices of vertex labels at their endpoints. For example, items p , q , and r form a unique itemset $\{p,q,r\}$, but three edges with labels p , q , and r can form multiple graphs, some examples of which are shown in [Figure 6.13](#).

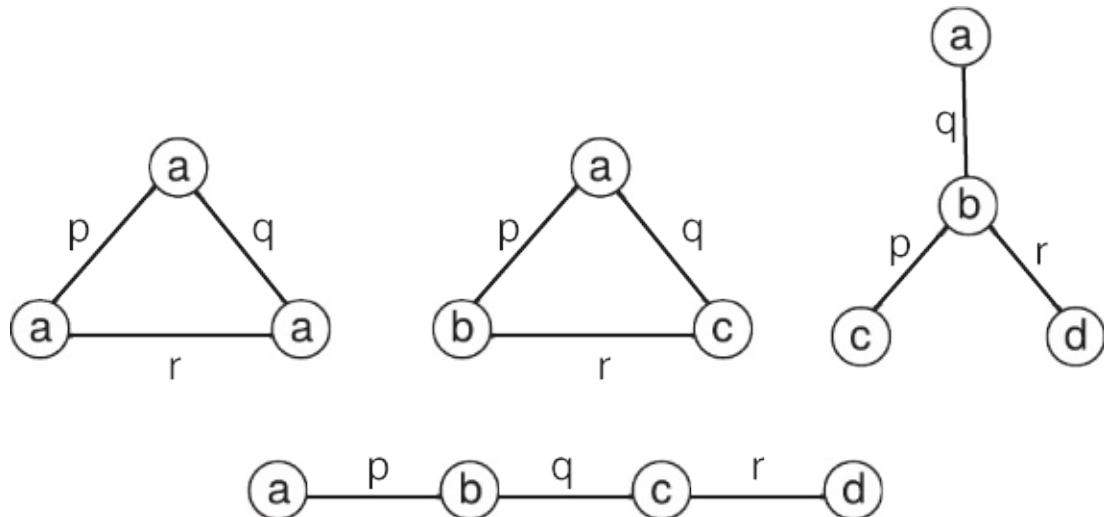


Figure 6.13.

Examples of graphs generated using three edges with labels p , q , and r .

2. An item can appear at most once in an itemset but an edge label can appear multiple times in a graph, because different arrangements of edges with the same edge label represent different graphs. For example, an item p can only generate a single candidate itemset, which is the item itself. However, using a single edge label p and vertex label a , we can generate a number of graphs with different sizes, as shown in [Figure 6.14](#).

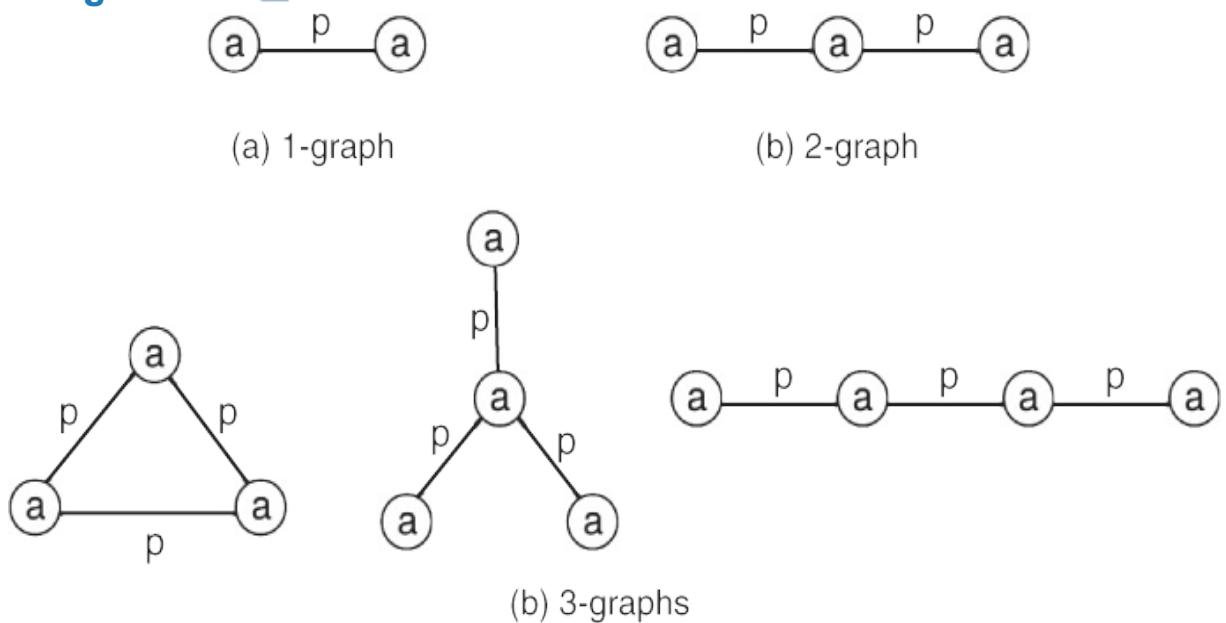


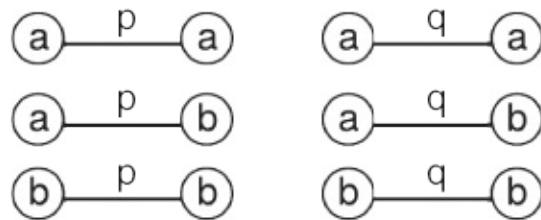
Figure 6.14.

Graphs of sizes one to three generated using a single edge label p and vertex label a .

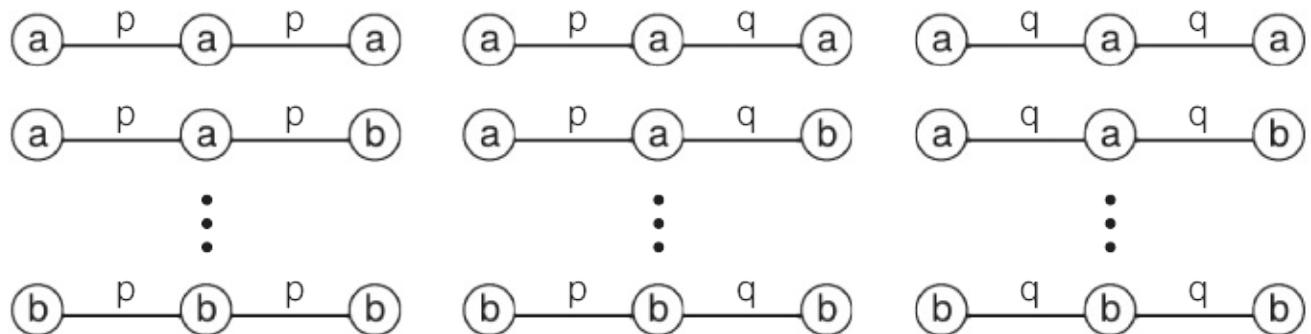
Because of the above reasons, it is challenging to enumerate all possible subgraphs that can be generated using a given set of vertex and edge labels.

[Figure 6.15](#) shows some examples of 1-graphs, 2-graphs, and 3-graphs that can be generated using vertex labels $\{a,b\}$ and edge labels $\{p,q\}$. It can be seen that even using two vertex and edge labels, enumerating all possible graphs becomes difficult even for size two. Hence, it is highly impractical to use a brute-force method for frequent subgraph mining, that enumerates all possible subgraphs and counts their respective supports.

$k = 1$



$k = 2$



$k = 3$

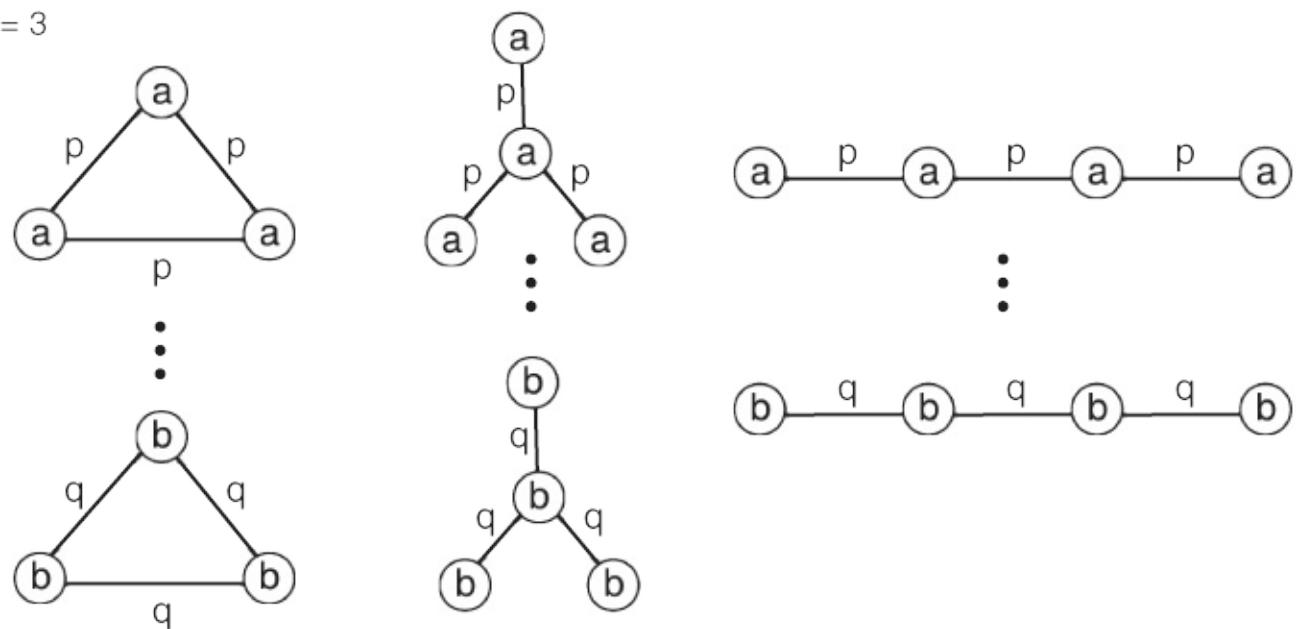


Figure 6.15.

Examples of graphs generated using two edge labels, p and q , and two vertex labels, a and b , for sizes varying from one to three.

However, note that the *Apriori* principle still holds for subgraphs because a k -graph is frequent only if all of its $(k-1)$ -subgraphs are frequent. Hence, despite the computational challenges in enumerating all possible candidate subgraphs, we can use the *Apriori* principle to generate candidate k -subgraphs using frequent $(k-1)$ -subgraphs. **Algorithm 6.2** presents a generic *Apriori*-like approach for frequent subgraph mining. In the following, we briefly describe the three main steps of the algorithm: candidate generation, candidate pruning, and support counting.

Algorithm 6.2 *Apriori*-like algorithm for frequent subgraph mining.

1. $F_1 \leftarrow$ Find all frequent 1-subgraphs in G
2. $F_2 \leftarrow$ Find all frequent 2-subgraphs in G
3. $k=2$.
4. **repeat**
5. $k=k+1$.
6. $C_k = \text{candidate-gen } (F_{k-1})$. {Generate candidate k -subgraphs.}
7. $C_k = \text{candidate-prune } (C_{k-1}, F_{k-1})$. {Perform candidate pruning.}
8. **for** each graph $g \in G$ **do**
9. $C_t = \text{subgraph } (C_k, g)$. {Identify all candidates contained in t .}
10. **for** each candidate k -subgraph $c \in C_t$ **do**
11. $\sigma(c) = \sigma(c) + 1$. {Increment the support count.}
12. **end for**
13. **end for**
14. $F_k = \{ c | c \in C_k \wedge \sigma(c) \geq \text{minsup} \}$. {Extract the frequent k -subgraphs.}
15. **until** $F_k = 0$
16. Answer = $\cup F_k$.

6.5.3 Candidate Generation

A pair of frequent $(k-1)$ -subgraphs are merged to form a candidate k -subgraph if they share a common $(k-2)$ -subgraph, known as their **core**. Given a common core, the subgraph merging procedure can be described as follows:

Subgraph Merging Procedure

Let $G_i(k-1)$ and $G_j(k-1)$ be two frequent $(k-1)$ -subgraphs. Let $G_i(k-1)$ consist of a core $G_i(k-2)$ and an extra edge (u, u') , where u is part of the core. This is depicted in **Figure 6.16(a)** , where the core is represented by a square and the extra edge is represented by a line between u and u' . Similarly, let $G_j(k-1)$ consist of the core $G_j(k-2)$ and the extra edge, (v, v') , as shown in **Figure 6.16(b)** .

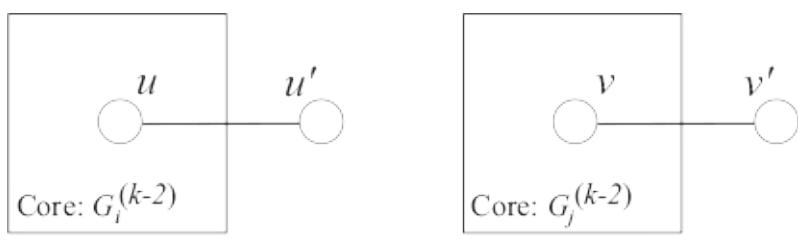
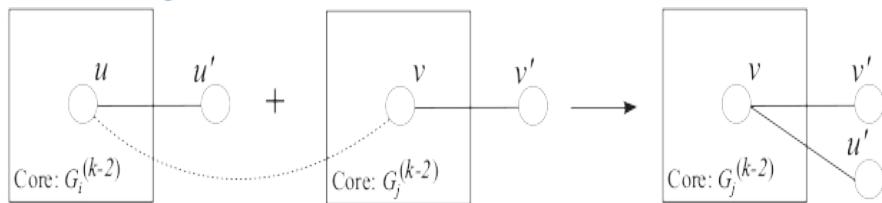


Figure 6.16.

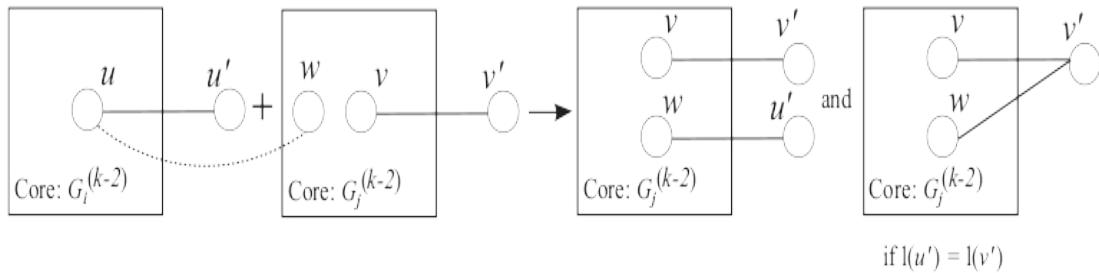
A compact representation of a pair of frequent $(k-1)$ -subgraphs considered for merging.

Using these cores, the two graphs are merged only if there exists an automorphism between the two cores: $(fv, fe): Gi(k-2) \rightarrow Gj(k-2)$. The resulting candidates are obtained by adding an edge to $Gi(k-1)$ as follows:

1. If $fv(u)=v$, i.e., u is mapped to v in the automorphism between the cores, then generate a candidate by adding (v,u') to $Gj(k-1)$, as shown in **Figure 6.17(a)**.



(a) Merging subgraphs when $f_v(u) = v$ (shown using dotted line).



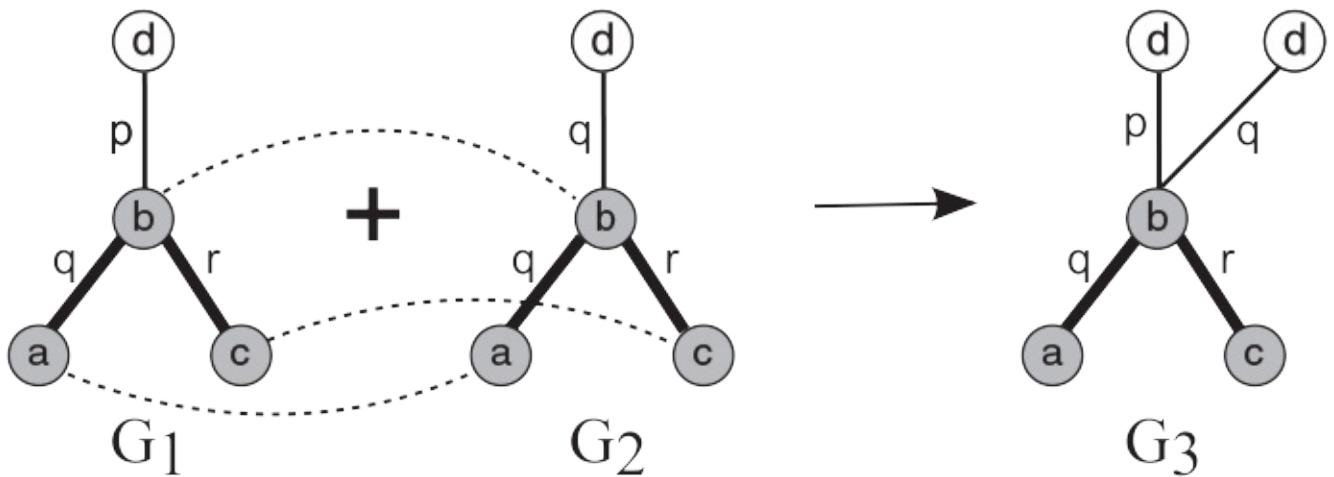
(b) Merging subgraphs when $f_v(u) = w \neq v$ (shown using dotted line).

Figure 6.17.

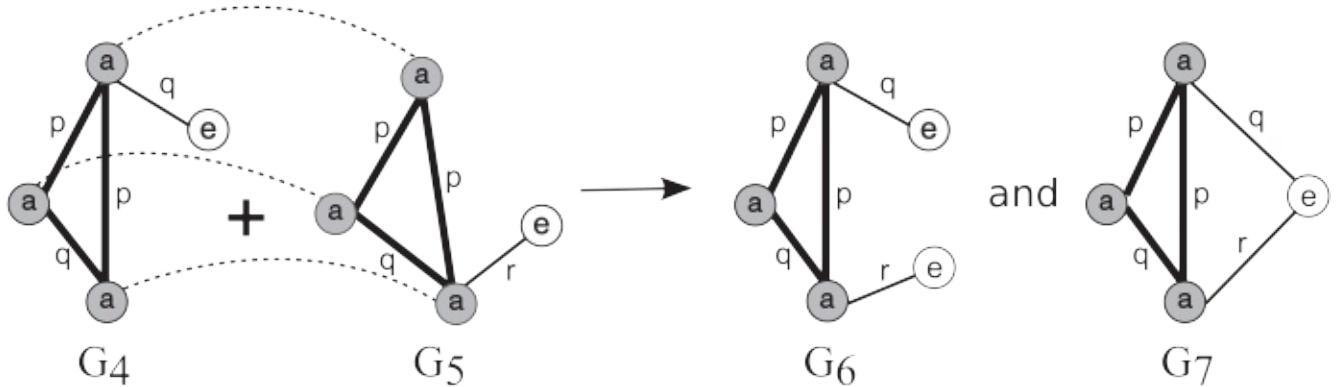
Illustration of Candidate Merging Procedures.

2. If $fv(u)=w \neq v$, i.e., u is not mapped to v but a different vertex w , then generate a candidate by adding (w,u') to $Gj(k-1)$. Additionally, if the labels of u' and v' are identical, then generate another candidate by adding (w,v') to $Gi(k-1)$, as shown in **Figure 6.17(b)**.

Figure 6.18(a) shows the candidate subgraphs generated by merging G1 and G2. The shaded vertices and thicker lines represent the core vertices and edges, respectively, of the two graphs, while the dotted lines represent the mapping between the two cores. Note that this example illustrates condition 1 of the subgraph merging procedure, since the endpoints of the extra edges in both the graphs are mapped to each other. This results in a single candidate subgraph, G3. On the other hand, **Figure 6.18(b)** shows an example of condition 2 of the subgraph merging procedure, where the endpoints of the extra edges do not map to each other and the labels of the new endpoints are identical. Merging the two graphs G4 and G5 thus results in two subgraphs, as shown in the Figure as G6 and G7.



(a) Merging graphs G_1 and G_2 when the endpoints of the extra edges are not mapped to each other, and the labels of the new endpoints are identical.



(b) Merging graphs G_4 and G_5 when the endpoints of the extra edges are mapped to each other.

Figure 6.18.

Two examples of candidate k -subgraph generation using a pair of $(k-1)$ -subgraphs.

The approach presented above of merging two frequent $(k-1)$ -subgraphs is similar to the $F_{k-1} \times F_{k-1}$ candidate generation strategy introduced for itemsets in [Chapter 5](#), and is guaranteed to exhaustively generate all frequent k -subgraphs as viable candidates (see [Exercise 18](#)). However,

there are several notable differences in the candidate generation procedures of itemsets and subgraphs.

1. **Merging with Self:** Unlike itemsets, a frequent $(k-1)$ -subgraph can be merged with itself to create a candidate k -subgraph. This is especially important when a k -graph contains repeating units of edges contained in a $(k-1)$ -subgraph. As an example, the 3-graphs shown in [Figure 6.14](#) can only be generated from the 2-graphs shown in [Figure 6.14](#), if self-merging is allowed.
2. **Multiplicity of Candidates:** As described in the subgraph merging procedure, a pair of frequent $(k-1)$ -subgraphs sharing a common core can generate multiple candidates. As an example, if the labels at the endpoints of the extra edges are identical, i.e., $l(u')=l(v')$, we will generate two candidates as shown in [Figure 6.18\(b\)](#). On the other hand, merging a pair of frequent itemsets or subsequences generates a unique candidate itemset or subsequence.
3. **Multiplicity of Cores:** Two frequent $(k-1)$ -subgraphs can share more than one core of size $k-2$ that is common in both the graphs. [Figure 6.19](#) shows an example of a pair of graphs that share two common cores. Since every choice of a common core can result in a different way of merging the two graphs, this can potentially contribute to the multiplicity of candidates generated by merging the same pair of subgraphs.

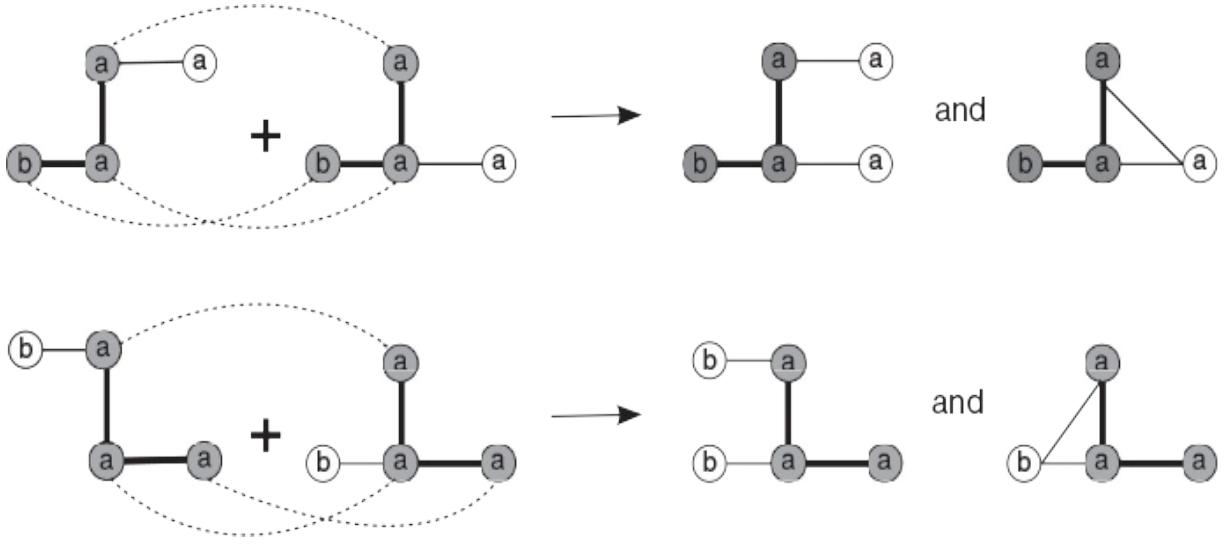


Figure 6.19.

Multiplicity of cores for the same pair of $(k-1)$ -subgraphs.

4. **Multiplicity of Automorphisms:** The common cores of the two graphs can be mapped to each other using multiple choices of mapping functions, each resulting in a different automorphism. To illustrate this, [Figure 6.20](#) shows a pair of graphs that share a common core of size four, represented as a square. The first core can exist in three different forms (rotated views), each resulting in a different mapping between the two cores. Since the choice of the mapping function affects the candidate generation procedure, every automorphism of the core can potentially result in a different set of candidates, as shown in [Figure 6.20](#).

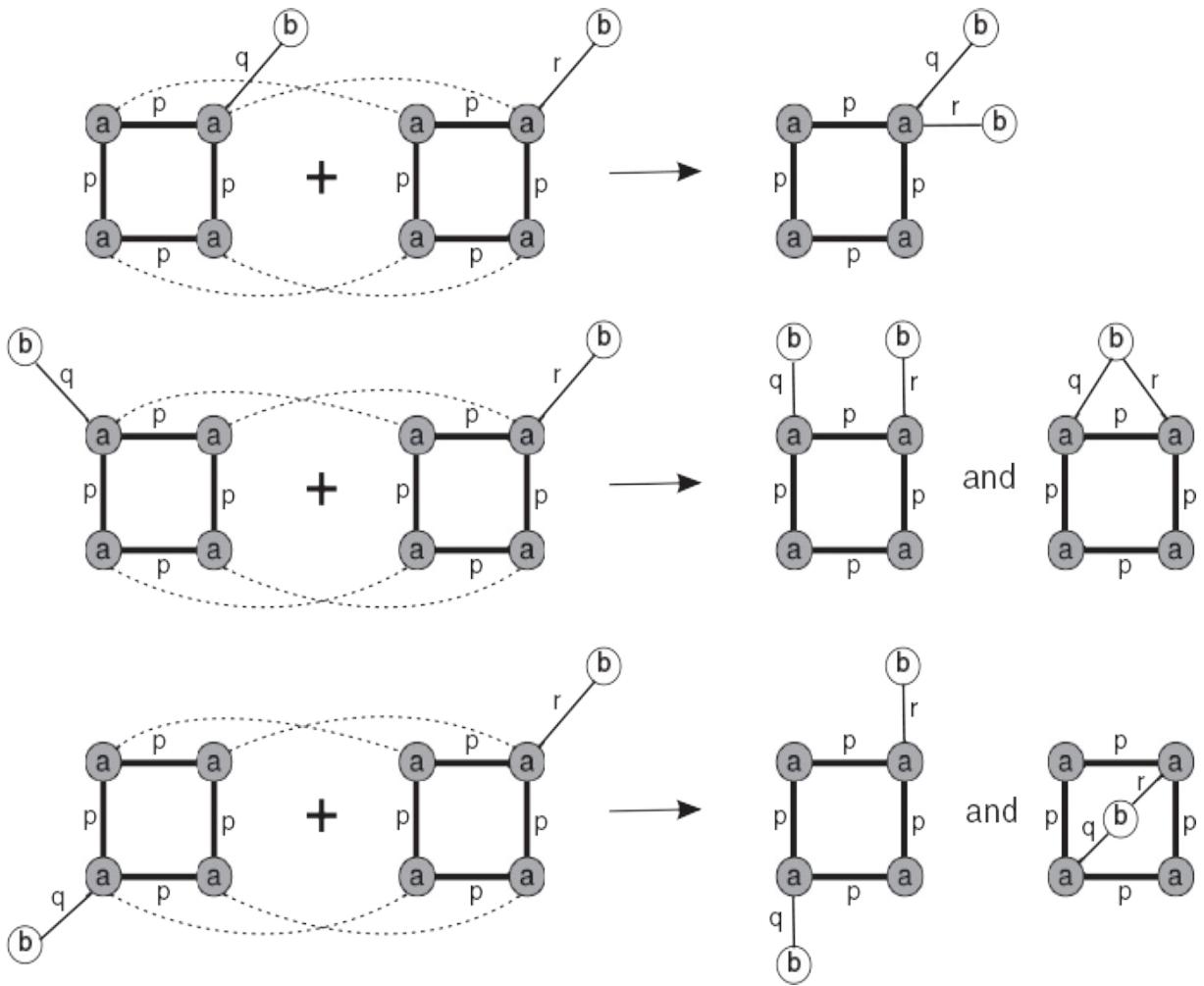


Figure 6.20.

An example showing multiple ways of mapping the cores of two $(k-1)$ -subgraphs with one another.

5. **Generation of Duplicate Candidates:** In the case of itemsets, generation of duplicate candidates is avoided by the use of lexicographic ordering, such that two frequent k -itemsets are merged only if their first $k-1$ items, arranged in lexicographic order, are identical. Unfortunately, in the case of subgraphs, there does not exist a notion of lexicographic ordering among the vertices or edges of a graph. Hence, the same candidate k -subgraph can be generated by merging two different pairs of $k-1$ -subgraphs. **Figure 6.21** shows an

example of a candidate 4-subgraph that can be generated in two different ways, using different pairs of frequent 3-subgraphs. Thus, it is necessary to check for duplicates and eliminate the redundant graphs during candidate pruning.

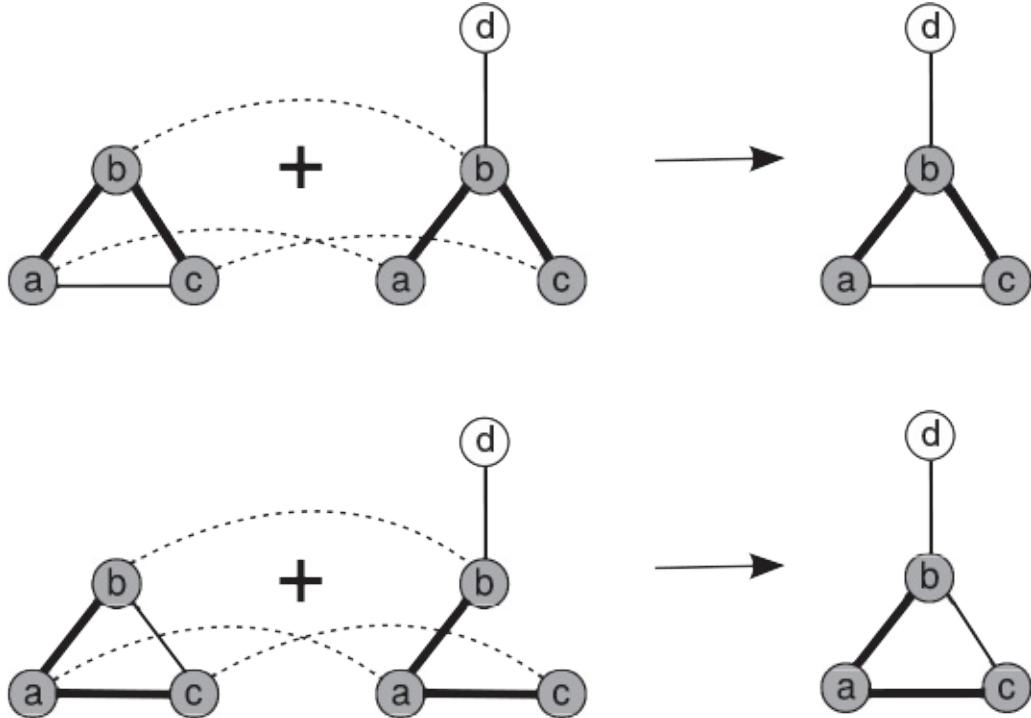


Figure 6.21.

Different pairs of $(k-1)$ -subgraphs can generate the same candidate k -subgraph, thus resulting in duplicate candidates.

Algorithm 6.3 presents the complete procedure for generating the set of all candidate k -subgraphs, C_k , using the set of frequent $(k-1)$ -subgraphs, F_{k-1} . We consider merging every pair of subgraphs in F_{k-1} , including pairs involving the same subgraph twice (to ensure self-merging). For every pair of $(k-1)$ -subgraphs, we consider all possible connected cores of size $k-2$, that can be constructed from the two graphs by removing an edge from each graph. If the two cores are isomorphic, we consider all possible mappings between the vertices and edges of the two cores. For every such mapping, we

employ the subgraph merging procedure to produce candidate k -subgraphs, that are added to C_k .

Algorithm 6.3 Procedure for candidate generation: candidate-gen (F_{k-1}).

1. $C_k = \emptyset$.
2. **for** each pair, $G_i(k-1) \in F_{k-1}$ and $G_j(k-1) \in F_{k-1}$, $i \leq j$ **do**
3. {Considering all pairs of frequent $(k-1)$ -subgraphs for merging.}
4. **for** each pair, $e_i \in G_i(k-1)$ and $e_j \in G_j(k-1)$ **do**
5. {Finding all common cores between a pair of frequent $(k-1)$ -subgraphs.}
6. $G_i(k-2) = G_i(k-1) \setminus e_i$. {Removing an edge from $G_i(k-1)$.}
7. $G_j(k-2) = G_j(k-1) \setminus e_j$. {Removing an edge from $G_j(k-1)$.}
8. **if** $G_i(k-2) \approx G_j(k-2)$ AND $G_i(k-2)$ and $G_j(k-2)$ are connected graphs **then**
9. { $G_i(k-2)$ and $G_j(k-2)$ are common cores of $G_i(k-1)$ and $G_j(k-1)$, respectively.}
10. **for** each $(f_v, f_e): G_i(k-2) \rightarrow G_j(k-2)$ **do**
11. {Generating candidates for every automorphism between the cores.}
12. $C_k = C_k \cup \text{subgraph-merge}(G_i(k-2), G_j(k-2), f_v, f_e, e_i, e_j)$.
13. **end for**
14. **end if**
15. **end for**
16. **end for**
17. Answer = C_k .

6.5.4 Candidate Pruning

After the candidate k -subgraphs are generated, the candidates whose $(k-1)$ -subgraphs are infrequent need to be pruned. The pruning step can be performed by identifying all possible connected $(k-1)$ -subgraphs that can be constructed by removing one edge from a candidate k -subgraph and then checking if they have already been identified as frequent. If any of the $(k-1)$ -subgraphs are infrequent, the candidate k -subgraph is discarded. Also, duplicate candidates need to be detected and eliminated. This can be done by comparing the canonical labels of candidate graphs, since the canonical labels of duplicate graphs will be identical. Canonical labels can also help in checking if a $(k-1)$ -subgraph contained in a candidate k -subgraph is frequent or not, by matching its canonical label with that of every frequent $(k-1)$ -subgraph in F_{k-1} .

6.5.5 Support Counting

Support counting is also a potentially costly operation because all the candidate subgraphs contained in each graph $G \in G$ must be determined. One way to speed up this operation is to maintain a list of graph IDs associated with each frequent $(k-1)$ -subgraph. Whenever a new candidate k -subgraph is generated by merging a pair of frequent $(k-1)$ -subgraphs, their corresponding lists of graph IDs are intersected. Finally, the subgraph isomorphism tests are performed on the graphs in the intersected list to determine whether they contain a particular candidate subgraph.

6.6 Infrequent Patterns*

The association analysis formulation described so far is based on the premise that the presence of an item in a transaction is more important than its absence. As a consequence, patterns that are rarely found in a database are often considered to be uninteresting and are eliminated using the support measure. Such patterns are known as infrequent patterns.

Definition 6.8 (Infrequent Pattern).

An infrequent pattern is an itemset or a rule whose support is less than the *minsup* threshold.

Although a vast majority of infrequent patterns are uninteresting, some of them might be useful to the analysts, particularly those that correspond to negative correlations in the data. For example, the sale of `DVD`s and `VCR`s together is low because any customer who buys a `DVD` will most likely not buy a `VCR`, and vice versa. Such negative-correlated patterns are useful to help identify **competing items**, which are items that can be substituted for one another. Examples of competing items include tea versus coffee, butter versus margarine, regular versus diet soda, and desktop versus laptop computers.

Some infrequent patterns may also suggest the occurrence of interesting rare events or exceptional situations in the data. For example, if `{Fire = Yes}` is

frequent but `{Fire = Yes, Alarm = On}` is infrequent, then the latter is an interesting infrequent pattern because it may indicate faulty alarm systems. To detect such unusual situations, the expected support of a pattern must be determined, so that, if a pattern turns out to have a considerably lower support than expected, it is declared as an interesting infrequent pattern.

Mining infrequent patterns is a challenging endeavor because there is an enormous number of such patterns that can be derived from a given data set. More specifically, the key issues in mining infrequent patterns are: (1) how to identify interesting infrequent patterns, and (2) how to efficiently discover them in large data sets. To get some perspective on various types of interesting infrequent patterns, two related concepts—negative patterns and negatively correlated patterns—are introduced in [Sections 6.6.1](#) and [6.6.2](#), respectively. The relationships among these patterns are elucidated in [Section 6.6.3](#). Finally, two classes of techniques developed for mining interesting infrequent patterns are presented in [Sections 6.6.5](#) and [6.6.6](#).

6.6.1 Negative Patterns

Let $I=\{i_1, i_2, \dots, i_d\}$ be a set of items. A **negative item**, i_k^- denotes the absence of item i_k from a given transaction. For example, `coffee^-` is a negative item whose value is 1 if a transaction does not contain `coffee`.

Definition 6.9 (Negative Itemset).

A negative itemset X is an itemset that has the following properties: (1) $X = A \cup B^-$, where A is a set of positive items, B^- is a set of negative items, $|B^-| \geq 1$, and (2) $s(X) \geq \text{minsup}$.

Definition 6.10 (Negative Association Rule).

A negative association rule is an association rule that has the following properties: (1) the rule is extracted from a negative itemset, (2) the support of the rule is greater than or equal to minsup , and (3) the confidence of the rule is greater than or equal to minconf .

The negative itemsets and negative association rules are collectively known as **negative patterns** throughout this chapter. An example of a negative association rule is $\text{tea} \rightarrow \text{coffee}^-$, which may suggest that people who drink tea tend to not drink coffee.

6.6.2 Negatively Correlated Patterns

[Section 5.7.1](#) on [page 402](#) described how correlation analysis can be used to analyze the relationship between a pair of categorical variables. Measures such as interest factor ([Equation 5.5](#)) and the ϕ -coefficient ([Equation](#)

[5.8](#)) were shown to be useful for discovering itemsets that are positively correlated. This section extends the discussion to negatively correlated patterns.

Definition 6.11 (Negatively Correlated Itemset).

An itemset X , which is defined as $X=\{x_1, x_2, \dots, x_k\}$, is negatively correlated if

$$s(X) < \prod_{j=1}^k s(x_j) = s(x_1) \times s(x_2) \times \dots \times s(x_k), \quad (6.3)$$

where $s(x)$ is the support of the item x .

Note that the support of an itemset is an estimate of the probability that a transaction contains the itemset. Hence, the right-hand side of the preceding expression, $\prod_{j=1}^k s(x_j)$, represents an estimate of the probability that all the items in X are statistically independent. [Definition 6.11](#) suggests that an itemset is negatively correlated if its support is below the expected support computed using the statistical independence assumption. The smaller $s(X)$, the more negatively correlated is the pattern.

Definition 6.12 (Negatively Correlated Association Rule).

An association rule $X \rightarrow Y$ is negatively correlated if

$$s(X \cup Y) < s(X)s(Y), \quad (6.4)$$

where X and Y are disjoint itemsets; i.e., $X \cup Y = 0$.

The preceding definition provides only a partial condition for negative correlation between items in X and items in Y . A full condition for negative correlation can be stated as follows:

$$s(X \cup Y) < \prod_i s(x_i) \prod_j s(y_j), \quad (6.5)$$

where $x_i \in X$ and $y_j \in Y$. Because the items in X (and in Y) are often positively correlated, it is more practical to use the partial condition to define a negatively correlated association rule instead of the full condition. For example, although the rule

```
{eyeglass, lens cleaner} → {contact lens, saline solution}
```

is negatively correlated according to Inequality 6.4, `eyeglass` is positively correlated with `lens cleaner` and `contact lens` is positively correlated with `saline solution`. If Inequality 6.5 is applied instead, such a rule could be missed because it may not satisfy the full condition for negative correlation.

The condition for negative correlation can also be expressed in terms of the support for positive and negative itemsets. Let X^- and Y^- denote the corresponding negative itemsets for X and Y , respectively. Since

$$\begin{aligned}
 s(X \cup Y) - s(X)s(Y) &= s(X \cup Y) - [s(X \cup Y) + s(X \cup Y^-)] [s(X \cup Y) + s(X^- \cup Y)] = s(X \cup Y)[\\
 1 - s(X \cup Y) - s(X \cup Y^-) - s(X^- \cup Y)] - s(X \cup Y^-)s(X^- \cup Y) = s(X \cup Y)s(X^- \cup Y^-) \\
 &\quad - s(X \cup Y^-)s(X^- \cup Y),
 \end{aligned}$$

the condition for negative correlation can be stated as follows:

$$s(X \cup Y)s(X^- \cup Y^-) < s(X \cup Y^-)s(X^- \cup Y). \quad (6.6)$$

The negatively correlated itemsets and association rules are known as **negatively correlated patterns** throughout this chapter.

6.6.3 Comparisons among Infrequent Patterns, Negative Patterns, and Negatively Correlated Patterns

Infrequent patterns, negative patterns, and negatively correlated patterns are three closely related concepts. Although infrequent patterns and negatively correlated patterns refer only to itemsets or rules that contain positive items, while negative patterns refer to itemsets or rules that contain both positive and negative items, there are certain commonalities among these concepts, as illustrated in [Figure 6.22](#).

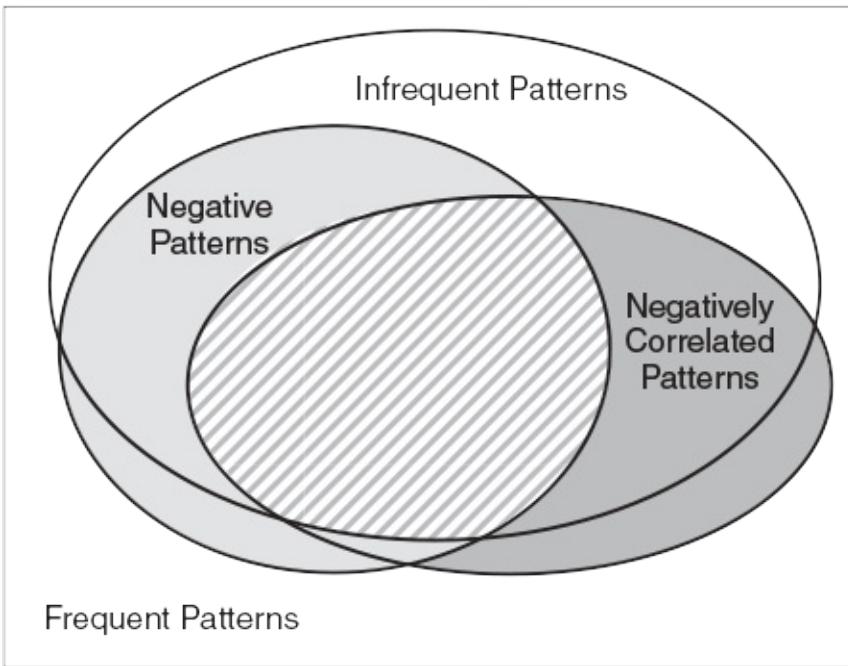


Figure 6.22.

Comparisons among infrequent patterns, negative patterns, and negatively correlated patterns.

First, note that many infrequent patterns have corresponding negative patterns. To understand why this is the case, consider the contingency table shown in [Table 6.9](#). If $X \cup Y$ is infrequent, then it is likely to have a corresponding negative itemset unless minsup is too high. For example, assuming that $\text{minsup} \leq 0.25$, if $X \cup Y$ is infrequent, then the support for at least one of the following itemsets, $X \cup Y^-$, $X^- \cup Y$, or $X^- \cup Y^-$, must be higher than minsup since the sum of the supports in a contingency table is 1.

Table 6.9. A two-way contingency table for the association rule $X \rightarrow Y$.

	Y	Y^-	
X	$s(X \cup Y)$	$s(X \cup Y^-)$	$s(X)$
X^-	$s(X^- \cup Y)$	$s(X^- \cup Y^-)$	$s(X^-)$

	$s(Y)$		$s(Y^-)$		1
--	--------	--	----------	--	---

Second, note that many negatively correlated patterns also have corresponding negative patterns. Consider the contingency table shown in [Table 6.9](#) and the condition for negative correlation stated in Inequality [6.6](#). If X and Y have strong negative correlation, then

$$s(X \cup Y^-) \times s(X^- \cup Y) \gg s(X \cup Y) \times s(X^- \cup Y^-).$$

Therefore, either $X \cup Y^-$ or $X^- \cup Y$, or both, must have relatively high support when X and Y are negatively correlated. These itemsets correspond to the negative patterns. Finally, because the lower the support of $X \cup Y$, the more negatively correlated is the pattern, infrequent patterns tend to be stronger negatively correlated patterns than frequent ones.

6.6.4 Techniques for Mining Interesting Infrequent Patterns

In principle, infrequent itemsets are given by all itemsets that are not extracted by standard frequent itemset generation algorithms such as *Apriori* and FP-growth. These itemsets correspond to those located below the frequent itemset border shown in [Figure 6.23](#).

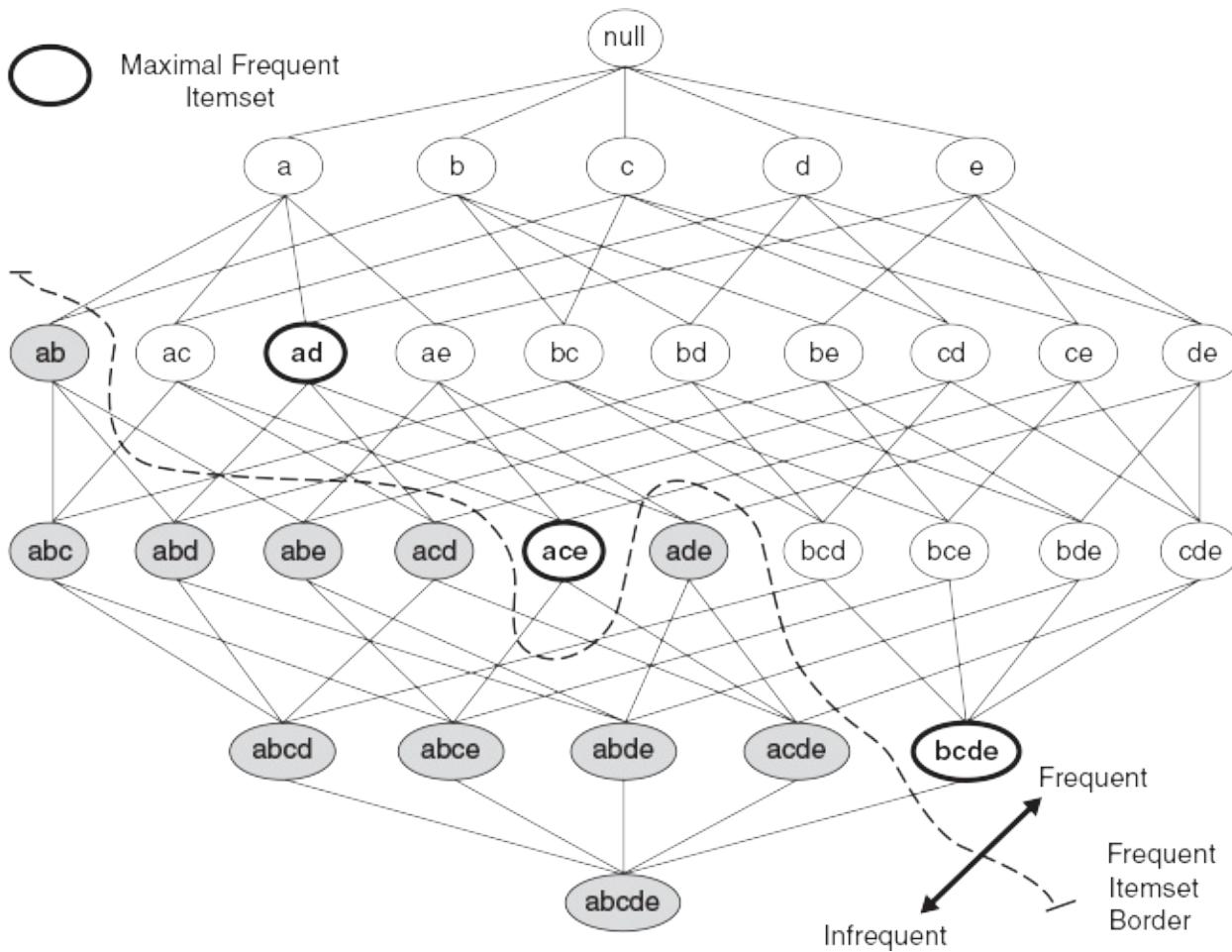


Figure 6.23.

Frequent and infrequent itemsets.

Since the number of infrequent patterns can be exponentially large, especially for sparse, high-dimensional data, techniques developed for mining infrequent patterns focus on finding only interesting infrequent patterns. An example of such patterns includes the negatively correlated patterns discussed in

Section 6.6.2. These patterns are obtained by eliminating all infrequent itemsets that fail the negative correlation condition provided in Inequality 6.3. This approach can be computationally intensive because the supports for all infrequent itemsets must be computed in order to determine whether they are negatively correlated. Unlike the support measure used for mining frequent itemsets, correlation-based measures used for mining negatively correlated itemsets do not possess an anti-monotone property that can be

exploited for pruning the exponential search space. Although an efficient solution remains elusive, several innovative methods have been developed, as mentioned in the Bibliographic Notes provided at the end of this chapter.

The remainder of this chapter presents two classes of techniques for mining interesting infrequent patterns. [Section 6.6.5](#) describes methods for mining negative patterns in data, while [Section 6.6.6](#) describes methods for finding interesting infrequent patterns based on support expectation.

6.6.5 Techniques Based on Mining Negative Patterns

The first class of techniques developed for mining infrequent patterns treats every item as a symmetric binary variable. Using the approach described in [Section 6.1](#), the transaction data can be binarized by augmenting it with negative items. [Figure 6.24](#) shows an example of transforming the original data into transactions having both positive and negative items. By applying existing frequent itemset generation algorithms such as *Apriori* on the augmented transactions, all the negative itemsets can be derived.

The diagram illustrates the process of augmenting a data set with negative items. On the left, a table titled "Original Transactions" shows five transactions (TID 1 to 5) with their respective item sets. An arrow points from this table to a larger table on the right titled "Transactions with Negative Items". The right table has TID as its primary key and includes columns for each item (A, B, C, D) and their negations (\bar{A} , \bar{B} , \bar{C} , \bar{D}). The data is as follows:

TID	Items	TID	A	\bar{A}	B	\bar{B}	C	\bar{C}	D	\bar{D}
1	{A,B}	1	1	0	1	0	0	1	0	1
2	{A,B,C}	2	1	0	1	0	1	0	0	1
3	{C}	3	0	1	0	1	1	0	0	1
4	{B,C}	4	0	1	1	0	1	0	0	1
5	{B,D}	5	0	1	1	0	0	1	1	0

Original Transactions

Transactions with Negative Items

Figure 6.24.

Augmenting a data set with negative items.

Such an approach is feasible only if a few variables are treated as symmetric binary (i.e., we look for negative patterns involving the negation of only a small number of items). If every item must be treated as symmetric binary, the problem becomes computationally intractable due to the following reasons.

1. The number of items doubles when every item is augmented with its corresponding negative item. Instead of exploring an itemset lattice of size $2d$, where d is the number of items in the original data set, the lattice becomes considerably larger, as shown in [Exercise 22](#) on [page 522](#).
2. Support-based pruning is no longer effective when negative items are augmented. For each variable x , either x or x^- has support greater than or equal to 50%. Hence, even if the support threshold is as high as 50%, half of the items will remain frequent. For lower thresholds, many more items and possibly itemsets containing them will be frequent. The support-based pruning strategy employed by *Apriori* is effective only when the support for most itemsets is low; otherwise, the number of frequent itemsets grows exponentially.

- The width of each transaction increases when negative items are augmented. Suppose there are d items available in the original data set. For sparse data sets such as market basket transactions, the width of each transaction tends to be much smaller than d . As a result, the maximum size of a frequent itemset, which is bounded by the maximum transaction width, w_{max} , tends to be relatively small. When negative items are included, the width of the transactions increases to d because an item is either present in the transaction or absent from the transaction, but not both. Since the maximum transaction width has grown from w_{max} to d , this will increase the number of frequent itemsets exponentially. As a result, many existing algorithms tend to break down when they are applied to the extended data set.

The previous brute-force approach is computationally expensive because it forces us to determine the support for a large number of positive and negative patterns. Instead of augmenting the data set with negative items, another approach is to determine the support of the negative itemsets based on the support of their corresponding positive items. For example, the support for $\{p, q^-, r^-\}$ can be computed in the following way:

$$s(\{p, q^-, r^-\}) = s(\{p\}) - s(\{p, q\}) - s(\{p, r\}) + s(\{p, q, r\}).$$

More generally, the support for any itemset $X \cup Y^-$ can be obtained as follows:

$$s(X \cup Y^-) = s(X) + \sum_{i=1}^n \sum_{Z \subseteq Y, |Z|=i} (-1)^i \times s(X \cup Z). \quad (6.7)$$

To apply [Equation 6.7](#), $s(X \cup Z)$ must be determined for every Z that is a subset of Y . The support for any combination of X and Z that exceeds the $minsup$ threshold can be found using the *Apriori* algorithm. For all other combinations, the supports must be determined explicitly, e.g., by scanning the entire set of transactions. Another possible approach is to either ignore the

support for any infrequent itemset $X \cup Z$ or to approximate it with the $minsup$ threshold.

Several optimization strategies are available to further improve the performance of the mining algorithms. First, the number of variables considered as symmetric binary can be restricted. More specifically, a negative item y^- is considered interesting only if y is a frequent item. The rationale for this strategy is that rare items tend to produce a large number of infrequent patterns and many of which are uninteresting. By restricting the set Y^- given in [Equation 6.7](#) to variables whose positive items are frequent, the number of candidate negative itemsets considered by the mining algorithm can be substantially reduced. Another strategy is to restrict the type of negative patterns. For example, the algorithm may consider only a negative pattern $X \cup Y^-$ if it contains at least one positive item (i.e., $|X| \geq 1$). The rationale for this strategy is that if the data set contains very few positive items with support greater than 50%, then most of the negative patterns of the form $X^- \cup Y^-$ will become frequent, thus degrading the performance of the mining algorithm.

6.6.6 Techniques Based on Support Expectation

Another class of techniques considers an infrequent pattern to be interesting only if its actual support is considerably smaller than its expected support. For negatively correlated patterns, the expected support is computed based on the statistical independence assumption. This section describes two alternative approaches for determining the expected support of a pattern

using (1) a concept hierarchy and (2) a neighborhood-based approach known as **indirect association**.

Support Expectation Based on Concept Hierarchy

Objective measures alone may not be sufficient to eliminate uninteresting infrequent patterns. For example, suppose `bread` and `laptop computer` are frequent items. Even though the itemset `{bread, laptop computer}` is infrequent and perhaps negatively correlated, it is not interesting because their lack of support seems obvious to domain experts. Therefore, a subjective approach for determining expected support is needed to avoid generating such infrequent patterns.

In the preceding example, `bread` and `laptop computer` belong to two completely different product categories, which is why it is not surprising to find that their support is low. This example also illustrates the advantage of using domain knowledge to prune uninteresting patterns. For market basket data, the domain knowledge can be inferred from a concept hierarchy such as the one shown in [Figure 6.25](#). The basic assumption of this approach is that items from the same product family are expected to have similar types of interaction with other items. For example, since `ham` and `bacon` belong to the same product family, we expect the association between `ham` and `chips` to be somewhat similar to the association between `bacon` and `chips`. If the actual support for any one of these pairs is less than their expected support, then the infrequent pattern is interesting.

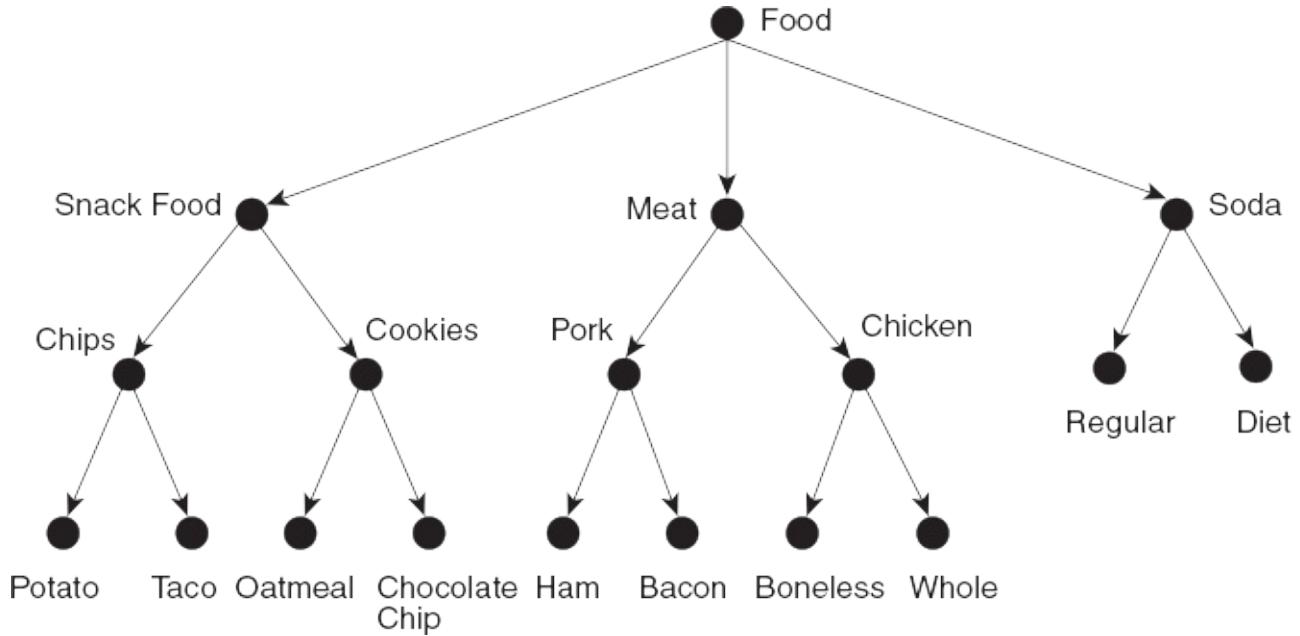


Figure 6.25.

Example of a concept hierarchy.

To illustrate how to compute the expected support, consider the diagram shown in [Figure 6.26](#). Suppose the itemset $\{ C, G \}$ is frequent. Let $s(\cdot)$ denote the actual support of a pattern and $\epsilon(\cdot)$ denote its expected support. The expected support for any children or siblings of C and G can be computed using the formula shown below.

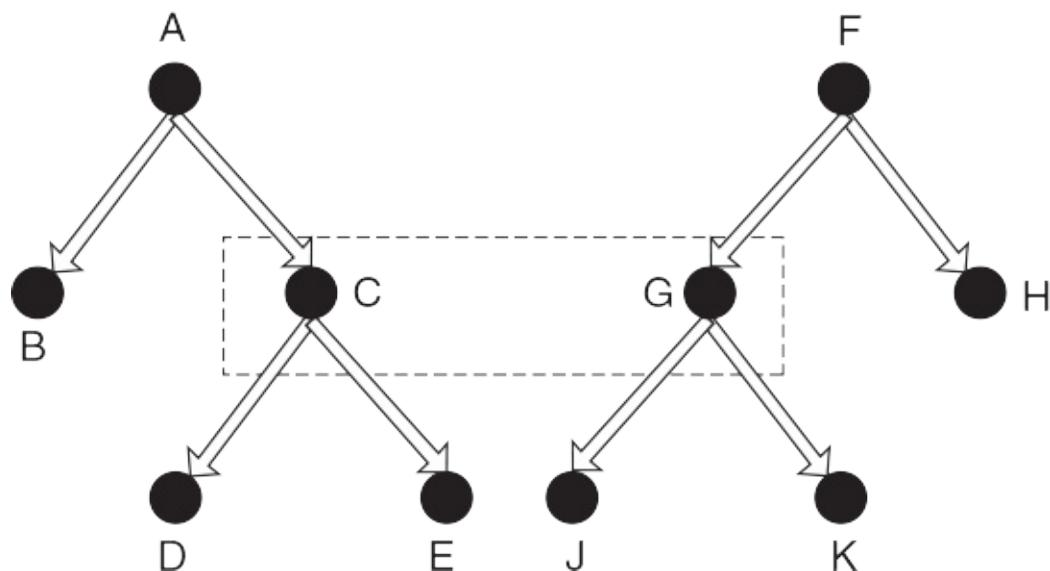


Figure 6.26.

Mining interesting negative patterns using a concept hierarchy.

$$\epsilon(s(E,J)) = s(C,G) \times s(E)s(C) \times s(J)s(G) \quad (6.8)$$

$$\epsilon(s(C,J)) = s(C,G) \times s(J)s(G) \quad (6.9)$$

$$\epsilon(s(C,H)) = s(C,G) \times s(H)s(G) \quad (6.10)$$

For example, if `soda` and `snack food` are frequent, then the expected support between `diet soda` and `chips` can be computed using **Equation 6.8**  because these items are children of `soda` and `snack food`, respectively. If the actual support for `diet soda` and `chips` is considerably lower than their expected value, then `diet soda` and `chips` form an interesting infrequent pattern.

Support Expectation Based on Indirect Association

Consider a pair of items, (a, b) , that are rarely bought together by customers. If a and b are unrelated items, such as `bread` and `DVD player`, then their support is expected to be low. On the other hand, if a and b are related items, then their support is expected to be high. The expected support was previously computed using a concept hierarchy. This section presents an approach for determining the expected support between a pair of items by looking at other items commonly purchased together with these two items.

For example, suppose customers who buy a `sleeping bag` also tend to buy other camping equipment, whereas those who buy a `desktop computer` also

tend to buy other computer accessories such as an optical mouse or a printer. Assuming there is no other item frequently bought together with both a sleeping bag and a desktop computer, the support for these unrelated items is expected to be low. On the other hand, suppose `diet` and `regular soda` are often bought together with `chips` and `cookies`. Even without using a concept hierarchy, both items are expected to be somewhat related and their support should be high. Because their actual support is low, `diet` and `regular soda` form an interesting infrequent pattern. Such patterns are known as **indirect association** patterns.

A high-level illustration of indirect association is shown in [Figure 6.27](#). Items a and b correspond to `diet soda` and `regular soda`, while Y , which is known as the **mediator set**, contains items such as `chips` and `cookies`. A formal definition of indirect association is presented next.

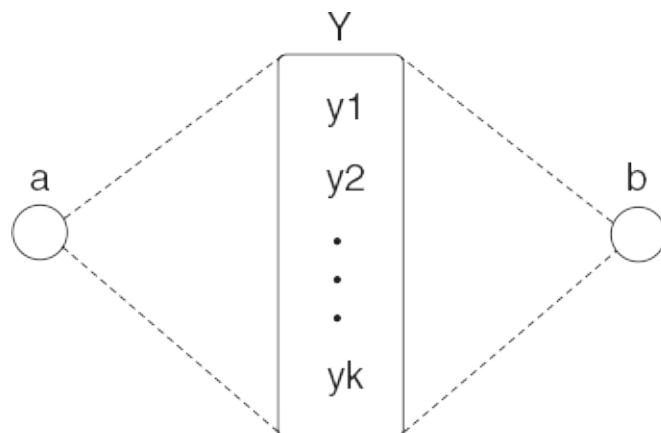


Figure 6.27.

An indirect association between a pair of items.

Definition 6.13 (Indirect Association).

A pair of items a , b is indirectly associated via a mediator set Y if the following conditions hold:

1. $s(\{a,b\}) < ts$ (Itempair support condition).
2. $\exists Y \neq \emptyset$ such that:
 - a. $s(\{a\} \cup Y) \geq tf$ and $s(\{b\} \cup Y) \geq tf$ (Mediator support condition).
 - b. $d(\{a\} \cup Y) \geq td, d(\{b\} \cup Y) \geq td$, where $d(X, Z)$ is an objective measure of the association between X and Z (Mediator dependence condition).

Note that the mediator support and dependence conditions are used to ensure that items in Y form a close neighborhood to both a and b . Some of the dependence measures that can be used include interest, cosine or IS, Jaccard, and other measures previously described in [Section 5.7.1](#) on [page 402](#).

Indirect association has many potential applications. In the market basket domain, a and b may refer to competing items such as `desktop` and `laptop` `computers`. In text mining, indirect association can be used to identify synonyms, antonyms, or words that are used in different contexts. For example, given a collection of documents, the word `data` may be indirectly associated with `gold` via the mediator `mining`. This pattern suggests that the word `mining` can be used in two different contexts—data mining versus gold mining.

Indirect associations can be generated in the following way. First, the set of frequent itemsets is generated using standard algorithms such as *Apriori* or

FP-growth. Each pair of frequent k -itemsets are then merged to obtain a candidate indirect association (a,b,Y) , where a and b are a pair of items and Y is their common mediator. For example, if $\{p,q,r\}$ and $\{p,q,s\}$ are frequent 3-itemsets, then the candidate indirect association $(r,s,\{p,q\})$ is obtained by merging the pair of frequent itemsets. Once the candidates have been generated, it is necessary to verify that they satisfy the itempair support and mediator dependence conditions provided in [Definition 6.13](#). However, the mediator support condition does not have to be verified because the candidate indirect association is obtained by merging a pair of frequent itemsets. A summary of the algorithm is shown in [Algorithm 6.4](#).

Algorithm 6.4 Algorithm for mining indirect associations.

1. Generate F_k , the set of frequent itemsets.
2. **for** $k=2$ to k_{max} **do**
3. $C_k = \{(a, b, Y) | \{a\} \cup Y \in F_k, \{b\} \cup Y \in F_k, a \neq b\}$
4. **for** each candidate $(a, b, Y) \in C_k$ **do**
5. **if** $s(\{a, b\}) < ts \wedge d(\{a\}, Y) \geq td \wedge d(\{b\}, Y) \geq td$ **then**
6. $I_k = I_k \cup \{(a, b, Y)\}$
7. **end if**
8. **end for**
9. **end for**
10. Result = $\cup I_k$.

6.7 Bibliographic Notes

The problem of mining association rules from categorical and continuous data was introduced by [Srikant and Agrawal in \[495\]](#). Their strategy was to binarize the categorical attributes and to apply equal-frequency discretization to the continuous attributes. A **partial completeness** measure was also proposed to determine the amount of information loss as a result of discretization. This measure was then used to determine the number of discrete intervals needed to ensure that the amount of information loss can be kept at a certain desired level. Following this work, numerous other formulations have been proposed for mining quantitative association rules. Instead of discretizing the quantitative attributes, a statistical-based approach was developed by [Aumann and Lindell \[465\]](#), where summary statistics such as mean and standard deviation are computed for the quantitative attributes of the rules. This formulation was later extended by other authors including [Webb \[501\]](#) and [Zhang et al. \[506\]](#). The min-*Apriori* algorithm was developed by [Han et al. \[474\]](#) for finding association rules in continuous data without discretization. Following the min-*Apriori*, a range of techniques for capturing different types of associations among continuous attributes have been explored. For example, the RAnge support Patterns (RAP) developed by [Pandey et al. \[487\]](#) finds groups of attributes that show coherent values across multiple rows of the data matrix. The RAP framework was extended by to deal with noisy data by [Gupta et al. \[473\]](#). Since the rules can be designed to satisfy multiple objectives, evolutionary algorithms for mining quantitative association rules [484, 485] have also been developed. Other techniques include those proposed by [Fukuda et al. \[471\]](#), [Lent et al. \[480\]](#), [Wang et al. \[500\]](#), [Ruckert et al. \[490\]](#) and [Miller and Yang \[486\]](#).

The method described in [Section 6.3](#) for handling concept hierarchy using extended transactions was developed by [Srikant and Agrawal \[494\]](#). An alternative algorithm was proposed by [Han and Fu \[475\]](#), where frequent itemsets are generated one level at a time. More specifically, their algorithm initially generates all the frequent 1-itemsets at the top level of the concept hierarchy. The set of frequent 1-itemsets is denoted as $L(1, 1)$. Using the frequent 1-itemsets in $L(1, 1)$, the algorithm proceeds to generate all frequent 2-itemsets at level 1, $L(1, 2)$. This procedure is repeated until all the frequent itemsets involving items from the highest level of the hierarchy, $L(1, k)$ ($k > 1$), are extracted. The algorithm then continues to extract frequent itemsets at the next level of the hierarchy, $L(2, 1)$, based on the frequent itemsets in $L(1, 1)$. The procedure is repeated until it terminates at the lowest level of the concept hierarchy requested by the user.

The sequential pattern formulation and algorithm described in [Section 6.4](#) was proposed by Agrawal and Srikant in [\[463, 496\]](#). Similarly, [Mannila et al. \[483\]](#) introduced the concept of frequent episode, which is useful for mining sequential patterns from a long stream of events. Another formulation of sequential pattern mining based on regular expressions was proposed by [Garofalakis et al. in \[472\]](#). Joshi et al. have attempted to reconcile the differences between various sequential pattern formulations [\[477\]](#). The result was a universal formulation of sequential pattern with the different counting schemes described in [Section 6.4.4](#). Alternative algorithms for mining sequential patterns were also proposed by [Pei et al. \[489\]](#), [Ayres et al. \[466\]](#), [Cheng et al. \[468\]](#), and [Seno et al. \[492\]](#). A review on sequential pattern mining algorithms can be found in [\[482\]](#) and [\[493\]](#). Extensions of the formulation to maximal [\[470, 481\]](#) and closed [\[499, 504\]](#) sequential pattern mining have also been developed in recent years.

The frequent subgraph mining problem was initially introduced by [Inokuchi et al. in \[476\]](#). They used a vertex-growing approach for generating frequent

induced subgraphs from a graph data set. The edge-growing strategy was developed by **Kuramochi and Karypis in [478]**, where they also presented an *Apriori*-like algorithm called FSG that addresses issues such as multiplicity of candidates, canonical labeling, and vertex invariant schemes. Another frequent subgraph mining algorithm known as gSpan was developed by **Yan and Han in [503]**. The authors proposed using a minimum DFS code for encoding the various subgraphs. Other variants of the frequent subgraph mining problems were proposed by **Zaki in [505]**, **Parthasarathy and Coatney in [488]**, and **Kuramochi and Karypis in [479]**. A recent review on graph mining is given by **Cheng et al. in [469]**.

The problem of mining infrequent patterns has been investigated by many authors. **Savasere et al. [491]** examined the problem of mining negative association rules using a concept hierarchy. **Tan et al. [497]** proposed the idea of mining indirect associations for sequential and non-sequential data. Efficient algorithms for mining negative patterns have also been proposed by **Boulicaut et al. [467]**, **Teng et al. [498]**, **Wu et al. [502]**, and **Antonie and Zaïane [464]**.

Bibliography

- [463] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of Intl. Conf. on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995.
- [464] M.-L. Antonie and O. R. Zaïane. Mining Positive and Negative Association Rules: An Approach for Confined Rules. In *Proc. of the 8th European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 27–38, Pisa, Italy, September 2004.
- [465] Y. Aumann and Y. Lindell. A Statistical Theory for Quantitative Association Rules. In *KDD99*, pages 261–270, San Diego, CA, August 1999.
- [466] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential Pattern mining using a bitmap representation. In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 429–435, Edmonton, Canada, July 2002.
- [467] J.-F. Boulicaut, A. Bykowski, and B. Jeudy. Towards the Tractable Discovery of Association Rules with Negations. In *Proc. of the 4th Intl. Conf on Flexible Query Answering Systems FQAS'00*, pages 425–434, Warsaw, Poland, October 2000.
- [468] H. Cheng, X. Yan, and J. Han. IncSpan: incremental mining of sequential patterns in large database. In *Proc. of the 10th Intl. Conf. on*

Knowledge Discovery and Data Mining, pages 527–532, Seattle, WA, August 2004.

- [469] H. Cheng, X. Yan, and J. Han. Mining Graph Patterns. In C. Aggarwal and J. Han, editors, *Frequent Pattern Mining*, pages 307–338. Springer, 2014.
- [470] P. Fournier-Viger, C.-W. Wu, A. Gomariz, and V. S. Tseng. VMSP: Efficient vertical mining of maximal sequential patterns. In *Proceedings of the Canadian Conference on Artificial Intelligence*, pages 83–94, 2014.
- [471] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining Optimized Association Rules for Numeric Attributes. In *Proc. of the 15th Symp. on Principles of Database Systems*, pages 182–191, Montreal, Canada, June 1996.
- [472] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential Pattern Mining with Regular Expression Constraints. In *Proc. of the 25th VLDB Conf.*, pages 223–234, Edinburgh, Scotland, 1999.
- [473] R. Gupta, N. Rao, and V. Kumar. Discovery of error-tolerant biclusters from noisy gene expression data. *BMC bioinformatics*, 12(12):1, 2011.
- [474] E.-H. Han, G. Karypis, and V. Kumar. Min-Apriori: An Algorithm for Finding Association Rules in Data with Continuous Attributes. <http://www.cs.umn.edu/~han>, 1997.

- [475] J. Han and Y. Fu. Mining Multiple-Level Association Rules in Large Databases. *IEEE Trans. on Knowledge and Data Engineering*, 11(5):798–804, 1999.
- [476] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data. In *Proc. of the 4th European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 13–23, Lyon, France, 2000.
- [477] M. V. Joshi, G. Karypis, and V. Kumar. A Universal Formulation of Sequential Patterns. In *Proc. of the KDD'2001 workshop on Temporal Data Mining*, San Francisco, CA, August 2001.
- [478] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 313–320, San Jose, CA, November 2001.
- [479] M. Kuramochi and G. Karypis. Discovering Frequent Geometric Subgraphs. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 258–265, Maebashi City, Japan, December 2002.
- [480] B. Lent, A. Swami, and J. Widom. Clustering Association Rules. In *Proc. of the 13th Intl. Conf. on Data Engineering*, pages 220–231, Birmingham, U.K, April 1997.
- [481] C. Luo and S. M. Chung. Efficient mining of maximal sequential patterns using multiple samples. In *Proceedings of the SIAM International Conference on Data Mining*, pages 415–426, 2005.

- [482] N. R. Mabroukeh and C. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Survey*, 43(1), 2010.
- [483] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, November 1997.
- [484] D. Martin, A. Rosete, J. Alcalá-Fdez, and F. Herrera. A new multiobjective evolutionary algorithm for mining a reduced set of interesting positive and negative quantitative association rules. *IEEE Transactions on Evolutionary Computation*, 18 (1):54–69, 2014.
- [485] J. Mata, J. L. Alvarez, and J. C. Riquelme. Mining Numeric Association Rules with Genetic Algorithms. In *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 264–267, Prague, Czech Republic, 2001. Springer.
- [486] R. J. Miller and Y. Yang. Association Rules over Interval Data. In *Proc. of 1997 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 452–461, Tucson, AZ, May 1997.
- [487] G. Pandey, G. Atluri, M. Steinbach, C. L. Myers, and V. Kumar. An association analysis approach to biclustering. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 677–686. ACM, 2009.

- [488] S. Parthasarathy and M. Coatney. Efficient Discovery of Common Substructures in Macromolecules. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 362–369, Maebashi City, Japan, December 2002.
- [489] J. Pei, J. Han, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. PrefixSpan: Mining Sequential Patterns efficiently by prefix-projected pattern growth. In *Proc of the 17th Intl. Conf. on Data Engineering*, Heidelberg, Germany, April 2001.
- [490] U. Ruckert, L. Richter, and S. Kramer. Quantitative association rules based on half-spaces: An optimization approach. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 507–510, 2004.
- [491] A. Savasere, E. Omiecinski, and S. Navathe. Mining for Strong Negative Associations in a Large Database of Customer Transactions. In *Proc. of the 14th Intl. Conf. on Data Engineering*, pages 494–502, Orlando, Florida, February 1998.
- [492] M. Seno and G. Karypis. SLPMiner: An Algorithm for Finding Frequent Sequential Patterns Using Length-Decreasing Support Constraint. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 418–425, Maebashi City, Japan, December 2002.
- [493] W. Shen, J. Wang, and J. Han. Sequential Pattern Mining. In C. Aggarwal and J. Han, editors, *Frequent Pattern Mining*, pages 261–282. Springer, 2014.

- [494] R. Srikant and R. Agrawal. Mining Generalized Association Rules. In *Proc. of the 21st VLDB Conf.*, pages 407–419, Zurich, Switzerland, 1995.
- [495] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of 1996 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 1–12, Montreal, Canada, 1996.
- [496] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the 5th Intl Conf. on Extending Database Technology (EDBT'96)*, pages 18–32, Avignon, France, 1996.
- [497] P. N. Tan, V. Kumar, and J. Srivastava. Indirect Association: Mining Higher Order Dependencies in Data. In *Proc. of the 4th European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 632–637, Lyon, France, 2000.
- [498] W. G. Teng, M. J. Hsieh, and M.-S. Chen. On the Mining of Substitution Rules for Statistically Dependent Items. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 442–449, Maebashi City, Japan, December 2002.
- [499] P. Tzvetkov, X. Yan, and J. Han. TSP: Mining top-k closed sequential patterns. *Knowledge and Information Systems*, 7(4):438–457, 2005.
- [500] K. Wang, S. H. Tay, and B. Liu. Interestingness-Based Interval Merger for Numeric Association Rules. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 121–128, New York, NY, August 1998.

- [501] G. I. Webb. Discovering associations with numeric variables. In *Proc. of the 7th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 383–388, San Francisco, CA, August 2001.
- [502] X. Wu, C. Zhang, and S. Zhang. Mining Both Positive and Negative Association Rules. *ACM Trans. on Information Systems*, 22(3):381–405, 2004.
- [503] X. Yan and J. Han. gSpan: Graph-based Substructure Pattern Mining. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 721–724, Maebashi City, Japan, December 2002.
- [504] X. Yan, J. Han, and R. Afshar. CloSpan: Mining: Closed sequential patterns in large datasets. In *Proceedings of the SIAM International Conference on Data Mining*, pages 166–177, 2003.
- [505] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 71–80, Edmonton, Canada, July 2002.
- [506] H. Zhang, B. Padmanabhan, and A. Tuzhilin. On the Discovery of Significant Statistical Quantitative Rules. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 374–383, Seattle, WA, August 2004.

6.8 Exercises

1. Consider the traffic accident data set shown in [Table 6.10](#).

Table 6.10. Traffic accident data set.

WeatherCondition	Driver'sCondition	TrafficViolation	Seat Belt	CrashSeverity
Good	Alcohol-impaired	Exceed speed limit	No	Major
Bad	Sober	None	Yes	Minor
Good	Sober	Disobey stop sign	Yes	Minor
Good	Sober	Exceed speed limit	Yes	Major
Bad	Sober	Disobey traffic signal	No	Major
Good	Alcohol-impaired	Disobey stop sign	Yes	Minor
Bad	Alcohol-impaired	None	Yes	Major
Good	Sober	Disobey traffic signal	Yes	Major
Good	Alcohol-impaired	None	No	Major
Bad	Sober	Disobey traffic signal	No	Major
Good	Alcohol-impaired	Exceed speed limit	Yes	Major
Bad	Sober	Disobey stop sign	Yes	Minor

- Show a binarized version of the data set.
- What is the maximum width of each transaction in the binarized data?

- c. Assuming that the support threshold is 30%, how many candidate and frequent itemsets will be generated?
- d. Create a data set that contains only the following asymmetric binary attributes: (`Weather = Bad`, `Driver's condition = Alcohol-impaired`,
`Traffic violation = Yes`, `Seat Belt = No`, `Crash Severity = Major`). For
`Traffic violation`, only None has a value of 0. The rest of the attribute
values are assigned to 1. Assuming that the support threshold is 30%,
how many candidate and frequent itemsets will be generated?
- e. Compare the number of candidate and frequent itemsets generated in
parts (c) and (d).

2.

- a. Consider the data set shown in [Table 6.11](#). Suppose we apply the following discretization strategies to the continuous attributes of the data set.

Table 6.11. Data set for Exercise 2

TID	Temperature	Pressure	Alarm 1	Alarm 2	Alarm 3
1	95	1105	0	0	1
2	85	1040	1	1	0
3	103	1090	1	1	1
4	97	1084	1	0	0
5	80	1038	0	1	1
6	100	1080	1	1	0
7	83	1025	1	0	1

8	86	1030	1	0	0
9	101	1100	1	1	1

D1:	Partition the range of each continuous attribute into 3 equal-sized bins.
D2:	Partition the range of each continuous attribute into 3 bins; where each bin contains an equal number of transactions

For each strategy, answer the following questions:

- i. Construct a binarized version of the data set.
- ii. Derive all the frequent itemsets having support $\geq 30\%$.
- b. The continuous attribute can also be discretized using a clustering approach.
 - i. Plot a graph of temperature versus pressure for the data points shown in **Table 6.11** [□](#).
 - ii. How many natural clusters do you observe from the graph? Assign a label (C1,C2, etc.) to each cluster in the graph.
 - iii. What type of clustering algorithm do you think can be used to identify the clusters? State your reasons clearly.
 - iv. Replace the temperature and pressure attributes in **Table 6.11** with asymmetric binary attributes C1,C2, etc. Construct a transaction matrix using the new attributes (along with attributes Alarm1, Alarm2, and Alarm3).
 - v. Derive all the frequent itemsets having support $\geq 30\%$ from the binarized data.

3. Consider the data set shown in [Table 6.12](#). The first attribute is continuous, while the remaining two attributes are asymmetric binary. A rule is considered to be strong if its support exceeds 15% and its confidence exceeds 60%. The data given in [Table 6.12](#) supports the following two strong rules:

Table 6.12. Data set for Exercise 3.

A	B	C
1	1	1
2	1	1
3	1	0
4	1	0
5	1	1
6	0	1
7	0	0
8	1	1
9	0	0
10	0	0
11	0	0
12	0	1

$$\{(1 \leq A \leq 2), B=1\} \rightarrow \{C=1\}$$

$$|\quad \{(5 \leq A \leq 8), B=1\} \rightarrow \{C=1\}$$

- a. Compute the support and confidence for both rules.
- b. To find the rules using the traditional *Apriori* algorithm, we need to discretize the continuous attribute A. Suppose we apply the equal width binning approach to discretize the data, with bin-width=2,3,4. For each *bin-width*, state whether the above two rules are discovered by the *Apriori* algorithm. (Note that the rules may not be in the same exact form as before because it may contain wider or narrower intervals for A.) For each rule that corresponds to one of the above two rules, compute its support and confidence.
- c. Comment on the effectiveness of using the equal width approach for classifying the above data set. Is there a *bin-width* that allows you to find both rules satisfactorily? If not, what alternative approach can you take to ensure that you will find both rules?

4. Consider the data set shown in [Table 6.13](#).

Table 6.13. Data set for Exercise 4.

Age (A)	Number of Hours Online per Week (B)				
	0 – 5	5 – 10	10 – 20	20 – 30	30 – 40
10 – 15	2	3	5	3	2
15 – 25	2	5	10	10	3
25 – 35	10	15	5	3	2
35 – 50	4	6	5	3	2

- a. For each combination of rules given below, specify the rule that has the highest confidence.

- i. $15 < A < 25 \rightarrow 10 < B < 20$, $10 < A < 25 \rightarrow 10 < B < 20$, and $15 < A < 35 \rightarrow 10 < B < 20$.
 - ii. $15 < A < 25 \rightarrow 10 < B < 20$, $15 < A < 25 \rightarrow 5 < B < 20$, and $15 < A < 25 \rightarrow 5 < B < 30$
 - iii. $15 < A < 25 \rightarrow 10 < B < 20$ and $10 < A < 35 \rightarrow 5 < B < 30$
- b. Suppose we are interested in finding the average number of hours spent online per week by Internet users between the age of 15 and 35. Write the corresponding statistics-based association rule to characterize the segment of users. To compute the average number of hours spent online, approximate each interval by its midpoint value (e.g., use $B=7.5$ to represent the interval $5 < B < 10$).
- c. Test whether the quantitative association rule given in part (b) is statistically significant by comparing its mean against the average number of hours spent online by other users who do not belong to the age group.
5. For the data set with the attributes given below, describe how you would convert it into a binary transaction data set appropriate for association analysis. Specifically, indicate for each attribute in the original data set
- a. how many binary attributes it would correspond to in the transaction data set,
 - b. how the values of the original attribute would be mapped to values of the binary attributes, and
 - c. if there is any hierarchical structure in the data values of an attribute that could be useful for grouping the data into fewer binary attributes.

The following is a list of attributes for the data set along with their possible values. Assume that all attributes are collected on a per-student basis:

- **Year:** Freshman, Sophomore, Junior, Senior, Graduate:Masters, Graduate:PhD, Professional
- **Zip code:** zip code for the home address of a U.S. student, zip code for the local address of a non-U.S. student
- **College:** Agriculture, Architecture, Continuing Education, Education, Liberal Arts, Engineering, Natural Sciences, Business, Law, Medical, Dentistry, Pharmacy, Nursing, Veterinary Medicine
- **On Campus:** 1 if the student lives on campus, 0 otherwise
- Each of the following is a separate attribute that has a value of 1 if the person speaks the language and a value of 0, otherwise.
 - Arabic
 - Bengali
 - Chinese Mandarin
 - English
 - Portuguese
 - Russian
 - Spanish

6. Consider the data set shown in **Table 6.14**. Suppose we are interested in extracting the following association rule:

$$\{\alpha_1 \leq \text{Age} \leq \alpha_2, \text{Play Piano} = \text{Yes}\} \rightarrow \{\text{Enjoy Classical Music} = \text{Yes}\}$$

Table 6.14. Data set for **Exercise 6**.

Age	Play Piano	Enjoy Classical Music

9	Yes	Yes
11	Yes	Yes
14	Yes	No
17	Yes	No
19	Yes	Yes
21	No	No
25	No	No
29	Yes	Yes
33	No	No
39	No	Yes
41	No	No
47	No	Yes

To handle the continuous attribute, we apply the equal-frequency approach with 3, 4, and 6 intervals. Categorical attributes are handled by introducing as many new asymmetric binary attributes as the number of categorical values. Assume that the support threshold is 10% and the confidence threshold is 70%.

- Suppose we discretize the Age attribute into 3 equal-frequency intervals. Find a pair of values for α_1 and α_2 that satisfy the minimum support and minimum confidence requirements.
- Repeat part (a) by discretizing the Age attribute into 4 equal-frequency intervals. Compare the extracted rules against the ones you had obtained

in part (a).

- c. Repeat part (a) by discretizing the Age attribute into 6 equal-frequency intervals. Compare the extracted rules against the ones you had obtained in part (a).
 - d. From the results in part (a), (b), and (c), discuss how the choice of discretization intervals will affect the rules extracted by association rule mining algorithms.
7. Consider the transactions shown in [Table 6.15](#), with an item taxonomy given in [Figure 6.25](#).

Table 6.15. Example of market basket transactions.

Transaction ID	Items Bought
1	Chips, Cookies, Regular Soda, Ham
2	Chips, Ham, Boneless Chicken, Diet Soda
3	Ham, Bacon, Whole Chicken, Regular Soda
4	Chips, Ham, Boneless Chicken, Diet Soda
5	Chips, Bacon, Boneless Chicken
6	Chips, Ham, Bacon, Whole Chicken, Regular Soda
7	Chips, Cookies, Boneless Chicken, Diet Soda

- a. What are the main challenges of mining association rules with item taxonomy?
- b. Consider the approach where each transaction t is replaced by an extended transaction t' that contains all the items in t as well as their

respective ancestors. For example, the transaction $t = \{\text{Chips}, \text{ Cookies}\}$ will be replaced by $t = \{\text{Chips}, \text{ Cookies}, \text{ Snack Food}, \text{ Food}\}$. Use this approach to derive all frequent itemsets (up to size 4) with support $\geq 70\%$.

- c. Consider an alternative approach where the frequent itemsets are generated one level at a time. Initially, all the frequent itemsets involving items at the highest level of the hierarchy are generated. Next, we use the frequent itemsets discovered at the higher level of the hierarchy to generate candidate itemsets involving items at the lower levels of the hierarchy. For example, we generate the candidate itemset $\{\text{Chips}, \text{ Diet Soda}\}$ only if $\{\text{Snack Food}, \text{ Soda}\}$ is frequent. Use this approach to derive all frequent itemsets (up to size 4) with support $\geq 70\%$.
- d. Compare the frequent itemsets found in parts (b) and (c). Comment on the efficiency and completeness of the algorithms.

8. The following questions examine how the support and confidence of an association rule may vary in the presence of a concept hierarchy.

- a. Consider an item x in a given concept hierarchy. Let x^-1, x^-2, \dots, x^-k denote the k children of x in the concept hierarchy. Show that $s(x) \leq \sum_{i=1}^k s(x^-i)$, where $s(\cdot)$ is the support of an item. Under what conditions will the inequality become an equality?
- b. Let p and q denote a pair of items, while p^\wedge and q^\wedge are their corresponding parents in the concept hierarchy. If $s(\{p, q\}) > \text{minsup}$, which of the following itemsets are guaranteed to be frequent? (i) $s(\{p^\wedge, q\})$, (ii) $s(\{p, q^\wedge\})$, and (iii) $s(\{p^\wedge, q^\wedge\})$.
- c. Consider the association rule $\{p\} \rightarrow \{q\}$. Suppose the confidence of the rule exceeds minconf . Which of the following rules are guaranteed to have confidence higher than minconf ? (i) $\{p\} \rightarrow \{q^\wedge\}$, (ii) $\{p^\wedge\} \rightarrow \{q\}$, and (iii) $\{p^\wedge\} \rightarrow \{q^\wedge\}$.

9.

- a. List all the 4-subsequences contained in the following data sequence:

{ 1,3 } { 2 } { 2,3 } { 4 },

assuming no timing constraints.

- b. List all the 3-element subsequences contained in the data sequence for part (a) assuming that no timing constraints are imposed.
- c. List all the 4-subsequences contained in the data sequence for part (a) (assuming the timing constraints are flexible).
- d. List all the 3-element subsequences contained in the data sequence for part (a) (assuming the timing constraints are flexible).

10. Find all the frequent subsequences with support $\geq 50\%$ given the sequence database shown in **Table 6.16**. Assume that there are no timing constraints imposed on the sequences.

Table 6.16. Example of event sequences generated by various sensors.

Sensor	Timestamp	Events
S1	1	A, B
	2	C
	3	D, E
	4	C
S2	1	A, B
	2	C, D

	3	E
S3	1	B
	2	A
	3	B
	4	D, E
S4	1	C
	2	D, E
	3	C
	4	E
S5	1	B
	2	A
	3	B, C
	4	A, D

11.

- a. For each of the sequences $w = \langle e_1e_2\dots e_i\dots e_{i+1}\dots e_{last} \rangle$ given below, determine whether they are subsequences of the sequence

$\langle \{1,2,3\}\{2,4\}\{2,4,5\}\{3,5\}\{6\} \rangle$

subjected to the following timing constraints:

mingap = 0	(interval between last event in e_i and first event in e_{i+1} is > 0)

maxgap = 3	(interval between first event in e_i and last event in e_{i+1} is ≤ 3)
maxspan = 5	(interval between first event in e_i and last event in e_{last} is ≤ 5)
ws = 1	(time between first and last events in e_i is ≤ 1)

- $w = \langle \{1\}\{2\}\{3\} \rangle$
- $w = \langle \{1,2,3,4\}\{5,6\} \rangle$
- $w = \langle \{2,4\}\{2,4\}\{6\} \rangle$
- $w = \langle \{1\}\{2,4\}\{6\} \rangle$
- $w = \langle \{1,2\}\{3,4\}\{5,6\} \rangle$

b. Determine whether each of the subsequences w given in the previous question are contiguous subsequences of the following sequences s .

- $s = \langle \{1,2,3,4,5,6\}\{1,2,3,4,5,6\}\{1,2,3,4,5,6\} \rangle$
- $s = \langle \{1,2,3,4\}\{1,2,3,4,5,6\}\{3,4,5,6\} \rangle$
- $s = \langle \{1,2\}\{1,2,3,4\}\{3,4,5,6\}\{5,6\} \rangle$
- $s = \langle \{1,2,3\}\{2,3,4,5\}\{4,5,6\} \rangle$

12. For each of the sequence $w = \langle e_1, \dots, e_{last} \rangle$ below, determine whether they are subsequences of the following data sequence:

$\langle \{A,B\}\{C,D\}\{A,B\}\{C,D\}\{A,B\}\{C,D\} \rangle$

subjected to the following timing constraints:

mingap = 0	(interval between last event in e_i and first event in e_{i+1} is > 0)
maxgap = 2	(interval between first event in e_i and last event in e_{i+1} is ≤ 2)

maxspan = 6	(interval between first event in ei and last event in elast is ≤ 6)
ws = 1	(time between first and last events in ei is ≤ 1)

- a. $w = \langle \{A\}\{B\}\{C\}\{D\} \rangle$
- b. $w = \langle \{A\}\{B,C,D\}\{A\} \rangle$
- c. $w = \langle \{A\}\{A,B,C,D\}\{A\} \rangle$
- d. $w = \langle \{B,C\}\{A,D\}\{B,C\} \rangle$
- e. $w = \langle \{A,B,C,D\}\{A,B,C,D\} \rangle$

13. Consider the following frequent 3-sequences:

$\langle \{1,2,3\} \rangle, \langle \{1,2\}\{3\} \rangle, \langle \{1\}\{2,3\} \rangle, \langle \{1,2\}\{4\} \rangle, \langle \{1,3\}\{4\} \rangle, \langle \{1,2,4\} \rangle, \langle \{2,3\}\{3\} \rangle, \langle \{2,3\}\{4\} \rangle, \langle \{2\}\{3\}\{4\} \rangle$, and $\langle \{2\}\{3\}\{4\} \rangle$.

- a. List all the candidate 4-sequences produced by the candidate generation step of the GSP algorithm.
- b. List all the candidate 4-sequences pruned during the candidate pruning step of the GSP algorithm (assuming no timing constraints).
- c. List all the candidate 4-sequences pruned during the candidate pruning step of the GSP algorithm (assuming $maxgap = 1$).

14. Consider the data sequence shown in [Table 6.17](#) for a given object. Count the number of occurrences for the sequence $\langle \{p\}\{q\}\{r\} \rangle$ according to the following counting methods:

Table 6.17. Example of event sequence data for Exercise 14

Timestamp	Events
1	p, q

2	r
3	s
4	p, q
5	r, s
6	p
7	q, r
8	q, s
9	p
10	q, r, s

- a. COBJ (one occurrence per object).
- b. CWIN (one occurrence per sliding window).
- c. CMINWIN (number of minimal windows of occurrence).
- d. CDIST_O (distinct occurrences with possibility of event-timestamp overlap).
- e. CDIST (distinct occurrences with no event timestamp overlap allowed).

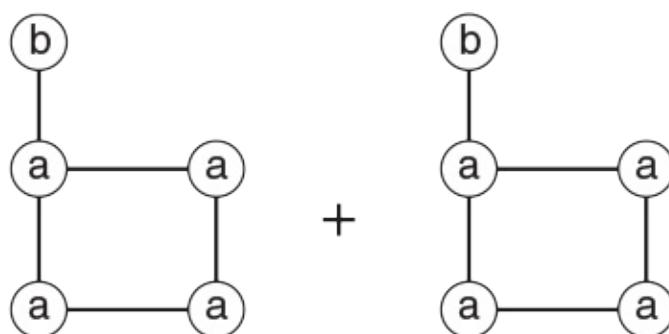
15. Describe the types of modifications necessary to adapt the frequent subgraph mining algorithm to handle:

- a. Directed graphs
- b. Unlabeled graphs
- c. Acyclic graphs

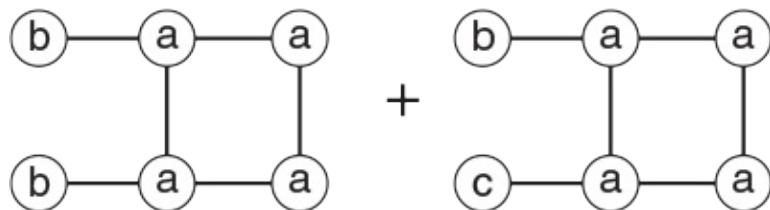
d. Disconnected graphs

For each type of graph given above, describe which step of the algorithm will be affected (candidate generation, candidate pruning, and support counting), and any further optimization that can help improve the efficiency of the algorithm.

16. Draw all candidate subgraphs obtained from joining the pair of graphs shown in [Figure 6.28](#).



(a)



(b)

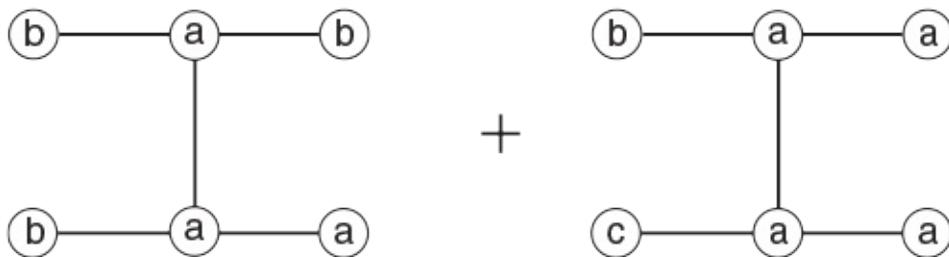
Figure 6.28.

Graphs for [Exercise 16](#).

17. Draw all the candidate subgraphs obtained by joining the pair of graphs shown in [Figure 6.29](#).



(a)



(b)

Figure 6.29.

Graphs for **Exercise 17**

18. Show that the candidate generation procedure introduced in [Section 6.5.3](#) for frequent subgraph mining is complete, i.e., no frequent k -subgraph can be missed from being generated if every pair of frequent $(k-1)$ -subgraphs is considered for merging.

19.

- If support is defined in terms of induced subgraph relationship, show that the confidence of the rule $g_1 \rightarrow g_2$ can be greater than 1 if g_1 and g_2 are allowed to have overlapping vertex sets.
- What is the time complexity needed to determine the canonical label of a graph that contains $|V|$ vertices?
- The core of a subgraph can have multiple automorphisms. This will increase the number of candidate subgraphs obtained after merging two

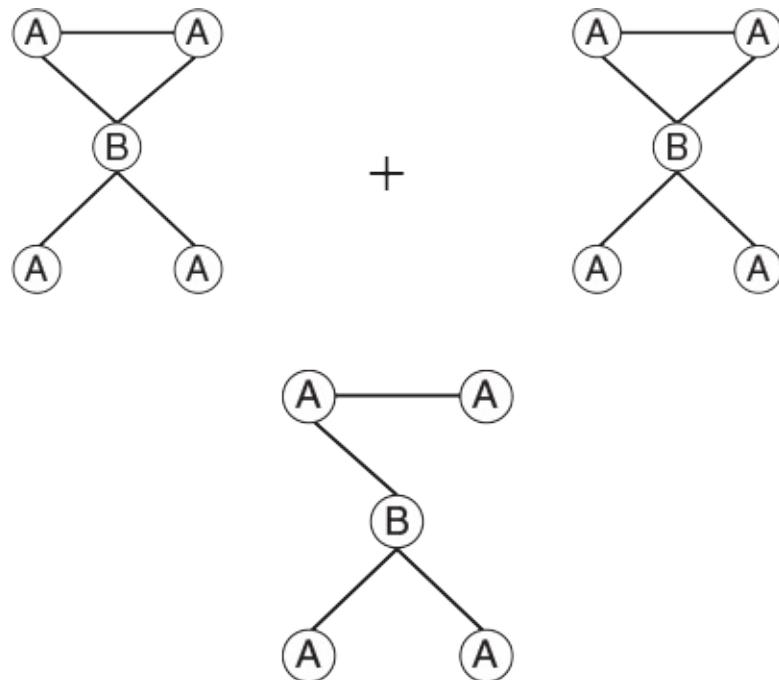
- frequent subgraphs that share the same core. Determine the maximum number of candidate subgraphs obtained due to automorphism of a core of size k .
- d. Two frequent subgraphs of size k may share multiple cores. Determine the maximum number of cores that can be shared by the two frequent subgraphs.

20. (a) Consider the two graphs shown below.

Draw all the distinct cores obtained when merging the two subgraphs.

How many candidates are generated using the following core?

21. The original association rule mining framework considers only presence of items together in the same transaction. There are situations in which itemsets that are infrequent may also be informative. For instance, the itemset TV, DVD, \neg VCR suggests that many customers who buy TVs and DVDs do not buy VCRs.



In this problem, you are asked to extend the association rule framework to negative itemsets (i.e., itemsets that contain both presence and absence of items). We will use the negation symbol (\neg) to refer to absence of items.

- a. A naïve way for deriving negative itemsets is to extend each transaction to include absence of items as shown in [Table 6.18](#).

Table 6.18. Example of numeric data set.

TID	TV	$\neg TV$	DVD	$\neg DVD$	VCR	$\neg VCR$...
1	1	0	0	1	0	1	...
2	1	0	0	1	0	1	...

- i. Suppose the transaction database contains 1000 distinct items. What is the total number of positive itemsets that can be generated from these items? (Note: A positive itemset does not contain any negated items).
- ii. What is the maximum number of frequent itemsets that can be generated from these transactions? (Assume that a frequent itemset may contain positive, negative, or both types of items)
- iii. Explain why such a naïve method of extending each transaction with negative items is not practical for deriving negative itemsets.

- b. Consider the database shown in [Table 6.15](#). What are the support and confidence values for the following negative association rules involving regular and diet soda?

 - i. $\neg \text{Regular} \rightarrow \text{Diet}$.
 - ii. $\text{Regular} \rightarrow \neg \text{Diet}$.
 - iii. $\neg \text{Diet} \rightarrow \text{Regular}$.

iv. $\text{Diet} \rightarrow \neg \text{Regular}$.

22. Suppose we would like to extract positive and negative itemsets from a data set that contains d items.

- a. Consider an approach where we introduce a new variable to represent each negative item. With this approach, the number of items grows from d to $2d$. What is the total size of the itemset lattice, assuming that an itemset may contain both positive and negative items of the same variable?
- b. Assume that an itemset must contain positive or negative items of different variables. For example, the itemset $\{ a, a^-, b, c^- \}$ is invalid because it contains both positive and negative items for variable a . What is the total size of the itemset lattice?

23. For each type of pattern defined below, determine whether the support measure is monotone, anti-monotone, or non-monotone (i.e., neither monotone nor anti-monotone) with respect to increasing itemset size.

- a. Itemsets that contain both positive and negative items such as $\{ a, b, c^-, d^- \}$. Is the support measure monotone, anti-monotone, or non-monotone when applied to such patterns?
- b. Boolean logical patterns such as $\{ (a \vee b \vee c), d, e \}$, which may contain both disjunctions and conjunctions of items. Is the support measure monotone, anti-monotone, or non-monotone when applied to such patterns?

24. Many association analysis algorithms rely on an *Apriori*-like approach for finding frequent patterns. The overall structure of the algorithm is given below.

Suppose we are interested in finding Boolean logical rules such as

$$\{ a \vee b \} \rightarrow \{ c, d \},$$

which may contain both disjunctions and conjunctions of items. The corresponding itemset can be written as $\{ (a \vee b), c, d \}$.

- a. Does the *Apriori* principle still hold for such itemsets?
- b. How should the candidate generation step be modified to find such patterns?
- c. How should the candidate pruning step be modified to find such patterns?
- d. How should the support counting step be modified to find such patterns?

Algorithm 6.5 *Apriori*-like algorithm.

1. $k=1$
2. $F_k = \{ i | i \in I \wedge \sigma(i) \geq \text{minsup} \}$ {Find frequent 1-patterns.}
3. **repeat**
4. $k=k+1$.
5. $C_k = \text{genCandidate}(F_{k-1})$. {Candidate Generation }
6. $C_k = \text{pruneCandidate}(C_k, F_{k-1})$. {Candidate Pruning }
7. $C_k = \text{count}(C_k, D)$. {Support Counting}
8. $F_k = \{ c | c \in C_k \wedge \sigma(c) \geq \text{minsup} \}$. {Extract frequent patterns}
9. **until** $F_k = \emptyset$
10. Answer = $\cup F_k$.

7 Cluster Analysis: Basic Concepts and Algorithms

Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both. If meaningful groups are the goal, then the clusters should capture the natural structure of the data. In some cases, however, cluster analysis is used for data summarization in order to reduce the size of the data. Whether for understanding or utility, cluster analysis has long played an important role in a wide variety of fields: psychology and other social sciences, biology, statistics, pattern recognition, information retrieval, machine learning, and data mining.

There have been many applications of cluster analysis to practical problems. We provide some specific examples, organized by whether the purpose of the clustering is understanding or utility.

Clustering for Understanding Classes, or conceptually meaningful groups of objects that share common characteristics, play an important role in how people analyze and describe the world. Indeed, human beings are skilled at

dividing objects into groups (clustering) and assigning particular objects to these groups (classification). For example, even relatively young children can quickly label the objects in a photograph. In the context of understanding data, clusters are potential classes and cluster analysis is the study of techniques for automatically finding classes. The following are some examples:

- **Biology.** Biologists have spent many years creating a taxonomy (hierarchical classification) of all living things: kingdom, phylum, class, order, family, genus, and species. Thus, it is perhaps not surprising that much of the early work in cluster analysis sought to create a discipline of mathematical taxonomy that could automatically find such classification structures. More recently, biologists have applied clustering to analyze the large amounts of genetic information that are now available. For example, clustering has been used to find groups of genes that have similar functions.
- **Information Retrieval.** The World Wide Web consists of billions of web pages, and the results of a query to a search engine can return thousands of pages. Clustering can be used to group these search results into a small number of clusters, each of which captures a particular aspect of the query. For instance, a query of “movie” might return web pages grouped into categories such as reviews, trailers, stars, and theaters. Each category (cluster) can be broken into subcategories (subclusters), producing a hierarchical structure that further assists a user’s exploration of the query results.
- **Climate.** Understanding the Earth’s climate requires finding patterns in the atmosphere and ocean. To that end, cluster analysis has been applied to find patterns in atmospheric pressure and ocean temperature that have a significant impact on climate.
- **Psychology and Medicine.** An illness or condition frequently has a number of variations, and cluster analysis can be used to identify these different subcategories. For example, clustering has been used to identify

different types of depression. Cluster analysis can also be used to detect patterns in the spatial or temporal distribution of a disease.

- **Business.** Businesses collect large amounts of information about current and potential customers. Clustering can be used to segment customers into a small number of groups for additional analysis and marketing activities.

Clustering for Utility Cluster analysis provides an abstraction from individual data objects to the clusters in which those data objects reside. Additionally, some clustering techniques characterize each cluster in terms of a cluster prototype; i.e., a data object that is representative of the objects in the cluster. These cluster prototypes can be used as the basis for a number of additional data analysis or data processing techniques. Therefore, in the context of utility, cluster analysis is the study of techniques for finding the most representative cluster prototypes.

- **Summarization.** Many data analysis techniques, such as regression or principal component analysis, have a time or space complexity of $O(m^2)$ or higher (where m is the number of objects), and thus, are not practical for large data sets. However, instead of applying the algorithm to the entire data set, it can be applied to a reduced data set consisting only of cluster prototypes. Depending on the type of analysis, the number of prototypes, and the accuracy with which the prototypes represent the data, the results can be comparable to those that would have been obtained if all the data could have been used.
- **Compression.** Cluster prototypes can also be used for data compression. In particular, a table is created that consists of the prototypes for each cluster; i.e., each prototype is assigned an integer value that is its position (index) in the table. Each object is represented by the index of the prototype associated with its cluster. This type of compression is known as **vector quantization** and is often applied to image, sound, and video data,

where (1) many of the data objects are highly similar to one another, (2) some loss of information is acceptable, and (3) a substantial reduction in the data size is desired.

- **Efficiently Finding Nearest Neighbors.** Finding nearest neighbors can require computing the pairwise distance between all points. Often clusters and their cluster prototypes can be found much more efficiently. If objects are relatively close to the prototype of their cluster, then we can use the prototypes to reduce the number of distance computations that are necessary to find the nearest neighbors of an object. Intuitively, if two cluster prototypes are far apart, then the objects in the corresponding clusters cannot be nearest neighbors of each other. Consequently, to find an object's nearest neighbors, it is necessary to compute only the distance to objects in nearby clusters, where the nearness of two clusters is measured by the distance between their prototypes. This idea is made more precise in [Exercise 25](#) of [Chapter 2](#), which is on page [111](#).

This chapter provides an introduction to cluster analysis. We begin with a high-level overview of clustering, including a discussion of the various approaches to dividing objects into sets of clusters and the different types of clusters. We then describe three specific clustering techniques that represent broad categories of algorithms and illustrate a variety of concepts: K-means, agglomerative hierarchical clustering, and DBSCAN. The final section of this chapter is devoted to cluster validity—methods for evaluating the goodness of the clusters produced by a clustering algorithm. More advanced clustering concepts and algorithms will be discussed in [Chapter 8](#). Whenever possible, we discuss the strengths and weaknesses of different schemes. In addition, the Bibliographic Notes provide references to relevant books and papers that explore cluster analysis in greater depth.

7.1 Overview

Before discussing specific clustering techniques, we provide some necessary background. First, we further define cluster analysis, illustrating why it is difficult and explaining its relationship to other techniques that group data. Then we explore two important topics: (1) different ways to group a set of objects into a set of clusters, and (2) types of clusters.

7.1.1 What Is Cluster Analysis?

Cluster analysis groups data objects based on information found only in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering.

In many applications, the notion of a cluster is not well defined. To better understand the difficulty of deciding what constitutes a cluster, consider [Figure 7.1](#), which shows 20 points and three different ways of dividing them into clusters. The shapes of the markers indicate cluster membership. [Figures 7.1\(b\)](#) and [7.1\(d\)](#) divide the data into two and six parts, respectively. However, the apparent division of each of the two larger clusters into three subclusters may simply be an artifact of the human visual system. Also, it may not be unreasonable to say that the points form four clusters, as shown in [Figure 7.1\(c\)](#). This figure illustrates that the definition of a cluster

is imprecise and that the best definition depends on the nature of data and the desired results.



(a) Original points.



(b) Two clusters.



(c) Four clusters.



(d) Six clusters.

Figure 7.1.

Three different ways of clustering the same set of points.

Cluster analysis is related to other techniques that are used to divide data objects into groups. For instance, clustering can be regarded as a form of classification in that it creates a labeling of objects with class (cluster) labels. However, it derives these labels only from the data. In contrast, classification in the sense of [Chapter 3](#) is **supervised classification**; i.e., new, unlabeled objects are assigned a class label using a model developed from objects with known class labels. For this reason, cluster analysis is sometimes referred to as **unsupervised classification**. When the term classification is used without any qualification within data mining, it typically refers to supervised classification.

Also, while the terms **segmentation** and **partitioning** are sometimes used as synonyms for clustering, these terms are frequently used for approaches outside the traditional bounds of cluster analysis. For example, the term

partitioning is often used in connection with techniques that divide graphs into subgraphs and that are not strongly connected to clustering. Segmentation often refers to the division of data into groups using simple techniques; e.g., an image can be split into segments based only on pixel intensity and color, or people can be divided into groups based on their income. Nonetheless, some work in graph partitioning and in image and segmentation is related to cluster analysis.

7.1.2 Different Types of Clusterings

An entire collection of clusters is commonly referred to as a **clustering**, and in this section, we distinguish various types of clusterings: hierarchical (nested) versus partitional (unnested), exclusive versus overlapping versus fuzzy, and complete versus partial.

Hierarchical versus Partitional

The most commonly discussed distinction among different types of clusterings is whether the set of clusters is nested or unnested, or in more traditional terminology, hierarchical or partitional. A **partitional clustering** is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Taken individually, each collection of clusters in [Figures 7.1 \(b–d\)](#) is a partitional clustering.

If we permit clusters to have subclusters, then we obtain a **hierarchical clustering**, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects. Often, but not always, the leaves of the tree are singleton clusters of

individual data objects. If we allow clusters to be nested, then one interpretation of [Figure 7.1\(a\)](#), each of which, in turn, has three subclusters ([Figure 7.1\(b\)](#)), when taken in that order, also form a hierarchical (nested) clustering with, respectively, 1, 2, 4, and 6 clusters on each level. Finally, note that a hierarchical clustering can be viewed as a sequence of partitional clusterings and a partitional clustering can be obtained by taking any member of that sequence; i.e., by cutting the hierarchical tree at a particular level.

Exclusive versus Overlapping versus Fuzzy

The clusterings shown in [Figure 7.1](#) are all **exclusive**, as they assign each object to a single cluster. There are many situations in which a point could reasonably be placed in more than one cluster, and these situations are better addressed by non-exclusive clustering. In the most general sense, an **overlapping or non-exclusive clustering** is used to reflect the fact that an object can *simultaneously* belong to more than one group (class). For instance, a person at a university can be both an enrolled student and an employee of the university. A non-exclusive clustering is also often used when, for example, an object is “between” two or more clusters and could reasonably be assigned to any of these clusters. Imagine a point halfway between two of the clusters of [Figure 7.1](#). Rather than make a somewhat arbitrary assignment of the object to a single cluster, it is placed in all of the “equally good” clusters.

In a **fuzzy clustering** ([Section 8.2.1](#)), every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn’t belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. (Mathematically, a fuzzy set is one in which an object belongs to every set with a weight that is between 0 and 1. In fuzzy clustering, we often impose the

additional constraint that the sum of the weights for each object must equal 1.) Similarly, probabilistic clustering techniques ([Section 8.2.2](#)) compute the probability with which each point belongs to each cluster, and these probabilities must also sum to 1. Because the membership weights or probabilities for any object sum to 1, a fuzzy or probabilistic clustering does not address true multiclass situations, such as the case of a student employee, where an object belongs to multiple classes. Instead, these approaches are most appropriate for avoiding the arbitrariness of assigning an object to only one cluster when it is close to several. In practice, a fuzzy or probabilistic clustering is often converted to an exclusive clustering by assigning each object to the cluster in which its membership weight or probability is highest.

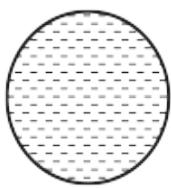
Complete versus Partial

A **complete clustering** assigns every object to a cluster, whereas a **partial clustering** does not. The motivation for a partial clustering is that some objects in a data set may not belong to well-defined groups. Many times objects in the data set represent noise, outliers, or “uninteresting background.” For example, some newspaper stories share a common theme, such as global warming, while other stories are more generic or one-of-a-kind. Thus, to find the important topics in last month’s stories, we often want to search only for clusters of documents that are tightly related by a common theme. In other cases, a complete clustering of the objects is desired. For example, an application that uses clustering to organize documents for browsing needs to guarantee that all documents can be browsed.

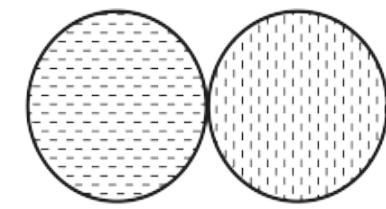
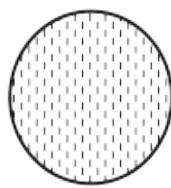
a

7.1.3 Different Types of Clusters

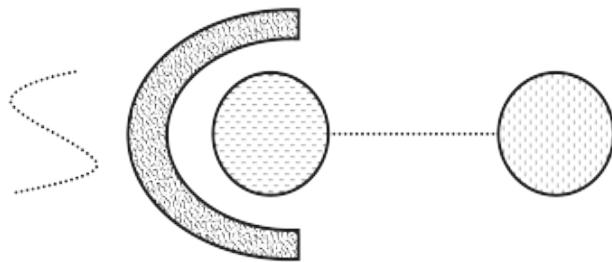
Clustering aims to find useful groups of objects (clusters), where usefulness is defined by the goals of the data analysis. Not surprisingly, several different notions of a cluster prove useful in practice. In order to visually illustrate the differences among these types of clusters, we use two-dimensional points, as shown in [Figure 7.2](#), as our data objects. We stress, however, that the types of clusters described here are equally valid for other kinds of data.



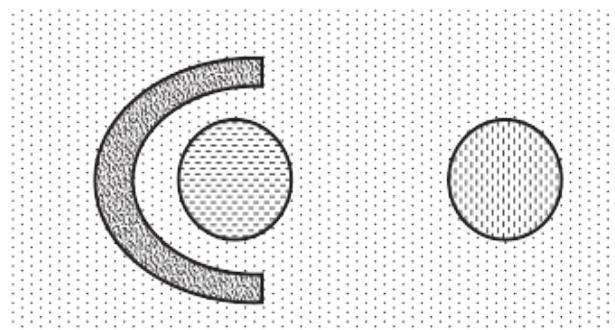
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



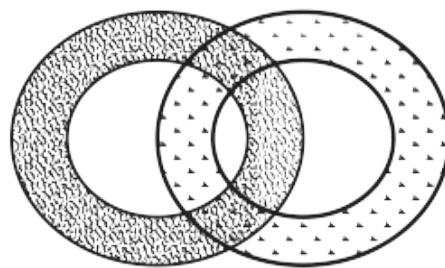
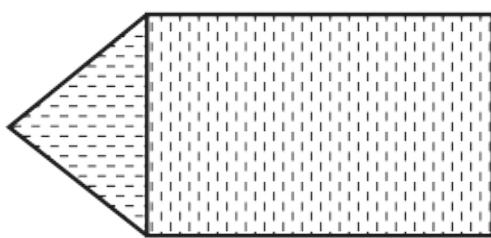
(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.



(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.



(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.



(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

Figure 7.2.

Different types of clusters as illustrated by sets of two-dimensional points.

Well-Separated

A cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster.

Sometimes a threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another. This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other. [Figure 7.2\(a\)](#) gives an example of well-separated clusters that consists of two groups of points in a two-dimensional space. The distance between any two points in different groups is larger than the distance between any two points within a group. Well-separated clusters do not need to be globular, but can have any shape.

Prototype-Based

A cluster is a set of objects in which each object is closer (more similar) to the prototype that defines the cluster than to the prototype of any other cluster.

For data with continuous attributes, the prototype of a cluster is often a centroid, i.e., the average (mean) of all the points in the cluster. When a centroid is not meaningful, such as when the data has categorical attributes, the prototype is often a medoid, i.e., the most representative point of a cluster. For many types of data, the prototype can be regarded as the most central point, and in such instances, we commonly refer to prototype-based clusters as **center-based clusters**. Not surprisingly, such clusters tend to be globular.

[Figure 7.2\(b\)](#) shows an example of center-based clusters.

Graph-Based

If the data is represented as a graph, where the nodes are objects and the links represent connections among objects (see [Section 2.1.2](#)), then a cluster can be defined as a **connected component**; i.e., a group of objects

that are connected to one another, but that have no connection to objects outside the group. An important example of graph-based clusters is a **contiguity-based cluster**, where two objects are connected only if they are within a specified distance of each other. This implies that each object in a contiguity-based cluster is closer to some other object in the cluster than to any point in a different cluster. [Figure 7.2\(c\)](#) shows an example of such clusters for two-dimensional points. This definition of a cluster is useful when clusters are irregular or intertwined. However, this approach can have trouble when noise is present since, as illustrated by the two spherical clusters of [Figure 7.2\(c\)](#), a small bridge of points can merge two distinct clusters.

Other types of graph-based clusters are also possible. One such approach ([Section 7.3.2](#)) defines a cluster as a **clique**; i.e., a set of nodes in a graph that are completely connected to each other. Specifically, if we add connections between objects in the order of their distance from one another, a cluster is formed when a set of objects forms a clique. Like prototype-based clusters, such clusters tend to be globular.

Density-Based

A cluster is a dense region of objects that is surrounded by a region of low density. [Figure 7.2\(d\)](#) shows some density-based clusters for data created by adding noise to the data of [Figure 7.2\(c\)](#). The two circular clusters are not merged, as in [Figure 7.2\(c\)](#), because the bridge between them fades into the noise. Likewise, the curve that is present in [Figure 7.2\(c\)](#) also fades into the noise and does not form a cluster in [Figure 7.2\(d\)](#). A density-based definition of a cluster is often employed when the clusters are irregular or intertwined, and when noise and outliers are present. By contrast, a contiguity-based definition of a cluster would not work well for the data of [Figure 7.2\(d\)](#) because the noise would tend to form bridges between clusters.

Shared-Property (Conceptual Clusters)

More generally, we can define a cluster as a set of objects that share some property. This definition encompasses all the previous definitions of a cluster; e.g., objects in a center-based cluster share the property that they are all closest to the same centroid or medoid. However, the shared-property approach also includes new types of clusters. Consider the clusters shown in [Figure 7.2\(e\)](#). A triangular area (cluster) is adjacent to a rectangular one, and there are two intertwined circles (clusters). In both cases, a clustering algorithm would need a very specific concept of a cluster to successfully detect these clusters. The process of finding such clusters is called conceptual clustering. However, too sophisticated a notion of a cluster would take us into the area of pattern recognition, and thus, we only consider simpler types of clusters in this book.

Road Map

In this chapter, we use the following three simple, but important techniques to introduce many of the concepts involved in cluster analysis.

- **K-means.** This is a prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids.
- **Agglomerative Hierarchical Clustering.** This clustering approach refers to a collection of closely related clustering techniques that produce a hierarchical clustering by starting with each point as a singleton cluster and then repeatedly merging the two closest clusters until a single, all-encompassing cluster remains. Some of these techniques have a natural interpretation in terms of graph-based clustering, while others have an interpretation in terms of a prototype-based approach.

- **DBSCAN.** This is a density-based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm. Points in low-density regions are classified as noise and omitted; thus, DBSCAN does not produce a complete clustering.

7.2 K-means

Prototype-based clustering techniques create a one-level partitioning of the data objects. There are a number of such techniques, but two of the most prominent are K-means and K-medoid. K-means defines a prototype in terms of a centroid, which is usually the mean of a group of points, and is typically applied to objects in a continuous n -dimensional space. K-medoid defines a prototype in terms of a medoid, which is the most representative point for a group of points, and can be applied to a wide range of data since it requires only a proximity measure for a pair of objects. While a centroid almost never corresponds to an actual data point, a medoid, by its definition, must be an actual data point. In this section, we will focus solely on K-means, which is one of the oldest and most widely-used clustering algorithms.

7.2.1 The Basic K-means Algorithm

The K-means clustering technique is simple, and we begin with a description of the basic algorithm. We first choose K initial centroids, where K is a user-specified parameter, namely, the number of clusters desired. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

K-means is formally described by [Algorithm 7.1](#). The operation of K-means is illustrated in [Figure 7.3](#), which shows how, starting from three

centroids, the final clusters are found in four assignment-update steps. In these and other figures displaying K-means clustering, each subfigure shows (1) the centroids at the start of the iteration and (2) the assignment of the points to those centroids. The centroids are indicated by the “+” symbol; all points belonging to the same cluster have the same marker shape.

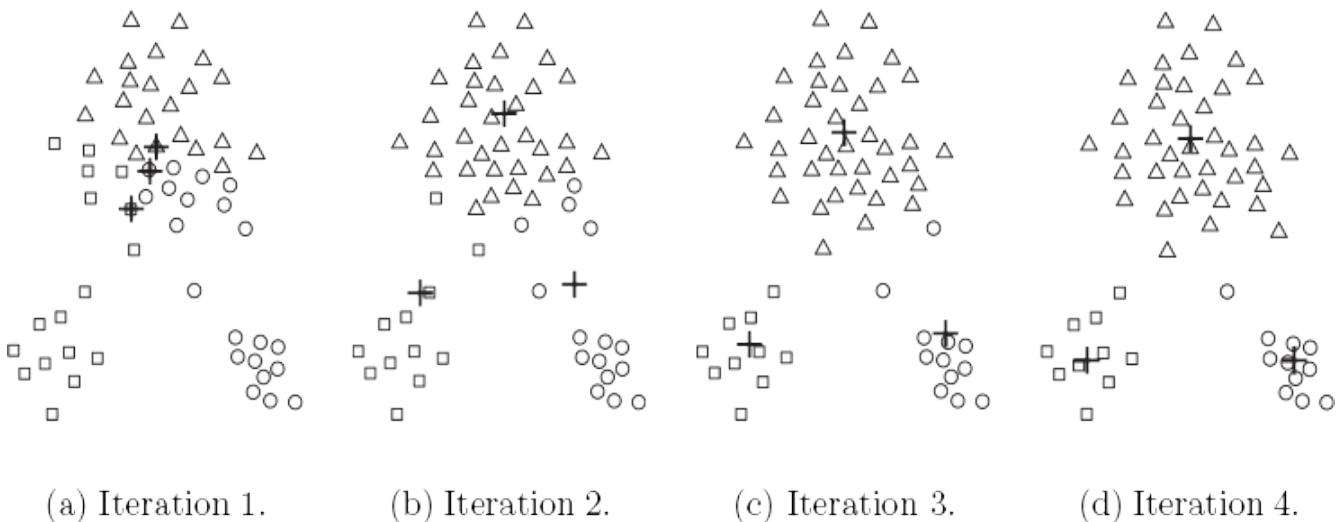


Figure 7.3.

Using the K-means algorithm to find three clusters in sample data.

Algorithm 7.1 Basic K-means algorithm.

- 1: Select K points as initial centroids.
- 2: **repeat**
- 3: Form K clusters by assigning each point to its closest centroid.
- 4: Recompute the centroid of each cluster.
- 5: **until** Centroids do not change.

In the first step, shown in [Figure 7.3\(a\)](#), points are assigned to the initial centroids, which are all in the larger group of points. For this example, we use the mean as the centroid. After points are assigned to a centroid, the centroid

is then updated. Again, the figure for each step shows the centroid at the beginning of the step and the assignment of points to those centroids. In the second step, points are assigned to the updated centroids, and the centroids are updated again. In steps 2, 3, and 4, which are shown in [Figures 7.3 \(b\)](#), [\(c\)](#), and [\(d\)](#), respectively, two of the centroids move to the two small groups of points at the bottom of the figures. When the K-means algorithm terminates in [Figure 7.3\(d\)](#), because no more changes occur, the centroids have identified the natural groupings of points.

For a number of combinations of proximity functions and types of centroids, K-means always converges to a solution; i.e., K-means reaches a state in which no points are shifting from one cluster to another, and hence, the centroids don't change. Because most of the convergence occurs in the early steps, however, the condition on line 5 of [Algorithm 7.1](#) is often replaced by a weaker condition, e.g., repeat until only 1% of the points change clusters.

We consider each of the steps in the basic K-means algorithm in more detail and then provide an analysis of the algorithm's space and time complexity.

Assigning Points to the Closest Centroid

To assign a point to the closest centroid, we need a proximity measure that quantifies the notion of "closest" for the specific data under consideration. Euclidean (L2) distance is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents. However, several types of proximity measures can be appropriate for a given type of data. For example, Manhattan (L1) distance can be used for Euclidean data, while the Jaccard measure is often employed for documents.

Usually, the similarity measures used for K-means are relatively simple since the algorithm repeatedly calculates the similarity of each point to each

centroid. In some cases, however, such as when the data is in low-dimensional Euclidean space, it is possible to avoid computing many of the similarities, thus significantly speeding up the K-means algorithm. Bisecting K-means (described in [Section 7.2.3](#)) is another approach that speeds up K-means by reducing the number of similarities computed.

Centroids and Objective Functions

Step 4 of the K-means algorithm was stated rather generally as “recompute the centroid of each cluster,” since the centroid can vary, depending on the proximity measure for the data and the goal of the clustering. The goal of the clustering is typically expressed by an objective function that depends on the proximities of the points to one another or to the cluster centroids; e.g., minimize the squared distance of each point to its closest centroid. We illustrate this with two examples. However, the key point is this: after we have specified a proximity measure and an objective function, the centroid that we should choose can often be determined mathematically. We provide mathematical details in [Section 7.2.6](#), and provide a non-mathematical discussion of this observation here.

Data in Euclidean Space

Consider data whose proximity measure is Euclidean distance. For our objective function, which measures the quality of a clustering, we use the **sum of the squared error (SSE)**, which is also known as scatter. In other words, we calculate the error of each data point, i.e., its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors. Given two different sets of clusters that are produced by two different runs of K-means, we prefer the one with the smallest squared error since this means that the prototypes (centroids) of this clustering are a better representation of

the points in their cluster. Using the notation in [Table 7.1](#), the SSE is formally defined as follows:

Table 7.1. Table of notation.

Symbol	Description
x	An object.
C_i	The i th cluster.
c_i	The centroid of cluster C_i .
c	The centroid of all points.
m_i	The number of objects in the i th cluster.
m	The number of objects in the data set.
K	The number of clusters.

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist(c_i, x)^2 \quad (7.1)$$

where $dist$ is the standard Euclidean (L2) distance between two objects in Euclidean space.

Given these assumptions, it can be shown (see [Section 7.2.6](#)) that the centroid that minimizes the SSE of the cluster is the mean. Using the notation in [Table 7.1](#), the centroid (mean) of the i th cluster is defined by [Equation 7.2](#).

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x \quad (7.2)$$

To illustrate, the centroid of a cluster containing the three two-dimensional points, (1,1), (2,3), and (6,2), is $((1+2+6)/3, ((1+3+2)/3)=(3,2)$.

Steps 3 and 4 of the K-means algorithm directly attempt to minimize the SSE (or more generally, the objective function). Step 3 forms clusters by assigning points to their nearest centroid, which minimizes the SSE for the given set of centroids. Step 4 recomputes the centroids so as to further minimize the SSE. However, the actions of K-means in Steps 3 and 4 are guaranteed to only find a local minimum with respect to the SSE because they are based on optimizing the SSE for specific choices of the centroids and clusters, rather than for all possible choices. We will later see an example in which this leads to a suboptimal clustering.

Document Data

To illustrate that K-means is not restricted to data in Euclidean space, we consider document data and the cosine similarity measure. Here we assume that the document data is represented as a document-term matrix as described on page 37. Our objective is to maximize the similarity of the documents in a cluster to the cluster centroid; this quantity is known as the **cohesion** of the cluster. For this objective it can be shown that the cluster centroid is, as for Euclidean data, the mean. The analogous quantity to the total SSE is the total cohesion, which is given by [Equation 7.3](#).

$$\text{Total Cohesion} = \sum_{i=1}^K \sum_{x \in C_i} \text{cosine}(x, c_i) \quad (7.3)$$

The General Case

There are a number of choices for the proximity function, centroid, and objective function that can be used in the basic K-means algorithm and that are guaranteed to converge. [Table 7.2](#) shows some possible choices,

including the two that we have just discussed. Notice that for Manhattan (L1) distance and the objective of minimizing the sum of the distances, the appropriate centroid is the median of the points in a cluster.

Table 7.2. K-means: Common choices for proximity, centroids, and objective functions.

Proximity Function	Centroid	Objective Function
Manhattan (L1)	median	Minimize sum of the L1 distance of an object to its cluster centroid
Squared Euclidean (L22)	mean	Minimize sum of the squared L2 distance of an object to its cluster centroid
cosine	mean	Maximize sum of the cosine similarity of an object to its cluster centroid
Bregman divergence	mean	Minimize sum of the Bregman divergence of an object to its cluster centroid

The last entry in the table, Bregman divergence ([Section 2.4.8](#)), is actually a class of proximity measures that includes the squared Euclidean distance, L22, the Mahalanobis distance, and cosine similarity. The importance of Bregman divergence functions is that any such function can be used as the basis of a K-means style clustering algorithm with the mean as the centroid. Specifically, if we use a Bregman divergence as our proximity function, then the resulting clustering algorithm has the usual properties of K-means with respect to convergence, local minima, etc. Furthermore, the properties of such a clustering algorithm can be developed for all possible Bregman divergences. For example, K-means algorithms that use cosine similarity or squared Euclidean distance are particular instances of a general clustering algorithm based on Bregman divergences.

For the rest of our K-means discussion, we use two-dimensional data since it is easy to explain K-means and its properties for this type of data. But, as suggested by the last few paragraphs, K-means is a general clustering algorithm and can be used with a wide variety of data types, such as documents and time series.

Choosing Initial Centroids

When random initialization of centroids is used, different runs of K-means typically produce different total SSEs. We illustrate this with the set of two-dimensional points shown in [Figure 7.3](#), which has three natural clusters of points. [Figure 7.4\(a\)](#) shows a clustering solution that is the global minimum of the SSE for three clusters, while [Figure 7.4\(b\)](#) shows a suboptimal clustering that is only a local minimum.

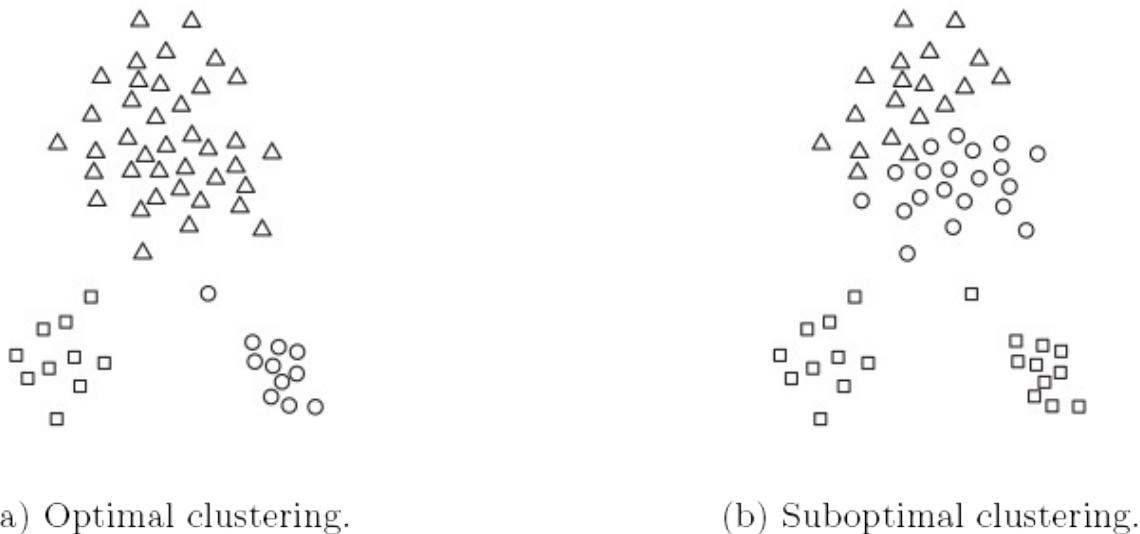


Figure 7.4.
Three optimal and non-optimal clusters.

Choosing the proper initial centroids is the key step of the basic K-means procedure. A common approach is to choose the initial centroids randomly, but the resulting clusters are often poor.

Example 7.1 (Poor Initial Centroids).

Randomly selected initial centroids can be poor. We provide an example of this using the same data set used in [Figures 7.3](#) and [7.4](#). [Figures 7.3](#) and [7.5](#) show the clusters that result from two particular choices of initial centroids. (For both figures, the positions of the cluster centroids in the various iterations are indicated by crosses.) In [Figure 7.3](#), even though all the initial centroids are from one natural cluster, the minimum SSE clustering is still found. In [Figure 7.5](#), however, even though the initial centroids seem to be better distributed, we obtain a suboptimal clustering, with higher squared error.

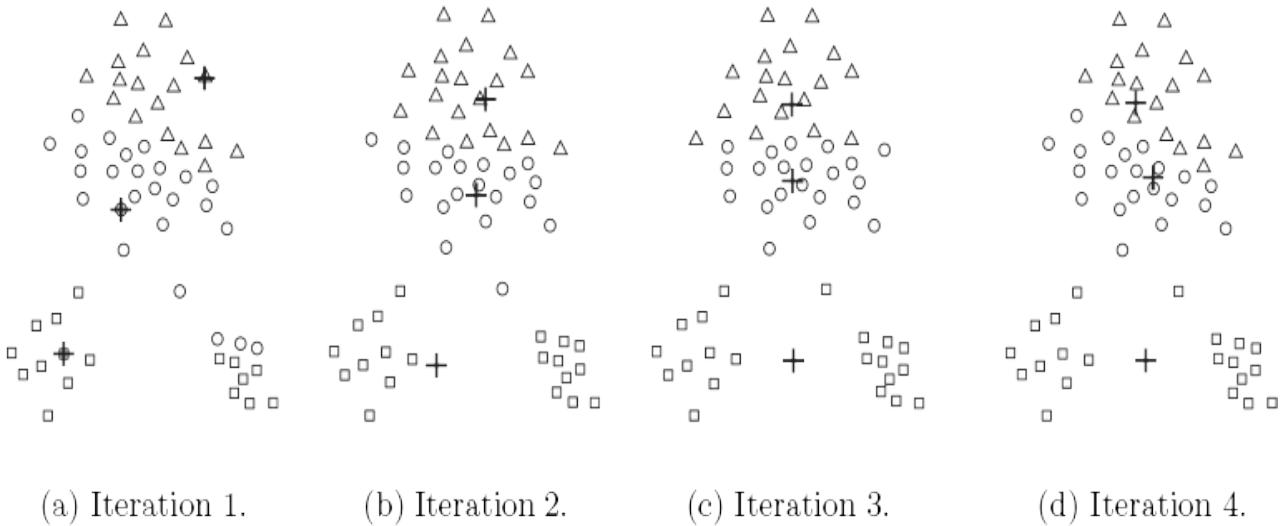


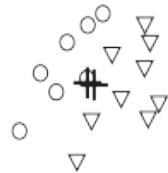
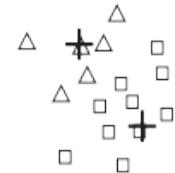
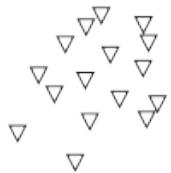
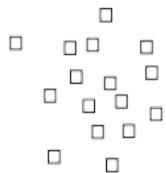
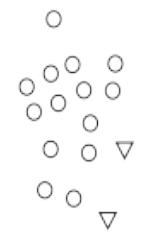
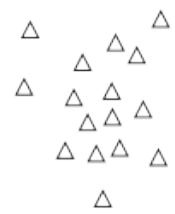
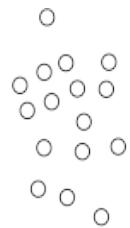
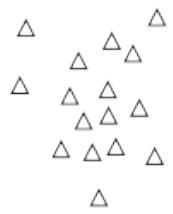
Figure 7.5.
Poor starting centroids for K-means.

Example 7.2 (Limits of Random Initialization).

One technique that is commonly used to address the problem of choosing initial centroids is to perform multiple runs, each with a different set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE. While simple, this strategy might not work very well, depending on the data set and the number of clusters sought. We

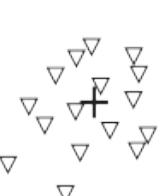
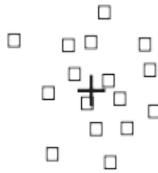
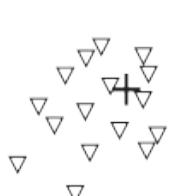
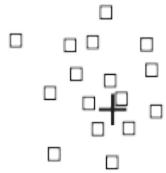
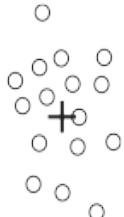
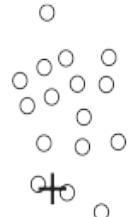
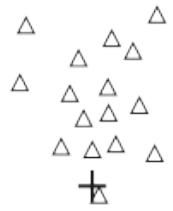
demonstrate this using the sample data set shown in [Figure 7.6\(a\)](#). The data consists of two pairs of clusters, where the clusters in each (top-bottom) pair are closer to each other than to the clusters in the other pair.

[Figure 7.6 \(b-d\)](#) shows that if we start with two initial centroids per pair of clusters, then even when both centroids are in a single cluster, the centroids will redistribute themselves so that the “true” clusters are found. However, [Figure 7.7](#) shows that if a pair of clusters has only one initial centroid and the other pair has three, then two of the true clusters will be combined and one true cluster will be split.



(a) Initial points.

(b) Iteration 1.

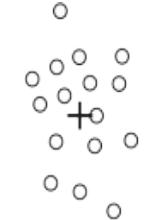
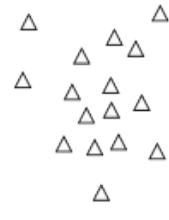
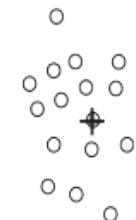
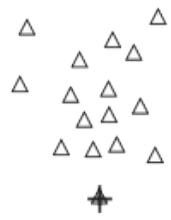


(c) Iteration 2.

(d) Iteration 3.

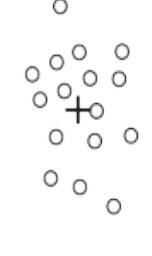
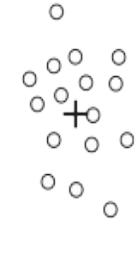
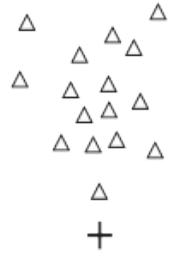
Figure 7.6.

Two pairs of clusters with a pair of initial centroids within each pair of clusters.



(a) Iteration 1.

(b) Iteration 2.



(c) Iteration 3.

(d) Iteration 4.

Figure 7.7.

Two pairs of clusters with more or fewer than two initial centroids within a pair of clusters.

Note that an optimal clustering will be obtained as long as two initial centroids fall anywhere in a pair of clusters, since the centroids will redistribute themselves, one to each cluster. Unfortunately, as the number of clusters becomes larger, it is increasingly likely that at least one pair of clusters will have only one initial centroid—see [Exercise 4](#) on page **603**.

In this case, because the pairs of clusters are farther apart than clusters within a pair, the K-means algorithm will not redistribute the centroids between pairs of clusters, and thus, only a local minimum will be achieved.

Because of the problems with using randomly selected initial centroids, which even repeated runs might not overcome, other techniques are often employed for initialization. One effective approach is to take a sample of points and cluster them using a hierarchical clustering technique. K clusters are extracted from the hierarchical clustering, and the centroids of those clusters are used as the initial centroids. This approach often works well, but is practical only if (1) the sample is relatively small, e.g., a few hundred to a few thousand (hierarchical clustering is expensive), and (2) K is relatively small compared to the sample size.

The following procedure is another approach to selecting initial centroids. Select the first point at random or take the centroid of all points. Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected. In this way, we obtain a set of initial centroids that is guaranteed to be not only randomly selected but also well separated. Unfortunately, such an approach can select outliers, rather than points in dense regions (clusters), which can lead to a situation where many clusters have just one point—an outlier—which reduces the number of centroids for forming clusters for the majority of points. Also, it is expensive to compute the farthest point from the current set of initial centroids. To overcome these problems, this approach is often applied to a sample of the points. Because outliers are rare, they tend not to show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small. Also, the computation involved in finding the initial centroids is greatly reduced because the sample size is typically much smaller than the number of points.

K-means++

More recently, a new approach for initializing K-means, called K-means++, has been developed. This procedure is guaranteed to find a K-means clustering solution that is optimal to within a factor of $O \log(k)$, which in practice translates into noticeably better clustering results in terms of lower SSE. This technique is similar to the idea just discussed of picking the first centroid at random and then picking each remaining centroid as the point as far from the remaining centroids as possible. Specifically, K-means++ picks centroids incrementally until k centroids have been picked. At every such step, each point has a probability of being picked as the new centroid that is proportional to the square of its distance to its closest centroid. It might seem that this approach might tend to choose outliers for centroids, but because outliers are rare, by definition, this is unlikely.

The details of K-means++ initialization are given by [Algorithm 7.2](#). □ The rest of the algorithm is the same as ordinary K-means.

Algorithm 7.2 K-means++ initialization algorithm.

- 1: For the first centroid, pick one of the points at random.
- 2: **for** $i=1$ to *number of trials* **do**
- 3: Compute the distance, $d(x)$, of each point to its closest centroid.
- 4: Assign each point a probability proportional to each point's $d(x)^2$.
- 5: Pick new centroid from the remaining points using the weighted probabilities.
- 6: **end for**

Later, we will discuss two other approaches that are also useful for producing better-quality (lower SSE) clusterings: using a variant of K-means that is less susceptible to initialization problems (bisecting K-means) and using postprocessing to “fix up” the set of clusters produced. K-means++ could be combined with either approach.

Time and Space Complexity

The space requirements for K-means are modest because only the data points and centroids are stored. Specifically, the storage required is $O((m+K)n)$, where m is the number of points and n is the number of attributes. The time requirements for K-means are also modest—basically linear in the number of data points. In particular, the time required is $O(I \times K \times m \times n)$, where I is the number of iterations required for convergence. As mentioned, I is often small and can usually be safely bounded, as most changes typically occur in the first few iterations. Therefore, K-means is linear in m , the number of points, and is efficient as well as simple provided that K , the number of clusters, is significantly less than m .

7.2.2 K-means: Additional Issues

Handling Empty Clusters

One of the problems with the basic K-means algorithm is that empty clusters can be obtained if no points are allocated to a cluster during the assignment step. If this happens, then a strategy is needed to choose a replacement centroid, since otherwise, the squared error will be larger than necessary. One approach is to choose the point that is farthest away from any current centroid. If nothing else, this eliminates the point that currently contributes

most to the total squared error. (A K-means++ approach could be used as well.) Another approach is to choose the replacement centroid at random from the cluster that has the highest SSE. This will typically split the cluster and reduce the overall SSE of the clustering. If there are several empty clusters, then this process can be repeated several times.

Outliers

When the squared error criterion is used, outliers can unduly influence the clusters that are found. In particular, when outliers are present, the resulting cluster centroids (prototypes) are typically not as representative as they otherwise would be and thus, the SSE will be higher. Because of this, it is often useful to discover outliers and eliminate them beforehand. It is important, however, to appreciate that there are certain clustering applications for which outliers should not be eliminated. When clustering is used for data compression, every point must be clustered, and in some cases, such as financial analysis, apparent outliers, e.g., unusually profitable customers, can be the most interesting points.

An obvious issue is how to identify outliers. A number of techniques for identifying outliers will be discussed in [Chapter 9](#). If we use approaches that remove outliers before clustering, we avoid clustering points that will not cluster well. Alternatively, outliers can also be identified in a postprocessing step. For instance, we can keep track of the SSE contributed by each point, and eliminate those points with unusually high contributions, especially over multiple runs. Also, we often want to eliminate small clusters because they frequently represent groups of outliers.

Reducing the SSE with Postprocessing

An obvious way to reduce the SSE is to find more clusters, i.e., to use a larger K . However, in many cases, we would like to improve the SSE, but don't want to increase the number of clusters. This is often possible because K-means typically converges to a local minimum. Various techniques are used to "fix up" the resulting clusters in order to produce a clustering that has lower SSE. The strategy is to focus on individual clusters since the total SSE is simply the sum of the SSE contributed by each cluster. (We will use the terms *total SSE* and *cluster SSE*, respectively, to avoid any potential confusion.) We can change the total SSE by performing various operations on the clusters, such as splitting or merging clusters. One commonly used approach is to employ alternate cluster splitting and merging phases. During a splitting phase, clusters are divided, while during a merging phase, clusters are combined. In this way, it is often possible to escape local SSE minima and still produce a clustering solution with the desired number of clusters. The following are some techniques used in the splitting and merging phases.

Two strategies that decrease the total SSE by increasing the number of clusters are the following:

Split a cluster: The cluster with the largest SSE is usually chosen, but we could also split the cluster with the largest standard deviation for one particular attribute.

Introduce a new cluster centroid: Often the point that is farthest from any cluster center is chosen. We can easily determine this if we keep track of the SSE contributed by each point. Another approach is to choose randomly from all points or from the points with the highest SSE with respect to their closest centroids.

Two strategies that decrease the number of clusters, while trying to minimize the increase in total SSE, are the following:

Disperse a cluster: This is accomplished by removing the centroid that corresponds to the cluster and reassigning the points to other clusters. Ideally, the cluster that is dispersed should be the one that increases the total SSE the least.

Merge two clusters: The clusters with the closest centroids are typically chosen, although another, perhaps better, approach is to merge the two clusters that result in the smallest increase in total SSE. These two merging strategies are the same ones that are used in the hierarchical clustering techniques known as the centroid method and Ward's method, respectively. Both methods are discussed in [Section 7.3](#).

Updating Centroids Incrementally

Instead of updating cluster centroids after all points have been assigned to a cluster, the centroids can be updated incrementally, after each assignment of a point to a cluster. Notice that this requires either zero or two updates to cluster centroids at each step, since a point either moves to a new cluster (two updates) or stays in its current cluster (zero updates). Using an incremental update strategy guarantees that empty clusters are not produced because all clusters start with a single point, and if a cluster ever has only one point, then that point will always be reassigned to the same cluster.

In addition, if incremental updating is used, the relative weight of the point being added can be adjusted; e.g., the weight of points is often decreased as the clustering proceeds. While this can result in better accuracy and faster convergence, it can be difficult to make a good choice for the relative weight, especially in a wide variety of situations. These update issues are similar to those involved in updating weights for artificial neural networks.

Yet another benefit of incremental updates has to do with using objectives other than “minimize SSE.” Suppose that we are given an arbitrary objective function to measure the goodness of a set of clusters. When we process an individual point, we can compute the value of the objective function for each possible cluster assignment, and then choose the one that optimizes the objective. Specific examples of alternative objective functions are given in [Section 7.5.2](#).

On the negative side, updating centroids incrementally introduces an order dependency. In other words, the clusters produced usually depend on the order in which the points are processed. Although this can be addressed by randomizing the order in which the points are processed, the basic K-means approach of updating the centroids after all points have been assigned to clusters has no order dependency. Also, incremental updates are slightly more expensive. However, K-means converges rather quickly, and therefore, the number of points switching clusters quickly becomes relatively small.

7.2.3 Bisecting K-means

The bisecting K-means algorithm is a straightforward extension of the basic K-means algorithm that is based on a simple idea: to obtain K clusters, split the set of all points into two clusters, select one of these clusters to split, and so on, until K clusters have been produced. The details of bisecting K-means are given by [Algorithm 7.3](#).

There are a number of different ways to choose which cluster to split. We can choose the largest cluster at each step, choose the one with the largest SSE, or use a criterion based on both size and SSE. Different choices result in different clusters.

Because we are using the K-means algorithm “locally,” i.e., to bisect individual clusters, the final set of clusters does not represent a clustering that is a local minimum with respect to the total SSE. Thus, we often refine the resulting clusters by using their cluster centroids as the initial centroids for the standard K-means algorithm.

Algorithm 7.3 Bisecting K-means algorithm.

- 1: Initialize the list of clusters to contain the cluster consisting of all points.
- 2: **repeat**
- 3: Remove a cluster from the list of clusters.
- 4: {Perform several “trial” bisections of the chosen cluster.}
- 5: **for** $i=1$ to *number of trials* **do**
- 6: Bisect the selected cluster using basic K-means.
- 7: **end for**
- 8: Select the two clusters from the bisection with the lowest total SSE.
- 9: Add these two clusters to the list of clusters.
- 10: **until** The list of clusters contains K clusters.

Example 7.3 (Bisecting K-means and Initialization).

To illustrate that bisecting K-means is less susceptible to initialization problems, we show, in [Figure 7.8](#), how bisecting K-means finds four clusters in the data set originally shown in [Figure 7.6\(a\)](#). In iteration 1, two pairs of clusters are found; in iteration 2, the rightmost pair of clusters

is split; and in iteration 3, the leftmost pair of clusters is split. Bisection K-means has less trouble with initialization because it performs several trial bisections and takes the one with the lowest SSE, and because there are only two centroids at each step.

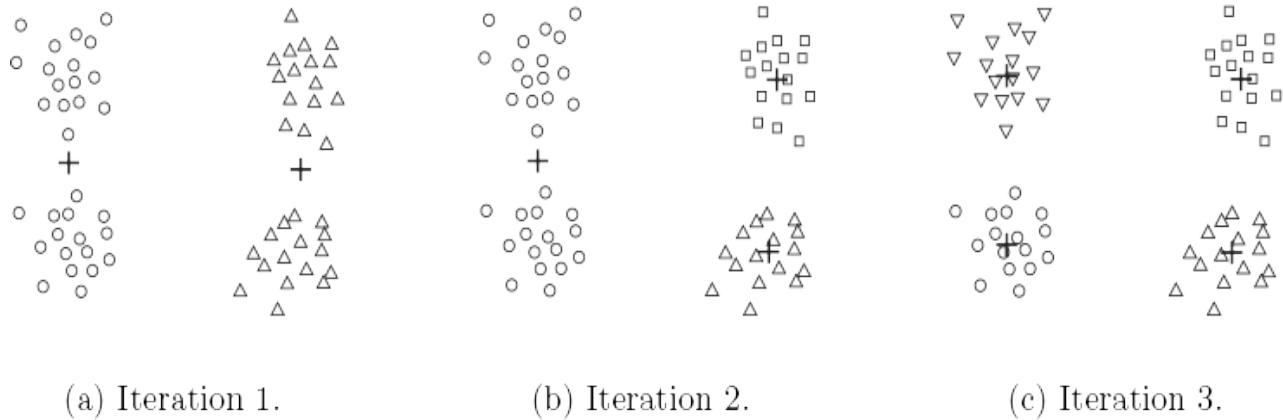


Figure 7.8.

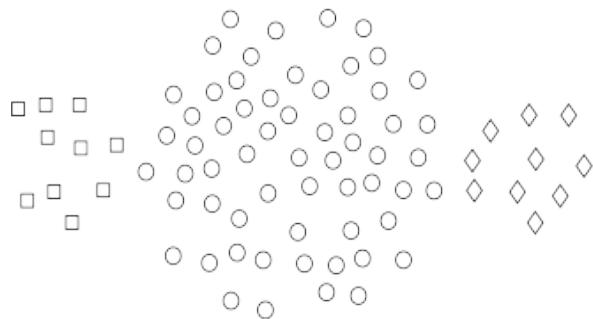
Bisection K-means on the four clusters example.

Finally, by recording the sequence of clusterings produced as K-means bisects clusters, we can also use bisection K-means to produce a hierarchical clustering.

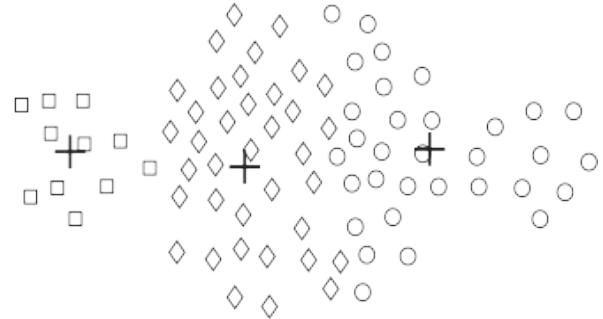
7.2.4 K-means and Different Types of Clusters

K-means and its variations have a number of limitations with respect to finding different types of clusters. In particular, K-means has difficulty detecting the “natural” clusters, when clusters have non-spherical shapes or widely different sizes or densities. This is illustrated by [Figures 7.9](#), [7.10](#), and [7.11](#). In [Figure 7.9](#), K-means cannot find the three natural clusters because one of the clusters is much larger than the other two, and hence, the larger cluster is

broken, while one of the smaller clusters is combined with a portion of the larger cluster. In [Figure 7.10](#), K-means fails to find the three natural clusters because the two smaller clusters are much denser than the larger cluster. Finally, in [Figure 7.11](#), K-means finds two clusters that mix portions of the two natural clusters because the shape of the natural clusters is not globular.



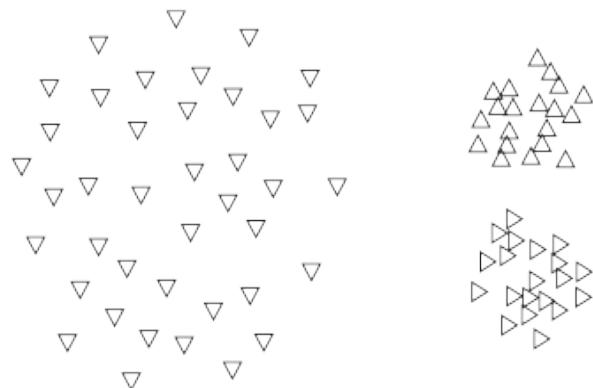
(a) Original points.



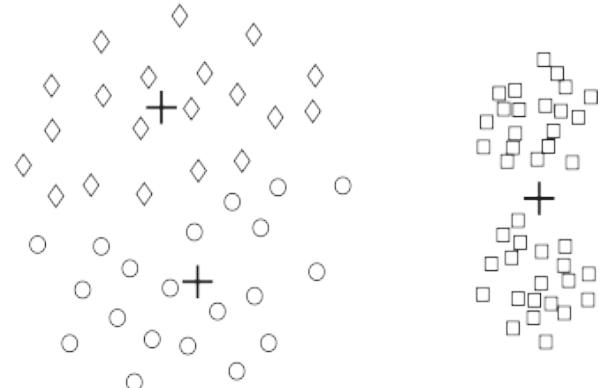
(b) Three K-means clusters.

Figure 7.9.

K-means with clusters of different size.



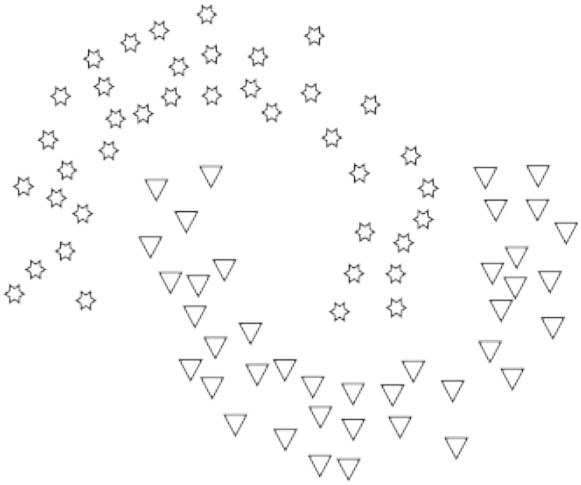
(a) Original points.



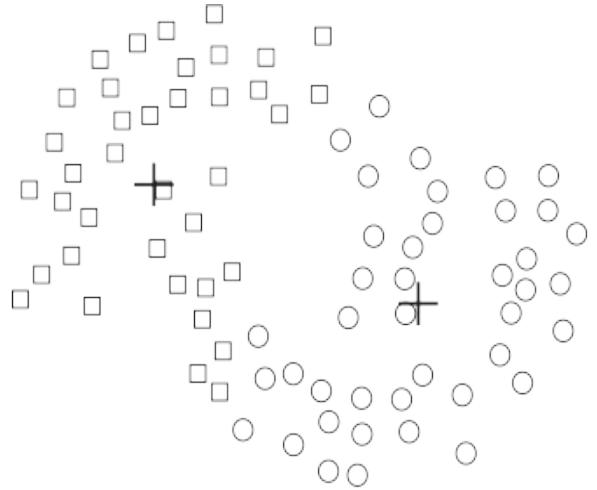
(b) Three K-means clusters.

Figure 7.10.

K-means with clusters of different density.



(a) Original points.

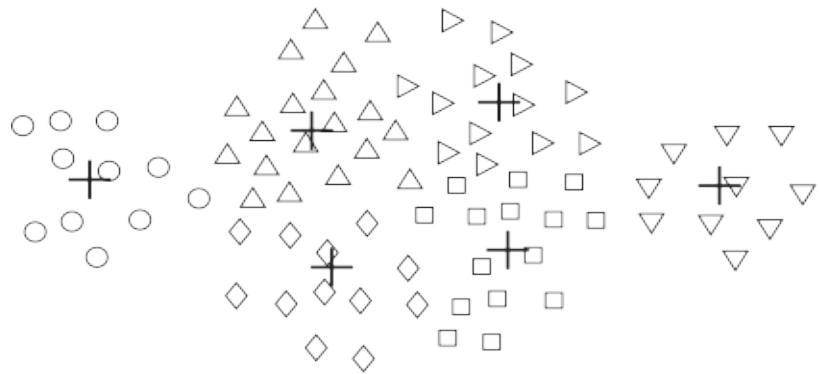


(b) Two K-means clusters.

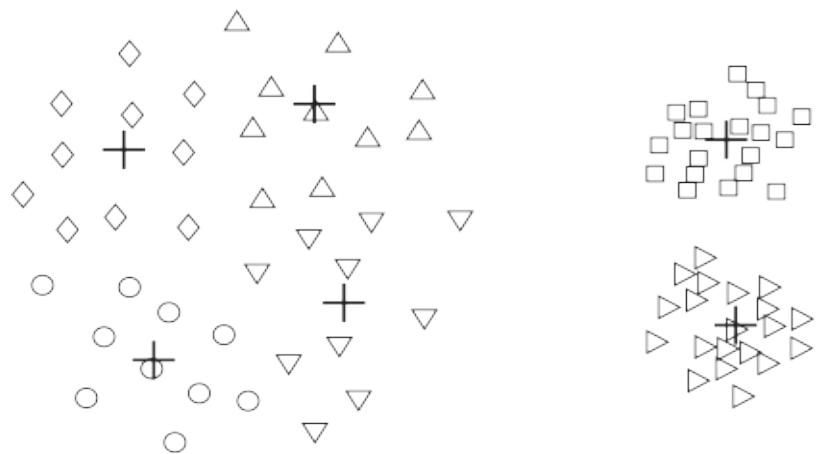
Figure 7.11.

K-means with non-globular clusters.

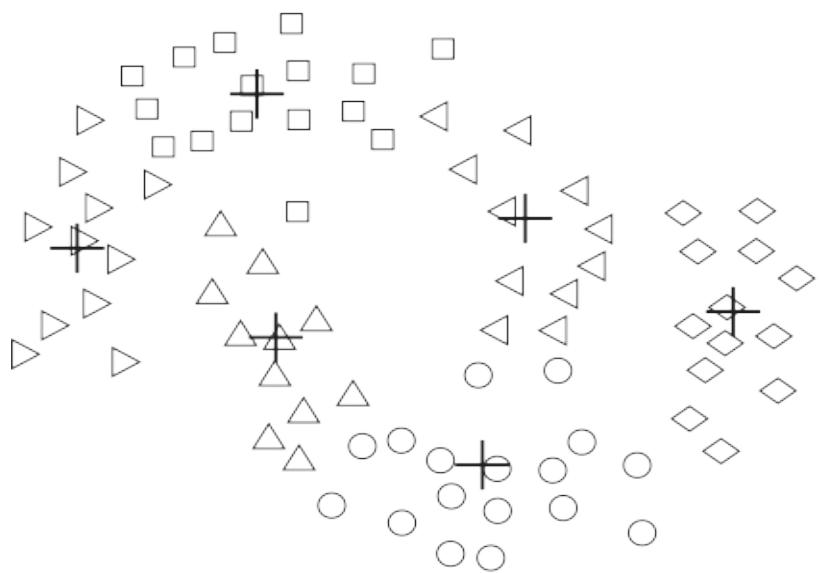
The difficulty in these three situations is that the K-means objective function is a mismatch for the kinds of clusters we are trying to find because it is minimized by globular clusters of equal size and density or by clusters that are well separated. However, these limitations can be overcome, in some sense, if the user is willing to accept a clustering that breaks the natural clusters into a number of subclusters. [Figure 7.12](#) shows what happens to the three previous data sets if we find six clusters instead of two or three. Each smaller cluster is pure in the sense that it contains only points from one of the natural clusters.



(a) Unequal sizes.



(b) Unequal densities.



(c) Non-spherical shapes.

Figure 7.12.

Using K-means to find clusters that are subclusters of the natural clusters.

7.2.5 Strengths and Weaknesses

K-means is simple and can be used for a wide variety of data types. It is also quite efficient, even though multiple runs are often performed. Some variants, including bisecting K-means, are even more efficient, and are less susceptible to initialization problems. K-means is not suitable for all types of data, however. It cannot handle non-globular clusters or clusters of different sizes and densities, although it can typically find pure subclusters if a large enough number of clusters is specified. K-means also has trouble clustering data that contains outliers. Outlier detection and removal can help significantly in such situations. Finally, K-means is restricted to data for which there is a notion of a center (centroid). A related technique, K-medoid clustering, does not have this restriction, but is more expensive.

7.2.6 K-means as an Optimization Problem

Here, we delve into the mathematics behind K-means. This section, which can be skipped without loss of continuity, requires knowledge of calculus through partial derivatives. Familiarity with optimization techniques, especially those based on gradient descent, can also be helpful.

As mentioned earlier, given an objective function such as “minimize SSE,” clustering can be treated as an optimization problem. One way to solve this problem—to find a global optimum—is to enumerate all possible ways of dividing the points into clusters and then choose the set of clusters that best satisfies the objective function, e.g., that minimizes the total SSE. Of course, this exhaustive strategy is computationally infeasible and as a result, a more practical approach is needed, even if such an approach finds solutions that are not guaranteed to be optimal. One technique, which is known as **gradient descent**, is based on picking an initial solution and then repeating the following two steps: compute the change to the solution that best optimizes the objective function and then update the solution.

We assume that the data is one-dimensional, i.e., $\text{dist}(x,y)=(x-y)^2$. This does not change anything essential, but greatly simplifies the notation.

Derivation of K-means as an Algorithm to Minimize the SSE

In this section, we show how the centroid for the K-means algorithm can be mathematically derived when the proximity function is Euclidean distance and the objective is to minimize the SSE. Specifically, we investigate how we can best update a cluster centroid so that the cluster SSE is minimized. In mathematical terms, we seek to minimize [Equation 7.1](#), which we repeat here, specialized for one-dimensional data.

$$\text{SSE} = \sum_{i=1}^K \sum_{x \in C_i} (c_i - x)^2 \quad (7.4)$$

Here, C_i is the i th cluster, x is a point in C_i , and c_i is the mean of the i th cluster. See [Table 7.1](#) for a complete list of notation.

We can solve for the k th centroid c_k , which minimizes [Equation 7.4](#), by differentiating the SSE, setting it equal to 0, and solving, as indicated below.

$$\partial \partial c_k \text{SSE} = \partial \partial c_k \sum_{i=1}^K \sum_{x \in C_i} (c_i - x)^2 = \sum_{i=1}^K \sum_{x \in C_i} \partial \partial c_k (c_i - x)^2 = \sum_{x \in C_k} 2 \times (c_k - x) = 0$$

$$\sum_{x \in C_k} 2 \times (c_k - x) = 0 \Rightarrow m_k c_k = \sum_{x \in C_k} x \Rightarrow c_k = \frac{1}{m_k} \sum_{x \in C_k} x$$

Thus, as previously indicated, the best centroid for minimizing the SSE of a cluster is the mean of the points in the cluster.

Derivation of K-means for SAE

To demonstrate that the K-means algorithm can be applied to a variety of different objective functions, we consider how to partition the data into K clusters such that the sum of the Manhattan (L1) distances of points from the center of their clusters is minimized. We are seeking to minimize the sum of the L1 absolute errors (SAE) as given by the following equation, where dist_{L1} is the L1 distance. Again, for notational simplicity, we use one-dimensional data, i.e., $\text{dist}_{\text{L1}} = |c_i - x|$.

$$\text{SAE} = \sum_{i=1}^K \sum_{x \in C_i} \text{dist}_{\text{L1}}(c_i, x) \tag{7.5}$$

We can solve for the k th centroid c_k , which minimizes [Equation 7.5](#), by differentiating the SAE, setting it equal to 0, and solving.

$$\partial \partial c_k \text{SAE} = \partial \partial c_k \sum_{i=1}^K \sum_{x \in C_i} |c_i - x| = \sum_{i=1}^K \sum_{x \in C_i} \partial \partial c_k |c_i - x| = \sum_{x \in C_k} \partial \partial c_k |c_k - x| = 0$$

$$\sum_{x \in C_k} \partial \partial c_k |c_k - x| = 0 \Rightarrow \sum_{x \in C_k} \text{sign}(x - c_k) = 0$$

If we solve for c_k , we find that $c_k = \text{median}\{x \in C_k\}$, the median of the points in the cluster. The median of a group of points is straightforward to compute and less susceptible to distortion by outliers.

7.3 Agglomerative Hierarchical Clustering

Hierarchical clustering techniques are a second important category of clustering methods. As with K-means, these approaches are relatively old compared to many clustering algorithms, but they still enjoy widespread use. There are two basic approaches for generating a hierarchical clustering:

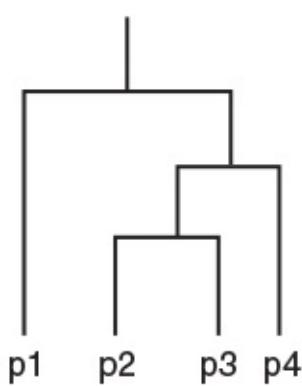
Agglomerative: Start with the points as individual clusters and, at each step, merge the closest pair of clusters. This requires defining a notion of cluster proximity.

Divisive: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide which cluster to split at each step and how to do the splitting.

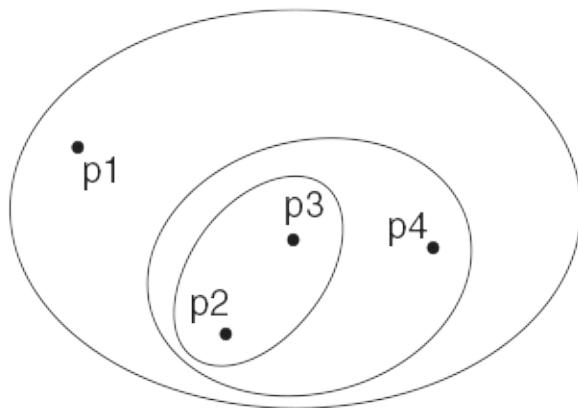
Agglomerative hierarchical clustering techniques are by far the most common, and, in this section, we will focus exclusively on these methods. A divisive hierarchical clustering technique is described in [Section 8.4.2](#).

A hierarchical clustering is often displayed graphically using a tree-like diagram called a **dendrogram**, which displays both the cluster-subcluster relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view). For sets of two-dimensional points, such as those that we will use as examples, a hierarchical clustering can also be graphically represented using a nested cluster diagram. [Figure 7.13](#) shows an example of these two types of figures for a set of four two-dimensional points.

These points were clustered using the single-link technique that is described in [Section 7.3.2](#).



(a) Dendrogram.



(b) Nested cluster diagram.

Figure 7.13.

A hierarchical clustering of four points shown as a dendrogram and as nested clusters.

7.3.1 Basic Agglomerative Hierarchical Clustering Algorithm

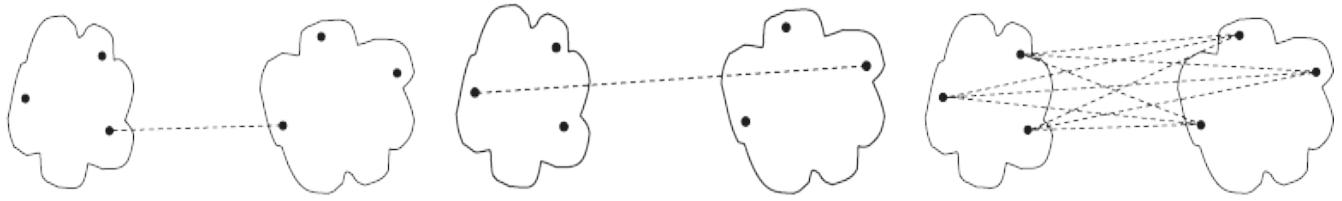
Many agglomerative hierarchical clustering techniques are variations on a single approach: starting with individual points as clusters, successively merge the two closest clusters until only one cluster remains. This approach is expressed more formally in [Algorithm 7.4](#).

Algorithm 7.4 Basic agglomerative hierarchical clustering algorithm.

- 1: Compute the proximity matrix, if necessary.
- 2: **repeat**
- 3: Merge the closest two clusters.
- 4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
- 5: **until** Only one cluster remains.

Defining Proximity between Clusters

The key operation of [Algorithm 7.4](#) is the computation of the proximity between two clusters, and it is the definition of cluster proximity that differentiates the various agglomerative hierarchical techniques that we will discuss. Cluster proximity is typically defined with a particular type of cluster in mind—see [Section 7.1.3](#). For example, many agglomerative hierarchical clustering techniques, such as MIN, MAX, and Group Average, come from a graph-based view of clusters. **MIN** defines cluster proximity as the proximity between the closest two points that are in different clusters, or using graph terms, the shortest edge between two nodes in different subsets of nodes. This yields contiguity-based clusters as shown in [Figure 7.2\(c\)](#). Alternatively, **MAX** takes the proximity between the farthest two points in different clusters to be the cluster proximity, or using graph terms, the longest edge between two nodes in different subsets of nodes. (If our proximities are distances, then the names, MIN and MAX, are short and suggestive. For similarities, however, where higher values indicate closer points, the names seem reversed. For that reason, we usually prefer to use the alternative names, **single link** and **complete link**, respectively.) Another graph-based approach, the **group average** technique, defines cluster proximity to be the average pairwise proximities (average length of edges) of all pairs of points from different clusters. [Figure 7.14](#) illustrates these three approaches.



(a) MIN (single link).

(b) MAX (complete link).

(c) Group average.

Figure 7.14.

Graph-based definitions of cluster proximity.

If, instead, we take a prototype-based view, in which each cluster is represented by a centroid, different definitions of cluster proximity are more natural. When using centroids, the cluster proximity is commonly defined as the proximity between cluster centroids. An alternative technique, **Ward's** method, also assumes that a cluster is represented by its centroid, but it measures the proximity between two clusters in terms of the increase in the SSE that results from merging the two clusters. Like K-means, Ward's method attempts to minimize the sum of the squared distances of points from their cluster centroids.

Time and Space Complexity

The basic agglomerative hierarchical clustering algorithm just presented uses a proximity matrix. This requires the storage of $12 m^2$ proximities (assuming the proximity matrix is symmetric) where m is the number of data points. The space needed to keep track of the clusters is proportional to the number of clusters, which is $m-1$, excluding singleton clusters. Hence, the total space complexity is $O(m^2)$.

The analysis of the basic agglomerative hierarchical clustering algorithm is also straightforward with respect to computational complexity. $O(m^2)$ time is required to compute the proximity matrix. After that step, there are $m-1$

iterations involving steps 3 and 4 because there are m clusters at the start and two clusters are merged during each iteration. If performed as a linear search of the proximity matrix, then for the i th iteration, Step 3 requires $O((m-i+1)^2)$ time, which is proportional to the current number of clusters squared. Step 4 requires $O(m-i+1)$ time to update the proximity matrix after the merger of two clusters. (A cluster merger affects $O(m-i+1)$ proximities for the techniques that we consider.) Without modification, this would yield a time complexity of $O(m^3)$. If the distances from each cluster to all other clusters are stored as a sorted list (or heap), it is possible to reduce the cost of finding the two closest clusters to $O(m-i+1)$. However, because of the additional complexity of keeping data in a sorted list or heap, the overall time required for a hierarchical clustering based on [Algorithm 7.4](#) is $O(m^2 \log m)$.

The space and time complexity of hierarchical clustering severely limits the size of data sets that can be processed. We discuss scalability approaches for clustering algorithms, including hierarchical clustering techniques, in [Section 8.5](#). Note, however, that the bisecting K-means algorithm presented in [Section 7.2.3](#) is a scalable algorithm that produces a hierarchical clustering.

7.3.2 Specific Techniques

Sample Data

To illustrate the behavior of the various hierarchical clustering algorithms, we will use sample data that consists of six two-dimensional points, which are shown in [Figure 7.15](#). The x and y coordinates of the points and the Euclidean distances between them are shown in [Tables 7.3](#) and [7.4](#), respectively.

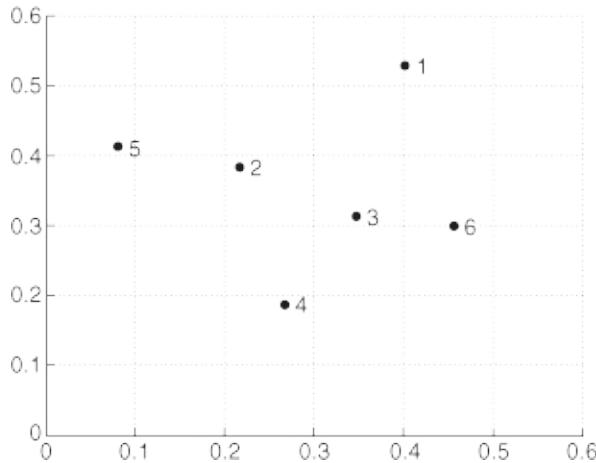


Figure 7.15.

Set of six two-dimensional points.

Table 7.3. xy-coordinates of six points.

Point	x Coordinate	y Coordinate
p1	0.4005	0.5306
p2	0.2148	0.3854
p3	0.3457	0.3156
p4	0.2652	0.1875
p5	0.0789	0.4139
p6	0.4548	0.3022

Table 7.4. Euclidean distance matrix for six points.

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25

p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Single Link or MIN

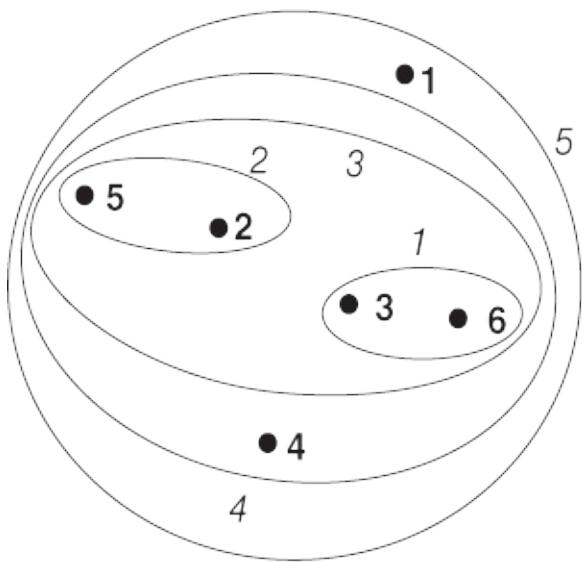
For the single link or MIN version of hierarchical clustering, the proximity of two clusters is defined as the minimum of the distance (maximum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then these single links combine the points into clusters. The single link technique is good at handling non-elliptical shapes, but is sensitive to noise and outliers.

Example 7.4 (Single Link).

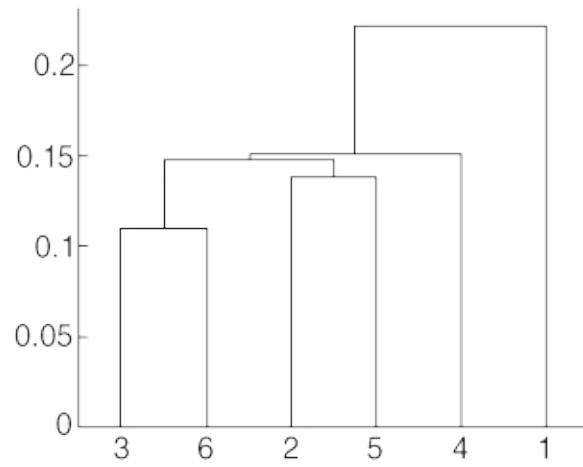
Figure 7.16 shows the result of applying the single link technique to our example data set of six points. **Figure 7.16(a)** shows the nested clusters as a sequence of nested ellipses, where the numbers associated with the ellipses indicate the order of the clustering. **Figure 7.16(b)** shows the same information, but as a dendrogram. The height at which two clusters are merged in the dendrogram reflects the distance of the two clusters. For instance, from **Table 7.4**, we see that the distance between points 3 and 6 is 0.11, and that is the height at which they are joined into one cluster in the dendrogram. As another example, the distance between clusters {3,6} and {2,5} is given by

$$\text{dist}(\{3,6\}, \{2,5\}) = \min(\text{dist}(3,2), \text{dist}(6,2), \text{dist}(3,5), \text{dist}(6,5)) =$$

$$\min(0.15, 0.25, 0.28, 0.39) = 0.15.$$



(a) Single link clustering.



(b) Single link dendrogram.

Figure 7.16.

Single link clustering of the six points shown in [Figure 7.15](#).

Complete Link or MAX or CLIQUE

For the complete link or MAX version of hierarchical clustering, the proximity of two clusters is defined as the maximum of the distance (minimum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then a group of points is not a cluster until all the points in it are completely linked, i.e., form a *clique*. Complete link is less susceptible to noise and outliers, but it can break large clusters and it favors globular shapes.

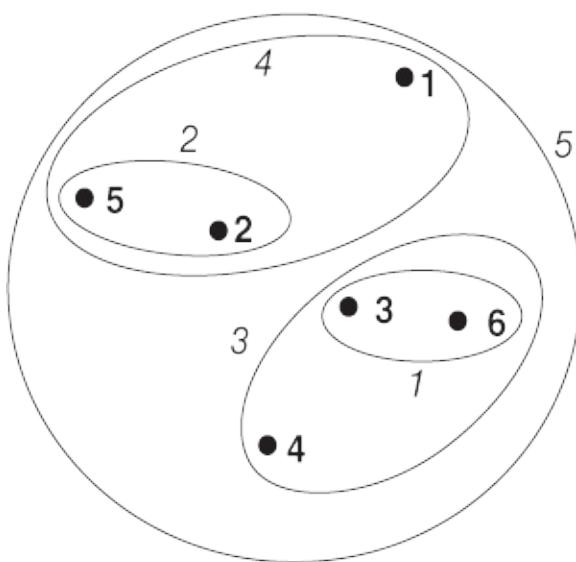
Example 7.5 (Complete Link).

Figure 7.17 shows the results of applying MAX to the sample data set of six points. As with single link, points 3 and 6 are merged first. However, $\{3,6\}$ is merged with $\{4\}$, instead of $\{2,5\}$ or $\{1\}$ because

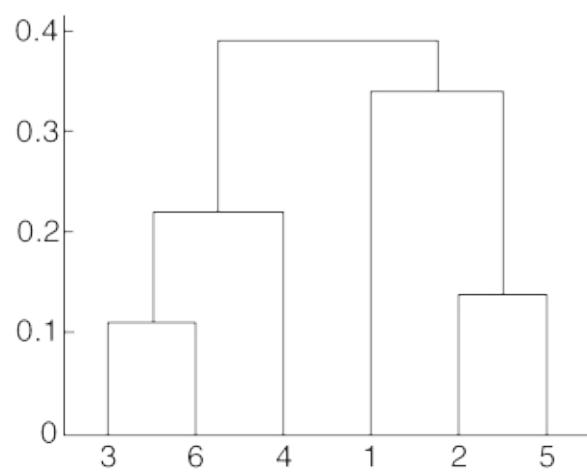
$$\text{dist}(\{3,6\}, \{4\}) = \max(\text{dist}(3,4), \text{dist}(6,4)) = \max(0.15, 0.22) = 0.22.$$

$$\text{dist}(\{3,6\}, \{2,5\}) = \max(\text{dist}(3,2), \text{dist}(6,2), \text{dist}(3,5), \text{dist}(6,5)) =$$

$$\max(0.15, 0.25, 0.28, 0.39) = 0.39. \text{ dist}(\{3,6\}, \{1\}) = \max(\text{dist}(3,1), \text{dist}(6,1)) = \max(0.22, 0.23) = 0.23.$$



(a) Complete link clustering.



(b) Complete link dendrogram.

Figure 7.17.

Complete link clustering of the six points shown in **Figure 7.15**.

Group Average

For the group average version of hierarchical clustering, the proximity of two clusters is defined as the average pairwise proximity among all pairs of points in the different clusters. This is an intermediate approach between the single and complete link approaches. Thus, for group average, the cluster proximity

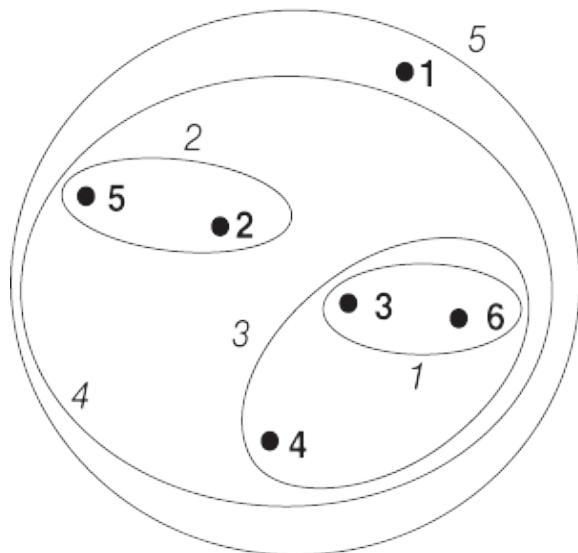
$\text{proximity}(\text{Ci}, \text{Cj})$ of clusters Ci and Cj , which are of size m_i and m_j , respectively, is expressed by the following equation:

$$\text{proximity}(\text{Ci}, \text{Cj}) = \sum_{y \in \text{Cj}} \sum_{x \in \text{Ci}} \text{proximity}(x, y) m_i \times m_j . \quad (7.6)$$

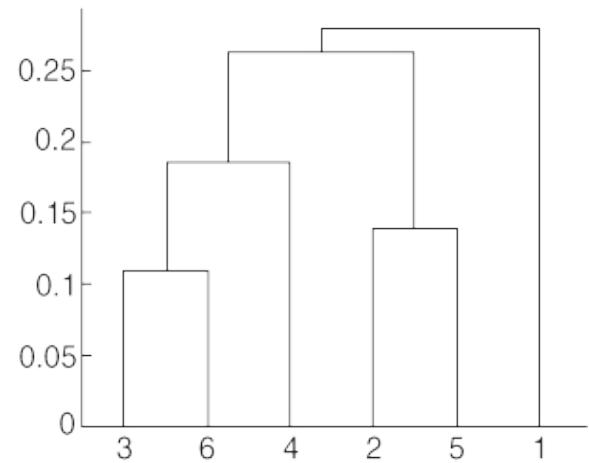
Example 7.6 (Group Average).

Figure 7.18 shows the results of applying the group average approach to the sample data set of six points. To illustrate how group average works, we calculate the distance between some clusters.

$$\begin{aligned} \text{dist}(\{3,6,4\}, \{1\}) &= (0.22+0.37+0.23)/(3 \times 1) = 0.28 \\ \text{dist}(\{2,5\}, \{1\}) &= (0.24+0.34)/(2 \times 1) = 0.29 \\ \text{dist}(\{3,6,4\}, \{2,5\}) &= (0.15+0.28+0.25+0.39+0.20+0.29)/(3 \times 2) = 0.26 \end{aligned}$$



(a) Group average clustering.



(b) Group average dendrogram.

Figure 7.18.

Group average clustering of the six points shown in **Figure 7.15**.

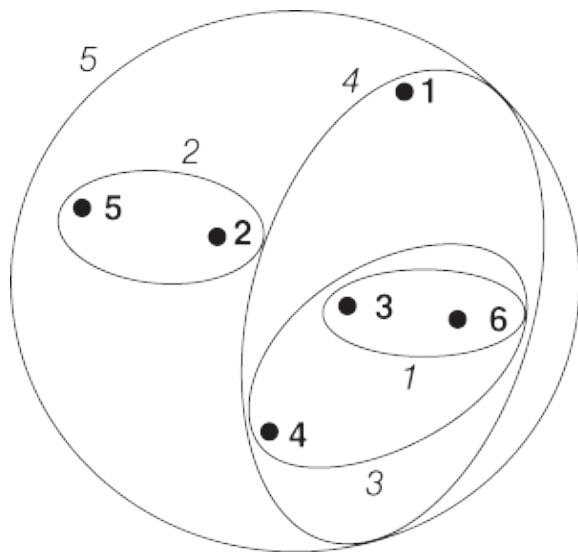
Because $\text{dist}(\{3,6,4\}, \{2,5\})$ is smaller than $\text{dist}(\{3,6,4\}, \{1\})$ and $\text{dist}(\{2,5\}, \{1\})$, clusters $\{3,6,4\}$ and $\{2,5\}$ are merged at the fourth stage.

Ward's Method and Centroid Methods

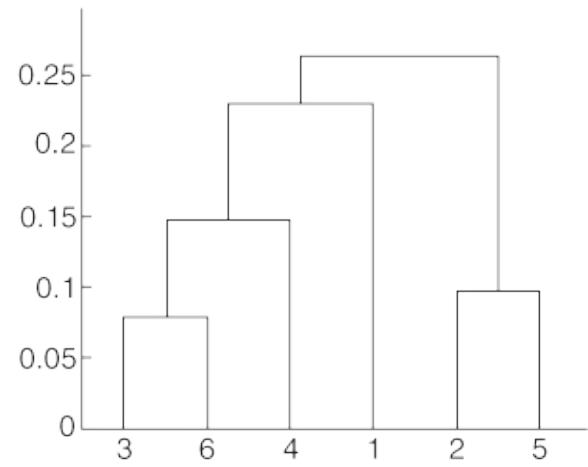
For Ward's method, the proximity between two clusters is defined as the increase in the squared error that results when two clusters are merged. Thus, this method uses the same objective function as K-means clustering. While it might seem that this feature makes Ward's method somewhat distinct from other hierarchical techniques, it can be shown mathematically that Ward's method is very similar to the group average method when the proximity between two points is taken to be the square of the distance between them.

Example 7.7 (Ward's Method).

Figure 7.19 shows the results of applying Ward's method to the sample data set of six points. The clustering that is produced is different from those produced by single link, complete link, and group average.



(a) Ward's clustering.



(b) Ward's dendrogram.

Figure 7.19.

Ward's clustering of the six points shown in [Figure 7.15](#).

Centroid methods calculate the proximity between two clusters by calculating the distance between the centroids of clusters. These techniques may seem similar to K-means, but as we have remarked, Ward's method is the correct hierarchical analog.

Centroid methods also have a characteristic—often considered bad—that is not possessed by the other hierarchical clustering techniques that we have discussed: the possibility of **inversions**. Specifically, two clusters that are merged can be more similar (less distant) than the pair of clusters that were merged in a previous step. For the other methods, the distance between merged clusters monotonically increases (or is, at worst, non-increasing) as we proceed from singleton clusters to one all-inclusive cluster.

7.3.3 The Lance-Williams Formula for Cluster Proximity

Any of the cluster proximities that we have discussed in this section can be viewed as a choice of different parameters (in the Lance-Williams formula shown below in [Equation 7.7](#)) for the proximity between clusters Q and R , where R is formed by merging clusters A and B . In this equation, $p(.,.)$ is a proximity function, while m_A , m_B , and m_Q are the number of points in clusters A , B , and Q , respectively. In other words, after we merge clusters A and B to form cluster R , the proximity of the new cluster, R , to an existing cluster, Q , is a linear function of the proximities of Q with respect to the

original clusters A and B . **Table 7.5** shows the values of these coefficients for the techniques that we have discussed.

$$p(R,Q) = \alpha_A p(A,Q) + \alpha_B p(B,Q) + \beta p(A,B) + \gamma |p(A,Q) - p(B,Q)| \quad (7.7)$$

Table 7.5. Table of Lance-Williams coefficients for common hierarchical clustering approaches.

Clustering Method	α_A	α_B	β	γ
Single Link	1/2	1/2	0	-1/2
Complete Link	1/2	1/2	0	1/2
Group Average	$m_A m_A + m_B$	$m_B m_A + m_B$	0	0
Centroid	$m_A m_A + m_B$	$m_B m_A + m_B$	$-m_A m_B (m_A + m_B)^2$	0
Ward's	$m_A + m_Q m_A + m_B + m_Q$	$m_B + m_Q m_A + m_B + m_Q$	$-m_Q m_A + m_B + m_Q$	0

Any hierarchical clustering technique that can be expressed using the Lance-Williams formula does not need to keep the original data points. Instead, the proximity matrix is updated as clustering occurs. While a general formula is appealing, especially for implementation, it is easier to understand the different hierarchical methods by looking directly at the definition of cluster proximity that each method uses.

7.3.4 Key Issues in Hierarchical

Clustering

Lack of a Global Objective Function

We previously mentioned that agglomerative hierarchical clustering cannot be viewed as globally optimizing an objective function. Instead, agglomerative hierarchical clustering techniques use various criteria to decide locally, at each step, which clusters should be merged (or split for divisive approaches). This approach yields clustering algorithms that avoid the difficulty of attempting to solve a hard combinatorial optimization problem. (It can be shown that the general clustering problem for an objective function such as “minimize SSE” is computationally infeasible.) Furthermore, such approaches do not have difficulties in choosing initial points. Nonetheless, the time complexity of $O(m^2 \log m)$ and the space complexity of $O(m^2)$ are prohibitive in many cases.

Ability to Handle Different Cluster Sizes

One aspect of agglomerative hierarchical clustering that we have not yet discussed is how to treat the relative sizes of the pairs of clusters that are merged. (This discussion applies only to cluster proximity schemes that involve sums, such as centroid, Ward’s, and group average.) There are two approaches: **weighted**, which treats all clusters equally, and **unweighted**, which takes the number of points in each cluster into account. Note that the terminology of weighted or unweighted refers to the data points, not the clusters. In other words, treating clusters of unequal size equally—the weighted approach—gives different weights to the points in different clusters, while taking the cluster size into account—the unweighted approach—gives points in different clusters the same weight.

We will illustrate this using the group average technique discussed in [Section 7.3.2](#), which is the unweighted version of the group average technique. In the clustering literature, the full name of this approach is the Unweighted Pair Group Method using Arithmetic averages (UPGMA). In [Table 7.5](#), which gives the formula for updating cluster similarity, the coefficients for UPGMA involve the size, m_A and m_B of each of the clusters, A and B that were merged: $\alpha_A = m_A/(m_A+m_B)$, $\alpha_B = m_B/(m_A+m_B)$, $\beta=0$, $\gamma=0$. For the weighted version of group average—known as WPGMA—the coefficients are constants that are independent of the cluster sizes: $\alpha_A = 1/2$, $\alpha_B = 1/2$, $\beta=0$, $\gamma=0$. In general, unweighted approaches are preferred unless there is reason to believe that individual points should have different weights; e.g., perhaps classes of objects have been unevenly sampled.

Merging Decisions Are Final

Agglomerative hierarchical clustering algorithms tend to make good local decisions about combining two clusters because they can use information about the pairwise similarity of all points. However, once a decision is made to merge two clusters, it cannot be undone at a later time. This approach prevents a local optimization criterion from becoming a global optimization criterion. For example, although the “minimize squared error” criterion from K-means is used in deciding which clusters to merge in Ward’s method, the clusters at each level do not represent local minima with respect to the total SSE. Indeed, the clusters are not even stable, in the sense that a point in one cluster can be closer to the centroid of some other cluster than it is to the centroid of its current cluster. Nonetheless, Ward’s method is often used as a robust method of initializing a K-means clustering, indicating that a local “minimize squared error” objective function does have a connection to a global “minimize squared error” objective function.

There are some techniques that attempt to overcome the limitation that merges are final. One approach attempts to fix up the hierarchical clustering by moving branches of the tree around so as to improve a global objective function. Another approach uses a partitional clustering technique such as K-means to create many small clusters, and then performs hierarchical clustering using these small clusters as the starting point.

7.3.5 Outliers

Outliers pose the most serious problems for Ward's method and centroid-based hierarchical clustering approaches because they increase SSE and distort centroids. For clustering approaches, such as single link, complete link, and group average, outliers are potentially less problematic. As hierarchical clustering proceeds for these algorithms, outliers or small groups of outliers tend to form singleton or small clusters that do not merge with any other clusters until much later in the merging process. By discarding singleton or small clusters that are not merging with other clusters, outliers can be removed.

7.3.6 Strengths and Weaknesses

The strengths and weaknesses of specific agglomerative hierarchical clustering algorithms were discussed above. More generally, such algorithms are typically used because the underlying application, e.g., creation of a taxonomy, requires a hierarchy. Also, some studies have suggested that these algorithms can produce better-quality clusters. However, agglomerative hierarchical clustering algorithms are expensive in terms of their computational and storage requirements. The fact that all merges are final can

also cause trouble for noisy, high-dimensional data, such as document data. In turn, these two problems can be addressed to some degree by first partially clustering the data using another technique, such as K-means.

7.4 DBSCAN

Density-based clustering locates regions of high density that are separated from one another by regions of low density. DBSCAN is a simple and effective density-based clustering algorithm that illustrates a number of important concepts that are important for any density-based clustering approach. In this section, we focus solely on DBSCAN after first considering the key notion of density. Other algorithms for finding density-based clusters are described in the next chapter.

7.4.1 Traditional Density: Center-Based Approach

Although there are not as many approaches for defining density as there are for defining similarity, there are several distinct methods. In this section we discuss the center-based approach on which DBSCAN is based. Other definitions of density will be presented in [Chapter 8](#).

In the center-based approach, density is estimated for a particular point in the data set by counting the number of points within a specified radius, Eps , of that point. This includes the point itself. This technique is graphically illustrated by [Figure 7.20](#). The number of points within a radius of Eps of point A is 7, including A itself.

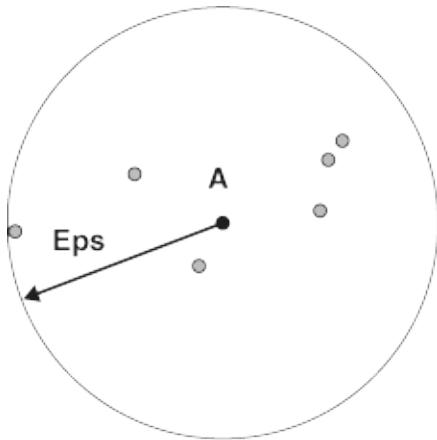


Figure 7.20.
Center-based density.

This method is simple to implement, but the density of any point will depend on the specified radius. For instance, if the radius is large enough, then all points will have a density of m , the number of points in the data set. Likewise, if the radius is too small, then all points will have a density of 1. An approach for deciding on the appropriate radius for low-dimensional data is given in the next section in the context of our discussion of DBSCAN.

Classification of Points According to Center-Based Density

The center-based approach to density allows us to classify a point as being (1) in the interior of a dense region (a core point), (2) on the edge of a dense region (a border point), or (3) in a sparsely occupied region (a noise or background point). [Figure 7.21](#) graphically illustrates the concepts of core, border, and noise points using a collection of two-dimensional points. The following text provides a more precise description.

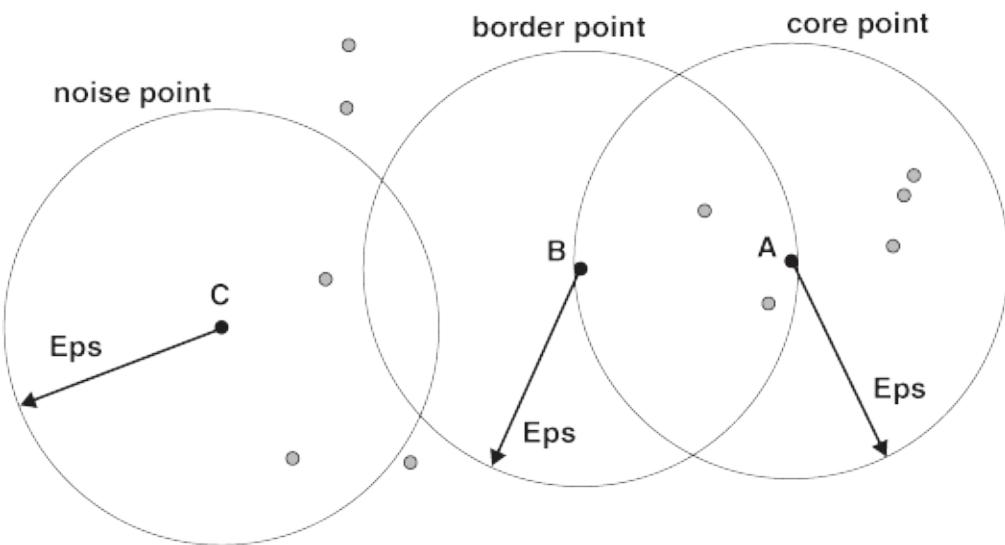


Figure 7.21.

Core, border, and noise points.

Core points: These points are in the interior of a density-based cluster. A point is a core point if there are at least MinPts within a distance of Eps , where MinPts and Eps are user-specified parameters. In [Figure 7.21](#), point A is a core point for the radius (Eps) if $\text{MinPts} \geq 7$.

Border points: A border point is not a core point, but falls within the neighborhood of a core point. In [Figure 7.21](#), point B is a border point. A border point can fall within the neighborhoods of several core points.

Noise points: A noise point is any point that is neither a core point nor a border point. In [Figure 7.21](#), point C is a noise point.

7.4.2 The DBSCAN Algorithm

Given the previous definitions of core points, border points, and noise points, the DBSCAN algorithm can be informally described as follows. Any two core points that are close enough—within a distance Eps of one another—are put

in the same cluster. Likewise, any border point that is close enough to a core point is put in the same cluster as the core point. (Ties need to be resolved if a border point is close to core points from different clusters.) Noise points are discarded. The formal details are given in [Algorithm 7.5](#). This algorithm uses the same concepts and finds the same clusters as the original DBSCAN, but is optimized for simplicity, not efficiency.

Algorithm 7.5 DBSCAN algorithm.

- 1: Label all points as core, border, or noise points.
- 2: Eliminate noise points.
- 3: Put an edge between all core points within a distance Eps of each other.
- 4: Make each group of connected core points into a separate cluster.
- 5: Assign each border point to one of the clusters of its associated core points.

Time and Space Complexity

The basic time complexity of the DBSCAN algorithm is $O(m \times \text{time to find points in the } Eps\text{-neighborhood})$, where m is the number of points. In the worst case, this complexity is $O(m^2)$. However, in low-dimensional spaces (especially 2D space), data structures such as kd-trees allow efficient retrieval of all points within a given distance of a specified point, and the time complexity can be as low as $O(m \log m)$ in the average case. The space requirement of DBSCAN, even for high-dimensional data, is $O(m)$ because it is necessary to keep only a small amount of data for each point, i.e., the cluster label and the identification of each point as a core, border, or noise point.

Selection of DBSCAN Parameters

There is, of course, the issue of how to determine the parameters *Eps* and *MinPts*. The basic approach is to look at the behavior of the distance from a point to its k th nearest neighbor, which we will call the k -dist. For points that belong to some cluster, the value of k -dist will be small if k is not larger than the cluster size. Note that there will be some variation, depending on the density of the cluster and the random distribution of points, but on average, the range of variation will not be huge if the cluster densities are not radically different. However, for points that are not in a cluster, such as noise points, the k -dist will be relatively large. Therefore, if we compute the k -dist for all the data points for some k , sort them in increasing order, and then plot the sorted values, we expect to see a sharp change at the value of k -dist that corresponds to a suitable value of *Eps*. If we select this distance as the *Eps* parameter and take the value of k as the *MinPts* parameter, then points for which k -dist is less than *Eps* will be labeled as core points, while other points will be labeled as noise or border points.

Figure 7.22 shows a sample data set, while the k -dist graph for the data is given in **Figure 7.23**. The value of *Eps* that is determined in this way depends on k , but does not change dramatically as k changes. If the value of k is too small, then even a small number of closely spaced points that are noise or outliers will be incorrectly labeled as clusters. If the value of k is too large, then small clusters (of size less than k) are likely to be labeled as noise. The original DBSCAN algorithm used a value of $k=4$, which appears to be a reasonable value for most two-dimensional data sets.

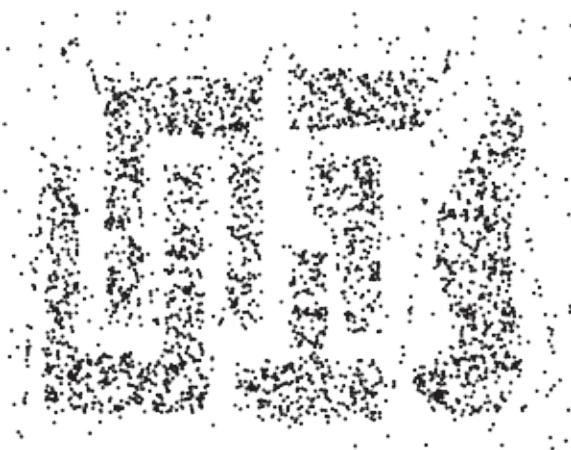


Figure 7.22.

Sample data.

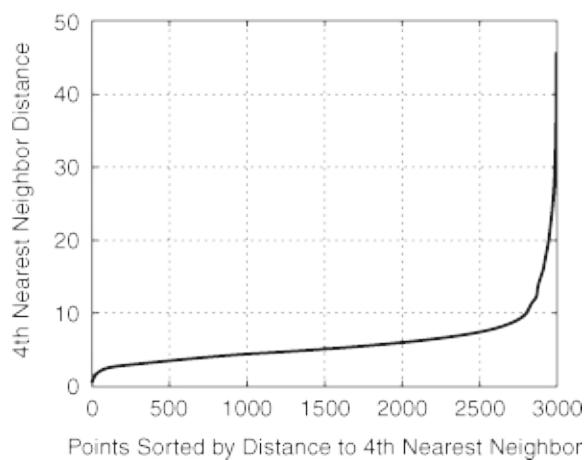


Figure 7.23.

K-dist plot for sample data.

Clusters of Varying Density

DBSCAN can have trouble with density if the density of clusters varies widely.

Consider [Figure 7.24](#), which shows four clusters embedded in noise. The density of the clusters and noise regions is indicated by their darkness. The noise around the pair of denser clusters, A and B, has the same density as clusters C and D. For a fixed *MinPts*, if the *Eps* threshold is chosen so that DBSCAN finds C and D as distinct clusters, with the points surrounding them

as noise, then A and B and the points surrounding them will become a single cluster. If the Eps threshold is set so that DBSCAN finds A and B as separate clusters, and the points surrounding them are marked as noise, then C , D , and the points surrounding them will also be marked as noise.

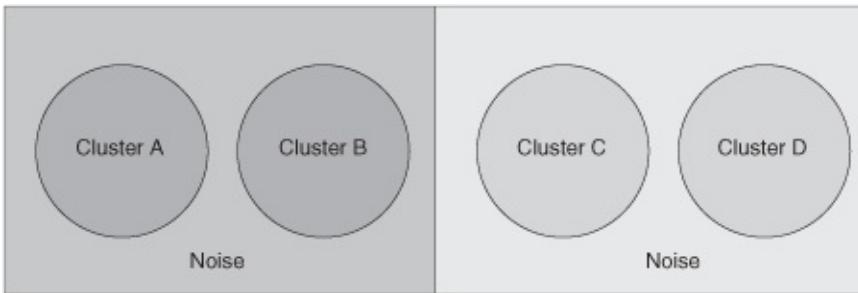
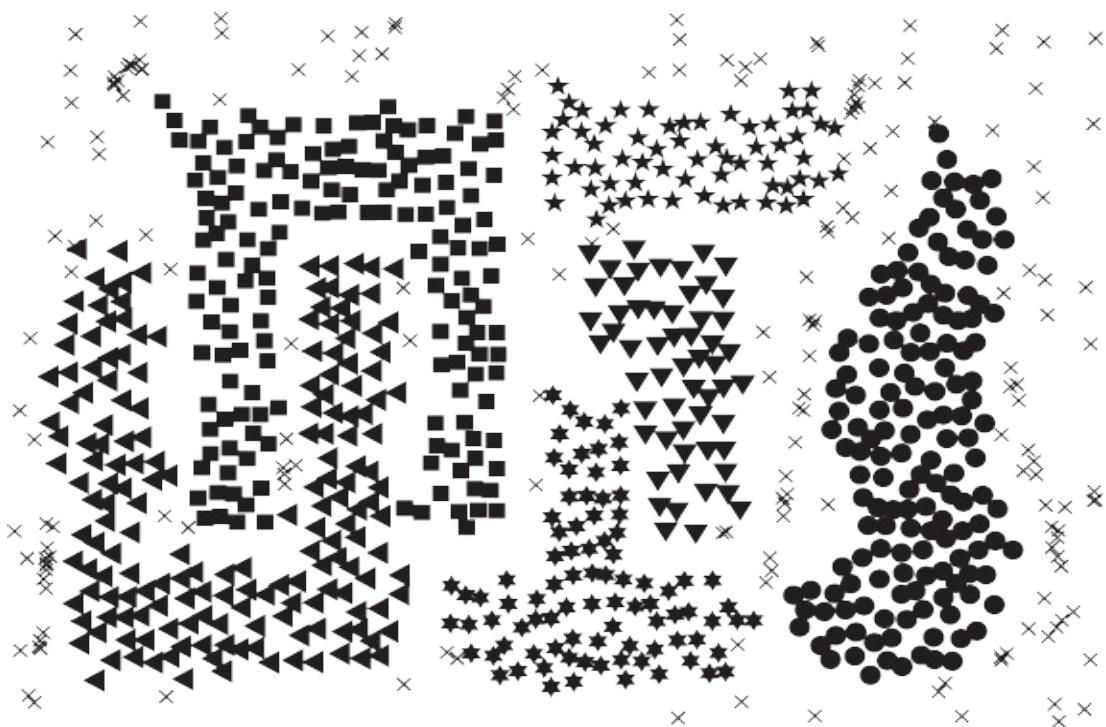


Figure 7.24.

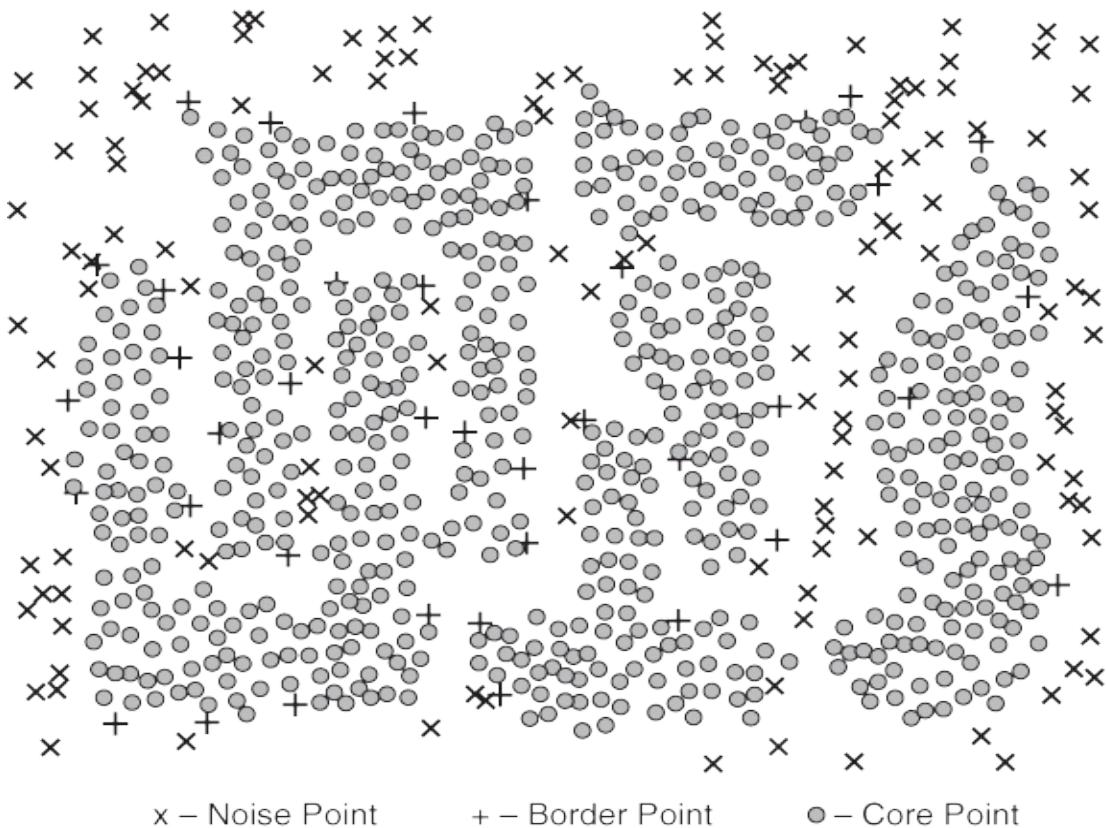
Four clusters embedded in noise.

An Example

To illustrate the use of DBSCAN, we show the clusters that it finds in the relatively complicated two-dimensional data set shown in [Figure 7.22](#). This data set consists of 3000 two-dimensional points. The Eps threshold for this data was found by plotting the sorted distances of the fourth nearest neighbor of each point ([Figure 7.23](#)) and identifying the value at which there is a sharp increase. We selected $Eps=10$, which corresponds to the knee of the curve. The clusters found by DBSCAN using these parameters, i.e., $\text{MinPts}=4$ and $Eps=10$, are shown in [Figure 7.25\(a\)](#). The core points, border points, and noise points are displayed in [Figure 7.25\(b\)](#).



(a) Clusters found by DBSCAN.



(b) Core, border, and noise points.

Figure 7.25.

DBSCAN clustering of 3000 two-dimensional points.

7.4.3 Strengths and Weaknesses

Because DBSCAN uses a density-based definition of a cluster, it is relatively resistant to noise and can handle clusters of arbitrary shapes and sizes. Thus, DBSCAN can find many clusters that could not be found using K-means, such as those in [Figure 7.22](#). As indicated previously, however, DBSCAN has trouble when the clusters have widely varying densities. It also has trouble with high-dimensional data because density is more difficult to define for such data. One possible approach to dealing with such issues is given in [Section 8.4.9](#). Finally, DBSCAN can be expensive when the computation of nearest neighbors requires computing all pairwise proximities, as is usually the case for high-dimensional data.

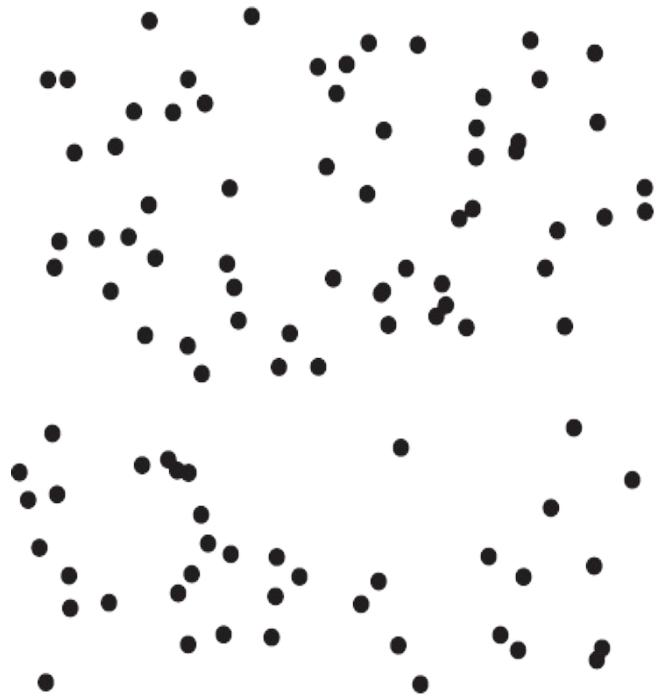
7.5 Cluster Evaluation

In supervised classification, the evaluation of the resulting classification model is an integral part of the process of developing a classification model, and there are well-accepted evaluation measures and procedures, e.g., accuracy and cross-validation, respectively. However, because of its very nature, cluster evaluation is not a well-developed or commonly used part of cluster analysis. Nonetheless, cluster evaluation, or **cluster validation** as it is more traditionally called, is important, and this section will review some of the most common and easily applied approaches.

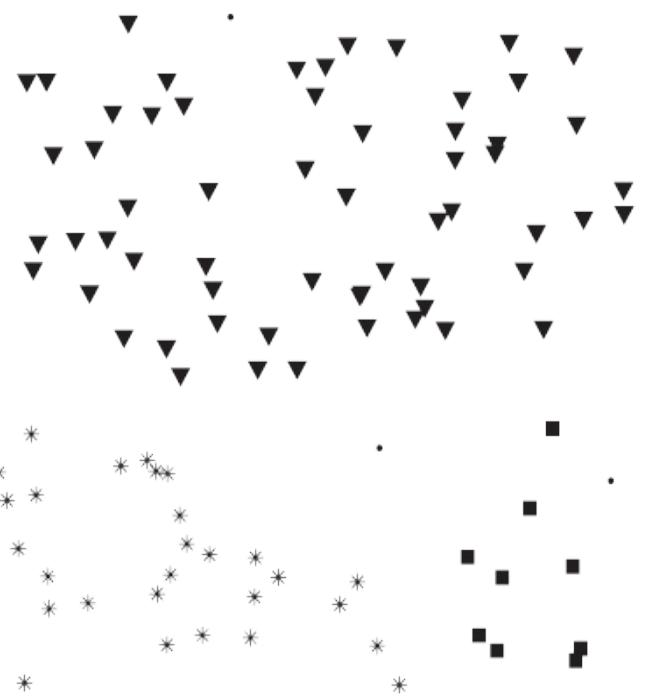
There might be some confusion as to why cluster evaluation is necessary. Many times, cluster analysis is conducted as a part of an exploratory data analysis. Hence, evaluation seems to be an unnecessarily complicated addition to what is supposed to be an informal process. Furthermore, because there are a number of different types of clusters—in some sense, each clustering algorithm defines its own type of cluster—it can seem that each situation might require a different evaluation measure. For instance, K-means clusters might be evaluated in terms of the SSE, but for density-based clusters, which need not be globular, SSE would not work well at all.

Nonetheless, cluster evaluation should be a part of any cluster analysis. A key motivation is that almost every clustering algorithm will find clusters in a data set, even if that data set has no natural cluster structure. For instance, consider [Figure 7.26](#), which shows the result of clustering 100 points that are randomly (uniformly) distributed on the unit square. The original points are shown in [Figure 7.26\(a\)](#), while the clusters found by DBSCAN, K-means, and complete link are shown in [Figures 7.26\(b\)](#), [7.26\(c\)](#), and [7.26\(d\)](#), respectively. Since DBSCAN found three clusters (after we set *Eps* by looking

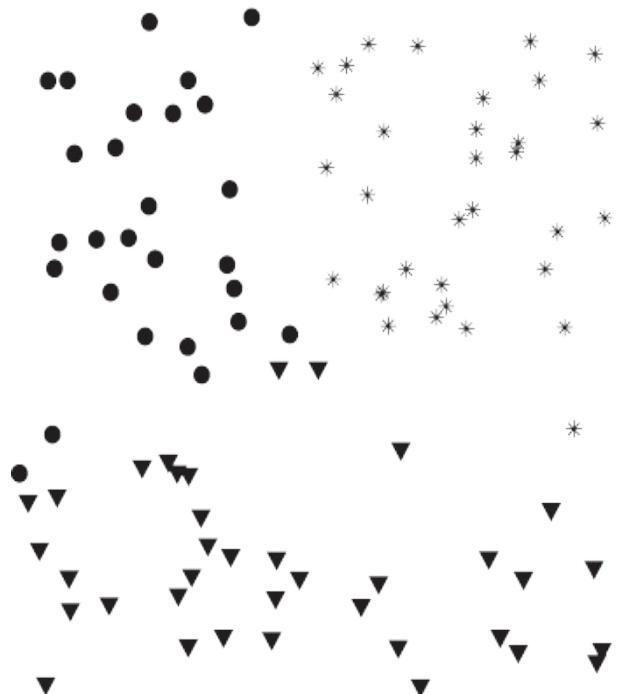
at the distances of the fourth nearest neighbors), we set K-means and complete link to find three clusters as well. (In [Figure 7.26\(b\)](#) the noise is shown by the small markers.) However, the clusters do not look compelling for any of the three methods. In higher dimensions, such problems cannot be so easily detected.



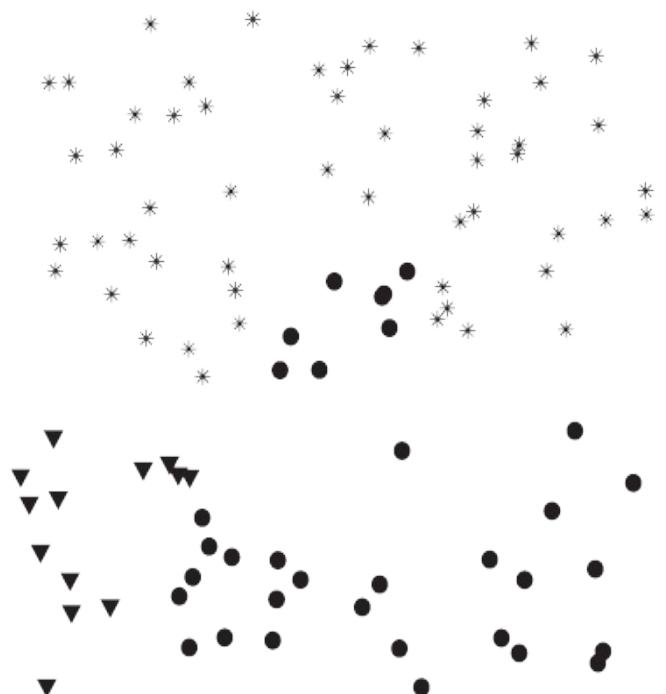
(a) Original points.



(b) Three clusters found by DBSCAN.



(c) Three clusters found by K-means.



(d) Three clusters found by complete link

Figure 7.26.

Clustering of 100 uniformly distributed points.

7.5.1 Overview

Being able to distinguish whether there is non-random structure in the data is just one important aspect of cluster validation. The following is a list of several important issues for cluster validation.

1. Determining the **clustering tendency** of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
2. Determining the correct number of clusters.
3. Evaluating how well the results of a cluster analysis fit the data *without* reference to external information.
4. Comparing the results of a cluster analysis to externally known results, such as externally provided class labels.
5. Comparing two sets of clusters to determine which is better.

Notice that items 1, 2, and 3 do not make use of any external information—they are unsupervised techniques—while item 4 requires external information. Item 5 can be performed in either a supervised or an unsupervised manner. A further distinction can be made with respect to items 3, 4, and 5: Do we want to evaluate the entire clustering or just individual clusters?

While it is possible to develop various numerical measures to assess the different aspects of cluster validity mentioned above, there are a number of challenges. First, a measure of cluster validity is sometimes quite limited in the scope of its applicability. For example, most work on measures of clustering tendency has been done for two- or three-dimensional spatial data. Second, we need a framework to interpret any measure. If we obtain a value

of 10 for a measure that evaluates how well cluster labels match externally provided class labels, does this value represent a good, fair, or poor match? The goodness of a match often can be measured by looking at the statistical distribution of this value, i.e., how likely it is that such a value occurs by chance. Finally, if a measure is too complicated to apply or to understand, then few will use it.

The evaluation measures, or indices, that are applied to judge various aspects of cluster validity are traditionally classified into the following three types.

Unsupervised. Measures the goodness of a clustering structure without respect to external information. An example of this is the SSE. Unsupervised measures of cluster validity are often further divided into two classes: measures of **cluster cohesion** (compactness, tightness), which determine how closely related the objects in a cluster are, and measures of **cluster separation** (isolation), which determine how distinct or well-separated a cluster is from other clusters. Unsupervised measures are often called **internal indices** because they use only information present in the data set.

Supervised. Measures the extent to which the clustering structure discovered by a clustering algorithm matches some external structure. An example of a supervised index is entropy, which measures how well cluster labels match externally supplied class labels. Supervised measures are often called **external indices** because they use information not present in the data set.

Relative. Compares different clusterings or clusters. A relative cluster evaluation measure is a supervised or unsupervised evaluation measure that is used for the purpose of comparison. Thus, relative measures are not actually a separate type of cluster evaluation measure, but are instead a specific use of such measures. As an example, two K-means clusterings can be compared using either the SSE or entropy.

In the remainder of this section, we provide specific details concerning cluster validity. We first describe topics related to unsupervised cluster evaluation, beginning with (1) measures based on cohesion and separation, and (2) two techniques based on the proximity matrix. Since these approaches are useful only for partitional sets of clusters, we also describe the popular cophenetic correlation coefficient, which can be used for the unsupervised evaluation of a hierarchical clustering. We end our discussion of unsupervised evaluation with brief discussions about finding the correct number of clusters and evaluating clustering tendency. We then consider supervised approaches to cluster validity, such as entropy, purity, and the Jaccard measure. We conclude this section with a short discussion of how to interpret the values of (unsupervised or supervised) validity measures.

7.5.2 Unsupervised Cluster Evaluation Using Cohesion and Separation

Many internal measures of cluster validity for partitional clustering schemes are based on the notions of cohesion or separation. In this section, we use cluster validity measures for prototype- and graph-based clustering techniques to explore these notions in some detail. In the process, we will also see some interesting relationships between prototype- and graph-based measures.

In general, we can consider expressing overall cluster validity for a set of K clusters as a weighted sum of the validity of individual clusters,

$$\text{overall validity} = \sum_{i=1}^K w_i \text{ validity}(C_i). \quad (7.8)$$

The *validity* function can be cohesion, separation, or some combination of these quantities. The weights will vary depending on the cluster validity measure. In some cases, the weights are simply 1 or the size of the cluster, while in other cases to be discussed a bit later, they reflect a more complicated property of the cluster.

Graph-Based View of Cohesion and Separation

From a graph-based view, the cohesion of a cluster can be defined as the sum of the weights of the links in the proximity graph that connect points within the cluster. See [Figure 7.27\(a\)](#). (Recall that the proximity graph has data objects as nodes, a link between each pair of data objects, and a weight assigned to each link that is the proximity between the two data objects connected by the link.) Likewise, the separation between two clusters can be measured by the sum of the weights of the links from points in one cluster to points in the other cluster. This is illustrated in [Figure 7.27\(b\)](#).

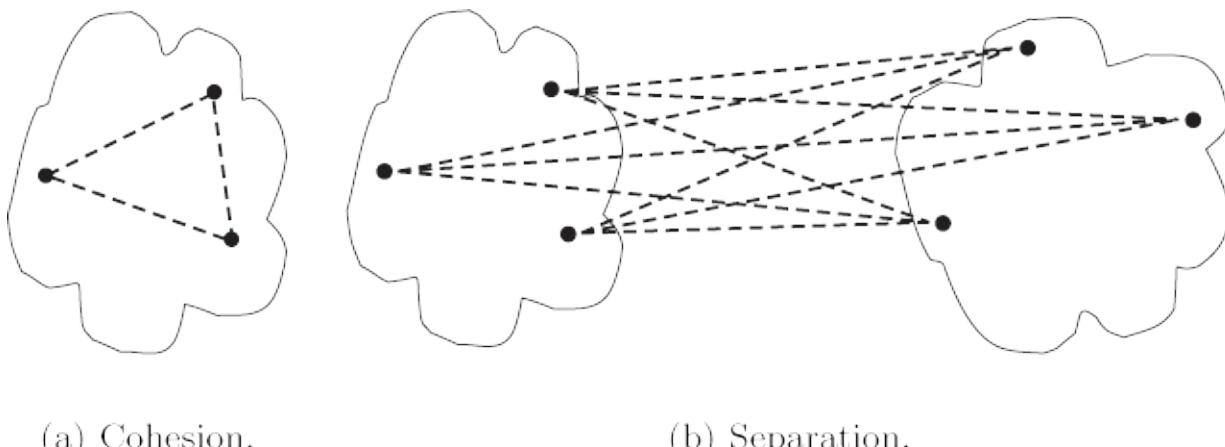


Figure 7.27.
Graph-based view of cluster cohesion and separation.

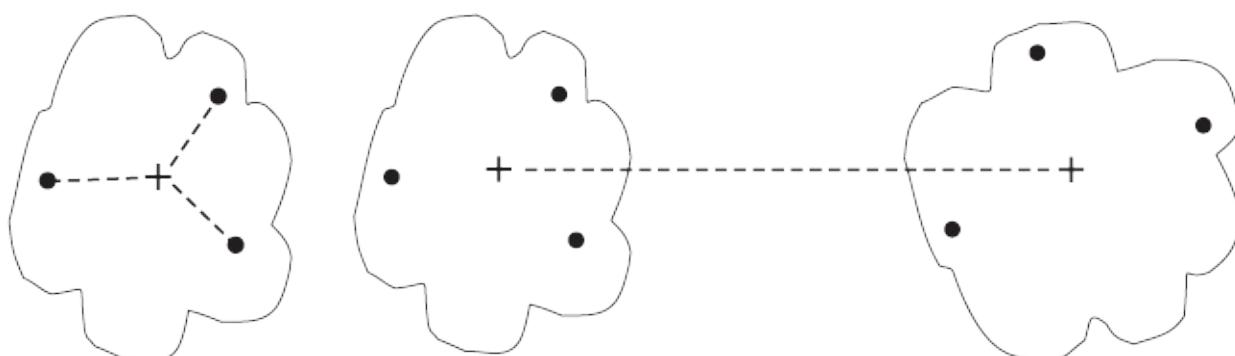
Most simply, the cohesion and separation for a graph-based cluster can be expressed using [Equations 7.9](#) and [7.10](#), respectively. The *proximity* function can be a similarity or a dissimilarity. For similarity, as in [Table 7.6](#), higher values are better for cohesion while lower values are better for separation. For dissimilarity, the opposite is true, i.e., lower values are better for cohesion while higher values are better for separation. More complicated approaches are possible but typically embody the basic ideas of [figure 7.27a](#) and [7.27b](#).

$$\text{cohesion}(C_i) = \sum_{x \in C_i} \sum_{y \in C_i} \text{proximity}(x, y) \quad (7.9)$$

$$\text{separation}(C_i, C_j) = \sum_{x \in C_i} \sum_{y \in C_j} \text{proximity}(x, y) \quad (7.10)$$

Prototype-Based View of Cohesion and Separation

For prototype-based clusters, the cohesion of a cluster can be defined as the sum of the proximities with respect to the prototype (centroid or medoid) of the cluster. Similarly, the separation between two clusters can be measured by the proximity of the two cluster prototypes. This is illustrated in [Figure 7.28](#), where the centroid of a cluster is indicated by a “+”.



(a) Cohesion.

(b) Separation.

Figure 7.28.

Prototype-based view of cluster cohesion and separation.

Cohesion for a prototype-based cluster is given in [Equation 7.11](#), while two measures for separation are given in [Equations 7.12](#) and [7.13](#), respectively, where c_i is the prototype (centroid) of cluster C_i and c is the overall prototype (centroid). There are two measures for separation because, as we will see in the next section, the separation of cluster prototypes from an overall prototype is sometimes directly related to the separation of cluster prototypes from one another. (This is true, for example, for Euclidean distance.) Note that [Equation 7.11](#) is the cluster SSE if we let proximity be the squared Euclidean distance.

$$\text{cohesion}(C_i) = \sum_{x \in C_i} \text{proximity}(x, c_i) \quad (7.11)$$

$$\text{separation}(C_i, C_j) = \text{proximity}(c_i, c_j) \quad (7.12)$$

$$\text{separation}(C_i) = \text{proximity}(c_i, c) \quad (7.13)$$

Relationship between Prototype-Based Cohesion and Graph-Based Cohesion

While the graph-based and prototype-based approaches to measuring the cohesion and separation of a cluster seem distinct, for some proximity measures they are equivalent. For instance, for the SSE and points in Euclidean space, it can be shown ([Equation 7.14](#)) that the average pairwise distance between the points in a cluster is equivalent to the SSE of the cluster. See [Exercise 27](#) on page [610](#).

$$\text{Cluster SSE} = \sum_{x \in C_i} \text{dist}(c_i, x)^2 = \frac{1}{m_i} \sum_{x \in C_i} \sum_{y \in C_i} \text{dist}(x, y)^2 \quad (7.14)$$

Relationship of the Two Approaches to Prototype-Based Separation

When proximity is measured by Euclidean distance, the traditional measure of separation between clusters is the between group sum of squares (SSB), which is the sum of the squared distance of a cluster centroid, c_i , to the overall mean, c , of all the data points. The SSB is given by [Equation 7.15](#), where c_i is the mean of the i th cluster and c is the overall mean. The higher the total SSB of a clustering, the more separated the clusters are from one another.

$$\text{Total SSB} = \sum_{i=1}^K m_i \text{dist}(c_i, c)^2 \quad (7.15)$$

It is straightforward to show that the total SSB is directly related to the pairwise distances between the centroids. In particular, if the cluster sizes are equal, i.e., $m_i = m/K$, then this relationship takes the simple form given by [Equation 7.16](#). (See [Exercise 28](#) on page [610](#).) It is this type of equivalence that motivates the definition of prototype separation in terms of both [Equations 7.12](#) and [7.13](#).

$$\text{Total SSB} = \frac{1}{2K} \sum_{i=1}^K \sum_{j=1}^K m_i m_j \text{dist}(c_i, c_j)^2 \quad (7.16)$$

Relationship between Cohesion and Separation

For some validity measures, there is also a strong relationship between cohesion and separation. Specifically, it is possible to show that the sum of the total SSE and the total SSB is a constant; i.e., that it is equal to the total sum of squares (TSS), which is the sum of squares of the distance of each

point to the overall mean of the data. The importance of this result is that minimizing SSE (cohesion) is equivalent to maximizing SSB (separation).

We provide the proof of this fact below, since the approach illustrates techniques that are also applicable to proving the relationships stated in the last two sections. To simplify the notation, we assume that the data is one-dimensional, i.e., $\text{dist}(x,y) = (x-y)^2$. Also, we use the fact that the cross-term $\sum_{i=1}^K \sum_{x \in C_i} (x - c_i)(c - c_i)$ is 0. (See [Exercise 29](#) on page [610](#).)

$$\begin{aligned} TSS &= \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)^2 = \sum_{i=1}^K \sum_{x \in C_i} ((x - c_i) - (c - c_i))^2 = \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)^2 - 2 \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)(c - c_i) + \sum_{i=1}^K \sum_{x \in C_i} (c - c_i)^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)^2 + \sum_{i=1}^K \sum_{x \in C_i} (c - c_i)^2 = \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)^2 + \sum_{i=1}^K |C_i| (c - c_i)^2 = SSE + SSB \end{aligned}$$

Relationship between Graph- and Centroid-Based Cohesion

It can also be shown that there is a relationship between graph- and centroid-based cohesion measures for Euclidean distance. For simplicity, we once again assume one-dimensional data. Recall that $c_i = 1/m_i \sum_{y \in C_i} y$.

$$\begin{aligned} m_i^2 \text{ cohesion}(C_i) &= m_i^2 \sum_{x \in C_i} \text{proximity}(x, c_i) = \sum_{x \in C_i} m_i^2 (x - c_i)^2 = \sum_{x \in C_i} (m_i x - m_i c_i)^2 = \sum_{x \in C_i} (m_i x - m_i (1/m_i \sum_{y \in C_i} y))^2 \\ &= \sum_{x \in C_i} \sum_{y \in C_i} (x - y)^2 = \sum_{x \in C_i} \sum_{y \in C_i} (x - y)^2 = \sum_{x \in C_i} \sum_{y \in C_i} \text{proximity}(x, y) \end{aligned}$$

More generally, in cases where a centroid makes sense for the data, the simple graph or centroid-based measures of cluster validity we presented are often related.

Overall Measures of Cohesion and Separation

The previous definitions of cluster cohesion and separation gave us some simple and well-defined measures of individual cluster validity that can be combined into an overall measure of cluster validity by using a weighted sum, as indicated in [Equation 7.8](#). However, we need to decide what weights to use. Not surprisingly, the weights used can vary widely. Often, but not always, they are either a function of cluster size or 1, which treats all clusters equally regardless of size.

The CLUstering TOOlkit (CLUTO) (see the Bibliographic Notes) uses the cluster evaluation measures described in [Table 7.6](#), as well as some other evaluation measures not mentioned here. Only similarity measures are used: cosine, correlation, Jaccard, and the inverse of Euclidean distance. \mathcal{J}_1 is a measure of cohesion in terms of the pairwise similarity of objects in the cluster. \mathcal{J}_2 is a measure of cohesion that can be expressed either in terms of the sum of the similarities of objects in the cluster to the cluster centroid or in terms of the pairwise similarities of objects in the cluster. ϵ_1 is a measure of separation. It can be defined in terms of the similarity of a cluster centroid to the overall centroid or in terms of the pairwise similarities of object in the cluster to objects in other clusters. (Although ϵ_1 is a measure of separation, the second definition shows that it also uses cluster cohesion, albeit in the cluster weight.) \mathcal{G}_1 , which is a cluster validity measure based on both cohesion and separation, is the sum of the pairwise similarities of all objects in the cluster with all objects outside the cluster—the total weight of the edges of the similarity graph that must be cut to separate the cluster from all other clusters—divided by the sum of the pairwise similarities of objects in the cluster.

Table 7.6. Table of graph-based cluster evaluation measures.

Name	Cluster Measure	Cluster Weight	Type
\mathcal{J}_1	$\sum_{x \in C_i} \sum_{y \in C_i} \text{sim}(x, y)$	$1/m_i$	graph-based cohesion
\mathcal{J}_2	$\sum_{x \in C_i} \text{sim}(x, c_i)$		prototype-based cohesion
\mathcal{J}_2	$\sum_{x \in C_i} \sum_{y \in C_i} \text{sim}(x, y)$		prototype-based cohesion
ε_1	$\text{sim}(c_i, c_i)$	m_i	prototype-based separation
ε_1	$\sum_{j=1}^k \sum_{x \in C_i} \sum_{y \in C_j} \text{sim}(x, y)$	$m_i(\sum_{x \in C_i} \sum_{y \in C_i} \text{sim}(x, y))$	graph-based separation
\mathcal{G}_1	$\sum_{j=1, j \neq i}^k \sum_{x \in C_i} \sum_{y \in C_j} \text{sim}(x, y)$	$1 \sum_{x \in C_i} \sum_{y \in C_i} \text{sim}(x, y)$	graph-based separation and cohesion

Note that any unsupervised measure of cluster validity potentially can be used as an objective function for a clustering algorithm and vice versa. CLUTO takes this approach by using an algorithm that is similar to the incremental K-means algorithm discussed in [Section 7.2.2](#). Specifically, each point is assigned to the cluster that produces the best value for the cluster evaluation function. The cluster evaluation measure \mathcal{J}_2 corresponds to traditional K-means and produces clusters that have good SSE values. The other measures produce clusters that are not as good with respect to SSE, but that are more optimal with respect to the specified cluster validity measure.

Evaluating Individual Clusters and Objects

So far, we have focused on using cohesion and separation in the overall evaluation of a group of clusters. Most of these measures of cluster validity also can be used to evaluate individual clusters and objects. For example, we can rank individual clusters according to their specific value of cluster validity, i.e., cluster cohesion or separation. A cluster that has a high value of cohesion

may be considered better than a cluster that has a lower value. This information often can be used to improve the quality of a clustering. If, for example, a cluster is not very cohesive, then we may want to split it into several subclusters. On the other hand, if two clusters are relatively cohesive, but not well separated, we may want to merge them into a single cluster.

We can also evaluate the objects within a cluster in terms of their contribution to the overall cohesion or separation of the cluster. Objects that contribute more to the cohesion and separation are near the “interior” of the cluster. Those objects for which the opposite is true are probably near the “edge” of the cluster. In the following section, we consider a cluster evaluation measure that uses an approach based on these ideas to evaluate points, clusters, and the entire set of clusters.

The Silhouette Coefficient

The popular method of silhouette coefficients combines both cohesion and separation. The following steps explain how to compute the silhouette coefficient for an individual point, a process that consists of the following three steps. We use distances, but an analogous approach can be used for similarities.

1. For the i th object, calculate its average distance to all other objects in its cluster. Call this value a_i .
2. For the i th object and any cluster not containing the object, calculate the object’s average distance to all the objects in the given cluster. Find the minimum such value with respect to all clusters; call this value b_i .
3. For the i th object, the silhouette coefficient is $s_i = (b_i - a_i) / \max(a_i, b_i)$.

The value of the silhouette coefficient can vary between -1 and 1 . A negative value is undesirable because this corresponds to a case in which a_i , the average distance to points in the cluster, is greater than b_i , the minimum average distance to points in another cluster. We want the silhouette coefficient to be positive ($a_i < b_i$), and for a_i to be as close to 0 as possible, since the coefficient assumes its maximum value of 1 when $a_i = 0$.

We can compute the average silhouette coefficient of a cluster by simply taking the average of the silhouette coefficients of points belonging to the cluster. An overall measure of the goodness of a clustering can be obtained by computing the average silhouette coefficient of all points.

Example 7.8 (Silhouette Coefficient).

Figure 7.29 shows a plot of the silhouette coefficients for points in 10 clusters. Darker shades indicate lower silhouette coefficients.

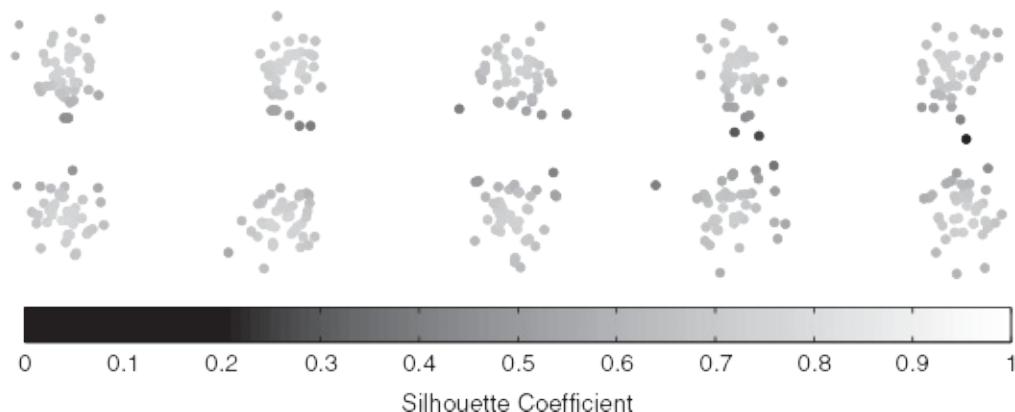


Figure 7.29.
Silhouette coefficients for points in ten clusters.

7.5.3 Unsupervised Cluster Evaluation

Using the Proximity Matrix

In this section, we examine a couple of unsupervised approaches for assessing cluster validity that are based on the proximity matrix. The first compares an actual and idealized proximity matrix, while the second uses visualization.

General Comments on Unsupervised Cluster Evaluation Measures

In addition to the measures presented above, a large number of other measures have been proposed as unsupervised cluster validity measures. Almost all these measures, including the measures presented above are intended for partitional, center-based clusters. In particular, none of them does well for continuity- or density-based clusters. Indeed, a recent evaluation—see Bibliographic Notes—of a dozen such measures found that although a number of them did well in terms of handling issues such as noise and differing sizes and density, none of them except a relatively recently proposed measure, Clustering Validation index based on Nearest Neighbors (CVNN), did well on handling arbitrary shapes. The silhouette index, however, did well on all other issues examined except for that.

Most unsupervised cluster evaluation measures, such as the silhouette coefficient, incorporate both cohesion (compactness) and separation. When used with a partitional clustering algorithm such as K-means, these measures will tend to decrease until the “natural” set of clusters is found and start increasing once clusters are being split “too finely” since separation will suffer and cohesion will not improve much. Thus, these measures can provide a way to determine the number of clusters. However, if the definition of a cluster

used by the clustering algorithm, differs from that of the cluster evaluation measure, then the set of clusters identified as optimal by the algorithm and validation measure can differ. For instance, CLUTO uses the measures described in [Table 7.6](#) to drive its clustering, and thus, the clustering produced will not usually match the optimal clusters according to the silhouette coefficient. Likewise for standard K-means and SSE. In addition, if there actually are subclusters that are not separated very well from one another, then methods that incorporate both may provide only a coarse view of the cluster structure of the data. Another consideration is that when clustering for summarization, we are not interested in the “natural” cluster structure of the data, but rather want to achieve a certain level of approximation, e.g., want to reduce SSE to a certain level.

More generally, if there are not too many clusters, then it can be better to examine cluster cohesion and separation independently. This can give a more comprehensive view of how cohesive each cluster is and how well each pair of clusters is separated from one another. For instance, given a centroid based clustering, we could compute the pairwise similarity or distance of the centroids, i.e., compute the distance or similarity matrix of the centroids. The approach just outlined is similar to looking at the confusion matrix for a classification problem instead of classification measures, such as accuracy or the F -measure.

Measuring Cluster Validity via Correlation

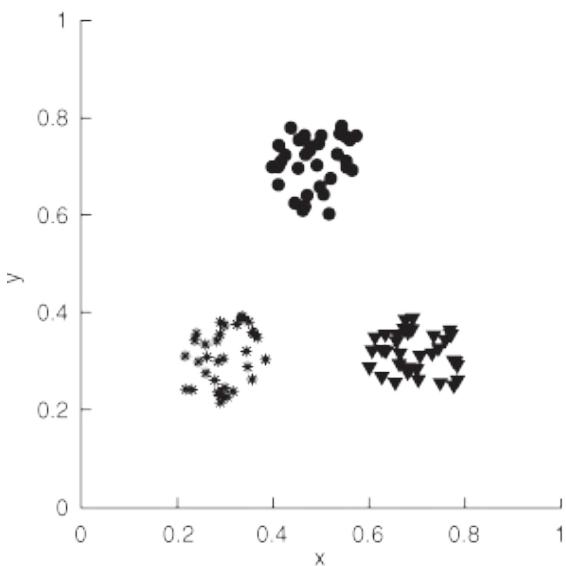
If we are given the similarity matrix for a data set and the cluster labels from a cluster analysis of the data set, then we can evaluate the “goodness” of the clustering by looking at the correlation between the similarity matrix and an ideal version of the similarity matrix based on the cluster labels. (With minor changes, the following applies to proximity matrices, but for simplicity, we discuss only similarity matrices.) More specifically, an ideal cluster is one

whose points have a similarity of 1 to all points in the cluster, and a similarity of 0 to all points in other clusters. Thus, if we sort the rows and columns of the similarity matrix so that all objects belonging to the same cluster are together, then an ideal cluster similarity matrix has a **block diagonal** structure. In other words, the similarity is non-zero, i.e., 1, inside the blocks of the similarity matrix whose entries represent intra-cluster similarity, and 0 elsewhere. The ideal cluster similarity matrix is constructed by creating a matrix that has one row and one column for each data point—just like an actual similarity matrix—and assigning a 1 to an entry if the associated pair of points belongs to the same cluster. All other entries are 0.

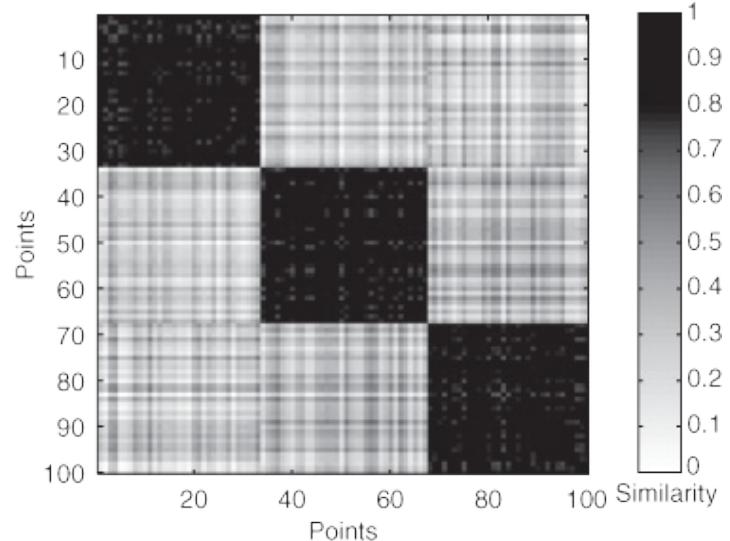
High correlation between the ideal and actual similarity matrices indicates that the points that belong to the same cluster are close to each other, while low correlation indicates the opposite. (Because the actual and ideal similarity matrices are symmetric, the correlation is calculated only among the $n(n-1)/2$ entries below or above the diagonal of the matrices.) Consequently, this is not a good measure for many density- or contiguity-based clusters, because they are not globular and can be closely intertwined with other clusters.

Example 7.9 (Correlation of Actual and Ideal Similarity Matrices).

To illustrate this measure, we calculated the correlation between the ideal and actual similarity matrices for the K-means clusters shown in [Figure 7.26\(c\)](#) (random data) and [Figure 7.30\(a\)](#) (data with three well-separated clusters). The correlations were 0.5810 and 0.9235, respectively, which reflects the expected result that the clusters found by K-means in the random data are worse than the clusters found by K-means in data with well-separated clusters.



(a) Well-separated clusters.



(b) Similarity matrix sorted by K-means cluster labels.

Figure 7.30.

Similarity matrix for well-separated clusters.

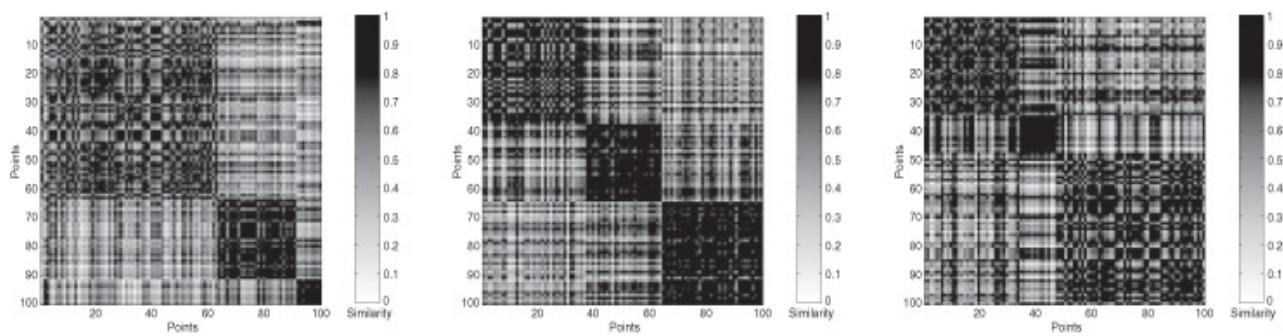
Judging a Clustering Visually by Its Similarity Matrix

The previous technique suggests a more general, qualitative approach to judging a set of clusters: Order the similarity matrix with respect to cluster labels and then plot it. In theory, if we have well-separated clusters, then the similarity matrix should be roughly block-diagonal. If not, then the patterns displayed in the similarity matrix can reveal the relationships between clusters. Again, all of this can be applied to dissimilarity matrices, but for simplicity, we will only discuss similarity matrices.

Example 7.10 (Visualizing a Similarity Matrix).

Consider the points in [Figure 7.30\(a\)](#), which form three well-separated clusters. If we use K-means to group these points into three clusters, then

we should have no trouble finding these clusters because they are well-separated. The separation of these clusters is illustrated by the reordered similarity matrix shown in [Figure 7.30\(b\)](#). (For uniformity, we have transformed the distances into similarities using the formula $s=1-(d-\min_d)/(\max_d-\min_d)$.) [Figure 7.31](#) shows the reordered similarity matrices for clusters found in the random data set of [Figure 7.26](#) by DBSCAN, K-means, and complete link.



(a) Similarity matrix sorted by DBSCAN cluster labels.

(b) Similarity matrix sorted by K-means cluster labels.

(c) Similarity matrix sorted by complete link cluster labels.

Figure 7.31.

Similarity matrices for clusters from random data.

The well-separated clusters in [Figure 7.30](#) show a very strong, block-diagonal pattern in the reordered similarity matrix. However, there are also weak block diagonal patterns—see [Figure 7.31](#)—in the reordered similarity matrices of the clusterings found by K-means, DBSCAN, and complete link in the random data. Just as people can find patterns in clouds, data mining algorithms can find clusters in random data. While it is entertaining to find patterns in clouds, it is pointless and perhaps embarrassing to find clusters in noise.

This approach may seem hopelessly expensive for large data sets, since the computation of the proximity matrix takes $O(m^2)$ time, where m is the number

of objects, but with sampling, this method can still be used. We can take a sample of data points from each cluster, compute the similarity between these points, and plot the result. It is sometimes necessary to oversample small clusters and undersample large ones to obtain an adequate representation of all clusters.

7.5.4 Unsupervised Evaluation of Hierarchical Clustering

The previous approaches to cluster evaluation are intended for partitional clusterings. Here we discuss the cophenetic correlation, a popular evaluation measure for hierarchical clusterings. The **cophenetic distance** between two objects is the proximity at which an agglomerative hierarchical clustering technique puts the objects in the same cluster for the first time. For example, if at some point in the agglomerative hierarchical clustering process, the smallest distance between the two clusters that are merged is 0.1, then all points in one cluster have a cophenetic distance of 0.1 with respect to the points in the other cluster. In a cophenetic distance matrix, the entries are the cophenetic distances between each pair of objects. The cophenetic distance is different for each hierarchical clustering of a set of points.

Example 7.11 (Cophenetic Distance Matrix).

Table 7.7 shows the cophenetic distance matrix for the single link clustering shown in **Figure 7.16**. (The data for this figure consists of the six two-dimensional points given in **Table 2.14**.)

Table 7.7. Cophenetic distance matrix for single link and data in **Table**

2.14 [on page 90](#).

Point	P1	P2	P3	P4	P5	P6
P1	0	0.222	0.222	0.222	0.222	0.222
P2	0.222	0	0.148	0.151	0.139	0.148
P3	0.222	0.148	0	0.151	0.148	0.110
P4	0.222	0.151	0.151	0	0.151	0.151
P5	0.222	0.139	0.148	0.151	0	0.148
P6	0.222	0.148	0.110	0.151	0.148	0

The **Cophenetic Correlation Coefficient** (CPCC) is the correlation between the entries of this matrix and the original dissimilarity matrix and is a standard measure of how well a hierarchical clustering (of a particular type) fits the data. One of the most common uses of this measure is to evaluate which type of hierarchical clustering is best for a particular type of data.

Example 7.12 (Cophenetic Correlation Coefficient).

We calculated the CPCC for the hierarchical clusterings shown in [Figures 7.16](#) [– 7.19](#). These values are shown in [Table 7.8](#). The hierarchical clustering produced by the single link technique seems to fit the data less well than the clusterings produced by complete link, group average, and Ward's method.

Table 7.8. Cophenetic correlation coefficient for data of [Table 2.14](#) and four agglomerative hierarchical clustering techniques.

Technique	CPCC
Single Link	0.44
Complete Link	0.63
Group Average	0.66
Ward's	0.64

7.5.5 Determining the Correct Number of Clusters

Various unsupervised cluster evaluation measures can be used to approximately determine the correct or natural number of clusters.

Example 7.13 (Number of Clusters).

The data set of [Figure 7.29](#) has 10 natural clusters. [Figure 7.32](#) shows a plot of the SSE versus the number of clusters for a (bisecting) K-means clustering of the data set, while [Figure 7.33](#) shows the average silhouette coefficient versus the number of clusters for the same data. There is a distinct knee in the SSE and a distinct peak in the silhouette coefficient when the number of clusters is equal to 10.

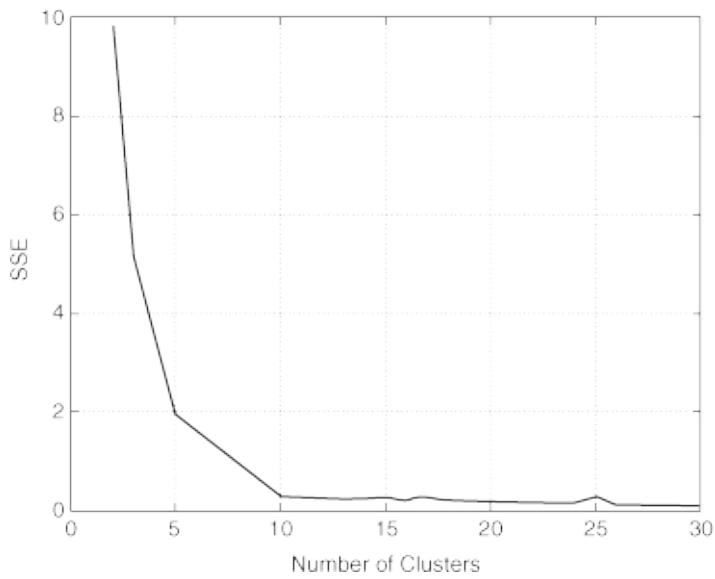


Figure 7.32.

SSE versus number of clusters for the data of [Figure 7.29](#) on page [582](#).

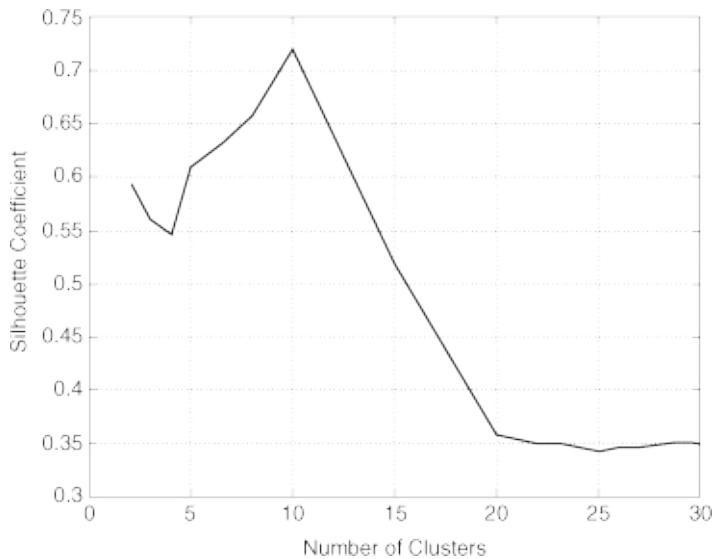


Figure 7.33.

Average silhouette coefficient versus number of clusters for the data of [Figure 7.29](#).

Thus, we can try to find the natural number of clusters in a data set by looking for the number of clusters at which there is a knee, peak, or dip in the plot of the evaluation measure when it is plotted against the number of clusters. Of course, such an approach does not always work well. Clusters can be

considerably more intertwined or overlapping than those shown in [Figure 7.29](#). Also, the data can consist of nested clusters. Actually, the clusters in [Figure 7.29](#) are somewhat nested; i.e., there are five pairs of clusters since the clusters are closer top to bottom than they are left to right. There is a knee that indicates this in the SSE curve, but the silhouette coefficient curve is not as clear. In summary, while caution is needed, the technique we have just described can provide insight into the number of clusters in the data.

7.5.6 Clustering Tendency

One obvious way to determine if a data set has clusters is to try to cluster it. However, almost all clustering algorithms will dutifully find clusters when given data. To address this issue, we could evaluate the resulting clusters and only claim that a data set has clusters if at least some of the clusters are of good quality. However, this approach does not address the fact the clusters in the data can be of a different type than those sought by our clustering algorithm. To handle this additional problem, we could use multiple algorithms and again evaluate the quality of the resulting clusters. If the clusters are uniformly poor, then this may indeed indicate that there are no clusters in the data.

Alternatively, and this is the focus of measures of clustering tendency, we can try to evaluate whether a data set has clusters without clustering. The most common approach, especially for data in Euclidean space, has been to use statistical tests for spatial randomness. Unfortunately, choosing the correct model, estimating the parameters, and evaluating the statistical significance of the hypothesis that the data is non-random can be quite challenging. Nonetheless, many approaches have been developed, most of them for points in low-dimensional Euclidean space.

Example 7.14 (Hopkins Statistic).

For this approach, we generate p points that are randomly distributed across the data space and also sample p actual data points. For both sets of points, we find the distance to the nearest neighbor in the original data set. Let the u_i be the nearest neighbor distances of the artificially generated points, while the w_i are the nearest neighbor distances of the sample of points from the original data set. The Hopkins statistic H is then defined by [Equation 7.17](#).

$$H = \frac{\sum_{i=1}^p w_i}{\sum_{i=1}^p u_i + \sum_{i=1}^p w_i} \quad (7.17)$$

If the randomly generated points and the sample of data points have roughly the same nearest neighbor distances, then H will be near 0.5. Values of H near 0 and 1 indicate, respectively, data that is highly clustered and data that is regularly distributed in the data space. To give an example, the Hopkins statistic for the data of [Figure 7.26](#) was computed for $p=20$ and 100 different trials. The average value of H was 0.56 with a standard deviation of 0.03. The same experiment was performed for the well-separated points of [Figure 7.30](#). The average value of H was 0.95 with a standard deviation of 0.006.

7.5.7 Supervised Measures of Cluster Validity

When we have external information about data, it is typically in the form of externally derived class labels for the data objects. In such cases, the usual procedure is to measure the degree of correspondence between the cluster

labels and the class labels. But why is this of interest? After all, if we have the class labels, then what is the point in performing a cluster analysis? Motivations for such an analysis include the comparison of clustering techniques with the “ground truth” or the evaluation of the extent to which a manual classification process can be automatically produced by cluster analysis, e.g., the clustering of news articles. Another potential motivation could be to evaluate whether objects in the same cluster tend to have the same label for semi-supervised learning techniques.

We consider two different kinds of approaches. The first set of techniques use measures from classification, such as entropy, purity, and the F -measure. These measures evaluate the extent to which a cluster contains objects of a single class. The second group of methods is related to the similarity measures for binary data, such as the Jaccard measure that we saw in [Chapter 2](#). These approaches measure the extent to which two objects that are in the same class are in the same cluster and vice versa. For convenience, we will refer to these two types of measures as **classification-oriented** and **similarity-oriented**, respectively.

Classification-Oriented Measures of Cluster Validity

There are a number of measures that are commonly used to evaluate the performance of a classification model. In this section, we will discuss five: entropy, purity, precision, recall, and the F -measure. In the case of classification, we measure the degree to which predicted class labels correspond to actual class labels, but for the measures just mentioned, nothing fundamental is changed by using cluster labels instead of predicted class labels. Next, we quickly review the definitions of these measures in the context of clustering.

Entropy: The degree to which each cluster consists of objects of a single class. For each cluster, the class distribution of the data is calculated first, i.e., for cluster i we compute p_{ij} , the probability that a member of cluster i belongs to class j as $p_{ij} = m_{ij}/m_i$, where m_i is the number of objects in cluster i and m_{ij} is the number of objects of class j in cluster i . Using this class distribution, the entropy of each cluster i is calculated using the standard formula, $e_i = -\sum_{j=1}^L p_{ij} \log_2 p_{ij}$, where L is the number of classes. The total entropy for a set of clusters is calculated as the sum of the entropies of each cluster weighted by the size of each cluster, i.e., $e = \sum_{i=1}^K m_i e_i / m$, where K is the number of clusters and m is the total number of data points.

Purity: Another measure of the extent to which a cluster contains objects of a single class. Using the previous terminology, the purity of cluster i is $\text{purity}(i) = \max_j p_{ij}$, the overall purity of a clustering is $\text{purity} = \sum_{i=1}^K m_i \text{purity}(i)$.

Precision: The fraction of a cluster that consists of objects of a specified class. The precision of cluster i with respect to class j is $\text{precision}(i,j) = p_{ij}$.

Recall: The extent to which a cluster contains all objects of a specified class. The recall of cluster i with respect to class j is $\text{recall}(i,j) = m_{ij}/m_j$, where m_j is the number of objects in class j .

F-measure A combination of both precision and recall that measures the extent to which a cluster contains *only* objects of a particular class and *all* objects of that class. The *F*-measure of cluster i with respect to class j is $F(i,j) = (2 \times \text{precision}(i,j) \times \text{recall}(i,j)) / (\text{precision}(i,j) + \text{recall}(i,j))$. The *F*-measure of a set of clusters, partitional or hierarchical is presented on page 594 when we discuss cluster validity for hierarchical clusterings.

Example 7.15 (Supervised Evaluation Measures).

We present an example to illustrate these measures. Specifically, we use K-means with the cosine similarity measure to cluster 3204 newspaper

articles from the *Los Angeles Times*. These articles come from six different classes: Entertainment, Financial, Foreign, Metro, National, and Sports.

Table 7.9 shows the results of a K-means clustering to find six clusters. The first column indicates the cluster, while the next six columns together form the confusion matrix; i.e., these columns indicate how the documents of each category are distributed among the clusters. The last two columns are the entropy and purity of each cluster, respectively.

Table 7.9. K-means clustering results for the *LA Times* document data set.

Cluster	Entertainment	Financial	Foreign	Metro	National	Sports	Entropy	Purity
1	3	5	40	506	96	27	1.2270	0.7474
2	4	7	280	29	39	2	1.1472	0.7756
3	1	1	1	7	4	671	0.1813	0.9796
4	10	162	3	119	73	2	1.7487	0.4390
5	331	22	5	70	13	23	1.3976	0.7134
6	5	358	12	212	48	13	1.5523	0.5525
Total	354	555	341	943	273	738	1.1450	0.7203

Ideally, each cluster will contain documents from only one class. In reality, each cluster contains documents from many classes. Nevertheless, many clusters contain documents primarily from just one class. In particular, cluster 3, which contains mostly documents from the Sports section, is exceptionally good, both in terms of purity and entropy. The purity and entropy of the other clusters is not as good, but can typically be greatly improved if the data is partitioned into a larger number of clusters.

Precision, recall, and the F -measure can be calculated for each cluster. To give a concrete example, we consider cluster 1 and the Metro class of [Table 7.9](#). The precision is $506/677=0.75$, recall is $506/943=0.26$, and hence, the F value is 0.39. In contrast, the F value for cluster 3 and Sports is 0.94. As in classification, the confusion matrix gives the most detailed information.

Similarity-Oriented Measures of Cluster Validity

The measures that we discuss in this section are all based on the premise that any two objects that are in the same cluster should be in the same class and vice versa. We can view this approach to cluster validity as involving the comparison of two matrices: (1) the **ideal cluster similarity matrix** discussed previously, which has a 1 in the ij th entry if two objects, i and j , are in the same cluster and 0, otherwise, and (2) a **class similarity matrix** defined with respect to class labels, which has a 1 in the ij th entry if two objects, i and j , belong to the same class, and a 0 otherwise. As before, we can take the correlation of these two matrices as the measure of cluster validity. This measure is known as Hubert's Γ statistic in the clustering validation literature.

Example 7.16 (Correlation between Cluster and Class Matrices).

To demonstrate this idea more concretely, we give an example involving five data points, p_1, p_2, p_3, p_4 , and p_5 , two clusters, $C_1 = \{ p_1, p_2, p_3 \}$ and $C_2 = \{ p_4, p_5 \}$, and two classes, $L_1 = \{ p_1, p_2 \}$ and $L_2 = \{ p_3, p_4, p_5 \}$. The ideal cluster and class similarity matrices are given in [Tables 7.10](#) and [7.11](#). The correlation between the entries of these two matrices is 0.359.

Table 7.10. Ideal cluster similarity matrix.

--	--	--	--	--	--

Point	p1	p2	p3	p4	p5
p1	1	1	1	0	0
p2	1	1	1	0	0
p3	1	1	1	0	0
p4	0	0	0	1	1
p5	0	0	0	1	1

Table 7.11. Class similarity matrix.

Point	p1	p2	p3	p4	p5
p1	1	1	0	0	0
p2	1	1	0	0	0
p3	0	0	1	1	1
p4	0	0	1	1	1
p5	0	0	1	1	1

More generally, we can use any of the measures for binary similarity that we saw in [Section 2.4.5](#). (For example, we can convert these two matrices into binary vectors by appending the rows.) We repeat the definitions of the four quantities used to define those similarity measures, but modify our descriptive text to fit the current context. Specifically, we need to compute the following four quantities for all pairs of distinct objects. (There are $m(m-1)/2$ such pairs, if m is the number of objects.)

f00=number of pairs of objects having a different class and a different clusterf0

In particular, the simple matching coefficient, which is known as the Rand statistic in this context, and the Jaccard coefficient are two of the most frequently used cluster validity measures.

$$\text{Rand statistic} = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}} \quad (7.18)$$

$$\text{Jaccard coefficient} = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \quad (7.19)$$

Example 7.17 (Rand and Jaccard Measures).

Based on these formulas, we can readily compute the Rand statistic and Jaccard coefficient for the example based on [Tables 7.10](#) and [7.11](#). Noting that $f_{00}=4$, $f_{01}=2$, $f_{10}=2$, and $f_{11}=2$, the Rand statistic $= (2+4)/10=0.6$ and the Jaccard coefficient $= 2/(2+2+2)=0.33$.

We also note that the four quantities, f_{00} , f_{01} , f_{10} , and f_{11} , define a *contingency table* as shown in [Table 7.12](#).

Table 7.12. Two-way contingency table for determining whether pairs of objects are in the same class and same cluster.

	Same Cluster	Different Cluster
Same Class	f_{11}	f_{10}
Different Class	f_{01}	f_{00}

Previously, in the context of association analysis—see [Section 5.7.1](#) on page [402](#)—we presented an extensive discussion of measures of association that can be used for this type of contingency table. (Compare [Table 7.12](#) on page [593](#) with [Table 5.6](#) on page [402](#).) Those measures can also be applied to cluster validity.

Cluster Validity for Hierarchical Clusterings

So far in this section, we have discussed supervised measures of cluster validity only for partitional clusterings. Supervised evaluation of a hierarchical clustering is more difficult for a variety of reasons, including the fact that a preexisting hierarchical structure often does not exist. In addition, although relatively pure clusters often exist at certain levels in the hierarchical clustering, as the clustering proceeds, the clusters will become impure. The key idea of the approach presented here, which is based on the *F*-measure, is to evaluate whether a hierarchical clustering contains, for each class, at least one cluster that is relatively pure and includes most of the objects of that class. To evaluate a hierarchical clustering with respect to this goal, we compute, for each class, the *F*-measure for each cluster in the cluster hierarchy, and then take the maximum *F*-measure attained for any cluster. Finally, we calculate an overall *F*-measure for the hierarchical clustering by computing the weighted average of all per-class *F*-measures, where the weights are based on the class sizes. More formally, this hierarchical *F*-measure is defined as follows:

$$F = \sum_j m_j \max_i F(i, j)$$

where the maximum is taken over all clusters *i* at all levels, *m_j* is the number of objects in class *j*, and *m* is the total number of objects. Note that this measure can also be applied for a partitional clustering without modification.

7.5.8 Assessing the Significance of Cluster Validity Measures

Cluster validity measures are intended to help us measure the goodness of the clusters that we have obtained. Indeed, they typically give us a single number as a measure of that goodness. However, we are then faced with the problem of interpreting the significance of this number, a task that might be even more difficult.

The minimum and maximum values of cluster evaluation measures can provide some guidance in many cases. For instance, by definition, a purity of 0 is bad, while a purity of 1 is good, at least if we trust our class labels and want our cluster structure to reflect the class structure. Likewise, an entropy of 0 is good, as is an SSE of 0.

Sometimes, however, there is no minimum or maximum value, or the scale of the data might affect the interpretation. Also, even if there are minimum and maximum values with obvious interpretations, intermediate values still need to be interpreted. In some cases, we can use an absolute standard. If, for example, we are clustering for utility, we are often willing to tolerate only a certain level of error in the approximation of our points by a cluster centroid.

But if this is not the case, then we must do something else. A common approach is to interpret the value of our validity measure in statistical terms. Specifically, we attempt to judge how likely it is that our observed value was achieved by random chance. The value is good if it is unusual; i.e., if it is unlikely to be the result of random chance. The motivation for this approach is that we are interested only in clusters that reflect non-random structure in the data, and such structures should generate unusually high (low) values of our cluster validity measure, at least if the validity measures are designed to reflect the presence of strong cluster structure.

Example 7.18 (Significance of SSE).

To show how this works, we present an example based on K-means and the SSE. Suppose that we want a measure of how good the well-separated clusters of [Figure 7.30](#) are with respect to random data. We generate many random sets of 100 points having the same range as the points in the three clusters, find three clusters in each data set using K-means, and accumulate the distribution of SSE values for these clusterings. By using this distribution of the SSE values, we can then estimate the probability of the SSE value for the original clusters. [Figure 7.34](#) shows the histogram of the SSE from 500 random runs. The lowest SSE shown in [Figure 7.34](#) is 0.0173. For the three clusters of [Figure 7.30](#), the SSE is 0.0050. We could therefore conservatively claim that there is less than a 1% chance that a clustering such as that of [Figure 7.30](#) could occur by chance.

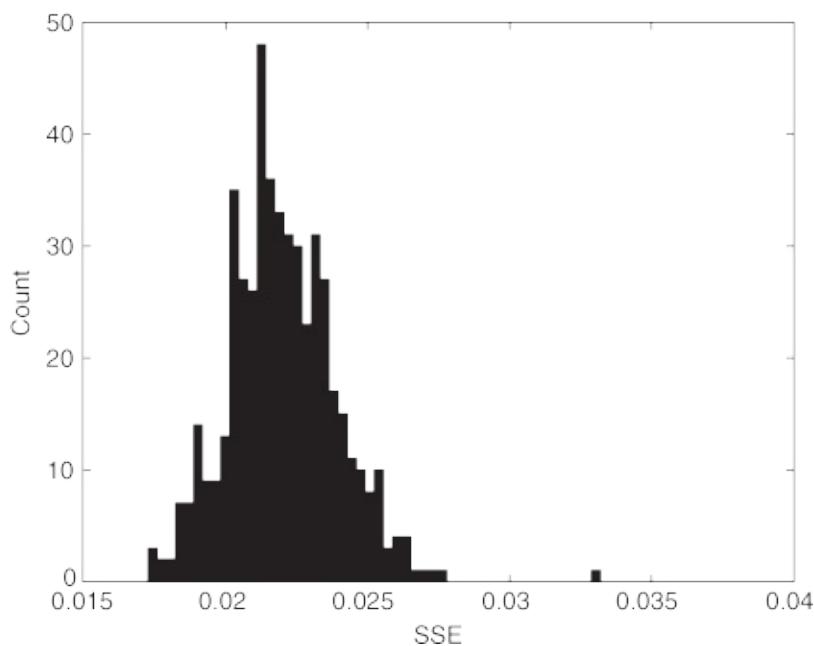


Figure 7.34.
Histogram of SSE for 500 random data sets.

In the previous example, randomization was used to evaluate the statistical significance of a cluster validity measure. However, for some measures, such

as Hubert's Γ statistic, the distribution is known and can be used to evaluate the measure. In addition, a normalized version of the measure can be computed by subtracting the mean and dividing by the standard deviation. See Bibliographic Notes for more details.

We stress that there is more to cluster evaluation (unsupervised or supervised) than obtaining a numerical measure of cluster validity. Unless this value has a natural interpretation based on the definition of the measure, we need to interpret this value in some way. If our cluster evaluation measure is defined such that lower (higher) values indicate stronger clusters, then we can use statistics to evaluate whether the value we have obtained is unusually low (high), provided we have a distribution for the evaluation measure. We have presented an example of how to find such a distribution, but there is considerably more to this topic, and we refer the reader to the Bibliographic Notes for more pointers.

Finally, even when an evaluation measure is used as a relative measure, i.e., to compare two clusterings, we still need to assess the significance in the difference between the evaluation measures of the two clusterings. Although one value will almost always be better than another, it can be difficult to determine if the difference is significant. Note that there are two aspects to this significance: whether the difference is statistically significant (repeatable) and whether the magnitude of the difference is meaningful with respect to the application. Many would not regard a difference of 0.1% as significant, even if it is consistently reproducible.

7.5.9 Choosing a Cluster Validity Measure

There are many more measures for evaluating cluster validity than have been discussed here. Various books and articles propose various measures as being superior to others. In this section, we offer some high-level guidance. First, it is important to distinguish whether the clustering is being performed for summarization or understanding. If summarization, typically class labels are not involved and the goal is maximum compression. This is often done by finding clusters that minimize the distance of objects to their closest cluster centroid. Indeed, the clustering process is often driven by the objective of minimizing representation error. Measures such as SSE are more natural for this application.

If the clustering is being performed for understanding, then the situation is more complicated. For the unsupervised case, virtually all measures try to maximize cohesion and separation. Some measures obtain a “best” value for a particular value of K , the number of clusters. Although this might seem appealing, such measures typically only identify one “right” number of clusters, even when subclusters are present. (Recall that cohesion and separation continue to increase for K-means until there is one cluster per point.) More generally, if the number of clusters is not too large, it can be useful to manually examine the cluster cohesion of each cluster and the pairwise separation of clusters. However, note that very few cluster validity measures are appropriate to contiguity or density-based clusters that can have irregular and intertwined shapes.

For the supervised case, where clustering is almost always performed with a goal of generating understandable clusters, the ideal result of clustering is to produce clusters that match the underlying class structure. Evaluating the match between a set of clusters and classes is a non-trivial problem. The F -measure and hierarchical F -measure discussed earlier, are examples of how to evaluate such a match. Other examples can be found in the references to cluster evaluation provided in the Bibliographic Notes. In any case, when the

number of clusters are relatively small, the confusion matrix can be more informative than any single measure of cluster validity since it can indicate which classes tend to appear in clusters with which other classes. Note that supervised cluster evaluation indices are independent of whether the clusters are center-, contiguity-, or density-based.

In conclusion, it is important to realize that clustering is often used as an exploratory data technique whose goal is often not to provide a crisp answer, but rather to provide some insight into the underlying structure of the data. In this situation, cluster validity indices are useful to the extent they are useful to that end goal.

7.6 Bibliographic Notes

Discussion in this chapter has been most heavily influenced by the books on cluster analysis written by [Jain and Dubes \[536\]](#), [Anderberg \[509\]](#), and [Kaufman and Rousseeuw \[540\]](#), as well as the more recent book edited by [and Aggarwal and Reddy \[507\]](#). Additional clustering books that may also be of interest include those by [Aldenderfer and Blashfield \[508\]](#), [Everitt et al. \[527\]](#), [Hartigan \[533\]](#), [Mirkin \[548\]](#), [Murtagh \[550\]](#), [Romesburg \[553\]](#), and [Späth \[557\]](#). A more statistically oriented approach to clustering is given by the pattern recognition book of [Duda et al. \[524\]](#), the machine learning book of [Mitchell \[549\]](#), and the book on statistical learning by [Hastie et al. \[534\]](#). General surveys of clustering are given by [Jain et al. \[537\]](#) and [Xu and Wunsch \[560\]](#), while a survey of spatial data mining techniques is provided by [Han et al. \[532\]](#). [Behrkin \[515\]](#) provides a survey of clustering techniques for data mining. A good source of references to clustering outside of the data mining field is the article by [Arabie and Hubert \[511\]](#). A paper by [Kleinberg \[541\]](#) provides a discussion of some of the trade-offs that clustering algorithms make and proves that it is impossible for a clustering algorithm to simultaneously possess three simple properties. A wide-ranging, retrospective article by Jain provides a look at clustering during the 50 years from the invention of K-means [\[535\]](#).

The K-means algorithm has a long history, but is still the subject of current research. The K-means algorithm was named by [MacQueen \[545\]](#), although its history is more extensive. Bock examines the origins of K-means and some of its extensions [\[516\]](#). The ISODATA algorithm by [Ball and Hall \[513\]](#) was an early, but sophisticated version of K-means that employed various pre- and postprocessing techniques to improve on the basic algorithm. The K-means algorithm and many of its variations are described in detail in the books by

Anderberg [509] and **Jain and Dubes [536]**. The bisecting K-means algorithm discussed in this chapter was described in a paper by **Steinbach et al. [558]**, and an implementation of this and other clustering approaches is freely available for academic use in the CLUTO (CLUstering TOolkit) package created by **Karypis [520]**. **Boley [517]** has created a divisive partitioning clustering algorithm (PDDP) based on finding the first principal direction (component) of the data, and Savaresi and **Boley [555]** have explored its relationship to bisecting K-means. Recent variations of K-means are a new incremental version of K-means (**Dhillon et al. [522]**), X-means (**Pelleg and Moore [552]**), and K-harmonic means (**Zhang et al [562]**). **Hamerly and Elkan [531]** discuss some clustering algorithms that produce better results than K-means. While some of the previously mentioned approaches address the initialization problem of K-means in some manner, other approaches to improving K-means initialization can also be found in the work of **Bradley and Fayyad [518]**. The K-means++ initialization approach was proposed by **Arthur and Vassilvitskii [512]**. **Dhillon and Modha [523]** present a generalization of K-means, called spherical K-means, which works with commonly used similarity functions. A general framework for K-means clustering that uses dissimilarity functions based on Bregman divergences was constructed by **Banerjee et al. [514]**.

Hierarchical clustering techniques also have a long history. Much of the initial activity was in the area of taxonomy and is covered in books by **Jardine and Sibson [538]** and **Sneath and Sokal [556]**. General-purpose discussions of hierarchical clustering are also available in most of the clustering books mentioned above. Agglomerative hierarchical clustering is the focus of most work in the area of hierarchical clustering, but divisive approaches have also received some attention. For example, **Zahn [561]** describes a divisive hierarchical technique that uses the minimum spanning tree of a graph. While both divisive and agglomerative approaches typically take the view that merging (splitting) decisions are final, there has been some work by **Fisher**

[528] and Karypis et al. [539] to overcome these limitations. Murtagh and Contreras provide a recent overview of hierarchical clustering algorithms [551] and have also proposed a linear time hierarchical clustering algorithm [521].

Ester et al. proposed DBSCAN [526], which was later generalized to the GDBSCAN algorithm by Sander et al. [554] in order to handle more general types of data and distance measures, such as polygons whose closeness is measured by the degree of intersection. An incremental version of DBSCAN was developed by Kriegel et al. [525]. One interesting outgrowth of DBSCAN is OPTICS (Ordering Points To Identify the Clustering Structure) (Ankerst et al. [510]), which allows the visualization of cluster structure and can also be used for hierarchical clustering. A recent discussion of density-based clustering by Kriegel et al. [542] provides a very readable synopsis of density-based clustering and recent developments.

An authoritative discussion of cluster validity, which strongly influenced the discussion in this chapter, is provided in Chapter 4 [□](#) of Jain and Dubes' clustering book [536]. A recent review of cluster validity measures by Xiong and Li can be found in [559]. Other recent reviews of cluster validity are those of Halkidi et al. [529, 530] and Milligan [547]. Silhouette coefficients are described in Kaufman and Rousseeuw's clustering book [540]. The source of the cohesion and separation measures in Table 7.6 [□](#) is a paper by Zhao and Karypis [563], which also contains a discussion of entropy, purity, and the hierarchical F -measure. The original source of the hierarchical F -measure is an article by Larsen and Aone [543]. The CVNN measure was introduced by Li et al. [544]. An axiomatic approach to clustering validity is presented in [546]. Many of the popular indices for cluster validation are implemented in the NbClust R package, which is described in the article by Charrad et al. [519].

Bibliography

- [507] C. C. Aggarwal and C. K. Reddy, editors. *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2013.
- [508] M. S. Aldenderfer and R. K. Blashfield. *Cluster Analysis*. Sage Publications, Los Angeles, 1985.
- [509] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, December 1973.
- [510] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proc. of 1999 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 49–60, Philadelphia, Pennsylvania, June 1999. ACM Press.
- [511] P. Arabie, L. Hubert, and G. D. Soete. An overview of combinatorial data analysis. In P. Arabie, L. Hubert, and G. D. Soete, editors, *Clustering and Classification*, pages 188–217. World Scientific, Singapore, January 1996.
- [512] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

- [513] G. Ball and D. Hall. A Clustering Technique for Summarizing Multivariate Data. *Behavior Science*, 12:153–155, March 1967.
- [514] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with Bregman Divergences. In *Proc. of the 2004 SIAM Intl. Conf. on Data Mining*, pages 234–245, Lake Buena Vista, FL, April 2004.
- [515] P. Berkhin. Survey Of Clustering Data Mining Techniques. Technical report, Accrue Software, San Jose, CA, 2002.
- [516] H.-H. Bock. Origins and extensions of the-means algorithm in cluster analysis. *Journal Électronique d'Histoire des Probabilités et de la Statistique [electronic only]*, 4(2):Article–14, 2008.
- [517] D. Boley. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.
- [518] P. S. Bradley and U. M. Fayyad. Refining Initial Points for K-Means Clustering. In *Proc. of the 15th Intl. Conf. on Machine Learning*, pages 91–99, Madison, WI, July 1998. Morgan Kaufmann Publishers Inc.
- [519] M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs. NbClust: an R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software*, 61(6):1–36, 2014.
- [520] CLUTO 2.1.2: Software for Clustering High-Dimensional Datasets. www.cs.umn.edu/~karypis, October 2016.

- [521] P. Contreras and F. Murtagh. Fast, linear time hierarchical clustering using the Baire metric. *Journal of classification*, 29(2):118–143, 2012.
- [522] I. S. Dhillon, Y. Guan, and J. Kogan. Iterative Clustering of High Dimensional Text Data Augmented by Local Search. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 131–138. IEEE Computer Society, 2002.
- [523] I. S. Dhillon and D. S. Modha. Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1/2):143–175, 2001.
- [524] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, second edition, 2001.
- [525] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental Clustering for Mining in a Data Warehousing Environment. In *Proc. of the 24th VLDB Conf.*, pages 323–333, New York City, August 1998. Morgan Kaufmann.
- [526] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of the 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, August 1996. AAAI Press.
- [527] B. S. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. Arnold Publishers, London, 4th edition, May 2001.

- [528] D. Fisher. Iterative Optimization and Simplification of Hierarchical Clusterings. *Journal of Artificial Intelligence Research*, 4:147–179, 1996.
- [529] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster validity methods: part I. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 31(2):40–45, June 2002.
- [530] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering validity checking methods: part II. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 31 (3):19–27, Sept. 2002.
- [531] G. Hamerly and C. Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proc. of the 11th Intl. Conf. on Information and Knowledge Management*, pages 600–607, McLean, Virginia, 2002. ACM Press.
- [532] J. Han, M. Kamber, and A. Tung. Spatial Clustering Methods in Data Mining: A review. In H. J. Miller and J. Han, editors, *Geographic Data Mining and Knowledge Discovery*, pages 188–217. Taylor and Francis, London, December 2001.
- [533] J. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.
- [534] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, New York, 2001.

- [535] A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern recognition letters*, 31 (8):651–666, 2010.
- [536] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series. Prentice Hall, March 1988.
- [537] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999.
- [538] N. Jardine and R. Sibson. *Mathematical Taxonomy*. Wiley, New York, 1971.
- [539] G. Karypis, E.-H. Han, and V. Kumar. Multilevel Refinement for Hierarchical Clustering. Technical Report TR 99-020, University of Minnesota, Minneapolis, MN, 1999.
- [540] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. John Wiley and Sons, New York, November 1990.
- [541] J. M. Kleinberg. An Impossibility Theorem for Clustering. In *Proc. of the 16th Annual Conf. on Neural Information Processing Systems*, December, 9–14 2002.
- [542] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

- [543] B. Larsen and C. Aone. Fast and Effective Text Mining Using Linear-Time Document Clustering. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 16–22, San Diego, California, 1999. ACM Press.
- [544] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, and S. Wu. Understanding and enhancement of internal clustering validation measures. *Cybernetics, IEEE Transactions on*, 43(3): 982–994, 2013.
- [545] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symp. on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.
- [546] M. Meilă. Comparing Clusterings: An Axiomatic View. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 577–584, New York, NY, USA, 2005. ACM.
- [547] G. W. Milligan. Clustering Validation: Results and Implications for Applied Analyses. In P. Arabie, L. Hubert, and G. D. Soete, editors, *Clustering and Classification*, pages 345–375. World Scientific, Singapore, January 1996.
- [548] B. Mirkin. *Mathematical Classification and Clustering*, volume 11 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, August 1996.

- [549] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [550] F. Murtagh. *Multidimensional Clustering Algorithms*. Physica-Verlag, Heidelberg and Vienna, 1985.
- [551] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.
- [552] D. Pelleg and A. W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proc. of the 17th Intl. Conf. on Machine Learning*, pages 727–734. Morgan Kaufmann, San Francisco, CA, 2000.
- [553] C. Romesburg. *Cluster Analysis for Researchers*. Life Time Learning, Belmont, CA, 1984.
- [554] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.
- [555] S. M. Savarese and D. Boley. A comparative analysis on the bisecting K-means and the PDDP clustering algorithms. *Intelligent Data Analysis*, 8(4):345–362, 2004.
- [556] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy*. Freeman, San Francisco, 1971.

[557] H. Späth. *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, volume 4 of *Computers and Their Application*. Ellis Horwood Publishers, Chichester, 1980. ISBN 0-85312-141-9.

[558] M. Steinbach, G. Karypis, and V. Kumar. A Comparison of Document Clustering Techniques. In *Proc. of KDD Workshop on Text Mining, Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, Boston, MA, August 2000.

[559] H. Xiong and Z. Li. Clustering Validation Measures. In C. C. Aggarwal and C. K. Reddy, editors, *Data Clustering: Algorithms and Applications*, pages 571–605. Chapman & Hall/CRC, 2013.

[560] R. Xu, D. Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[561] C. T. Zahn. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers*, C-20(1):68–86, Jan. 1971.

[562] B. Zhang, M. Hsu, and U. Dayal. K-Harmonic Means—A Data Clustering Algorithm. Technical Report HPL-1999-124, Hewlett Packard Laboratories, Oct. 29 1999.

[563] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.

7.7 Exercises

1. Consider a data set consisting of 220 data vectors, where each vector has 32 components and each component is a 4-byte value. Suppose that vector quantization is used for compression, and that 216 prototype vectors are used. How many bytes of storage does that data set take before and after compression and what is the compression ratio?
2. Find all well-separated clusters in the set of points shown in [Figure 7.35](#).



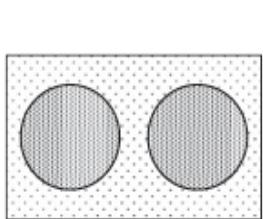
Figure 7.35.
Points for [Exercise 2](#).

3. Many partitional clustering algorithms that automatically determine the number of clusters claim that this is an advantage. List two situations in which this is not the case.
4. Given K equally sized clusters, the probability that a randomly chosen initial centroid will come from any given cluster is $1/K$, but the probability that each cluster will have exactly one initial centroid is much lower. (It should be clear that having one initial centroid in each cluster is a good starting situation for K-means.) In general, if there are K clusters and each cluster has n points, then the probability, p , of selecting in a sample of size K one initial centroid from each cluster is given by [Equation 7.20](#). (This assumes sampling with

replacement.) From this formula we can calculate, for example, that the chance of having one initial centroid from each of four clusters is $4!/44=0.0938$.

p=number of ways to select one centroid from each cluster
number of ways { $\binom{4}{1} \times \binom{3}{1} \times \binom{2}{1} \times \binom{1}{1}$ }

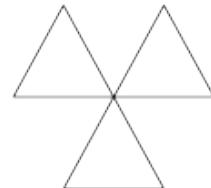
- a. Plot the probability of obtaining one point from each cluster in a sample of size K for values of K between 2 and 100.
 - b. For K clusters, $K=10, 100$, and 1000 , find the probability that a sample of size $2K$ contains at least one point from each cluster. You can use either mathematical methods or statistical simulation to determine the answer.
5. Identify the clusters in [Figure 7.36](#) using the center-, contiguity-, and density-based definitions. Also indicate the number of clusters for each case and give a brief indication of your reasoning. Note that darkness or the number of dots indicates density. If it helps, assume center-based means K-means, contiguity-based means single link, and density-based means DBSCAN.



(a)



(b)



(c)



(d)

Figure 7.36.
Clusters for [Exercise 5](#).

6. For the following sets of two-dimensional points, (1) provide a sketch of how they would be split into clusters by K-means for the given number of

clusters and (2) indicate approximately where the resulting centroids would be. Assume that we are using the squared error objective function. If you think that there is more than one possible solution, then please indicate whether each solution is a global or local minimum. Note that the label of each diagram in [Figure 7.37](#) matches the corresponding part of this question, e.g., [Figure 7.37\(a\)](#) goes with part (a).

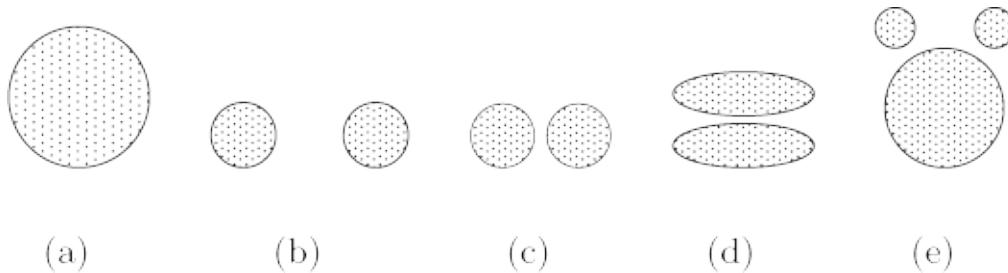


Figure 7.37.

Diagrams for [Exercise 6](#).

- a. K=2. Assuming that the points are uniformly distributed in the circle, how many possible ways are there (in theory) to partition the points into two clusters? What can you say about the positions of the two centroids? (Again, you don't need to provide exact centroid locations, just a qualitative description.)
- b. K=3. The distance between the edges of the circles is slightly greater than the radii of the circles.
- c. K=3. The distance between the edges of the circles is much less than the radii of the circles.
- d. K=2.
- e. K=3. Hint: Use the symmetry of the situation and remember that we are looking for a rough sketch of what the result would be.

7. Suppose that for a data set

- there are m points and K clusters,
- half the points and clusters are in “more dense” regions,
- half the points and clusters are in “less dense” regions, and
- the two regions are well-separated from each other.

For the given data set, which of the following should occur in order to minimize the squared error when finding K clusters:

- a. Centroids should be equally distributed between more dense and less dense regions.
- b. More centroids should be allocated to the less dense region.
- c. More centroids should be allocated to the denser region.

Note: Do not get distracted by special cases or bring in factors other than density. However, if you feel the true answer is different from any given above, justify your response.

8. Consider the mean of a cluster of objects from a binary transaction data set. What are the minimum and maximum values of the components of the mean? What is the interpretation of components of the cluster mean? Which components most accurately characterize the objects in the cluster?

9. Give an example of a data set consisting of three natural clusters, for which (almost always) K-means would likely find the correct clusters, but bisecting K-means would not.

10. Would the cosine measure be the appropriate similarity measure to use with K-means clustering for time series data? Why or why not? If not, what similarity measure would be more appropriate?

11. Total SSE is the sum of the SSE for each separate attribute. What does it mean if the SSE for one variable is low for all clusters? Low for just one cluster? High for all clusters? High for just one cluster? How could you use the per variable SSE information to improve your clustering?

12. The leader algorithm ([Hartigan \[533\]](#)) represents each cluster using a point, known as a *leader*, and assigns each point to the cluster corresponding to the closest leader, unless this distance is above a user-specified threshold. In that case, the point becomes the leader of a new cluster.

- a. What are the advantages and disadvantages of the leader algorithm as compared to K-means?
- b. Suggest ways in which the leader algorithm might be improved.

13. The Voronoi diagram for a set of K points in the plane is a partition of all the points of the plane into K regions, such that every point (of the plane) is assigned to the closest point among the K specified points—see [Figure 7.38](#). What is the relationship between Voronoi diagrams and K-means clusters? What do Voronoi diagrams tell us about the possible shapes of K-means clusters?

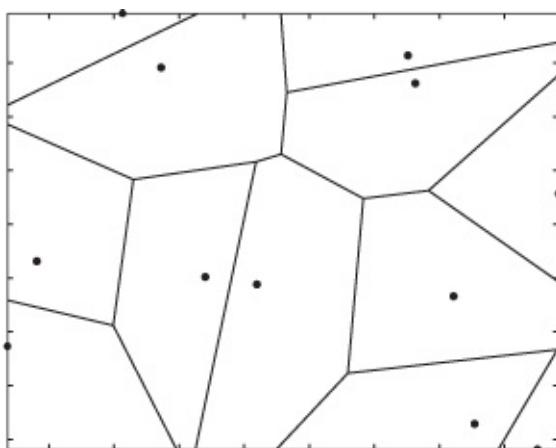


Figure 7.38.
Voronoi diagram for [Exercise 13](#).

14. You are given a data set with 100 records and are asked to cluster the data. You use K-means to cluster the data, but for all values of K , $1 \leq K \leq 100$, the K-means algorithm returns only one non-empty cluster. You then apply an incremental version of K-means, but obtain exactly the same result. How is this possible? How would single link or DBSCAN handle such data?
15. Traditional agglomerative hierarchical clustering routines merge two clusters at each step. Does it seem likely that such an approach accurately captures the (nested) cluster structure of a set of data points? If not, explain how you might postprocess the data to obtain a more accurate view of the cluster structure.
16. Use the similarity matrix in [Table 7.13](#) to perform single and complete link hierarchical clustering. Show your results by drawing a dendrogram. The dendrogram should clearly show the order in which the points are merged.

Table 7.13. Similarity matrix for Exercise 16

	p1	p2	p3	p4	p5
p1	1.00	0.10	0.41	0.55	0.35
p2	0.10	1.00	0.64	0.47	0.98
p3	0.41	0.64	1.00	0.44	0.85
p4	0.55	0.47	0.44	1.00	0.76
p5	0.35	0.98	0.85	0.76	1.00

17. Hierarchical clustering is sometimes used to generate K clusters, $K > 1$ by taking the clusters at the K th level of the dendrogram. (Root is at level 1.) By looking at the clusters produced in this way, we can evaluate the behavior of hierarchical clustering on different types of data and clusters, and also compare hierarchical approaches to K-means.

The following is a set of one-dimensional points: {6, 12, 18, 24, 30, 42, 48}.

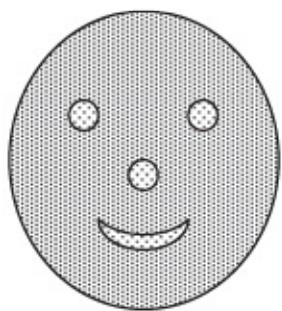
- a. For each of the following sets of initial centroids, create two clusters by assigning each point to the nearest centroid, and then calculate the total squared error for each set of two clusters. Show both the clusters and the total squared error for each set of centroids.
 - i. {18, 45}
 - ii. {15, 40}
 - b. Do both sets of centroids represent stable solutions; i.e., if the K-means algorithm was run on this set of points using the given centroids as the starting centroids, would there be any change in the clusters generated?
 - c. What are the two clusters produced by single link?
 - d. Which technique, K-means or single link, seems to produce the “most natural” clustering in this situation? (For K-means, take the clustering with the lowest squared error.)
 - e. What definition(s) of clustering does this natural clustering correspond to? (Well-separated, center-based, contiguous, or density.)
 - f. What well-known characteristic of the K-means algorithm explains the previous behavior?
18. Suppose we find K clusters using Ward’s method, bisecting K-means, and ordinary K-means. Which of these solutions represents a local or global minimum? Explain.
19. Hierarchical clustering algorithms require $O(m^2 \log(m))$ time, and consequently, are impractical to use directly on larger data sets. One possible technique for reducing the time required is to sample the data set. For example, if K clusters are desired and m points are sampled from the m

points, then a hierarchical clustering algorithm will produce a hierarchical clustering in roughly $O(m)$ time. K clusters can be extracted from this hierarchical clustering by taking the clusters on the K th level of the dendrogram. The remaining points can then be assigned to a cluster in linear time, by using various strategies. To give a specific example, the centroids of the K clusters can be computed, and then each of the $m - K$ remaining points can be assigned to the cluster associated with the closest centroid.

For each of the following types of data or clusters, discuss briefly if (1) sampling will cause problems for this approach and (2) what those problems are. Assume that the sampling technique randomly chooses points from the total set of m points and that any unmentioned characteristics of the data or clusters are as optimal as possible. In other words, focus only on problems caused by the particular characteristic mentioned. Finally, assume that K is very much less than m .

- a. Data with very different sized clusters.
- b. High-dimensional data.
- c. Data with outliers, i.e., atypical points.
- d. Data with highly irregular regions.
- e. Data with globular clusters.
- f. Data with widely different densities.
- g. Data with a small percentage of noise points.
- h. Non-Euclidean data.
- i. Euclidean data.
- j. Data with many and mixed attribute types.

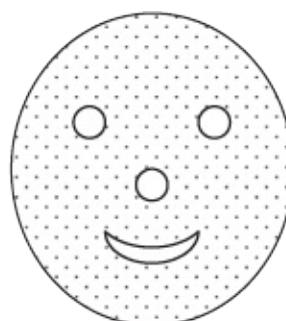
20. Consider the following four faces shown in [Figure 7.39](#). Again, darkness or number of dots represents density. Lines are used only to distinguish regions and do not represent points.



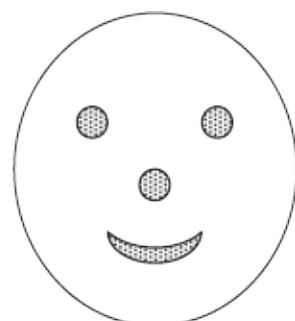
(a)



(b)



(c)



(d)

Figure 7.39.

Figure for [Exercise 20](#).

- For each figure, could you use single link to find the patterns represented by the nose, eyes, and mouth? Explain.
- For each figure, could you use K-means to find the patterns represented by the nose, eyes, and mouth? Explain.
- What limitation does clustering have in detecting all the patterns formed by the points in [Figure 7.39\(c\)](#)?

21. Compute the entropy and purity for the confusion matrix in [Table 7.14](#).

Table 7.14. Confusion matrix for Exercise 21.

Cluster	Entertainment	Financial	Foreign	Metro	National	Sports	Total
#1	1	1	0	11	4	676	693
#2	27	89	333	827	253	33	1562

#3	326	465	8	105	16	29	949
Total	354	555	341	943	273	738	3204

22. You are given two sets of 100 points that fall within the unit square. One set of points is arranged so that the points are uniformly spaced. The other set of points is generated from a uniform distribution over the unit square.

- a. Is there a difference between the two sets of points?
- b. If so, which set of points will typically have a smaller SSE for K=10 clusters?
- c. What will be the behavior of DBSCAN on the uniform data set? The random data set?

23. Using the data in [Exercise 24](#), compute the silhouette coefficient for each point, each of the two clusters, and the overall clustering.

24. Given the set of cluster labels and similarity matrix shown in [Tables 7.15](#) and [7.16](#), respectively, compute the correlation between the similarity matrix and the ideal similarity matrix, i.e., the matrix whose i,j th entry is 1 if two objects belong to the same cluster, and 0 otherwise.

Table 7.15. Table of cluster labels for Exercise 24.

Point	Cluster Label
P1	1
P2	1
P3	2
P4	2

Table 7.16. Similarity matrix for Exercise 24

Point	P1	P2	P3	P4
P1	1	0.8	0.65	0.55
P2	0.8	1	0.7	0.6
P3	0.65	0.7	1	0.9
P4	0.55	0.6	0.9	1

25. Compute the hierarchical F -measure for the eight objects $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ and hierarchical clustering shown in [Figure 7.40](#). Class A contains points p_1, p_2 , and p_3 , while p_4, p_5, p_6, p_7 , and p_8 belong to class B.

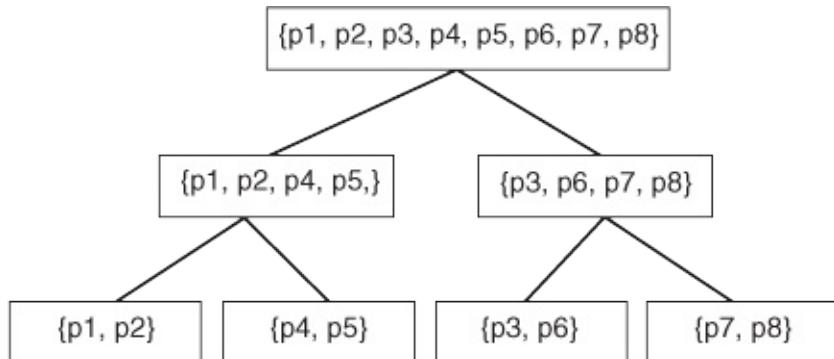


Figure 7.40.
Hierarchical clustering for [Exercise 25](#).

26. Compute the cophenetic correlation coefficient for the hierarchical clusterings in [Exercise 16](#). (You will need to convert the similarities into dissimilarities.)

27. Prove [Equation 7.14](#).

28. Prove [Equation 7.16](#).

29. Prove that $\sum_{i=1}^K \sum_{x \in C_i} (x - m_i)(m - m_i) = 0$. This fact was used in the proof that $TSS = SSE + SSB$ in [Section 7.5.2](#).

30. Clusters of documents can be summarized by finding the top terms (words) for the documents in the cluster, e.g., by taking the most frequent k terms, where k is a constant, say 10, or by taking all terms that occur more frequently than a specified threshold. Suppose that K-means is used to find clusters of both documents and words for a document data set.

- a. How might a set of term clusters defined by the top terms in a document cluster differ from the word clusters found by clustering the terms with K-means?
- b. How could term clustering be used to define clusters of documents?

31. We can represent a data set as a collection of object nodes and a collection of attribute nodes, where there is a link between each object and each attribute, and where the weight of that link is the value of the object for that attribute. For sparse data, if the value is 0, the link is omitted. Bipartite clustering attempts to partition this graph into disjoint clusters, where each cluster consists of a set of object nodes and a set of attribute nodes. The objective is to maximize the weight of links between the object and attribute nodes of a cluster, while minimizing the weight of links between object and attribute links in different clusters. This type of clustering is also known as **co-clustering** because the objects and attributes are clustered at the same time.

- a. How is bipartite clustering (co-clustering) different from clustering the sets of objects and attributes separately?
- b. Are there any cases in which these approaches yield the same clusters?
- c. What are the strengths and weaknesses of co-clustering as compared to ordinary clustering?

32. In **Figure 7.41**, match the similarity matrices, which are sorted according to cluster labels, with the sets of points. Differences in shading and marker shape distinguish between clusters, and each set of points contains 100 points and three clusters. In the set of points labeled 2, there are three very tight, equalsized clusters.

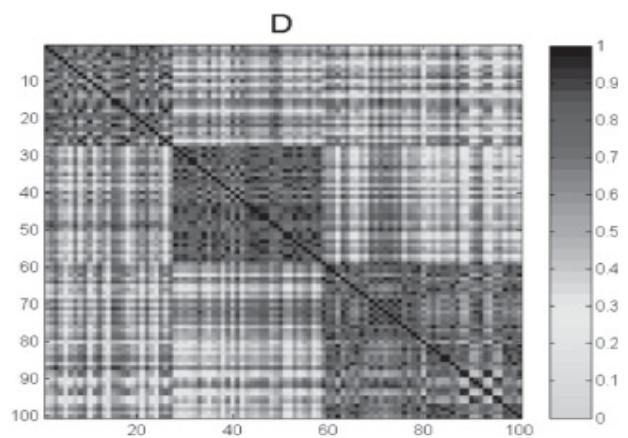
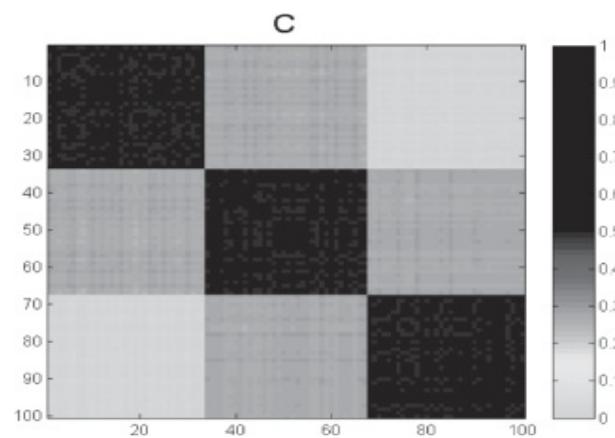
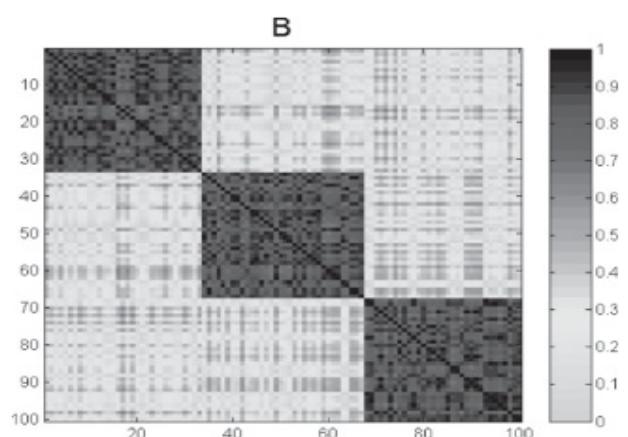
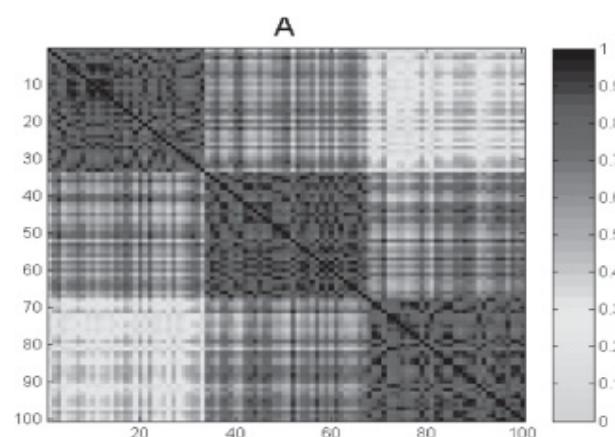
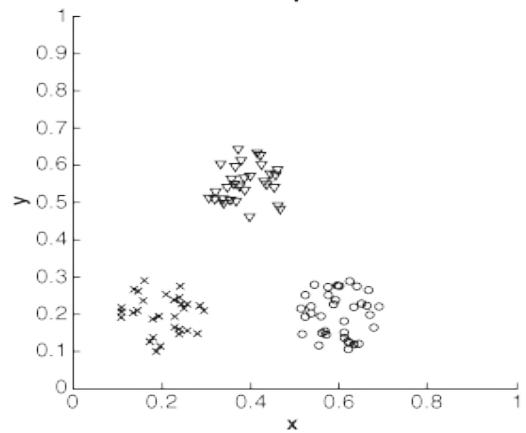
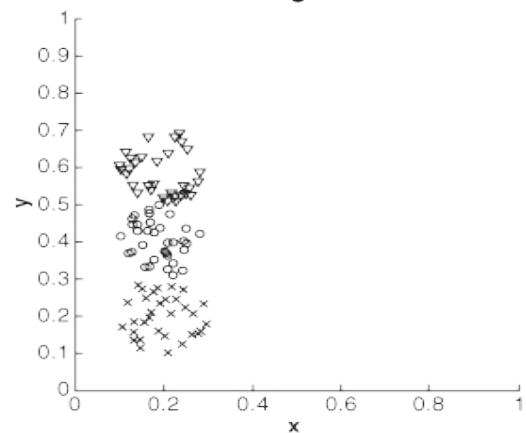
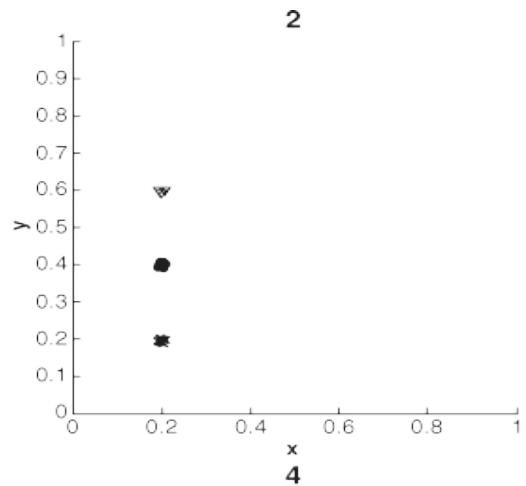
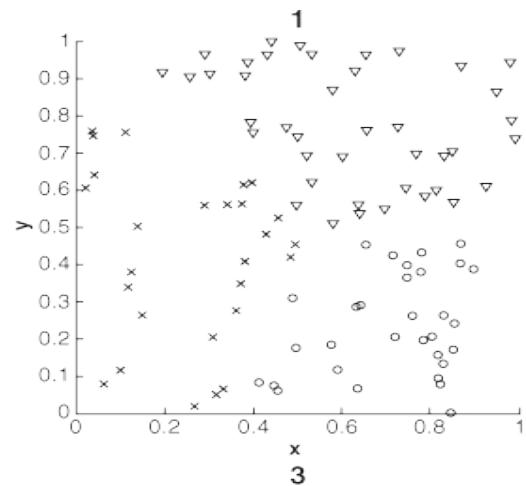


Figure 7.41.

Points and similarity matrices for [Exercise 32](#).

8 Cluster Analysis: Additional Issues and Algorithms

A large number of clustering algorithms have been developed in a variety of domains for different types of applications. No algorithm is suitable for all types of data, clusters, and applications. In fact, it seems that there is always room for a new clustering algorithm that is more efficient or better suited to a particular type of data, cluster, or application. Instead, we can only claim that we have techniques that work well in some situations. The reason is that, in many cases, what constitutes a good set of clusters is open to subjective interpretation. Furthermore, when an objective measure is employed to give a precise definition of a cluster, the problem of finding the optimal clustering is often computationally infeasible.

This chapter focuses on important issues in cluster analysis and explores the concepts and approaches that have been developed to address them. We begin with a discussion of the key issues of cluster analysis, namely, the characteristics of data, clusters, and algorithms that strongly impact clustering. These issues

are important for understanding, describing, and comparing clustering techniques, and provide the basis for deciding which technique to use in a specific situation. For example, many clustering algorithms have a time or space complexity of $O(m^2)$ (m being the number of objects) and, thus, are not suitable for large data sets. We then discuss additional clustering techniques. For each technique, we describe the algorithm, including the issues it addresses and the methods that it uses to address them. We conclude this chapter by providing some general guidelines for selecting a clustering algorithm for a given application.

8.1 Characteristics of Data, Clusters, and Clustering Algorithms

This section explores issues related to the characteristics of data, clusters, and algorithms that are important for a broad understanding of cluster analysis. Some of these issues represent challenges, such as handling noise and outliers. Other issues involve a desired feature of an algorithm, such as an ability to produce the same result regardless of the order in which the data objects are processed. The discussion in this section, along with the discussion of different types of clusterings in [Section 7.1.2](#) and different types of clusters in [Section 7.1.3](#), identifies a number of “dimensions” that can be used to describe and compare various clustering algorithms and the clustering results that they produce. To illustrate this, we begin this section with an example that compares two clustering algorithms that were described in the previous chapter, DBSCAN and K-means. This is followed by a more detailed description of the characteristics of data, clusters, and algorithms that impact cluster analysis.

8.1.1 Example: Comparing K-means and DBSCAN

To simplify the comparison, we assume that there are no ties in distances for either K-means or DBSCAN and that DBSCAN always assigns a border point that is associated with several core points to the closest core point.

- Both DBSCAN and K-means are partitional clustering algorithms that assign each object to a single cluster, but K-means typically clusters all the objects, while DBSCAN discards objects that it classifies as noise.
- K-means uses a prototype-based notion of a cluster; DBSCAN uses a density-based concept.
- DBSCAN can handle clusters of different sizes and shapes and is not strongly affected by noise or outliers. K-means has difficulty with non-globular clusters and clusters of different sizes. Both algorithms can perform poorly when clusters have widely differing densities.
- K-means can only be used for data that has a well-defined centroid, such as a mean or median. DBSCAN requires that its definition of density, which is based on the traditional Euclidean notion of density, be meaningful for the data.
- K-means can be applied to sparse, high-dimensional data, such as document data. DBSCAN typically performs poorly for such data because the traditional Euclidean definition of density does not work well for high-dimensional data.
- The original versions of K-means and DBSCAN were designed for Euclidean data, but both have been extended to handle other types of data.
- DBSCAN makes no assumption about the distribution of the data. The basic K-means algorithm is equivalent to a statistical clustering approach (mixture models) that assumes all clusters come from spherical Gaussian distributions with different means but the same covariance matrix. See [Section 8.2.2](#).
- DBSCAN and K-means both look for clusters using all attributes, that is, they do not look for clusters that involve only a subset of the attributes.
- K-means can find clusters that are not well separated, even if they overlap (see [Figure 7.2\(b\)](#)), but DBSCAN merges clusters that overlap.
- The K-means algorithm has a time complexity of $O(m)$, while DBSCAN takes $O(m^2)$ time, except for special cases such as low-dimensional

Euclidean data.

- DBSCAN produces the same set of clusters from one run to another, while K-means, which is typically used with random initialization of centroids, does not.
- DBSCAN automatically determines the number of clusters; for K-means, the number of clusters needs to be specified as a parameter. However, DBSCAN has two other parameters that must be specified, *Eps* and *MinPts*.
- K-means clustering can be viewed as an optimization problem; i.e., minimize the sum of the squared error of each point to its closest centroid, and as a specific case of a statistical clustering approach (mixture models). DBSCAN is not based on any formal model.

8.1.2 Data Characteristics

The following are some characteristics of data that can strongly affect cluster analysis.

High Dimensionality In high-dimensional data sets, the traditional Euclidean notion of density, which is the number of points per unit volume, becomes meaningless. To see this, consider that as the number of dimensions increases, the volume increases rapidly, and unless the number of points grows exponentially with the number of dimensions, the density tends to 0. (Volume is exponential in the number of dimensions. For instance, a hypersphere with radius, r , and dimension, d , has volume proportional to rd .) Also, proximity tends to become more uniform in high-dimensional spaces. Another way to view this fact is that there are more dimensions (attributes) that contribute to the proximity between two points and this tends to make the proximity more uniform. Since most clustering techniques are based on

proximity or density, they can often have difficulty with high-dimensional data. One way to address such problems is to employ dimensionality reduction techniques. Another approach, as discussed in [Sections 8.4.6](#) and [8.4.8](#), is to redefine the notions of proximity and density.

Size Many clustering algorithms that work well for small or medium-size data sets are unable to handle larger data sets. This is addressed further in the discussion of the characteristics of clustering algorithms—scalability is one such characteristic—and in [Section 8.5](#), which discusses scalable clustering algorithms.

Sparseness Sparse data often consists of asymmetric attributes, where zero values are not as important as non-zero values. Therefore, similarity measures appropriate for asymmetric attributes are commonly used. However, other, related issues also arise. For example, are the magnitudes of non-zero entries important, or do they distort the clustering? In other words, does the clustering work best when there are only two values, 0 and 1?

Noise and Outliers An atypical point (outlier) can often severely degrade the performance of clustering algorithms, especially algorithms such as K-means that are prototype-based. On the other hand, noise can cause techniques, such as single link, to join clusters that should not be joined. In some cases, algorithms for removing noise and outliers are applied before a clustering algorithm is used. Alternatively, some algorithms can detect points that represent noise and outliers during the clustering process and then delete them or otherwise eliminate their negative effects. In the previous chapter, for instance, we saw that DBSCAN automatically classifies low-density points as noise and removes them from the clustering process. Chameleon ([Section 8.4.4](#)), SNN density-based clustering ([Section 8.4.9](#)), and CURE ([Section 8.5.3](#)) are three of the algorithms in this chapter that explicitly deal with noise and outliers during the clustering process.

Type of Attributes and Data Set As discussed in [Chapter 2](#), data sets can be of various types, such as structured, graph, or ordered, while attributes are usually categorical (nominal or ordinal) or quantitative (interval or ratio), and are binary, discrete, or continuous. Different proximity and density measures are appropriate for different types of data. In some situations, data needs to be discretized or binarized so that a desired proximity measure or clustering algorithm can be used. Another complication occurs when attributes are of widely differing types, e.g., continuous and nominal. In such cases, proximity and density are more difficult to define and often more ad hoc. Finally, special data structures and algorithms are often needed to handle certain types of data efficiently.

Scale Different attributes, e.g., height and weight, are often measured on different scales. These differences can strongly affect the distance or similarity between two objects and, consequently, the results of a cluster analysis. Consider clustering a group of people based on their heights, which are measured in meters, and their weights, which are measured in kilograms. If we use Euclidean distance as our proximity measure, then height will have little impact and people will be clustered mostly based on the weight attribute. If, however, we standardize each attribute by subtracting off its mean and dividing by its standard deviation, then we will have eliminated effects due to the difference in scale. More generally, normalization techniques, such as those discussed in [Section 2.3.7](#), are typically used to handle these issues.

Mathematical Properties of the Data Space Some clustering techniques calculate the mean of a collection of points or use other mathematical operations that only make sense in Euclidean space or in other specific data spaces. Other algorithms require that the definition of density be meaningful for the data.

8.1.3 Cluster Characteristics

The different types of clusters, such as prototype-, graph-, and density-based, were described earlier in [Section 7.1.3](#). Here, we describe other important characteristics of clusters.

Data Distribution Some clustering techniques assume a particular type of distribution for the data. More specifically, they often assume that data can be modeled as arising from a mixture of distributions, where each cluster corresponds to a distribution. Clustering based on mixture models is discussed in [Section 8.2.2](#).

Shape Some clusters are regularly shaped, e.g., rectangular or globular, but in general, clusters can be of arbitrary shape. Techniques such as DBSCAN and single link can handle clusters of arbitrary shape, but prototype-based schemes and some hierarchical techniques, such as complete link and group average, cannot. Chameleon ([Section 8.4.4](#)) and CURE ([Section 8.5.3](#)) are examples of techniques that were specifically designed to address this problem.

Differing Sizes Many clustering methods, such as K-means, don't work well when clusters have different sizes. (See [Section 7.2.4](#).) This topic is discussed further in [Section 8.6](#).

Differing Densities Clusters that have widely varying density can cause problems for methods such as DBSCAN and K-means. The SNN density-based clustering technique presented in [Section 8.4.9](#) addresses this issue.

Poorly Separated Clusters When clusters touch or overlap, some clustering techniques combine clusters that should be kept separate. Even techniques that find distinct clusters arbitrarily assign points to one cluster or another. Fuzzy clustering, which is described in [Section 8.2.1](#), is one technique for dealing with data that does not form well-separated clusters.

Relationships among Clusters In most clustering techniques, there is no explicit consideration of the relationships between clusters, such as their relative position. Self-organizing maps (SOM), which are described in [Section 8.2.3](#), are a clustering technique that directly considers the relationships between clusters during the clustering process. Specifically, the assignment of a point to one cluster affects the definitions of nearby clusters.

Subspace Clusters Clusters may only exist in a subset of dimensions (attributes), and the clusters determined using one set of dimensions are frequently quite different from the clusters determined by using another set. While this issue can arise with as few as two dimensions, it becomes more acute as dimensionality increases, because the number of possible subsets of dimensions is exponential in the total number of dimensions. For that reason, it is not feasible to simply look for clusters in all possible subsets of dimensions unless the number of dimensions is relatively low.

One approach is to apply feature selection, which was discussed in [Section 2.3.4](#). However, this approach assumes that there is only one subset of dimensions in which the clusters exist. In reality, clusters can exist in many distinct subspaces (sets of dimensions), some of which overlap. [Section 8.3.2](#) considers techniques that address the general problem of subspace clustering, i.e., of finding both clusters and the dimensions they span.

8.1.4 General Characteristics of Clustering Algorithms

Clustering algorithms are quite varied. We provide a general discussion of important characteristics of clustering algorithms here, and make more specific comments during our discussion of particular techniques.

Order Dependence For some algorithms, the quality and number of clusters produced can vary, perhaps dramatically, depending on the order in which the data is processed. While it would seem desirable to avoid such algorithms, sometimes the order dependence is relatively minor or the algorithm has other desirable characteristics. SOM ([Section 8.2.3](#)) is an example of an algorithm that is order dependent.

Nondeterminism Clustering algorithms, such as K-means, are not order-dependent, but they produce different results for each run because they rely on an initialization step that requires a random choice. Because the quality of the clusters can vary from one run to another, multiple runs can be necessary.

Scalability It is not unusual for a data set to contain millions of objects, and the clustering algorithms used for such data sets should have linear or near-linear time and space complexity. Even algorithms that have a complexity of $O(m^2)$ are not practical for large data sets. Furthermore, clustering techniques for data sets cannot always assume that all the data will fit in main memory or that data elements can be randomly accessed. Such algorithms are infeasible for large data sets. [Section 8.5](#) is devoted to the issue of scalability.

Parameter Selection Most clustering algorithms have one or more parameters that need to be set by the user. It can be difficult to choose the

proper values; thus, the attitude is usually, “the fewer parameters, the better.” Choosing parameter values becomes even more challenging if a small change in the parameters drastically changes the clustering results. Finally, unless a procedure (which might involve user input) is provided for determining parameter values, a user of the algorithm is reduced to using trial and error to find suitable parameter values.

Perhaps the most well-known parameter selection problem is that of “choosing the right number of clusters” for partitional clustering algorithms, such as K-means. One possible approach to that issue is given in [Section 7.5.5](#), while references to others are provided in the Bibliographic Notes.

Transforming the Clustering Problem to Another Domain One approach taken by some clustering techniques is to map the clustering problem to a problem in a different domain. Graph-based clustering, for instance, maps the task of finding clusters to the task of partitioning a proximity graph into connected components.

Treating Clustering as an Optimization Problem Clustering is often viewed as an optimization problem: divide the points into clusters in a way that maximizes the goodness of the resulting set of clusters as measured by a user-specified objective function. For example, the K-means clustering algorithm ([Section 7.2](#)) tries to find the set of clusters that minimizes the sum of the squared distance of each point from its closest cluster centroid. In theory, such problems can be solved by enumerating all possible sets of clusters and selecting the one with the best value of the objective function, but this exhaustive approach is computationally infeasible. For this reason, many clustering techniques are based on heuristic approaches that produce good, but not optimal clusterings. Another approach is to use objective functions on a greedy or local basis. In particular, the hierarchical clustering techniques

discussed in [Section 7.3](#) proceed by making locally optimal (greedy) decisions at each step of the clustering process.

Road Map

We arrange our discussion of clustering algorithms in a manner similar to that of the previous chapter, grouping techniques primarily according to whether they are prototype-based, density-based, or graph-based. There is, however, a separate discussion for scalable clustering techniques. We conclude this chapter with a discussion of how to choose a clustering algorithm.

8.2 Prototype-Based Clustering

In prototype-based clustering, a cluster is a set of objects in which any object is closer to the prototype that defines the cluster than to the prototype of any other cluster. [Section 7.2](#) described K-means, a simple prototype-based clustering algorithm that uses the centroid of the objects in a cluster as the prototype of the cluster. This section discusses clustering approaches that expand on the concept of prototype-based clustering in one or more ways, as discussed next:

- Objects are allowed to belong to more than one cluster. More specifically, an object belongs to every cluster with some weight. Such an approach addresses the fact that some objects are equally close to several cluster prototypes.
- A cluster is modeled as a statistical distribution, i.e., objects are generated by a random process from a statistical distribution that is characterized by a number of statistical parameters, such as the mean and variance. This viewpoint generalizes the notion of a prototype and enables the use of well-established statistical techniques.
- Clusters are constrained to have fixed relationships. Most commonly, these relationships are constraints that specify neighborhood relationships; i.e., the degree to which two clusters are neighbors of each other. Constraining the relationships among clusters can simplify the interpretation and visualization of the data.

We consider three specific clustering algorithms to illustrate these extensions of prototype-based clustering. Fuzzy c-means uses concepts from the field of fuzzy logic and fuzzy set theory to propose a clustering scheme, which is much like K-means, but which does not require a hard assignment of a point

to only one cluster. Mixture model clustering takes the approach that a set of clusters can be modeled as a mixture of distributions, one for each cluster. The clustering scheme based on Self-Organizing Maps (SOM) performs clustering within a framework that requires clusters to have a prespecified relationship to one another, e.g., a two-dimensional grid structure.

8.2.1 Fuzzy Clustering

If data objects are distributed in well-separated groups, then a crisp classification of the objects into disjoint clusters seems like an ideal approach. However, in most cases, the objects in a data set cannot be partitioned into well-separated clusters, and there will be a certain arbitrariness in assigning an object to a particular cluster. Consider an object that lies near the boundary of two clusters, but is slightly closer to one of them. In many such cases, it might be more appropriate to assign a weight to each object and each cluster that indicates the degree to which the object belongs to the cluster. Mathematically, w_{ij} is the weight with which object x_i belongs to cluster C_j .

As shown in the next section, probabilistic approaches can also provide such weights. While probabilistic approaches are useful in many situations, there are times when it is difficult to determine an appropriate statistical model. In such cases, non-probabilistic clustering techniques are needed to provide similar capabilities. Fuzzy clustering techniques are based on fuzzy set theory and provide a natural technique for producing a clustering in which membership weights (the w_{ij}) have a natural (but not probabilistic) interpretation. This section describes the general approach of fuzzy clustering and provides a specific example in terms of fuzzy c-means (fuzzy K-means).

Fuzzy Sets

Lotfi Zadeh introduced **fuzzy set theory** and **fuzzy logic** in 1965 as a way of dealing with imprecision and uncertainty. Briefly, fuzzy set theory allows an object to belong to a set with a degree of membership between 0 and 1, while fuzzy logic allows a statement to be true with a degree of certainty between 0 and 1. Traditional set theory and logic are special cases of their fuzzy counterparts that restrict the degree of set membership or the degree of certainty to be either 0 or 1. Fuzzy concepts have been applied to many different areas, including control systems, pattern recognition, and data analysis (classification and clustering).

Consider the following example of fuzzy logic. The degree of truth of the statement “It is cloudy” can be defined to be the percentage of cloud cover in the sky, e.g., if the sky is 50% covered by clouds, then we would assign “It is cloudy” a degree of truth of 0.5. If we have two sets, “cloudy days” and “non-cloudy days,” then we can similarly assign each day a degree of membership in the two sets. Thus, if a day were 25% cloudy, it would have a 25% degree of membership in “cloudy days” and a 75% degree of membership in “non-cloudy days.”

Fuzzy Clusters

Assume that we have a set of data points $X=\{x_1, \dots, x_m\}$, where each point, x_i , is an n -dimensional point, i.e., $x_i=(x_{i1}, \dots, x_{in})$. A collection of fuzzy clusters, C_1, C_2, \dots, C_k is a subset of all possible fuzzy subsets of X . (This simply means that the membership weights (degrees), w_{ij} , have been assigned values between 0 and 1 for each point, x_i , and each cluster, C_j .) However, we also want to impose the following reasonable conditions on the clusters in order to ensure that the clusters form what is called a **fuzzy pseudo-partition**.

1. All the weights for a given point, x_i , add up to 1.

$$\sum_{j=1}^k w_{ij} = 1$$

2. Each cluster, C_j , contains, with non-zero weight, at least one point, but does not contain, with a weight of one, all of the points.

$$0 < \sum_{i=1}^m w_{ij} < m$$

Fuzzy c-means

While there are many types of fuzzy clustering—indeed, many data analysis algorithms can be “fuzzified”—we only consider the fuzzy version of K-means, which is called fuzzy c-means. In the clustering literature, the version of K-means that does not use incremental updates of cluster centroids is sometimes referred to as **c-means**, and this was the term adapted by the fuzzy community for the fuzzy version of K-means. The fuzzy c-means algorithm, also sometimes known as FCM, is given by [Algorithm 8.1](#).

Algorithm 8.1 Basic fuzzy c-means

algorithm.

- 1: Select an initial fuzzy pseudo-partition, i.e., assign values to all the w_{ij} .
- 2: **repeat**
- 3: Compute the centroid of each cluster using the fuzzy pseudo-partition.
- 4: Recompute the fuzzy pseudo-partition, i.e., the w_{ij} .
- 5: **until** The centroids don't change.
(Alternative stopping conditions are “if the change in the error is below a specified threshold” or “if the absolute change in any w_{ij} is below a given threshold.”)

After initialization, FCM repeatedly computes the centroids of each cluster and the fuzzy pseudo-partition until the partition does not change. FCM is similar in structure to the K-means algorithm, which after initialization, alternates between a step that updates the centroids and a step that assigns each object to the closest centroid. Specifically, computing a fuzzy pseudo-partition is equivalent to the assignment step. As with K-means, FCM can be interpreted as attempting to minimize the sum of the squared error (SSE), although FCM is based on a fuzzy version of SSE. Indeed, K-means can be regarded as a special case of FCM and the behavior of the two algorithms is quite similar. The details of FCM are described below.

Computing SSE

The definition of the sum of the squared error (SSE) is modified as follows:

$$\text{SSE}(C_1, C_2, \dots, C_k) = \sum_{j=1}^k \sum_{i=1}^m w_{ij} p^{\text{dist}}(x_i, c_j)^2 \quad (8.1)$$

where c_j is the centroid of the j th cluster and p , which is the exponent that determines the influence of the weights, has a value between 1 and ∞ . Note that this SSE is just a weighted version of the traditional K-means SSE given in [Equation 7.1](#).

Initialization

Random initialization is often used. In particular, weights are chosen randomly, subject to the constraint that the weights associated with any object must sum to 1. As with K-means, random initialization is simple, but often results in a clustering that represents a local minimum in terms of the SSE. [Section 7.2.1](#), which contains a discussion on choosing initial centroids for K-means, has considerable relevance for FCM as well.

Computing Centroids

The definition of the centroid given in [Equation 8.2](#) can be derived by finding the centroid that minimizes the fuzzy SSE as given by [Equation 8.1](#). (See the approach in [Section 7.2.6](#).) For a cluster, C_j , the corresponding centroid, c_j , is defined by the following equation:

$$c_j = \sum_{i=1}^m w_{ij} p_{xi} / \sum_{i=1}^m w_{ij} p \quad (8.2)$$

The fuzzy centroid definition is similar to the traditional definition except that all points are considered (any point can belong to any cluster, at least somewhat) and the contribution of each point to the centroid is weighted by its membership degree. In the case of traditional crisp sets, where all w_{ij} are either 0 or 1, this definition reduces to the traditional definition of a centroid.

There are a few considerations when choosing the value of p . Choosing $p=2$ simplifies the weight update formula—see [Equation 8.4](#). However, if p is chosen to be near 1, then fuzzy c-means behaves like traditional K-means. Going in the other direction, as p gets larger, all the cluster centroids approach the global centroid of all the data points. In other words, the partition becomes fuzzier as p increases.

Updating the Fuzzy Pseudo-partition

Because the fuzzy pseudo-partition is defined by the weight, this step involves updating the weights w_{ij} associated with the i th point and j th cluster. The weight update formula given in [Equation 8.3](#) can be derived by minimizing the SSE of [Equation 8.1](#) subject to the constraint that the weights sum to 1.

$$w_{ij} = (1/\text{dist}(x_i, c_j)^2)^{1/p-1} / \sum_{q=1}^k (1/\text{dist}(x_i, c_q)^2)^{1/p-1} \quad (8.3)$$

This formula might appear a bit mysterious. However, note that if $p=2$, then we obtain [Equation 8.4](#), which is somewhat simpler. We provide an intuitive explanation of [Equation 8.4](#), which, with a slight modification, also applies to [Equation 8.3](#).

$$w_{ij} = \frac{1}{\text{dist}(x_i, c_j)^2} / \sum_{q=1}^k \frac{1}{\text{dist}(x_i, c_q)^2} \quad (8.4)$$

Intuitively, the weight w_{ij} , which indicates the degree of membership of point x_i in cluster C_j , should be relatively high if x_i is close to centroid c_j (if $\text{dist}(x_i, c_j)$ is low) and relatively low if x_i is far from centroid c_j (if $\text{dist}(x_i, c_j)$ is high). If $w_{ij} = 1/\text{dist}(x_i, c_j)^2$, which is the numerator of [Equation 8.4](#), then this will indeed be the case. However, the membership weights for a point will not sum to one unless they are normalized; i.e., divided by the sum of all the weights as in [Equation 8.4](#). To summarize, the membership weight of a point in a cluster is just the reciprocal of the square of the distance between the point and the cluster centroid divided by the sum of all the membership weights of the point.

Now consider the impact of the exponent $1/(p-1)$ in [Equation 8.3](#). If $p>2$, then this exponent decreases the weight assigned to clusters that are close to the point. Indeed, as p goes to infinity, the exponent tends to 0 and weights tend to the value $1/k$. On the other hand, as p approaches 1, the exponent increases the membership weights of points to which the cluster is close. As p goes to 1, the membership weight goes to 1 for the closest cluster and to 0 for all the other clusters. This corresponds to K-means.

Example 8.1 (Fuzzy c-means on Three Circular Clusters).

[Figure 8.1](#) shows the result of applying fuzzy c-means to find three clusters for a two-dimensional data set of 100 points. Each point was

assigned to the cluster in which it had the largest membership weight. The points belonging to each cluster are shown by different marker shapes, while the degree of membership in the cluster is shown by the shading. The darker the points, the stronger their membership in the cluster to which they have been assigned. The membership in a cluster is strongest toward the center of the cluster and weakest for those points that are between clusters.

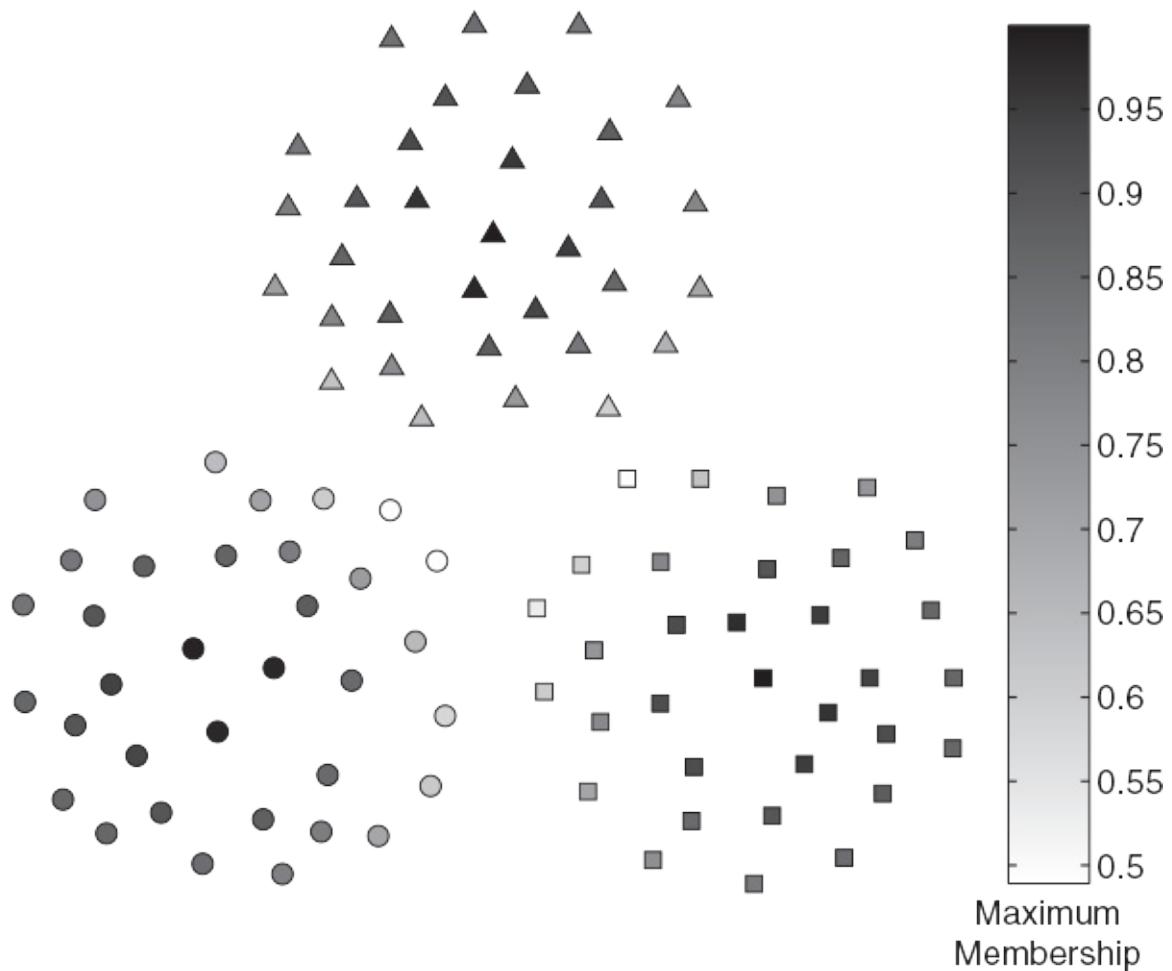


Figure 8.1.

Fuzzy c-means clustering of a two-dimensional point set.

Strengths and Limitations

A positive feature of FCM is that it produces a clustering that provides an indication of the degree to which any point belongs to any cluster. Otherwise, it has much the same strengths and weaknesses as K-means, although it is somewhat more computationally intensive.

8.2.2 Clustering Using Mixture Models

This section considers clustering based on statistical models. It is often convenient and effective to assume that data has been generated as a result of a statistical process and to describe the data by finding the statistical model that best fits the data, where the statistical model is described in terms of a distribution and a set of parameters for that distribution. At a high level, this process involves deciding on a statistical model for the data and estimating the parameters of that model from the data. This section describes a particular kind of statistical model, **mixture models**, which model the data by using a number of statistical distributions. Each distribution corresponds to a cluster and the parameters of each distribution provide a description of the corresponding cluster, typically in terms of its center and spread.

The discussion in this section proceeds as follows. After providing a description of mixture models, we consider how parameters can be estimated for statistical data models. We first describe how a procedure known as **maximum likelihood estimation (MLE)** can be used to estimate parameters for simple statistical models and then discuss how we can extend this approach for estimating the parameters of mixture models. Specifically, we describe the well-known **Expectation-Maximization (EM) algorithm**, which makes an initial guess for the parameters, and then iteratively improves these estimates. We present examples of how the EM algorithm can be used to

cluster data by estimating the parameters of a mixture model and discuss its strengths and limitations.

A firm understanding of statistics and probability, as covered in Appendix C, is essential for understanding this section. Also, for convenience in the following discussion, we use the term probability to refer to both probability and probability density.

Mixture Models

Mixture models view the data as a set of observations from a mixture of different probability distributions. The probability distributions can be anything, but are often taken to be multivariate normal, as this type of distribution is well understood, mathematically easy to work with, and has been shown to produce good results in many instances. These types of distributions can model ellipsoidal clusters.

Conceptually, mixture models correspond to the following process of generating data. Given several distributions, usually of the same type, but with different parameters, randomly select one of these distributions and generate an object from it. Repeat the process m times, where m is the number of objects.

More formally, assume that there are K distributions and m objects, $X=\{x_1, \dots, x_m\}$. Let the j th distribution have parameters θ_j , and let Θ be the set of all parameters, i.e., $\Theta=\{\theta_1, \dots, \theta_K\}$. Then, $\text{prob}(x_i|\theta_j)$ is the probability of the i th object if it comes from the j th distribution. The probability that the j th distribution is chosen to generate an object is given by the weight $w_j, 1 \leq j \leq K$, where these weights (probabilities) are subject to the constraint that they sum to one, i.e., $\sum_{j=1}^K w_j = 1$. Then, the probability of an object x is given by

Equation 8.5 

$$\text{prob}(x|\Theta) = \sum_{j=1}^K w_j p_j(x|\theta_j) \quad (8.5)$$

If the objects are generated in an independent manner, then the probability of the entire set of objects is just the product of the probabilities of each individual x_i .

$$\text{prob}(X|\Theta) = \prod_{i=1}^m \text{prob}(x_i|\Theta) = \prod_{i=1}^m \sum_{j=1}^K w_j p_j(x_i|\theta_j) \quad (8.6)$$

For mixture models, each distribution describes a different group, i.e., a different cluster. By using statistical methods, we can estimate the parameters of these distributions from the data and thus describe these distributions (clusters). We can also identify which objects belong to which clusters. However, mixture modeling does not produce a crisp assignment of objects to clusters, but rather gives the probability with which a specific object belongs to a particular cluster.

Example 8.2 (Univariate Gaussian Mixture).

We provide a concrete illustration of a mixture model in terms of Gaussian distributions. The probability density function for a one-dimensional Gaussian distribution at a point x is

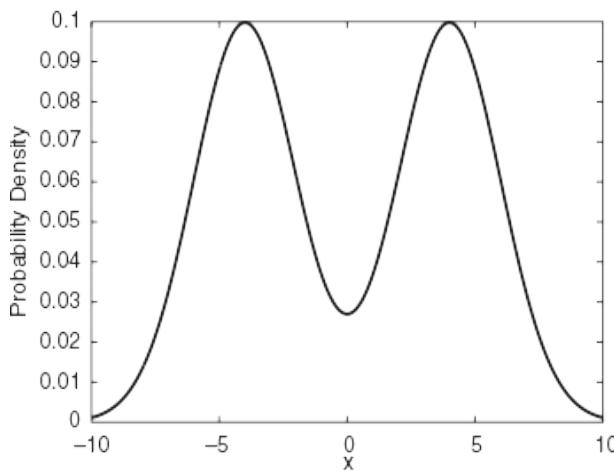
$$\text{prob}(x|\Theta) = \frac{1}{2\pi\sigma^2} e^{-(x-\mu)^2/2\sigma^2}. \quad (8.7)$$

The parameters of the Gaussian distribution are given by $\theta=(\mu, \sigma)$, where μ is the mean of the distribution and σ is the standard deviation. Assume that there are two Gaussian distributions, with a common standard deviation of 2 and means of -4 and 4, respectively. Also assume that each of the two distributions is selected with equal probability, i.e., $w_1=w_2=0.5$. Then

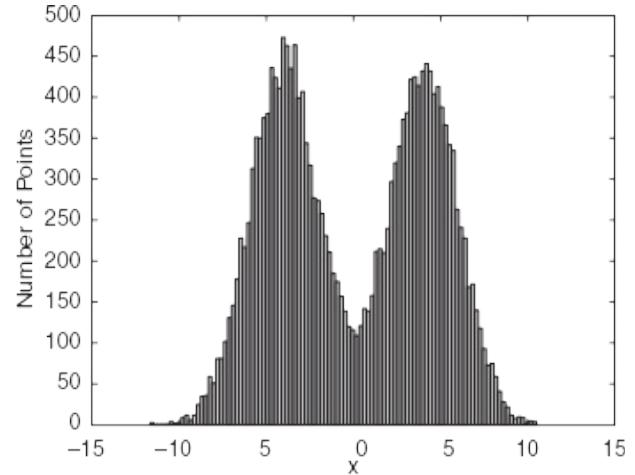
Equation 8.5 becomes the following:

$$\text{prob}(x|\Theta) = 122\pi e^{-(x+4)^2/28} + 122\pi e^{-(x-4)^2/28}. \quad (8.8)$$

Figure 8.2(a) shows a plot of the probability density function of this mixture model, while **Figure 8.2(b)** shows the histogram for 20,000 points generated from this mixture model.



(a) Probability density function for the mixture model.



(b) 20,000 points generated from the mixture model.

Figure 8.2.

Mixture model consisting of two normal distributions with means of -4 and 4, respectively. Both distributions have a standard deviation of 2.

Estimating Model Parameters Using Maximum Likelihood

Given a statistical model for the data, it is necessary to estimate the parameters of that model. A standard approach used for this task is maximum likelihood estimation, which we now explain.

Consider a set of m points that are generated from a one-dimensional Gaussian distribution. Assuming that the points are generated independently,

the probability of these points is just the product of their individual probabilities. (Again, we are dealing with probability densities, but to keep our terminology simple, we will refer to probabilities.) Using [Equation 8.7](#), we can write this probability as shown in [Equation 8.9](#). Because this probability would be a very small number, we typically will work with the log probability, as shown in [Equation 8.10](#).

$$\text{prob}(X|\Theta) = \prod_{i=1}^m \frac{1}{2\pi\sigma^2} e^{-(x_i - \mu)^2/2\sigma^2} \quad (8.9)$$

$$\log \text{prob}(X|\Theta) = -\sum_{i=1}^m (x_i - \mu)^2/2\sigma^2 - m\log(2\pi\sigma^2) \quad (8.10)$$

We would like to find a procedure to estimate μ and σ if they are unknown. One approach is to choose the values of the parameters for which the data is most probable (most likely). In other words, choose the μ and σ that maximize [Equation 8.9](#). This approach is known in statistics as the **maximum likelihood principle**, and the process of applying this principle to estimate the parameters of a statistical distribution from the data is known as **maximum likelihood estimation (MLE)**.

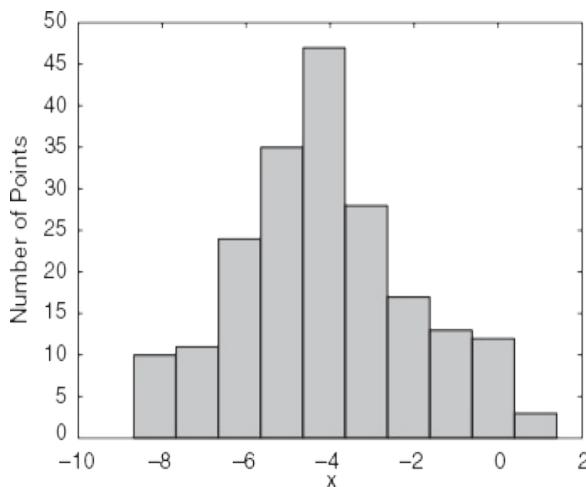
The principle is called the maximum likelihood principle because, given a set of data, the probability of the data, regarded as a function of the parameters, is called a **likelihood function**. To illustrate, we rewrite [Equation 8.9](#) as [Equation 8.11](#) to emphasize that we view the statistical parameters μ and σ as our variables and that the data is regarded as a constant. For practical reasons, the log likelihood is more commonly used. The log likelihood function derived from the log probability of [Equation 8.10](#) is shown in [Equation 8.12](#). Note that the parameter values that maximize the log likelihood also maximize the likelihood since log is a monotonically increasing function.

$$\text{likelihood}(\Theta|X) = L(\Theta|X) = \prod_{i=1}^m \frac{1}{2\pi\sigma^2} e^{-(x_i - \mu)^2/2\sigma^2} \quad (8.11)$$

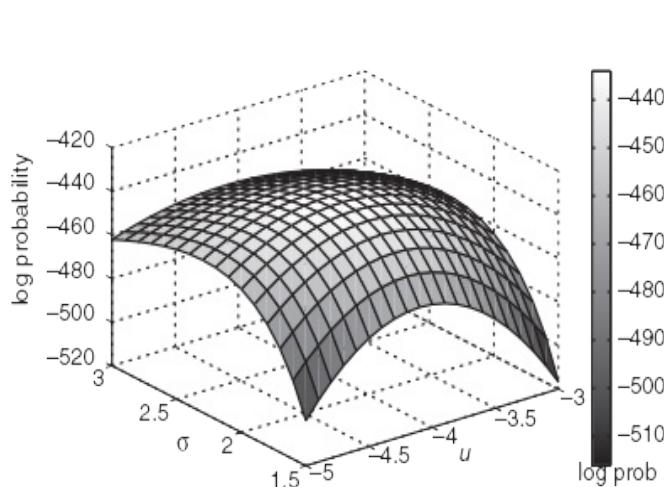
$$\log \text{likelihood}(\Theta|X) = \ell(\Theta|X) = -\sum_{i=1}^m (x_i - \mu)^2 / 2\sigma^2 - 0.5m \log 2\pi - m \log \sigma \quad (8.12)$$

Example 8.3 (Maximum Likelihood Parameter Estimation).

We provide a concrete illustration of the use of MLE for finding parameter values. Suppose that we have the set of 200 points whose histogram is shown in [Figure 8.3\(a\)](#). [Figure 8.3\(b\)](#) shows the maximum log likelihood plot for the 200 points under consideration. The values of the parameters for which the log probability is a maximum are $\mu=-4.1$ and $\sigma=2.1$, which are close to the parameter values of the underlying Gaussian distribution, $\mu=-4.0$ and $\sigma=2.0$.



(a) Histogram of 200 points from a Gaussian distribution.



(b) Log likelihood plot of the 200 points for different values of the mean and standard deviation.

Figure 8.3.

200 points from a Gaussian distribution and their log probability for different parameter values.

Graphing the likelihood of the data for different values of the parameters is not practical, at least if there are more than two parameters. Thus, standard

statistical procedure is to derive the maximum likelihood estimates of a statistical parameter by taking the derivative of likelihood function with respect to that parameter, setting the result equal to 0, and solving. In particular, for a Gaussian distribution, it can be shown that the mean and standard deviation of the sample points are the maximum likelihood estimates of the corresponding parameters of the underlying distribution. (See [Exercise 9](#) on 700.) Indeed, for the 200 points considered in our example, the parameter values that maximized the log likelihood were precisely the mean and standard deviation of the 200 points, i.e., $\mu=-4.1$ and $\sigma=2.1$.

Estimating Mixture Model Parameters Using Maximum Likelihood: The EM Algorithm

We can also use the maximum likelihood approach to estimate the model parameters for a mixture model. In the simplest case, we know which data objects come from which distributions, and the situation reduces to one of estimating the parameters of a single distribution given data from that distribution. For most common distributions, the maximum likelihood estimates of the parameters are calculated from simple formulas involving the data.

In a more general (and more realistic) situation, we do not know which points were generated by which distribution. Thus, we cannot directly calculate the probability of each data point, and hence, it would seem that we cannot use the maximum likelihood principle to estimate parameters. The solution to this problem is the EM algorithm, which is shown in [Algorithm 8.2](#). Briefly, given a guess for the parameter values, the EM algorithm calculates the probability that each point belongs to each distribution and then uses these probabilities to compute a new estimate for the parameters. (These parameters are the ones that maximize the likelihood.) This iteration continues until the estimates of the parameters either do not change or change very

little. Thus, we still employ maximum likelihood estimation, but via an iterative search.

Algorithm 8.2 EM algorithm.

- 1: Select an initial set of model parameters.
(As with K-means, this can be done randomly or in a variety of ways.)
- 2: **repeat**
- 3: **Expectation Step** For each object, calculate the probability that each object belongs to each distribution, i.e., calculate $\text{prob}(\text{distribution } j|x_i, \Theta)$.
- 4: **Maximization Step** Given the probabilities from the expectation step, find the new estimates of the parameters that maximize the expected likelihood.
- 5: **until** The parameters do not change.
(Alternatively, stop if the change in the parameters is below a specified threshold.)

The EM algorithm is similar to the K-means algorithm given in [Section 7.2.1](#). Indeed, the K-means algorithm for Euclidean data is a special case of the EM algorithm for spherical Gaussian distributions with equal covariance matrices, but different means. The expectation step corresponds to the K-means step of assigning each object to a cluster. Instead, each object is assigned to every cluster (distribution) with some probability. The maximization step corresponds to computing the cluster centroids. Instead, all the parameters of the distributions, as well as the weight parameters, are selected to maximize the likelihood. This process is often straightforward, as the parameters are typically computed using formulas derived from maximum likelihood estimation. For instance, for a single Gaussian distribution, the MLE estimate of the mean is the mean of the objects in the distribution. In the

context of mixture models and the EM algorithm, the computation of the mean is modified to account for the fact that every object belongs to a distribution with a certain probability. This is illustrated further in the following example.

Example 8.4 (Simple Example of EM Algorithm).

This example illustrates how EM operates when applied to the data in [Figure 8.2](#). To keep the example as simple as possible, we assume that we know that the standard deviation of both distributions is 2.0 and that points were generated with equal probability from both distributions. We will refer to the left and right distributions as distributions 1 and 2, respectively.

We begin the EM algorithm by making initial guesses for μ_1 and μ_2 , say, $\mu_1=-2$ and $\mu_2=3$. Thus, the initial parameters, $\theta=(\mu,\sigma)$, for the two distributions are, respectively, $\theta_1=(-2,2)$ and $\theta_2=(3,2)$. The set of parameters for the entire mixture model is $\Theta=\{\theta_1,\theta_2\}$. For the expectation step of EM, we want to compute the probability that a point came from a particular distribution; i.e., we want to compute $\text{prob}(\text{distribution } 1|x_i, \Theta)$ and $\text{prob}(\text{distribution } 2|x_i, \Theta)$. These values can be expressed by [Equation 8.13](#), which is a straightforward application of Bayes rule, which is described in Appendix C.

$$\text{prob}(\text{distribution } j|x_i, \Theta) = 0.5 \text{ prob}(x_i|\theta_j) 0.5 \text{ prob}(x_i|\theta_1) + 0.5 \text{ prob}(x_i|\theta_2), \quad (8.13)$$

where 0.5 is the probability (weight) of each distribution and j is 1 or 2.

For instance, assume one of the points is 0. Using the Gaussian density function given in [Equation 8.7](#), we compute that $\text{prob}(0|\theta_1)=0.12$ and $\text{prob}(0|\theta_2)=0.06$. (Again, we are really computing probability densities.) Using these values and [Equation 8.13](#), we find that

$\text{prob}(\text{distribution 1}|0, \Theta) = 0.12/(0.12+0.06) = 0.66$ and $\text{prob}(\text{distribution 2}|0, \Theta) = 0.06/(0.12+0.06) = 0.33$. This means that the point 0 is twice as likely to belong to distribution 1 as distribution 2 based on the current assumptions for the parameter values.

After computing the cluster membership probabilities for all 20,000 points, we compute new estimates for μ_1 and μ_2 (using [Equations 8.14](#) and [8.15](#)) in the maximization step of the EM algorithm. Notice that the new estimate for the mean of a distribution is just a weighted average of the points, where the weights are the probabilities that the points belong to the distribution, i.e., the $\text{prob}(\text{distribution } j|xi)$ values.

$$\mu_1 = \sum_{i=1}^{20,000} x_i \text{prob}(\text{distribution 1}|xi, \Theta) / \sum_{i=1}^{20,000} \text{prob}(\text{distribution 1}|xi, \Theta)$$

$$\mu_2 = \sum_{i=1}^{20,000} x_i \text{prob}(\text{distribution 2}|xi, \Theta) / \sum_{i=1}^{20,000} \text{prob}(\text{distribution 2}|xi, \Theta)$$

We repeat these two steps until the estimates of μ_1 and μ_2 either don't change or change very little. [Table 8.1](#) gives the first few iterations of the EM algorithm when it is applied to the set of 20,000 points. For this data, we know which distribution generated which point, so we can also compute the mean of the points from each distribution. The means are $\mu_1 = -3.98$ and $\mu_2 = 4.03$.

Table 8.1. First few iterations of the EM algorithm for the simple example.

Iteration	μ_1	μ_2
0	-2.00	3.00
1	-3.74	4.10
2	-3.94	4.07

3	-3.97	4.04
4	-3.98	4.03
5	-3.98	4.03

Example 8.5 (The EM Algorithm on Sample Data Sets).

We give three examples that illustrate the use of the EM algorithm to find clusters using mixture models. The first example is based on the data set used to illustrate the fuzzy c-means algorithm—see [Figure 8.1](#). We modeled this data as a mixture of three two-dimensional Gaussian distributions with different means and identical covariance matrices. We then clustered the data using the EM algorithm. The results are shown in [Figure 8.4](#). Each point was assigned to the cluster in which it had the largest membership weight. The points belonging to each cluster are shown by different marker shapes, while the degree of membership in the cluster is shown by the shading. Membership in a cluster is relatively weak for those points that are on the border of the two clusters, but strong elsewhere. It is interesting to compare the membership weights and probabilities of [Figures 8.4](#) and [8.1](#). (See [Exercise 11](#) on page [700](#).)

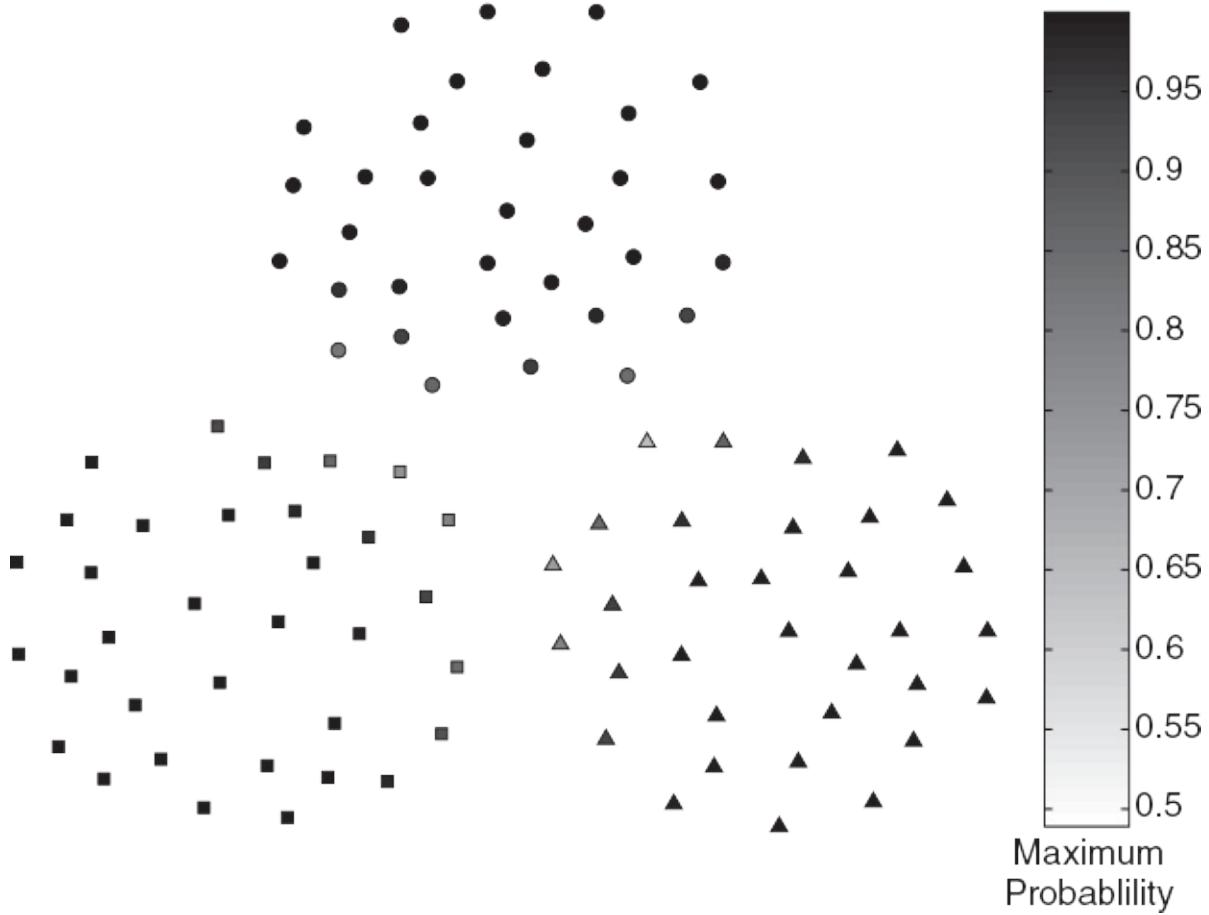


Figure 8.4.

EM clustering of a two-dimensional point set with three clusters.

For our second example, we apply mixture model clustering to data that contains clusters with different densities. The data consists of two natural clusters, each with roughly 500 points. This data was created by combining two sets of Gaussian data, one with a center at $(-4, 1)$ and a standard deviation of 2, and one with a center at $(0, 0)$ and a standard deviation of 0.5. [Figure 8.5](#) shows the clustering produced by the EM algorithm. Despite the differences in the density, the EM algorithm is quite successful at identifying the original clusters.

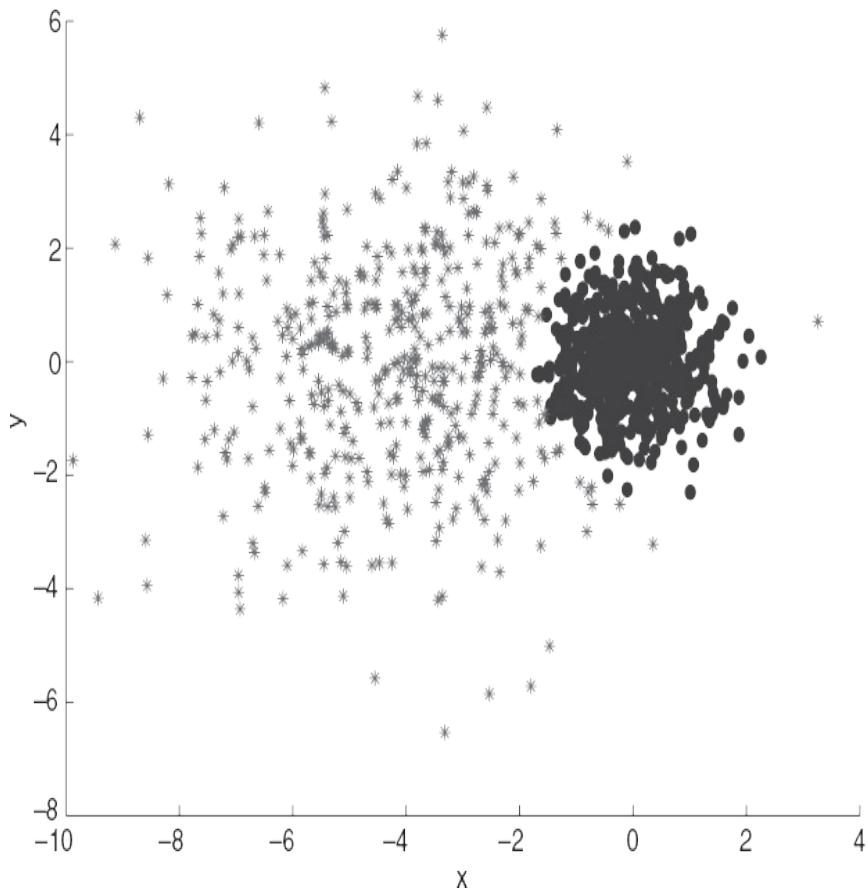
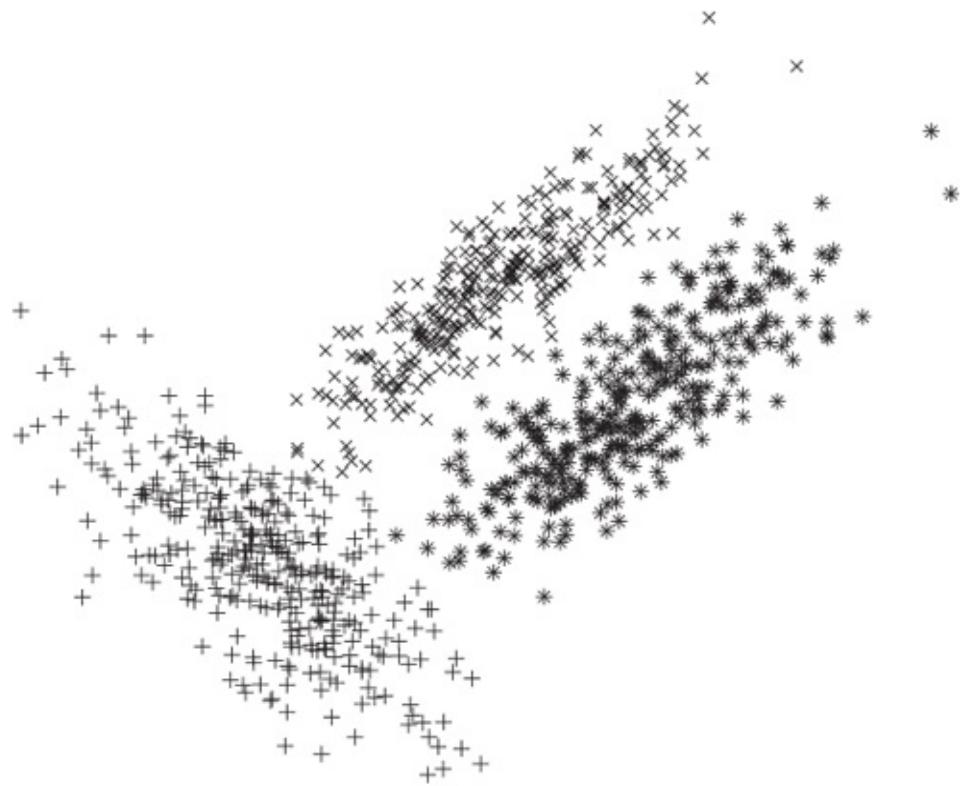


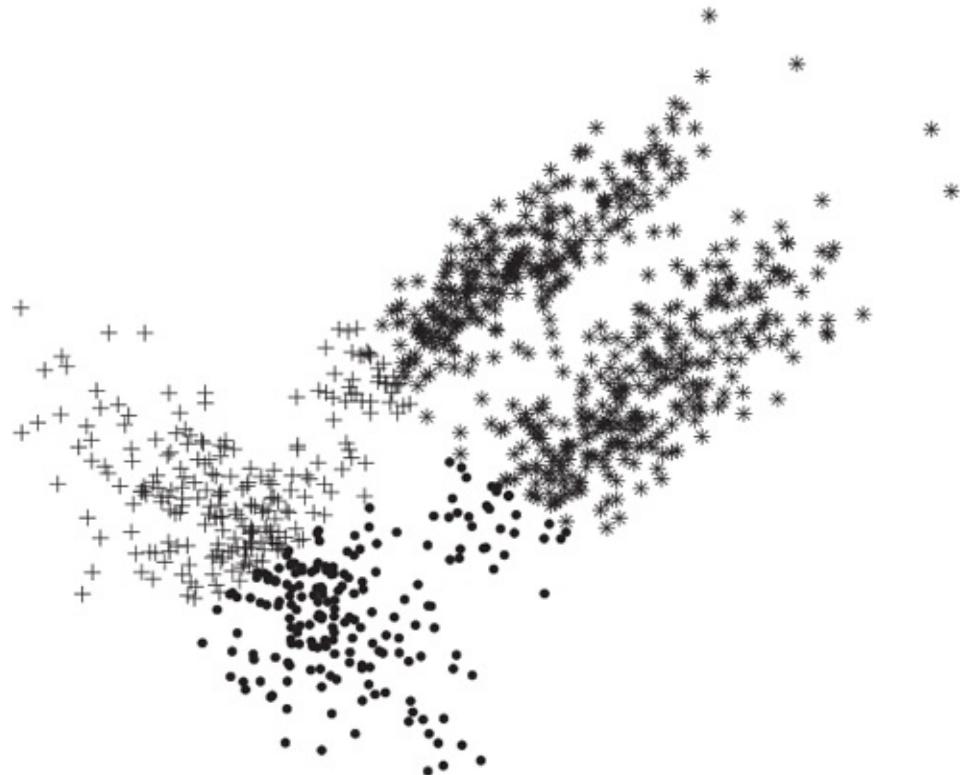
Figure 8.5.

EM clustering of a two-dimensional point set with two clusters of differing density.

For our third example, we use mixture model clustering on a data set that K-means cannot properly handle. [Figure 8.6\(a\)](#) shows the clustering produced by a mixture model algorithm, while [Figure 8.6\(b\)](#) shows the K-means clustering of the same set of 1,000 points. For mixture model clustering, each point has been assigned to the cluster for which it has the highest probability. In both figures, different markers are used to distinguish different clusters. Do not confuse the '+' and 'x' markers in [Figure 8.6\(a\)](#).



(a) Clusters produced by mixture model clustering.



(b) Clusters produced by K-means clustering.

Figure 8.6.

Mixture model and K-means clustering of a set of two-dimensional points.

Advantages and Limitations of Mixture Model Clustering Using the EM Algorithm

Finding clusters by modeling the data using mixture models and applying the EM algorithm to estimate the parameters of those models has a variety of advantages and disadvantages. On the negative side, the EM algorithm can be slow, it is not practical for models with large numbers of components, and it does not work well when clusters contain only a few data points or if the data points are nearly co-linear. There is also a problem in estimating the number of clusters or, more generally, in choosing the exact form of the model to use. This problem typically has been dealt with by applying a Bayesian approach, which, roughly speaking, gives the odds of one model versus another, based on an estimate derived from the data. Mixture models can also have difficulty with noise and outliers, although work has been done to deal with this problem.

On the positive side, mixture models are more general than K-means or fuzzy c-means because they can use distributions of various types. As a result, mixture models (based on Gaussian distributions) can find clusters of different sizes and elliptical shapes. Also, a model-based approach provides a disciplined way of eliminating some of the complexity associated with data. To see the patterns in data, it is often necessary to simplify the data, and fitting the data to a model is a good way to do that if the model is a good match for the data. Furthermore, it is easy to characterize the clusters produced, because they can be described by a small number of parameters. Finally, many sets of data are indeed the result of random processes, and thus should satisfy the statistical assumptions of these models.

8.2.3 Self-Organizing Maps (SOM)

The Kohonen Self-Organizing Feature Map (SOFM or SOM) is a clustering and data visualization technique based on a neural network viewpoint. Despite the neural network origins of SOM, it is more easily presented—at least in the context of this chapter—as a variation of prototype-based clustering. As with other types of centroid-based clustering, the goal of SOM is to find a set of centroids (**reference vectors** in SOM terminology) and to assign each object in the data set to the centroid that provides the best approximation of that object. In neural network terminology, there is one neuron associated with each centroid.

As with incremental K-means, data objects are processed one at a time and the closest centroid is updated. Unlike K-means, SOM imposes a topographic ordering on the centroids and nearby centroids are also updated.

Furthermore, SOM does not keep track of the current cluster membership of an object, and, unlike K-means, if an object switches clusters, there is no explicit update of the old cluster centroid. However, if the old cluster is in the neighborhood of the new cluster, it will be updated. The processing of points continues until some predetermined limit is reached or the centroids are not changing very much. The final output of the SOM technique is a set of centroids that implicitly define clusters. Each cluster consists of the points closest to a particular centroid. The following section explores the details of this process.

The SOM Algorithm

A distinguishing feature of SOM is that it imposes a topographic (spatial) organization on the centroids (neurons). [Figure 8.7](#) shows an example of a two-dimensional SOM in which the centroids are represented by nodes that

are organized in a rectangular lattice. Each centroid is assigned a pair of coordinates (i, j) . Sometimes, such a network is drawn with links between adjacent nodes, but that can be misleading because the influence of one centroid on another is via a neighborhood that is defined in terms of coordinates, not links. There are many types of SOM neural networks, but we restrict our discussion to two-dimensional SOMs with a rectangular or hexagonal organization of the centroids.

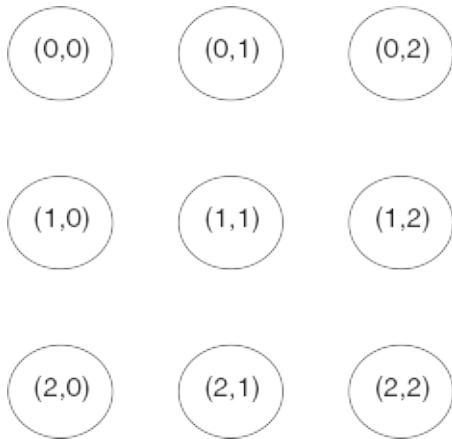


Figure 8.7.

Two-dimensional 3-by-3 rectangular SOM neural network.

Even though SOM is similar to K-means or other prototype-based approaches, there is a fundamental difference. Centroids used in SOM have a predetermined topographic ordering relationship. During the training process, SOM uses each data point to update the closest centroid and centroids that are nearby in the topographic ordering. In this way, SOM produces an ordered set of centroids for any given data set. In other words, the centroids that are close to each other in the SOM grid are more closely related to each other than to the centroids that are farther away. Because of this constraint, the centroids of a two-dimensional SOM can be viewed as lying on a two-dimensional surface that tries to fit the n -dimensional data as well as possible. The SOM centroids can also be thought of as the result of a nonlinear regression with respect to the data points.

At a high level, clustering using the SOM technique consists of the steps described in [Algorithm 8.3](#).

Algorithm 8.3 Basic SOM Algorithm.

- 1: Initialize the centroids.
- 2: **repeat**
- 3: Select the next object.
- 4: Determine the closest centroid to the object.
- 5: Update this centroid and the centroids that are close, i.e., in a specified neighborhood.
- 6: **until** The centroids don't change much or a threshold is exceeded.
- 7: Assign each object to its closest centroid and return the centroids and clusters.

Initialization

This step (line 1) can be performed in a number of ways. One approach is to choose each component of a centroid randomly from the range of values observed in the data for that component. While this approach works, it is not necessarily the best approach, especially for producing rapid convergence. Another approach is to randomly choose the initial centroids from the available data points. This is very much like randomly selecting centroids for K-means.

Selection of an Object

The first step in the loop (line 3) is the selection of the next object. This is fairly straightforward, but there are some difficulties. Because convergence can

require many steps, each data object may be used multiple times, especially if the number of objects is small. However, if the number of objects is large, then not every object needs to be used. It is also possible to enhance the influence of certain groups of objects by increasing their frequency in the training set.

Assignment

The determination of the closest centroid (line 4) is also relatively straightforward, although it requires the specification of a distance metric. The Euclidean distance metric is often used, as is the dot product metric. When using the dot product distance, the data vectors are typically normalized beforehand and the reference vectors are normalized at each step. In such cases, using the dot product metric is equivalent to using the cosine measure.

Update

The update step (line 5) is the most complicated. Let m_1, \dots, m_k be the centroids. (For a rectangular grid, note that k is the product of the number of rows and the number of columns.) For time step t , let $p(t)$ be the current object (point) and assume that the closest centroid to $p(t)$ is m_j . Then, for time $t+1$, the j th centroid is updated by using the following equation. (We will see shortly that the update is really restricted to centroids whose neurons are in a small neighborhood of m_j .)

$$m_j(t+1) = m_j(t) + h_j(t)(p(t) - m_j(t)) \quad (8.16)$$

Thus, at time t , a centroid $m_j(t)$ is updated by adding a term, $h_j(t)(p(t) - m_j(t))$, which is proportional to the difference, $p(t) - m_j(t)$, between the current object, $p(t)$, and centroid, $m_j(t)$. $h_j(t)$, determines the effect that the difference, $p(t) - m_j(t)$, will have and is chosen so that (1) it diminishes with time and (2) it enforces a neighborhood effect, i.e., the effect of an object is strongest on the

centroids closest to the centroid m_j . Here we are referring to the distance in the grid, not the distance in the data space. Typically, $h_j(t)$ is chosen to be one of the following two functions:

$$h_j(t) = \alpha(t) \exp(-\text{dist}(r_j, r_k)^2 / (2\sigma^2(t)) \quad (\text{Gaussian function})$$
$$h_j(t) = \alpha(t) \text{ if } \text{dist}(r_j, r_k) \leq t \quad (\text{step function})$$

These functions require more explanation. $\alpha(t)$ is a learning rate parameter, $0 < \alpha(t) < 1$, which decreases monotonically with time and controls the rate of convergence. $r_k = (x_k, y_k)$ is the two-dimensional point that gives the grid coordinates of the k th centroid. $\text{dist}(r_j, r_k)$ is the Euclidean distance between the grid location of the two centroids, i.e., $(x_j - x_k)^2 + (y_j - y_k)^2$. Consequently, for centroids whose grid locations are far from the grid location of centroid m_j , the influence of object $p(t)$ will be either greatly diminished or non-existent. Finally, note that σ is the typical Gaussian variance parameter and controls the width of the neighborhood, i.e., a small σ will yield a small neighborhood, while a large σ will yield a wide neighborhood. The threshold used for the step function also controls the neighborhood size.

Remember, it is the neighborhood updating technique that enforces a relationship (ordering) between centroids associated with neighboring neurons.

Termination

Deciding when we are close enough to a stable set of centroids is an important issue. Ideally, iteration should continue until convergence occurs, that is, until the reference vectors either do not change or change very little. The rate of convergence will depend on a number of factors, such as the data and $\alpha(t)$. We will not discuss these issues further, except to mention that, in general, convergence can be slow and is not guaranteed.

Example 8.6 (Document Data).

We present two examples. In the first case, we apply SOM with a 4-by-4 hexagonal grid to document data. We clustered 3204 newspaper articles from the *Los Angeles Times*, which come from 6 different sections: Entertainment, Financial, Foreign, Metro, National, and Sports. [Figure 8.8](#) shows the SOM grid. We have used a hexagonal grid, which allows each centroid to have six immediate neighbors instead of four. Each SOM grid cell (cluster) has been labeled with the majority class label of the associated points. The clusters of each particular category form contiguous groups, and their position relative to other categories of clusters gives us additional information, e.g., that the Metro section contains stories related to all other sections.

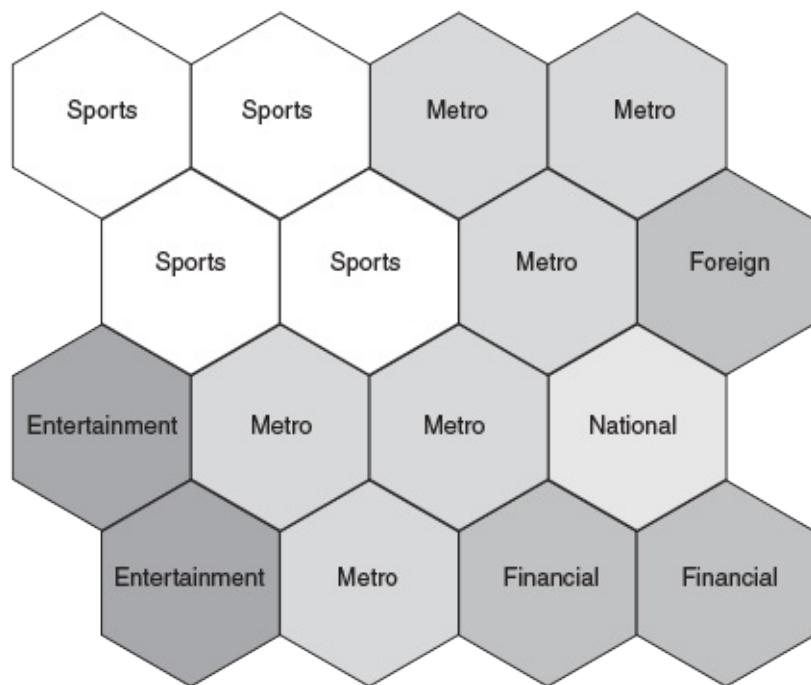
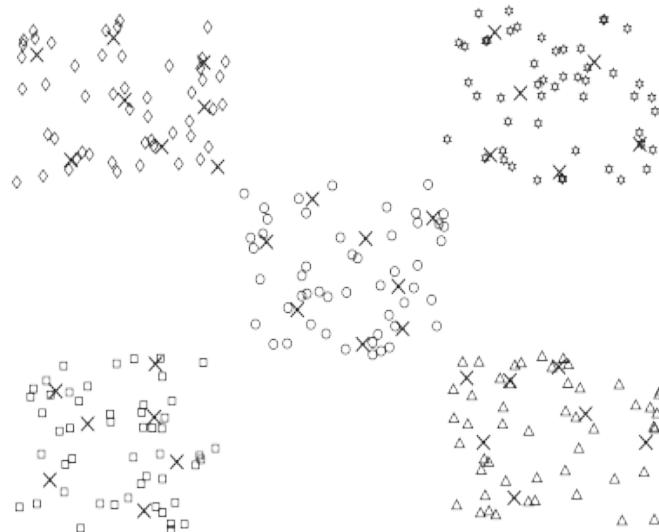


Figure 8.8.

Visualization of the relationships between SOM cluster for *Los Angeles Times* document data set.

Example 8.7 (Two-Dimensional Points).

In the second case, we use a rectangular SOM and a set of two-dimensional data points. [Figure 8.9\(a\)](#) shows the points and the positions of the 36 reference vectors (shown as x's) produced by SOM. The points are arranged in a checkerboard pattern and are split into five classes: circles, triangles, squares, diamonds, and hexagons (stars). A 6-by-6 two-dimensional rectangular grid of centroids was used with random initialization. As [Figure 8.9\(a\)](#) shows, the centroids tend to distribute themselves to the dense areas. [Figure 8.9\(b\)](#) indicates the majority class of the points associated with that centroid. The clusters associated with triangle points are in one contiguous area, as are the centroids associated with the four other types of points. This is a result of the neighborhood constraints enforced by SOM. While there are the same number of points in each of the five groups, notice also that the centroids are not evenly distributed. This is partly due to the overall distribution of points and partly an artifact of putting each centroid in a single cluster.



(a) Distribution of SOM reference vectors (\mathbf{X} 's) for a two-dimensional point set.

diamond	diamond	diamond	hexagon	hexagon	hexagon
diamond	diamond	diamond	circle	hexagon	hexagon
diamond	diamond	circle	circle	circle	hexagon
square	square	circle	circle	triangle	triangle
square	square	circle	circle	triangle	triangle
square	square	square	triangle	triangle	triangle

(b) Classes of the SOM centroids.

Figure 8.9.

SOM applied to two-dimensional data points.

Applications

Once the SOM vectors are found, they can be used for many purposes other than clustering. For example, with a two-dimensional SOM, it is possible to associate various quantities with the grid points associated with each centroid (cluster) and to visualize the results via various types of plots. For example, plotting the number of points associated with each cluster yields a plot that reveals the distribution of points among clusters. A two-dimensional SOM is a nonlinear projection of the original probability distribution function into two dimensions. This projection attempts to preserve topological features; thus, using SOM to capture the structure of the data has been compared to the process of “pressing a flower.”

Strengths and Limitations

SOM is a clustering technique that enforces neighborhood relationships on the resulting cluster centroids. Because of this, clusters that are neighbors are more related to one another than clusters that are not. Such relationships facilitate the interpretation and visualization of the clustering results. Indeed, this aspect of SOM has been exploited in many areas, such as visualizing web documents or gene array data.

SOM also has a number of limitations, which are listed next. Some of the listed limitations are only valid if we consider SOM to be a standard clustering technique that aims to find the true clusters in the data, rather than a technique that uses clustering to help discover the structure of the data. Also,

some of these limitations have been addressed either by extensions of SOM or by clustering algorithms inspired by SOM. (See the Bibliographic Notes.)

- The user must choose the settings of parameters, the neighborhood function, the grid type, and the number of centroids.
- A SOM cluster often does not correspond to a single natural cluster. In some cases, a SOM cluster might encompass several natural clusters, while in other cases a single natural cluster is split into several SOM clusters. This problem is partly due to the use of a grid of centroids and partly due to the fact that SOM, like other prototype-based clustering techniques, tends to split or combine natural clusters when they are of varying sizes, shapes, and densities.
- SOM lacks a specific objective function. SOM attempts to find a set of centroids that best approximate the data, subject to the topographic constraints among the centroids, but the success of SOM in doing this cannot be expressed by a function. This can make it difficult to compare different SOM clustering results.
- SOM is not guaranteed to converge, although, in practice, it typically does.

8.3 Density-Based Clustering

In [Section 7.4](#), we considered DBSCAN, a simple, but effective algorithm for finding density-based clusters, i.e., dense regions of objects that are surrounded by low-density regions. This section examines additional density-based clustering techniques that address issues of efficiency, finding clusters in subspaces, and more accurately modeling density. First, we consider grid-based clustering, which breaks the data space into grid cells and then forms clusters from cells that are sufficiently dense. Such an approach can be efficient and effective, at least for low-dimensional data. Next, we consider subspace clustering, which looks for clusters (dense regions) in subsets of all dimensions. For a data space with n dimensions, potentially $2n-1$ subspaces need to be searched, and thus an efficient technique is needed to do this. CLIQUE is a grid-based clustering algorithm that provides an efficient approach to subspace clustering based on the observation that dense areas in a high-dimensional space imply the existence of dense areas in lower-dimensional space. Finally, we describe DENCLUE, a clustering technique that uses kernel density functions to model density as the sum of the influences of individual data objects. While DENCLUE is not fundamentally a grid-based technique, it does employ a grid-based approach to improve efficiency.

8.3.1 Grid-Based Clustering

A grid is an efficient way to organize a set of data, at least in low dimensions. The idea is to split the possible values of each attribute into a number of contiguous intervals, creating a set of grid cells. (We are assuming, for this

discussion and the remainder of the section, that our attributes are ordinal, interval, or continuous.) Each object falls into a grid cell whose corresponding attribute intervals contain the values of the object. Objects can be assigned to grid cells in one pass through the data, and information about each cell, such as the number of points in the cell, can also be gathered at the same time.

There are a number of ways to perform clustering using a grid, but most approaches are based on density, at least in part, and thus, in this section, we will use grid-based clustering to mean density-based clustering using a grid.

Algorithm 8.4 describes a basic approach to grid-based clustering. Various aspects of this approach are explored next.

Algorithm 8.4 Basic grid-based clustering algorithm.

- 1: Define a set of grid cells.
- 2: Assign objects to the appropriate cells and compute the density of each cell.
- 3: Eliminate cells having a density below a specified threshold, T .
- 4: Form clusters from contiguous (adjacent) groups of dense cells.

Defining Grid Cells

This is a key step in the process, but also the least well defined, as there are many ways to split the possible values of each attribute into a number of contiguous intervals. For continuous attributes, one common approach is to

split the values into equal width intervals. If this approach is applied to each attribute, then the resulting grid cells all have the same volume, and the density of a cell is conveniently defined as the number of points in the cell.

However, more sophisticated approaches can also be used. In particular, for continuous attributes any of the techniques that are commonly used to discretize attributes can be applied. (See [Section 2.3.6](#).) In addition to the equal width approach already mentioned, this includes (1) breaking the values of an attribute into intervals so that each interval contains an equal number of points, i.e., equal frequency discretization, or (2) using clustering. Another approach, which is used by the subspace clustering algorithm MAFIA, initially breaks the set of values of an attribute into a large number of equal width intervals and then combines intervals of similar density.

Regardless of the approach taken, the definition of the grid has a strong impact on the clustering results. We will consider specific aspects of this later.

The Density of Grid Cells

A natural way to define the density of a grid cell (or a more generally shaped region) is as the number of points divided by the volume of the region. In other words, density is the number of points per amount of space, regardless of the dimensionality of that space. Specific, low-dimensional examples of density are the number of road signs per mile (one dimension), the number of eagles per square kilometer of habitat (two dimensions), and the number of molecules of a gas per cubic centimeter (three dimensions). As mentioned, however, a common approach is to use grid cells that have the same volume so that the number of points per cell is a direct measure of the cell's density.

Example 8.8 (Grid-Based Density).

Figure 8.10 shows two sets of two-dimensional points divided into 49 cells using a 7-by-7 grid. The first set contains 200 points generated from a uniform distribution over a circle centered at $(2, 3)$ of radius 2, while the second set has 100 points generated from a uniform distribution over a circle centered at $(6, 3)$ of radius 1. The counts for the grid cells are shown in **Table 8.2**. Since the cells have equal volume (area), we can consider these values to be the densities of the cells.

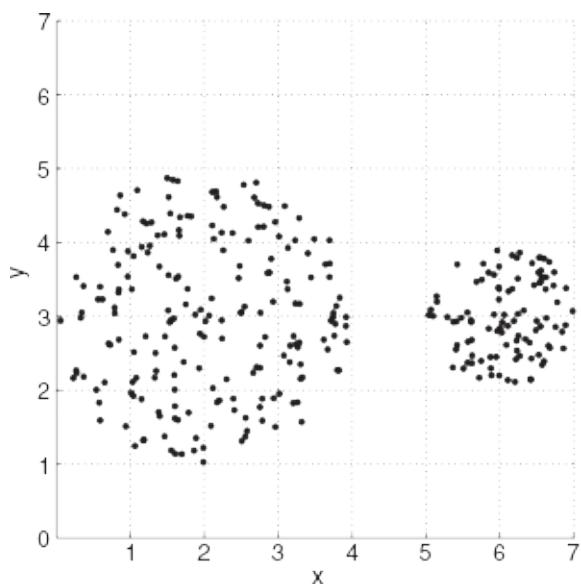


Figure 8.10.
Grid-based density.

Table 8.2. Point counts for grid cells.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
4	17	18	6	0	0	0
14	14	13	13	0	18	27
11	18	10	21	0	24	31

3	20	14	4	0	0	0
0	0	0	0	0	0	0

Forming Clusters from Dense Grid Cells

Forming clusters from adjacent groups of dense cells is relatively straightforward. (In [Figure 8.10](#), for example, it is clear that there would be two clusters.) There are, however, some issues. We need to define what we mean by adjacent cells. For example, does a two-dimensional grid cell have 4 adjacent cells or 8? Also, we need an efficient technique to find the adjacent cells, particularly when only occupied cells are stored.

The clustering approach defined by [Algorithm 8.4](#) has some limitations that could be addressed by making the algorithm slightly more sophisticated. For example, there are likely to be partially empty cells on the boundary of a cluster. Often, these cells are not dense. If so, they will be discarded and parts of a cluster will be lost. [Figure 8.10](#) and [Table 8.2](#) show that four parts of the larger cluster would be lost if the density threshold is 9. The clustering process could be modified to avoid discarding such cells, although this would require additional processing.

It is also possible to enhance basic grid-based clustering by using more than just density information. In many cases, the data has both spatial and non-spatial attributes. In other words, some of the attributes describe the location of objects in time or space, while other attributes describe other aspects of the objects. A common example is houses, which have both a location and a number of other characteristics, such as price or floor space in square feet. Because of spatial (or temporal) autocorrelation, objects in a particular cell often have similar values for their other attributes. In such cases, it is possible to filter the cells based on the statistical properties of one or more non-spatial

attributes, e.g., average house price, and then form clusters based on the density of the remaining points.

Strengths and Limitations

On the positive side, grid-based clustering can be very efficient and effective. Given a partitioning of each attribute, a single pass through the data can determine the grid cell of every object and the count of every grid. Also, even though the number of potential grid cells can be high, grid cells need to be created only for non-empty cells. Thus, the time and space complexity of defining the grid, assigning each object to a cell, and computing the density of each cell is only $O(m)$, where m is the number of points. If adjacent, occupied cells can be efficiently accessed, for example, by using a search tree, then the entire clustering process will be highly efficient, e.g., with a time complexity of $O(m \log m)$. For this reason, the grid-based approach to density clustering forms the basis of a number of clustering algorithms, such as STING, GRIDCLUS, WaveCluster, Bang-Clustering, CLIQUE, and MAFIA.

On the negative side, grid-based clustering, like most density-based clustering schemes, is very dependent on the choice of the density threshold τ . If τ is too high, then clusters will be lost. If τ is too low, two clusters that should be separate may be joined. Furthermore, if there are clusters and noise of differing densities, then it might not be possible to find a single value of τ that works for all parts of the data space.

There are also a number of issues related to the grid-based approach. In **Figure 8.10**, for example, the rectangular grid cells do not accurately capture the density of the circular boundary areas. We could attempt to alleviate this problem by making the grid finer, but the number of points in the grid cells associated with a cluster would likely show more fluctuation because points in the cluster are not evenly distributed. Indeed, some grid cells,

including those in the interior of the cluster, might even be empty. Another issue is that, depending on the placement or size of the cells, a group of points can appear in just one cell or be split between several different cells. The same group of points might be part of a cluster in the first case, but be discarded in the second. Finally, as dimensionality increases, the number of potential grid cells increases rapidly—exponentially in the number of dimensions. Even though it is not necessary to explicitly consider empty grid cells, it can easily happen that most grid cells contain a single object. In other words, grid-based clustering tends to work poorly for high-dimensional data.

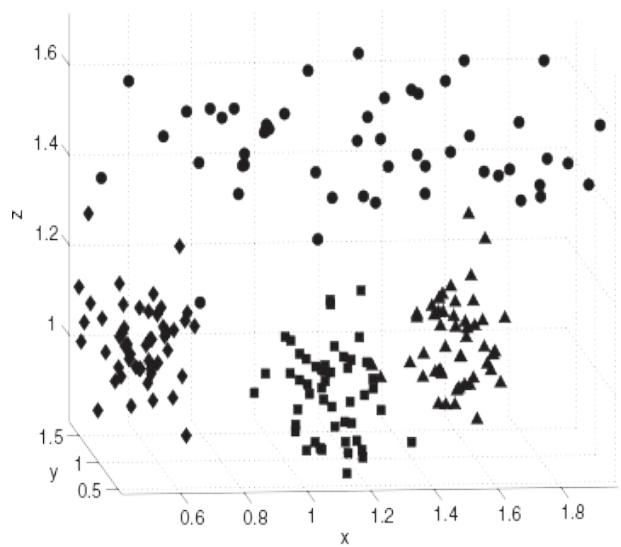
8.3.2 Subspace Clustering

The clustering techniques considered until now found clusters by using all of the attributes. However, if only subsets of the features are considered, i.e., subspaces of the data, then the clusters that we find can be quite different from one subspace to another. There are two reasons that subspace clusters might be interesting. First, the data may be clustered with respect to a small set of attributes, but randomly distributed with respect to the remaining attributes. Second, there are cases in which different clusters exist in different sets of dimensions. Consider a data set that records the sales of various items at various times. (The times are the dimensions and the items are the objects.) Some items might show similar behavior (cluster together) for particular sets of months, e.g., summer, but different clusters would likely be characterized by different months (dimensions).

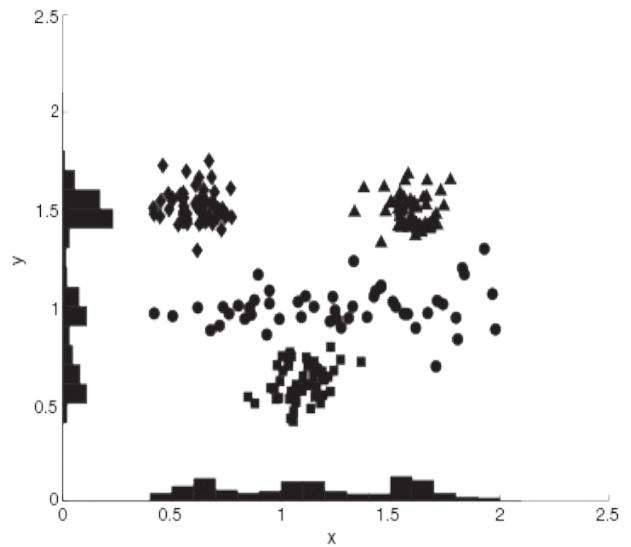
Example 8.9 (Subspace Clusters).

Figure 8.11(a) shows a set of points in three-dimensional space. There are three clusters of points in the full space, which are represented by

squares, diamonds, and triangles. In addition, there is one set of points, represented by circles, that is not a cluster in three-dimensional space. Each dimension (attribute) of the example data set is split into a fixed number (η) of equal width intervals. There are $\eta=20$ intervals, each of size 0.1. This partitions the data space into rectangular cells of equal volume, and thus, the density of each unit is the fraction of points it contains. Clusters are contiguous groups of dense cells. To illustrate, if the threshold for a dense cell is $\xi=0.06$, or 6% of the points, then three one-dimensional clusters can be identified in [Figure 8.12](#), which shows a histogram of the data points of [Figure 8.11\(a\)](#) for the x attribute.



(a) Four clusters in three dimensions.



(b) View in the xy plane.

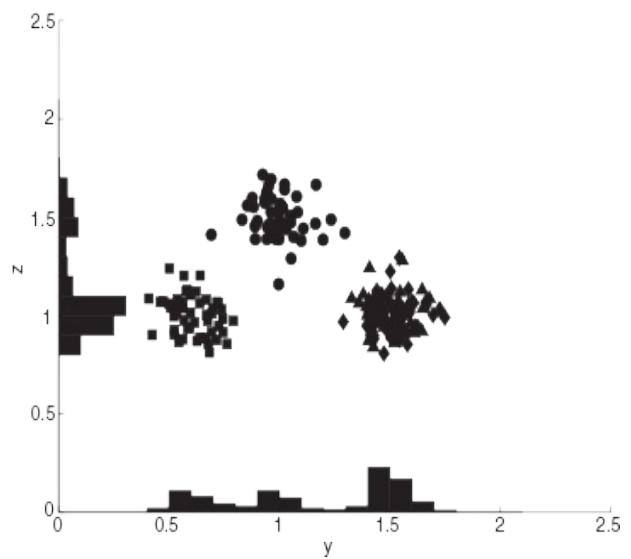
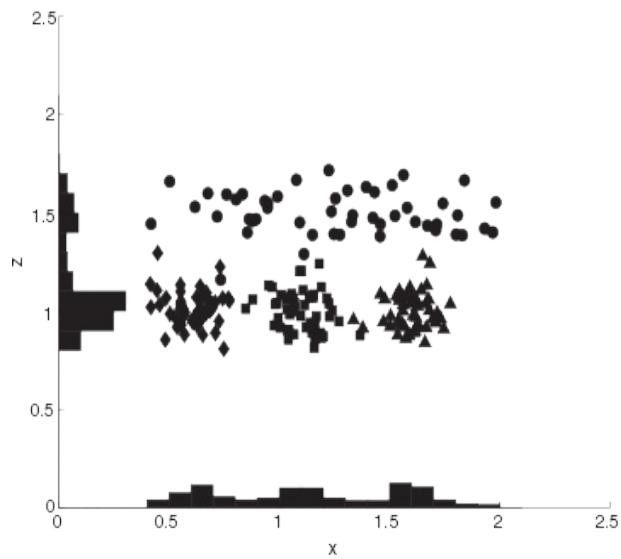


Figure 8.11.

Example figures for subspace clustering.

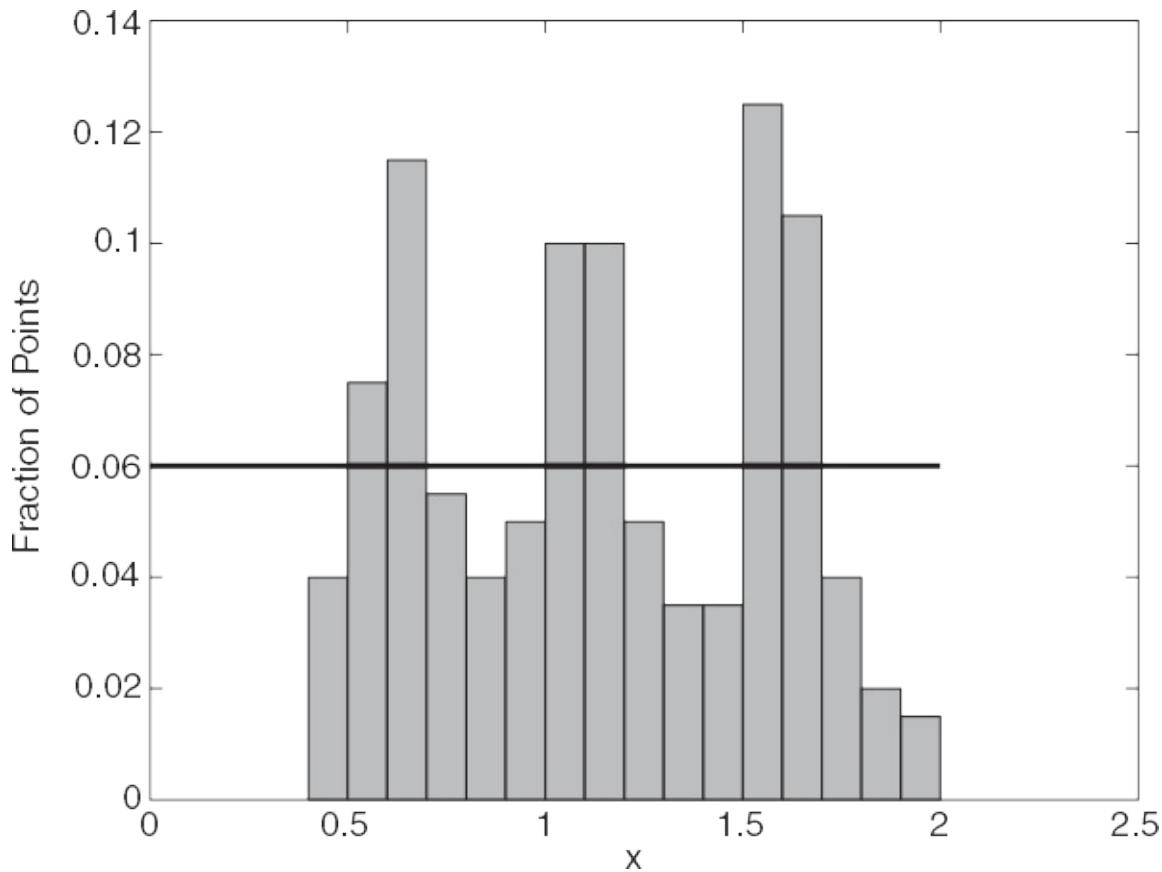


Figure 8.12.

Histogram showing the distribution of points for the x attribute.

Figure 8.11(b) shows the points plotted in the xy plane. (The z attribute is ignored.) This figure also contains histograms along the x and y axes that show the distribution of the points with respect to their x and y coordinates, respectively. (A higher bar indicates that the corresponding interval contains relatively more points, and vice versa.) When we consider the y axis, we see three clusters. One is from the circle points that do not form a cluster in the full space, one consists of the square points, and one consists of the diamond and triangle points. There are also three clusters in the x dimension; they correspond to the three clusters—diamonds, triangles, and squares—in the full space. These points also form distinct clusters in the xy plane. **Figure 8.11(c)** shows the points plotted in the xz plane. There are two clusters, if we consider only the z attribute. One cluster corresponds to the points represented by circles, while the other

consists of the diamond, triangle, and square points. These points also form distinct clusters in the xz plane. In [Figure 8.11\(d\)](#), there are three clusters when we consider both the y and z coordinates. One of these clusters consists of the circles; another consists of the points marked by squares. The diamonds and triangles form a single cluster in the yz plane.

These figures illustrate a couple of important facts. First, a set of points—the circles—may not form a cluster in the entire data space, but may form a cluster in a subspace. Second, clusters that exist in the full data space (or even a subspace) show up as clusters in lower-dimensional spaces. The first fact tells us that we need to look in subsets of dimensions to find clusters, while the second fact tells us that many of the clusters we find in subspaces are likely to be “shadows” (projections) of higher-dimensional clusters. The goal is to find the clusters and the dimensions in which they exist, but we are typically not interested in clusters that are projections of higher-dimensional clusters.

CLIQUE

CLIQUE (CLustering In QUEst) is a grid-based clustering algorithm that methodically finds subspace clusters. It is impractical to check each subspace for clusters because the number of such subspaces is exponential in the number of dimensions. Instead, CLIQUE relies on the following property:

Monotonicity property of density-based clusters If a set of points forms a density-based cluster in k dimensions (attributes), then the same set of points is also part of a density-based cluster in all possible subsets of those dimensions.

Consider a set of adjacent, k -dimensional cells that form a cluster; i.e., there is a collection of adjacent cells that have a density above the specified threshold

ξ . A corresponding set of cells in $k-1$ dimensions can be found by omitting one of the k dimensions (attributes). The lower-dimensional cells are still adjacent, and each low-dimensional cell contains all points of the corresponding high-dimensional cell. It can contain additional points as well. Thus, a low-dimensional cell has a density greater than or equal to that of its corresponding high-dimensional cell. Consequently, the low-dimensional cells form a cluster; i.e., the points form a cluster with the reduced set of attributes.

Algorithm 8.5  gives a simplified version of the steps involved in CLIQUE. Conceptually, the CLIQUE algorithm is similar to the *Apriori* algorithm for finding frequent itemsets. See **Chapter 5** .

Algorithm 8.5 CLIQUE.

- 1: Find all the dense areas in the one-dimensional spaces corresponding to each attribute. This is the set of dense one-dimensional cells.
- 2: $k \leftarrow 2$
- 3: **repeat**
- 4: Generate all candidate dense k -dimensional cells from dense $(k-1)$ -dimensional cells.
- 5: Eliminate cells that have fewer than ξ points.
- 6: $k \leftarrow k+1$
- 7: **until** There are no candidate dense k -dimensional cells.
- 8: Find clusters by taking the union of all adjacent, high-density cells.
- 9: Summarize each cluster using a small set of inequalities that describe the attribute ranges of the cells in the cluster.

Strengths and Limitations of CLIQUE

The most useful feature of CLIQUE is that it provides an efficient technique for searching subspaces for clusters. Since this approach is based on the well-known *Apriori* principle from association analysis, its properties are well understood. Another useful feature is CLIQUE's ability to summarize the list of cells that comprises a cluster with a small set of inequalities.

Many limitations of CLIQUE are identical to the previously discussed limitations of other grid-based density schemes. Other limitations are similar to those of the *Apriori* algorithm. Specifically, just as frequent itemsets can share items, the clusters found by CLIQUE can share objects. Allowing clusters to overlap can greatly increase the number of clusters and make interpretation difficult. Another issue is that *Apriori*—like CLIQUE—potentially has exponential time complexity. In particular, CLIQUE will have difficulty if too many dense cells are generated at lower values of k . Raising the density threshold ξ can alleviate this problem. Still another potential limitation of CLIQUE is explored in [Exercise 20](#) on [page 702](#).

8.3.3 DENCLUE: A Kernel-Based Scheme for Density-Based Clustering

DENCLUE (DENsity CLUstEring) is a density-based clustering approach that models the overall density of a set of points as the sum of influence functions associated with each point. The resulting overall density function will have local peaks, i.e., local density maxima, and these local peaks can be used to define clusters in a natural way. Specifically, for each data point, a hill-climbing procedure finds the nearest peak associated with that point, and the set of all

data points associated with a particular peak (called a **local density attractor**) becomes a cluster. However, if the density at a local peak is too low, then the points in the associated cluster are classified as noise and discarded. Also, if a local peak can be connected to a second local peak by a path of data points, and the density at each point on the path is above the minimum density threshold, then the clusters associated with these local peaks are merged. Therefore, clusters of any shape can be discovered.

Example 8.10 (DENCLUE Density).

We illustrate these concepts with [Figure 8.13](#), which shows a possible density function for a one-dimensional data set. Points A–E are the peaks of this density function and represent local density attractors. The dotted vertical lines delineate local regions of influence for the local density attractors. Points in these regions will become center-defined clusters. The dashed horizontal line shows a density threshold, ξ . All points associated with a local density attractor that has a density less than ξ , such as those associated with C, will be discarded. All other clusters are kept. Note that this can include points whose density is less than ξ , as long as they are associated with local density attractors whose density is greater than ξ . Finally, clusters that are connected by a path of points with a density above ξ are combined. Clusters A and B would remain separate, while clusters D and E would be combined.

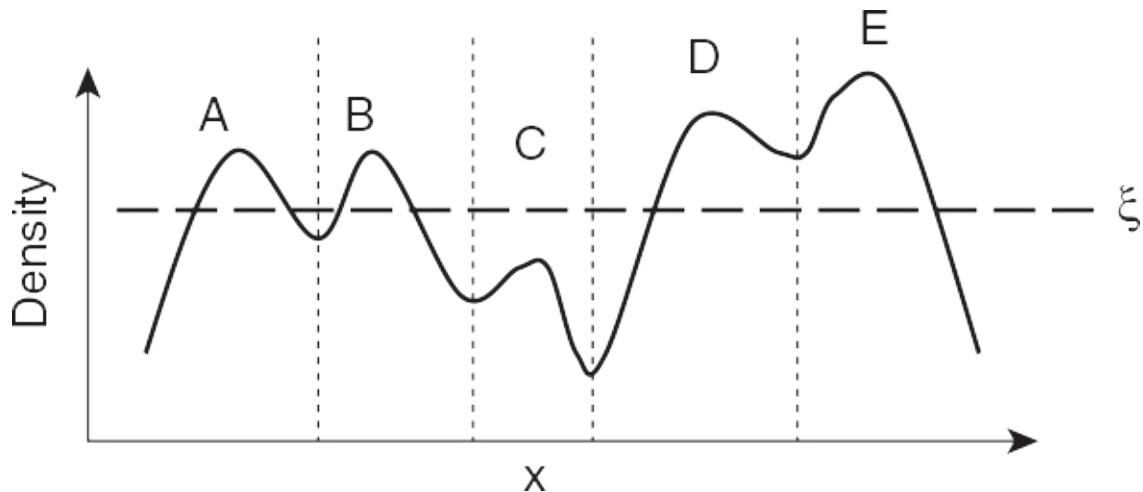


Figure 8.13.

Illustration of DENCLUE density concepts in one dimension.

The high-level details of the DENCLUE algorithm are summarized in

Algorithm 8.6 [□](#). Next, we explore various aspects of DENCLUE in more detail. First, we provide a brief overview of kernel density estimation and then present the grid-based approach that DENCLUE uses for approximating the density.

Algorithm 8.6 DENCLUE algorithm.

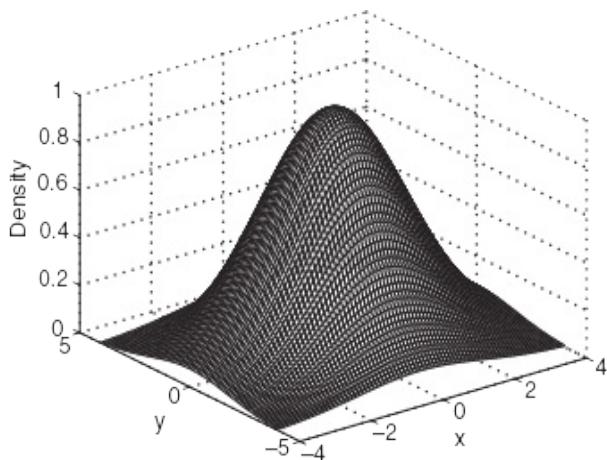
- 1: Derive a density function for the space occupied by the data points.
- 2: Identify the points that are local maxima. (These are the density attractors.)
- 3: Associate each point with a density attractor by moving in the direction of maximum increase in density.
- 4: Define clusters consisting of points associated with a particular density attractor.
- 5: Discard clusters whose density attractor has a density less than a user-specified threshold of ξ .

6: Combine clusters that are connected by a path of points that all have a density of ξ or higher.

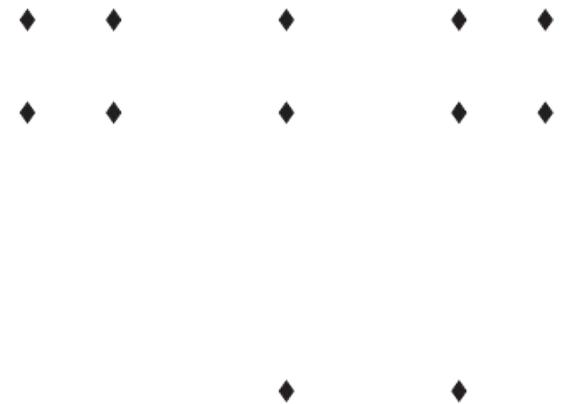
Kernel Density Estimation

DENCLUE is based on a well-developed area of statistics and pattern recognition that is known as **kernel density estimation**. The goal of this collection of techniques (and many other statistical techniques as well) is to describe the distribution of the data by a function. For kernel density estimation, the contribution of each point to the overall density function is expressed by an influence or **kernel function**. The overall density function is simply the sum of the influence functions associated with each point.

Typically, the influence or kernel function is symmetric (the same in all directions) and its value (contribution) decreases as the distance from the point increases. For example, for a particular point, x , the Gaussian function, $K(y)=e^{-\text{distance}(x,y)^2/2\sigma^2}$, is often used as a kernel function. σ is a parameter, analogous to standard deviation, which governs how quickly the influence of a point diminishes with distance. [Figure 8.14\(a\)](#) shows what a Gaussian density function would look like for a single point in two dimensions, while [Figures 8.14\(c\)](#) and [8.14\(d\)](#) show the overall density function produced by applying the Gaussian influence function to the set of points shown in [Figure 8.14\(b\)](#).



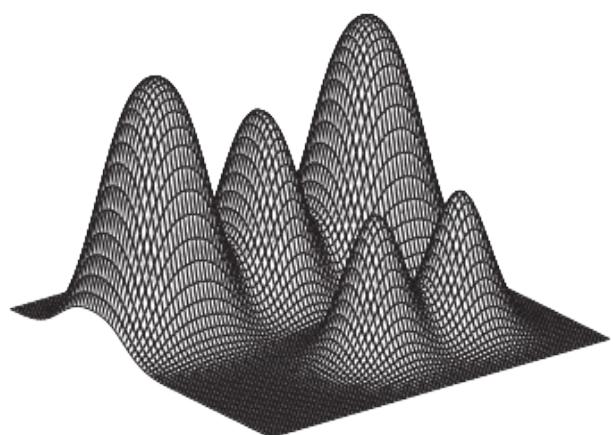
(a) Gaussian kernel



(b) Set of 12 points.



(c) Overall density—gray scale plot.



(d) Overall density—surface plot.

Figure 8.14.

Example of the Gaussian influence (kernel) function and an overall density function.

Implementation Issues

Computation of kernel density can be quite expensive, and DENCLUE uses a number of approximations to implement its basic approach efficiently. First, it explicitly computes density only at data points. However, this still would result in an $O(m^2)$ time complexity because the density at each point is a function of

the density contributed by every point. To reduce the time complexity, DENCLUE uses a grid-based implementation to efficiently define neighborhoods and thus limit the number of points that need to be considered to define the density at a point. First, a preprocessing step creates a set of grid cells. Only occupied cells are created, and these cells and their related information can be efficiently accessed via a search tree. Then, when computing the density of a point and finding its nearest density attractor, DENCLUE considers only the points in the neighborhood; i.e., points in the same cell and in cells that are connected to the point's cell. While this approach can sacrifice some accuracy with respect to density estimation, computational complexity is greatly reduced.

Strengths and Limitations of DENCLUE

DENCLUE has a solid theoretical foundation because it is based on the concept of kernel density estimation, which is a well-developed area of statistics. For this reason, DENCLUE provides a more flexible and potentially more accurate way to compute density than other grid-based clustering techniques and DBSCAN. (DBSCAN is a special case of DENCLUE.) An approach based on kernel density functions is inherently computationally expensive, but DENCLUE employs grid-based techniques to address such issues. Nonetheless, DENCLUE can be more computationally expensive than other density-based clustering techniques. Also, the use of a grid can adversely affect the accuracy of the density estimation, and it makes DENCLUE susceptible to problems common to grid-based approaches; e.g., the difficulty of choosing the proper grid size. More generally, DENCLUE shares many of the strengths and limitations of other density-based approaches. For instance, DENCLUE is good at handling noise and outliers and it can find clusters of different shapes and size, but it has trouble with high-dimensional data and data that contains clusters of widely different densities.

8.4 Graph-Based Clustering

Section 7.3 discussed a number of clustering techniques that took a graph-based view of data, in which data objects are represented by nodes and the proximity between two data objects is represented by the weight of the edge between the corresponding nodes. This section considers some additional graph-based clustering algorithms that use a number of key properties and characteristics of graphs. The following are some key approaches, different subsets of which are employed by these algorithms.

1. Sparsify the proximity graph to keep only the connections of an object with its nearest neighbors. This sparsification is useful for handling noise and outliers. It also allows the use of highly efficient graph partitioning algorithms that have been developed for sparse graphs.
2. Define a similarity measure between two objects based on the number of nearest neighbors that they share. This approach, which is based on the observation that an object and its nearest neighbors usually belong to the same class, is useful for overcoming problems with high dimensionality and clusters of varying density.
3. Define core objects and build clusters around them. To do this for graph-based clustering, it is necessary to introduce a notion of density-based on a proximity graph or a sparsified proximity graph. As with DBSCAN, building clusters around core objects leads to a clustering technique that can find clusters of differing shapes and sizes.
4. Use the information in the proximity graph to provide a more sophisticated evaluation of whether two clusters should be merged. Specifically, two clusters are merged only if the resulting cluster will have characteristics similar to the original two clusters.

We begin by discussing the sparsification of proximity graphs, providing three examples of techniques whose approach to clustering is based solely on this technique: MST, which is equivalent to the single link clustering algorithm, Opossum, and spectral clustering. We then discuss Chameleon, a hierarchical clustering algorithm that uses a notion of self-similarity to determine if clusters should be merged. We next define Shared Nearest Neighbor (SNN) similarity, a new similarity measure, and introduce the Jarvis-Patrick clustering algorithm, which uses this similarity. Finally, we discuss how to define density and core objects based on SNN similarity and introduce an SNN density-based clustering algorithm, which can be viewed as DBSCAN with a new similarity measure.

8.4.1 Sparsification

The m by m proximity matrix for m data points can be represented as a dense graph in which each node is connected to all others and the weight of the edge between any pair of nodes reflects their pairwise proximity. Although every object has some level of similarity to every other object, for most data sets, objects are highly similar to a small number of objects and weakly similar to most other objects. This property can be used to sparsify the proximity graph (matrix), by setting many of these low-similarity (high-dissimilarity) values to 0 before beginning the actual clustering process. The sparsification may be performed, for example, by breaking all links that have a similarity (dissimilarity) below (above) a specified threshold or by keeping only links to the k -nearest neighbors of point. This latter approach creates what is called a **k -nearest neighbor graph**.

Sparsification has several beneficial effects:

- **Data size is reduced.** The amount of data that needs to be processed to cluster the data is drastically reduced. Sparsification can often eliminate more than 99% of the entries in a proximity matrix. As a result, the size of problems that can be handled is increased.
- **Clustering often works better.** Sparsification techniques keep the connections to their nearest neighbors of an object while breaking the connections to more distant objects. This is in keeping with the **nearest neighbor principle** that the nearest neighbors of an object tend to belong to the same class (cluster) as the object itself. This reduces the impact of noise and outliers and sharpens the distinction between clusters.
- **Graph partitioning algorithms can be used.** There has been a considerable amount of work on heuristic algorithms for finding mincut partitionings of sparse graphs, especially in the areas of parallel computing and the design of integrated circuits. Sparsification of the proximity graph makes it possible to use graph partitioning algorithms for the clustering process. For example, Opossum and Chameleon use graph partitioning.

Sparsification of the proximity graph should be regarded as an initial step before the use of actual clustering algorithms. In theory, a perfect sparsification could leave the proximity matrix split into connected components corresponding to the desired clusters, but in practice, this rarely happens. It is easy for a single edge to link two clusters or for a single cluster to be split into several disconnected subclusters. As we shall see when we discuss Jarvis-Patrick and SNN density-based clustering, the sparse proximity graph is often modified to yield a new proximity graph. This new proximity graph can again be sparsified. Clustering algorithms work with the proximity graph that is the result of all these preprocessing steps. This process is summarized in [Figure 8.15](#).

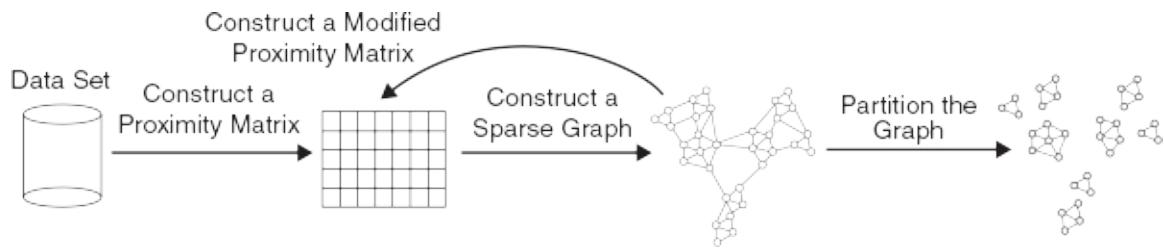


Figure 8.15.

Ideal process of clustering using sparsification.

8.4.2 Minimum Spanning Tree (MST) Clustering

In [Section 7.3](#), where we described agglomerative hierarchical clustering techniques, we mentioned that divisive hierarchical clustering algorithms also exist. We saw an example of one such technique, bisecting K-means, in [Section 7.2.3](#). Another divisive hierarchical technique, **MST**, starts with the minimum spanning tree of the proximity graph and can be viewed as an application of sparsification for finding clusters. We briefly describe this algorithm. Interestingly, this algorithm also produces the same clustering as single link agglomerative clustering. See [Exercise 13](#) on [page 700](#).

A **minimum spanning tree** of a graph is a subgraph that (1) has no cycles, i.e., is a tree, (2) contains all the nodes of the graph, and (3) has the minimum total edge weight of all possible spanning trees. The terminology, minimum spanning tree, assumes that we are working only with dissimilarities or distances, and we will follow this convention. This is not a limitation, however, since we can convert similarities to dissimilarities or modify the notion of a minimum spanning tree to work with similarities. An example of a minimum spanning tree for some two-dimensional points is shown in [Figure 8.16](#).

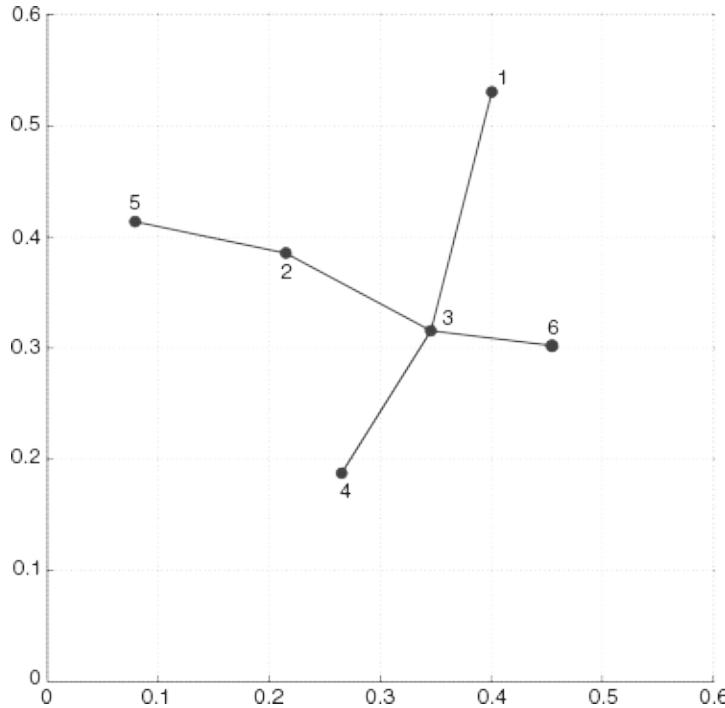


Figure 8.16.

Minimum spanning tree for a set of six two-dimensional points.

The MST divisive hierarchical algorithm is shown in [Algorithm 8.7](#). The first step is to find the MST of the original dissimilarity graph. Note that a minimum spanning tree can be viewed as a special type of sparsified graph. Step 3 can also be viewed as graph sparsification. Hence, MST can be viewed as a clustering algorithm based on the sparsification of the dissimilarity graph.

Algorithm 8.7 MST divisive hierarchical clustering algorithm.

- 1: Compute a minimum spanning tree for the dissimilarity graph.
- 2: **repeat**
- 3: Create a new cluster by breaking the link corresponding to the largest dissimilarity.

4: until Only singleton clusters remain.

8.4.3 OPOSSUM: Optimal Partitioning of Sparse Similarities Using METIS

OPOSSUM is a clustering technique for clustering sparse, high-dimensional data, e.g., document or market basket data. Like MST, it performs clustering based on the sparsification of a proximity graph. However, OPOSSUM uses the METIS algorithm, which was specifically created for partitioning sparse graphs. The steps of OPOSSUM are given in [Algorithm 8.8](#).

The similarity measures used are those appropriate for sparse, high-dimensional data, such as the extended Jaccard measure or the cosine measure. The METIS graph partitioning program partitions a sparse graph into k distinct components, where k is a user-specified parameter, in order to (1) minimize the weight of the edges (the similarity) between components and (2) fulfill a balance constraint. OPOSSUM uses one of the following two balance constraints: (1) the number of objects in each cluster must be roughly the same, or (2) the sum of the attribute values must be roughly the same. The second constraint is useful when, for example, the attribute values represent the cost of an item.

Algorithm 8.8 OPOSSUM clustering algorithm.

1: Compute a sparsified similarity graph.

2: Partition the similarity graph into k distinct components (clusters) using METIS.

Strengths and Weaknesses

OPOSSUM is simple and fast. It partitions the data into roughly equal-sized clusters, which, depending on the goal of the clustering, can be viewed as an advantage or a disadvantage. Because they are constrained to be of roughly equal size, clusters can be broken or combined. However, if OPOSSUM is used to generate a large number of clusters, then these clusters are typically relatively pure pieces of larger clusters. Indeed, OPOSSUM is similar to the initial step of the Chameleon clustering routine, which is discussed next.

8.4.4 Chameleon: Hierarchical Clustering with Dynamic Modeling

Agglomerative hierarchical clustering techniques operate by merging the two most similar clusters, where the definition of cluster similarity depends on the particular algorithm. Some agglomerative algorithms, such as group average, base their notion of similarity on the strength of the connections between the two clusters (e.g., the pairwise similarity of points in the two clusters), while other techniques, such as the single link method, use the closeness of the clusters (e.g., the minimum distance between points in different clusters) to measure cluster similarity. Although there are two basic approaches, using only one of these two approaches can lead to mistakes in merging clusters. Consider [Figure 8.17](#), which shows four clusters. If we use the closeness of clusters (as measured by the closest two points in different clusters) as our

merging criterion, then we would merge the two circular clusters, (c) and (d), which almost touch, instead of the rectangular clusters, (a) and (b), which are separated by a small gap. However, intuitively, we should have merged rectangular clusters, (a) and (b). [Exercise 15](#) on page 700 asks for an example of a situation in which the strength of connections likewise leads to an unintuitive result.



Figure 8.17.

Situation in which closeness is not the appropriate merging criterion. © 1999, IEEE

Another problem is that most clustering techniques have a global (static) model of clusters. For instance, K-means assumes that the clusters will be globular, while DBSCAN defines clusters based on a single density threshold. Clustering schemes that use such a global model cannot handle cases in which cluster characteristics, such as size, shape, and density, vary widely between clusters. As an example of the importance of the local (dynamic) modeling of clusters, consider [Figure 8.18](#). If we use the closeness of clusters to determine which pair of clusters should be merged, as would be the case if we used, for example, the single link clustering algorithm, then we would merge clusters (a) and (b). However, we have not taken into account the characteristics of each individual cluster. Specifically, we have ignored the density of the individual clusters. For clusters (a) and (b), which are relatively dense, the distance between the two clusters is significantly larger than the distance between a point and its nearest neighbors within the same cluster.

This is not the case for clusters (c) and (d), which are relatively sparse. Indeed, when clusters (c) and (d) are merged, they yield a cluster that seems more similar to the original clusters than the cluster that results from merging clusters (a) and (b).

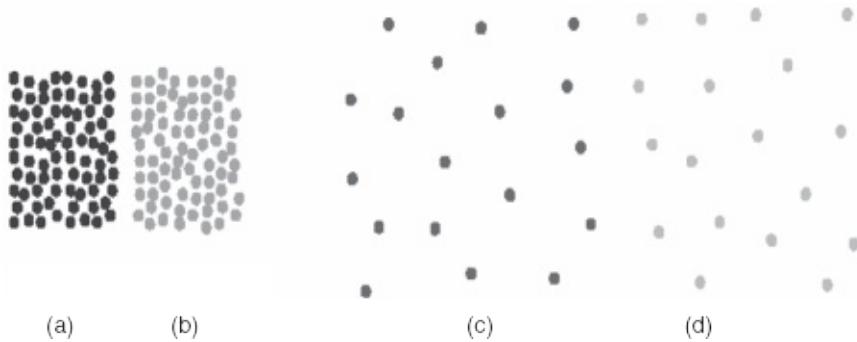


Figure 8.18.

Illustration of the notion of relative closeness. © 1999, IEEE

Chameleon is an agglomerative clustering algorithm that addresses the issues of the previous two paragraphs. It combines an initial partitioning of the data, using an efficient graph partitioning algorithm, with a novel hierarchical clustering scheme that uses the notions of closeness and interconnectivity, together with the local modeling of clusters. The key idea is that two clusters should be merged only if the resulting cluster is similar to the two original clusters. Self-similarity is described first, and then the remaining details of the Chameleon algorithm are presented.

Deciding Which Clusters to Merge

The agglomerative hierarchical clustering techniques considered in [Section 7.3](#) repeatedly combine the two closest clusters and are principally distinguished from one another by the way they define cluster proximity. In contrast, Chameleon aims to merge the pair of clusters that results in a cluster that is most similar to the original pair of clusters, as measured by closeness

and interconnectivity. Because this approach depends only on the pair of clusters and not on a global model, Chameleon can handle data that contains clusters with widely different characteristics.

Following are more detailed explanations of the properties of closeness and interconnectivity. To understand these properties, it is necessary to take a proximity graph viewpoint and to consider the number of the links and the strength of those links among points within a cluster and across clusters.

- **Relative Closeness (RC)** is the absolute closeness of two clusters normalized by the internal closeness of the clusters. Two clusters are combined only if the points in the resulting cluster are almost as close to each other as in each of the original clusters. Mathematically,

$$RC(C_i, C_j) = \frac{S^-EC(C_i, C_j)m_i m_j}{m_i S^-EC(C_i) + m_j S^-EC(C_j)}, \quad (8.17)$$

where m_i and m_j are the sizes of clusters C_i and C_j , respectively; $S^-EC(C_i, C_j)$ is the average weight of the edges (of the k -nearest neighbor graph) that connect clusters C_i and C_j ; $S^-EC(C_i)$ is the average weight of edges if we bisect cluster C_i ; and $S^-EC(C_j)$ is the average weight of edges if we bisect cluster C_j . (EC stands for edge cut.) [Figure 8.18](#) illustrates the notion of relative closeness. As discussed previously, while clusters (a) and (b) are closer in absolute terms than clusters (c) and (d), this is not true if the nature of the clusters is taken into account.

- **Relative Interconnectivity (RI)** is the absolute interconnectivity of two clusters normalized by the internal connectivity of the clusters. Two clusters are combined if the points in the resulting cluster are almost as strongly connected as points in each of the original clusters. Mathematically,

$$RI(C_i, C_j) = \frac{EC(C_i, C_j)}{2(EC(C_i) + EC(C_j))}, \quad (8.18)$$

where $EC(C_i, C_j)$ is the sum of the edges (of the k -nearest neighbor graph) that connect clusters C_i and C_j ; $EC(C_i)$ is the minimum sum of the cut edges if we bisect cluster C_i ; and $EC(C_j)$ is the minimum sum of the cut edges if we bisect cluster C_j . **Figure 8.19** illustrates the notion of relative interconnectivity. The two circular clusters, (c) and (d), have more connections than the rectangular clusters, (a) and (b). However, merging (c) and (d) produces a cluster that has connectivity quite different from that of (c) and (d). In contrast, merging (a) and (b) produces a cluster with connectivity very similar to that of (a) and (b).

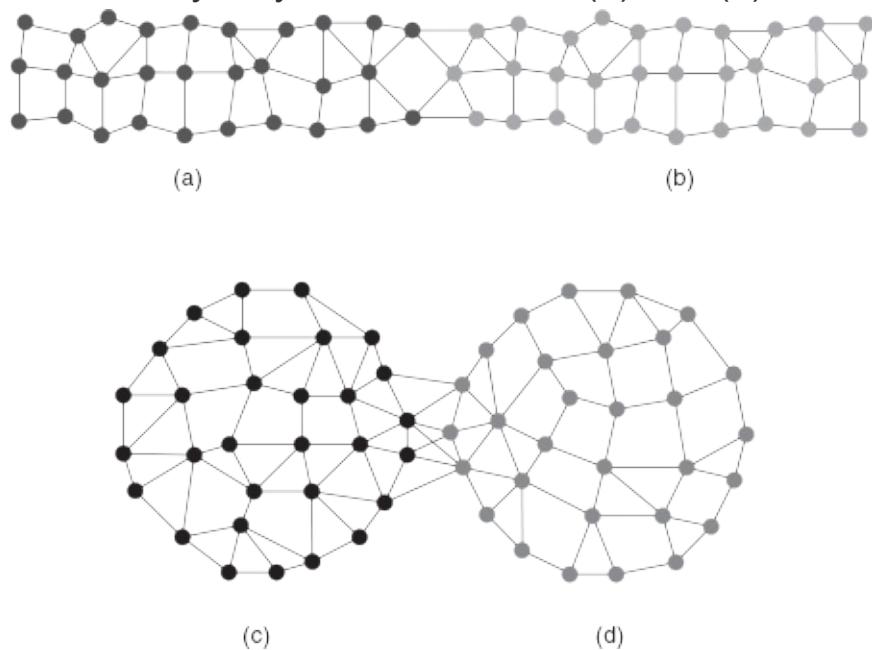


Figure 8.19.

Illustration of the notion of relative interconnectedness. © 1999, IEEE

RI and RC can be combined in many different ways to yield an overall measure of self-similarity. One approach used in Chameleon is to merge the pair of clusters that maximizes $RI(C_i, C_j) * RC(C_i, C_j)^\alpha$, where α is a user-specified parameter that is typically greater than 1.

Chameleon Algorithm

Chameleon consists of three key steps: sparsification, graph partitioning, and hierarchical clustering. [Algorithm 8.9](#) and [Figure 8.20](#) describe these steps.

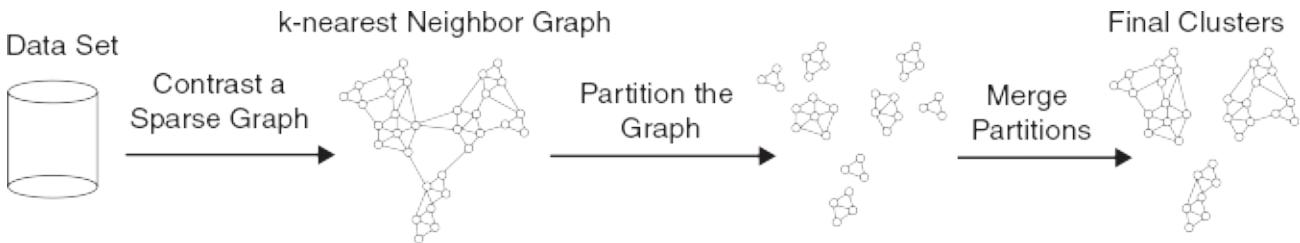


Figure 8.20.

Overall process by which Chameleon performs clustering. © 1999, IEEE

Algorithm 8.9 Chameleon algorithm.

- 1: Build a k -nearest neighbor graph.
- 2: Partition the graph using a multilevel graph partitioning algorithm.
- 3: **repeat**
- 4: Merge the clusters that best preserve the cluster self-similarity with respect to relative interconnectivity and relative closeness.
- 5: **until** No more clusters can be merged.

Sparsification

The first step in Chameleon is to generate a k -nearest neighbor graph. Conceptually, such a graph is derived from the proximity graph, and it contains links only between a point and its k -nearest neighbors, i.e., the points to which it is closest. As mentioned, working with a sparsified proximity graph

instead of the full proximity graph can significantly reduce the effects of noise and outliers and improve computational efficiency.

Graph Partitioning

Once a sparsified graph has been obtained, an efficient multilevel graph partitioning algorithm, such as METIS (see Bibliographic Notes), can be used to partition the data set. Chameleon starts with an all-inclusive graph (cluster) and then bisects the largest current subgraph (cluster) until no cluster has more than `MIN_SIZE` points, where `MIN_SIZE` is a user-specified parameter.

This process results in a large number of roughly equally sized groups of well-connected vertices (highly similar data points). The goal is to ensure that each partition contains objects mostly from one true cluster.

Agglomerative Hierarchical Clustering

As discussed previously, Chameleon merges clusters based on the notion of self-similarity. Chameleon can be parameterized to merge more than one pair of clusters in a single step and to stop before all objects have been merged into a single cluster.

Complexity

Assume that m is the number of data points and p is the number of partitions. Performing an agglomerative hierarchical clustering of the p partitions obtained from the graph partitioning requires time $O(p^2 \log p)$. (See [Section 7.3.1](#).) The amount of time required for partitioning the graph is $O(mp + m \log m)$. The time complexity of graph sparsification depends on how much time it takes to build the k -nearest neighbor graph. For low-dimensional data, this takes $O(m \log m)$ time if a k-d tree or a similar type of data structure

is used. Unfortunately, such data structures only work well for low-dimensional data sets, and thus, for high-dimensional data sets, the time complexity of the sparsification becomes $O(m^2)$. Because only the k -nearest neighbor list needs to be stored, the space complexity is $O(km)$ plus the space required to store the data.

Example 8.11.

Chameleon was applied to two data sets that clustering algorithms such as K-means and DBSCAN have difficulty clustering. The results of this clustering are shown in [Figure 8.21](#). The clusters are identified by the shading of the points. In [Figure 8.21\(a\)](#), the two clusters are irregularly shaped and quite close to each other. Also, noise is present. In [Figure 8.21\(b\)](#), the two clusters are connected by a bridge, and again, noise is present. Nonetheless, Chameleon identifies what most people would identify as the natural clusters. Chameleon has specifically been shown to be very effective for clustering spatial data. Finally, notice that Chameleon does not discard noise points, as do other clustering schemes, but instead assigns them to the clusters.



Figure 8.21.

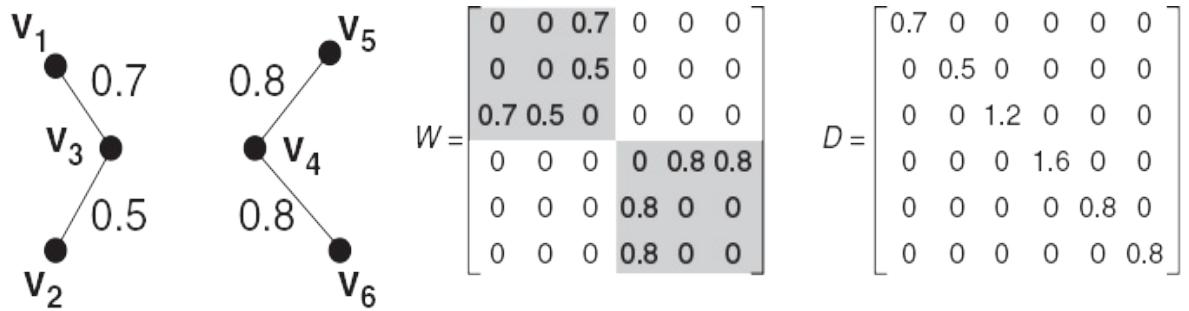
Chameleon applied to cluster a pair of two-dimensional sets of points. © 1999, IEEE

Strengths and Limitations

Chameleon can effectively cluster spatial data, even though noise and outliers are present and the clusters are of different shapes, sizes, and density. Chameleon assumes that the groups of objects produced by the sparsification and graph partitioning process are subclusters; i.e., that most of the points in a partition belong to the same true cluster. If not, then agglomerative hierarchical clustering will only compound the errors because it can never separate objects that have been wrongly put together. (See the discussion in [Section 7.3.4](#).) Thus, Chameleon has problems when the partitioning process does not produce subclusters, as is often the case for high-dimensional data.

8.4.5 Spectral Clustering

Spectral clustering is an elegant graph partitioning approach that exploits properties of the similarity graph to determine the cluster partitions. Specifically, it examines the graph's spectrum, i.e., eigenvalues and eigenvectors associated with the adjacency matrix of the graph, to identify the natural clusters of the data. To motivate the ideas behind this approach, consider the similarity graph shown in [Figure 8.22](#) for a data set that contains 6 data points. The link weights in the graph are computed based on some similarity measure, with a threshold applied to remove links with low similarity values. The sparsification produces a graph with two connected components, which trivially represent the two clusters in the data, $\{v_1, v_2, v_3\}$ and $\{v_4, v_5, v_6\}$.



$$W = \begin{bmatrix} 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0.7 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8 & 0.8 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.8 \end{bmatrix}$$

$$L = \begin{bmatrix} 0.7 & 0 & -0.7 & 0 & 0 & 0 \\ 0 & 0.5 & -0.5 & 0 & 0 & 0 \\ -0.7 & -0.5 & 1.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.6 & -0.8 & -0.8 \\ 0 & 0 & 0 & -0.8 & 0.8 & 0 \\ 0 & 0 & 0 & -0.8 & 0 & 0.8 \end{bmatrix} \quad \Lambda = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.58 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.4 \end{bmatrix} \quad V = \begin{bmatrix} 0.58 & 0 & 0.64 & 0 & -0.50 & 0 \\ 0.58 & 0 & -0.76 & 0 & -0.31 & 0 \\ 0.58 & 0 & 0.11 & 0 & 0.81 & 0 \\ 0 & -0.58 & 0 & 0 & 0 & -0.82 \\ 0 & -0.5 & 0 & -0.71 & 0 & 0.41 \\ 0 & -0.5 & 0 & 0.71 & 0 & 0.41 \end{bmatrix}$$

Figure 8.22.

Example of a similarity graph with two connected components along with its weighted adjacency matrix (\mathbf{W}), graph Laplacian matrix (\mathbf{L}), and eigendecomposition.

The top right-hand panel of the figure also shows the weighted adjacency matrix of the graph, denoted as \mathbf{W} , and a diagonal matrix, \mathbf{D} , whose diagonal elements correspond to the sum of the weights of the links incident to each node in the graph, i.e.,

$$D_{ij} = \begin{cases} \sum_k W_{ik}, & \text{if } i=j \\ 0, & \text{otherwise.} \end{cases}$$

Note that the rows and columns of the weighted adjacency matrix have been ordered in such a way that nodes belonging to the same connected component are next to each other. With this ordering, the matrix \mathbf{W} has a block structure of the form

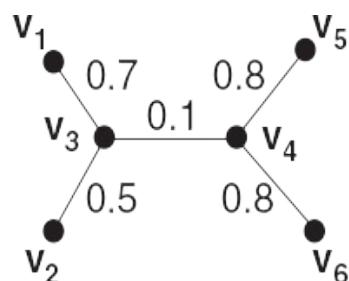
$$\mathbf{W} = (W_1 \ 0 \ 0 \ W_2),$$

in which the off-diagonal blocks are matrices of zero values since there are no links connecting a node from the first connected component to a node from the second connected component. Indeed, if the sparse graph contains k connected components, its weighted adjacency matrix can be re-ordered into the following block diagonal form:

$$W = (W_1 \cdots 0 \cdots W_2 \cdots 0 \cdots \cdots \cdots 0 \cdots 0 \cdots W_k), \quad (8.19)$$

This example suggests the possibility of identifying the inherent clusters of a data set by examining the block structure of its weighted adjacency matrix.

Unfortunately, unless the clusters are well-separated, the adjacency matrices associated with most similarity graphs are not in block diagonal form. For example, consider the graph shown in [Figure 8.23](#), in which there is a link between nodes v_3 and v_4 , with a low similarity value. If we are interested in generating two clusters, we could break the weakest link, located between (v_3, v_4) , to split the graph into two partitions. Because there is only one connected component in the graph, the block structure in W is harder to discern.



$$W = \begin{bmatrix} 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0.7 & 0.5 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.8 & 0.8 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.8 \end{bmatrix}$$

$$L = \begin{bmatrix} 0.7 & 0 & -0.7 & 0 & 0 & 0 \\ 0 & 0.5 & -0.5 & 0 & 0 & 0 \\ -0.7 & -0.5 & 1.3 & -0.1 & 0 & 0 \\ 0 & 0 & -0.1 & 1.7 & -0.8 & -0.8 \\ 0 & 0 & 0 & -0.8 & 0.8 & 0 \\ 0 & 0 & 0 & -0.8 & 0 & 0.8 \end{bmatrix} \quad \Lambda = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.06 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.58 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.80 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.47 \end{bmatrix} \quad V = \begin{bmatrix} 0.41 & \textbf{-0.41} & 0.65 & 0 & 0.48 & -0.04 \\ 0.41 & \textbf{-0.43} & -0.75 & 0 & 0.29 & 0.03 \\ 0.41 & \textbf{-0.38} & 0.11 & 0 & -0.81 & 0.10 \\ 0.41 & \textbf{0.38} & 0 & 0 & 0.08 & -0.82 \\ 0.41 & \textbf{0.42} & 0 & -0.71 & 0.06 & 0.39 \\ 0.41 & \textbf{0.42} & 0 & 0.71 & 0.06 & 0.39 \end{bmatrix}$$

Figure 8.23.

Example of a similarity graph with a single connected component along with its weighted adjacency matrix (**W**), graph Laplacian matrix (**L**), and eigendecomposition.

Fortunately, there is a more objective way to create the cluster partitions by considering the graph spectrum. First, we need to compute the graph Laplacian matrix, which is formally defined as follows:

$$L=D-W \quad (8.20)$$

The graph Laplacian matrices for the examples shown in [Figures 8.22](#) and [8.23](#) are depicted in the bottom left panel of both diagrams. The matrix has several notable properties:

1. It is a symmetric matrix since both **W** and **D** are symmetric.
2. It is a positive semi-definite matrix, which means $v^T L v \geq 0$ for any input vector **v**.
3. All eigenvalues of **L** must be non-negative. The eigenvalues and eigenvectors for the graphs shown in [Figure 8.22](#) and [8.23](#) are denoted in the diagrams as **Λ** and **V**, respectively. Note that the eigenvalues of the graph Laplacian matrix are given by the diagonal elements of **Λ**.
4. The smallest eigenvalue of **L** is zero, with the corresponding eigenvector **e**, which is a vector of 1s. This is because
$$We = (W_{11}W_{12}\cdots W_{1n}W_{21}W_{22}\cdots W_{2n}\cdots\cdots\cdots W_{n1}W_{n2}\cdots W_{nn})(11\cdots 1) = (\sum_j W_{1j}\sum_j W_{2j}\cdots \sum_j W_{nj})De = (\sum_j W_{1j}0\cdots 00\sum_j W_{2j}\cdots 0\cdots\cdots\cdots 00\cdots \sum_j W_{nj})(11\cdots 1) = (\sum_j W_{1j}\sum_j W_{2j}\cdots \sum_j W_{nj})$$

Thus, $We=De$, which is equivalent to $(D-W)e=0$. This can be simplified into the eigenvalue equation $Le=0e$ since $L=D-W$.

5. A graph with k connected components has an adjacency matrix \mathbf{W} in block diagonal form as shown in [Equation 8.19](#). Its graph Laplacian matrix also has a block diagonal form
- $$\mathbf{L} = (\mathbf{L}_1 \cdots \mathbf{0} \mathbf{0} \mathbf{L}_2 \cdots \mathbf{0} \cdots \cdots \cdots \mathbf{0} \mathbf{0} \cdots \mathbf{L}_k),$$

In addition, its graph Laplacian matrix has k eigenvalues of zeros, with the corresponding eigenvectors

$$(\mathbf{e}_1 \mathbf{0} \cdots \mathbf{0}), (\mathbf{0} \mathbf{e}_2 \cdots \mathbf{0}), \cdots, (\mathbf{0} \cdots \mathbf{0} \mathbf{e}_k),$$

where the \mathbf{e}_i 's are vectors of 1's and $\mathbf{0}$'s are vectors of 0's. For example, the graph shown in [Figure 8.22](#) contains two connected components, which is why its graph Laplacian matrix has two eigenvalues of zeros. More importantly, its first two eigenvectors (normalized to unit length),

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow (0.5800 \ 0.5800 \ 0.5800 \ -0.580 \ -0.580 \ -0.58),$$

corresponding to the first two columns in \mathbf{V} , provide information about the cluster membership of each node. A node that belongs to the first cluster has a positive value in its first eigenvector and a zero value in its second eigenvector, whereas a node that belongs to the second cluster has a zero value in the first eigenvector and a negative value in the second eigenvector.

The graph shown in [Figure 8.23](#) has one eigenvalue of zero because it has only one connected component. Nevertheless, if we examine the first two eigenvectors of its graph Laplacian matrix

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow (0.41 \ -0.41 \ 0.41 \ -0.43 \ 0.41 \ -0.38 \ 0.41 \ 0.38 \ 0.41 \ 0.42 \ 0).$$

the graph can be easily split into two clusters since the set of nodes $\{v_1, v_2, v_3\}$ has a negative value in the second eigenvector whereas $\{v_4, v_5, v_6\}$ has a

positive value in the second eigenvector. In short, the eigenvectors of the graph Laplacian matrix contain information that can be used to partition the graph into its underlying components. However, instead of manually checking the eigenvectors, it is common practice to apply a simple clustering algorithm such as K-means to help extract the clusters from the eigenvectors. A summary of the spectral clustering algorithm is given in [Algorithm 8.10](#).

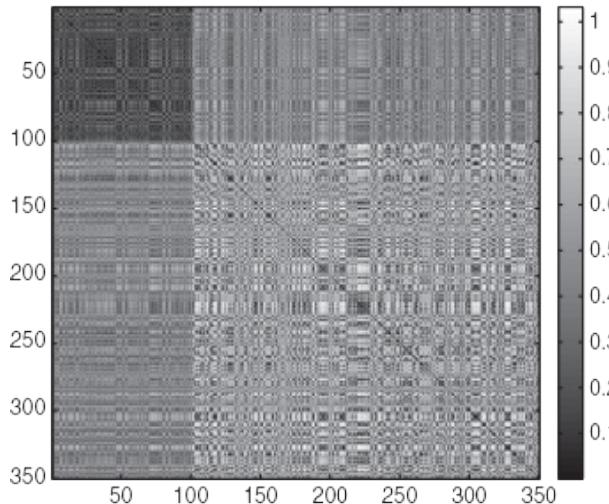
Algorithm 8.10 Spectral clustering algorithm.

- 1: Create a sparsified similarity graph G .
- 2: Compute the graph Laplacian for G , \mathbf{L} (see [Equation \(8.20\)](#)).
- 3: Create a matrix \mathbf{V} from the first k eigenvectors of \mathbf{L} .
- 4: Apply K-means clustering on \mathbf{V} to obtain the k clusters.

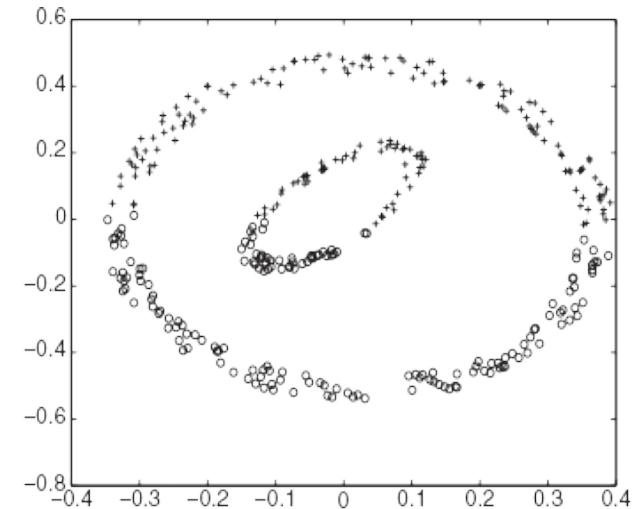
Example 8.12.

Consider the two-dimensional ring data shown in [Figure 8.24\(b\)](#), which contains 350 data points. The first 100 points belong to the inner ring while the remaining 250 points belong to the outer ring. A heat map showing the Euclidean distance between every pair of points is depicted in [Figure 8.24\(a\)](#). While the points in the inner ring are relatively close to each other, those located in the outer ring can be quite far from each other. As a result, standard clustering algorithms such as K-means perform poorly on the data. In contrast, applying spectral clustering on the sparsified similarity graph can produce the correct clustering results (see [Figure 8.24\(d\)](#)). Here, the similarity between points is calculated using the Gaussian radial basis function and the graph is sparsified by choosing the 10-nearest neighbors for each data point. The sparsification reduces the

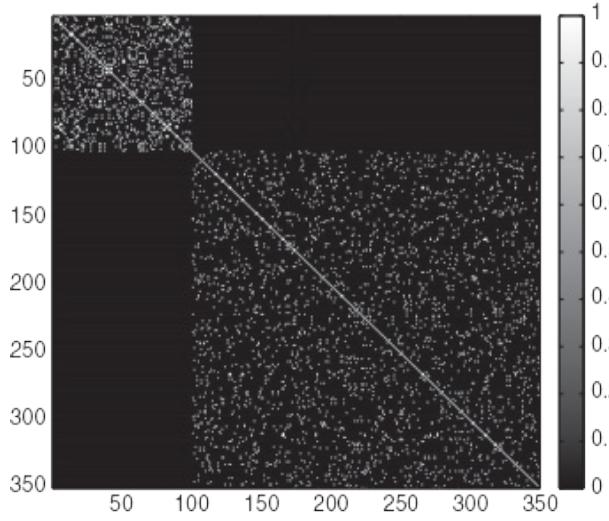
similarity between a data point located in the inner ring and a corresponding point in the outer ring, which enables spectral clustering to effectively partition the data set into two clusters.



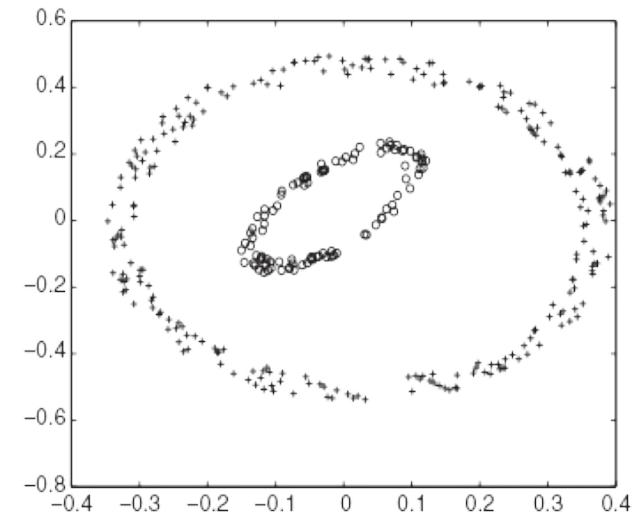
(a) Heat map of Euclidean distance.



(b) Results of K-means clustering.



(c) Heat map of sparsified similarity.



(d) Results of spectral clustering.

Figure 8.24.

Application of K-means and spectral clustering to a two-dimensional ring data.

Relationship between Spectral Clustering and Graph Partitioning

The objective of graph partitioning is to break the weak links in a graph until the desired number of cluster partitions is obtained. One way to assess the quality of the partitions is by summing up the weights of the links that were removed. The resulting measure is known as graph cut. Unfortunately, minimizing the graph cut of the partitions alone is insufficient as it tends to produce clusters with highly imbalanced sizes. For example, consider the graph shown in [Figure 8.25](#). Suppose we are interested in partitioning the graph into two connected components. The graph cut measure prefers to break the link between v_4 and v_5 because it has the lowest weight.

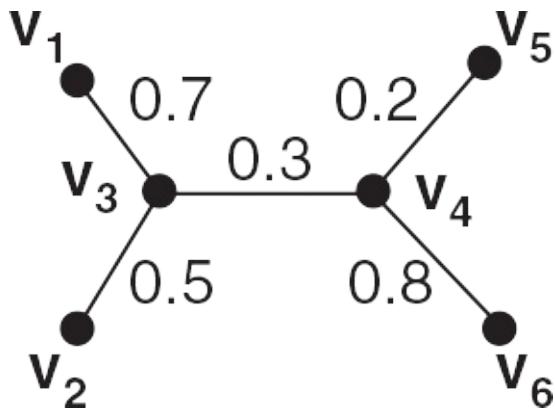


Figure 8.25.

Example to illustrate the limitation of using graph cut as evaluation measure for graph partitioning.

Unfortunately, such a split would create one cluster with a single isolated node and another cluster containing all the remaining nodes. To overcome this limitation, alternative measures have been proposed including

$$\text{Ratio cut}(C_1, C_2, \dots, C_k) = 12 \sum_{i=1}^k \sum_{p \in C_i, q \notin C_i} W_{pq} / |C_i|,$$

where C_1, C_2, \dots, C_k denote the cluster partitions. The numerator represents the sum of the weights of the broken links, i.e., the graph cut, while the denominator represents the size of each cluster partition. Such a measure can be used to ensure that the resulting clusters are more balanced in terms of their sizes. More importantly, it can be shown that minimizing the ratio cut for a graph is equivalent to finding a cluster membership matrix \mathbf{Y} that minimizes the expression $\text{Tr}[\mathbf{Y}^T \mathbf{L} \mathbf{Y}]$, where $\text{Tr}[\cdot]$ denotes the trace of a matrix and \mathbf{L} is the graph Laplacian, subject to the constraint $\mathbf{Y}^T \mathbf{Y} = \mathbf{I}$. By relaxing the requirement that \mathbf{Y} is a binary matrix, we can use the Lagrange multiplier method to solve the optimization problem.

$$\text{Lagrangian}, \mathcal{L} = \text{Tr}[\mathbf{Y}^T \mathbf{L} \mathbf{Y}] - \lambda (\text{Tr}[\mathbf{Y}^T \mathbf{Y} - \mathbf{I}]) \partial \mathcal{L} / \partial \mathbf{Y} = \mathbf{L} \mathbf{Y} - \lambda \mathbf{Y} = 0 \Rightarrow \mathbf{L} \mathbf{Y} = \lambda \mathbf{Y}$$

In other words, an approximate solution to the ratio cut minimization problem can be obtained by finding the eigenvectors of the graph Laplacian matrix, which is exactly the approach used by spectral clustering.

Strengths and Limitations

As shown in [Example 8.12](#), the strength of spectral clustering lies in its ability to detect clusters of varying sizes and shapes. However, the clustering performance depends on how the similarity graph is created and sparsified. In particular, tuning the parameters of the similarity function (e.g., Gaussian radial basis function) to produce an appropriate sparse graph for spectral clustering can be quite a challenge. The time complexity of the algorithm depends on how fast the eigenvectors of the graph Laplacian matrix can be computed. Efficient eigensolvers for sparse matrices are available, e.g., those based on Krylov subspace methods, especially when the number of clusters chosen is small. The storage complexity is $O(N^2)$, though it can be significantly reduced using a sparse representation for the graph Laplacian matrix. In many ways, spectral clustering behaves similarly to the K-means

clustering algorithm. First, they both require the user to specify the number of clusters as input parameter. Both methods are also susceptible to the presence of outliers, which tend to form their own connected components (clusters). Thus, preprocessing or postprocessing methods will be needed to handle outliers in the data.

8.4.6 Shared Nearest Neighbor Similarity

In some cases, clustering techniques that rely on standard approaches to similarity and density do not produce the desired clustering results. This section examines the reasons for this and introduces an indirect approach to similarity that is based on the following principle:

If two points are similar to many of the same points, then they are similar to one another, even if a direct measurement of similarity does not indicate this.

We motivate the discussion by first explaining two problems that an SNN version of similarity addresses: low similarity and differences in density.

Problems with Traditional Similarity in High-Dimensional Data

In high-dimensional spaces, it is not unusual for similarity to be low. Consider, for example, a set of documents such as a collection of newspaper articles that come from a variety of sections of the newspaper: Entertainment, Financial, Foreign, Metro, National, and Sports. As explained in [Chapter 2](#), these documents can be viewed as vectors in a high-dimensional space,

where each component of the vector (attribute) records the number of times that each word in a vocabulary occurs in a document. The cosine similarity measure is often used to assess the similarity between documents. For this example, which comes from a collection of articles from the *Los Angeles Times*, [Table 8.3](#) gives the average cosine similarity in each section and among the entire set of documents.

Table 8.3. Similarity among documents in different sections of a newspaper.

Section	Average Cosine Similarity
Entertainment	0.032
Financial	0.030
Foreign	0.030
Metro	0.021
National	0.027
Sports	0.036
All Sections	0.014

The similarity of each document to its most similar document (the first nearest neighbor) is better, 0.39 on average. However, a consequence of low similarity among objects of the same class is that their nearest neighbor is often not of the same class. In the collection of documents from which [Table 8.3](#) was generated, about 20% of the documents have a nearest neighbor of a different class. In general, if direct similarity is low, then it becomes an unreliable guide for clustering objects, especially for agglomerative hierarchical clustering, where the closest points are put together and cannot

be separated afterward. Nonetheless, it is still usually the case that a large majority of the nearest neighbors of an object belong to the same class; this fact can be used to define a proximity measure that is more suitable for clustering.

Problems with Differences in Density

Another problem relates to differences in densities between clusters. [Figure 8.26](#) shows a pair of two-dimensional clusters of points with differing density. The lower density of the rightmost cluster is reflected in a lower average distance among the points. Even though the points in the less dense cluster form an equally valid cluster, typical clustering techniques will have more difficulty finding such clusters. Also, normal measures of cohesion, such as SSE, will indicate that these clusters are less cohesive. To illustrate with a real example, the stars in a galaxy are no less real clusters of stellar objects than the planets in a solar system, even though the planets in a solar system are considerably closer to one another on average, than the stars in a galaxy.



Figure 8.26.

Two circular clusters of 200 uniformly distributed points.

SNN Similarity Computation

In both situations, the key idea is to take the context of points into account in defining the similarity measure. This idea can be made quantitative by using a **shared nearest neighbor** definition of similarity in the manner indicated by **Algorithm 8.11** . Essentially, the SNN similarity is the number of shared neighbors as long as the two objects are on each other's nearest neighbor lists. Note that the underlying proximity measure can be any meaningful similarity or dissimilarity measure.

Algorithm 8.11 Computing shared nearest neighbor similarity

- 1: Find the k -nearest neighbors of all points.
- 2: **if** two points, x and y , are *not* among the k -nearest neighbors of each other **then**
- 3: $similarity(x,y) \leftarrow 0$
- 4: **else**
- 5: $similarity(x,y) \leftarrow$ number of shared neighbors
- 6: **end if**

The computation of SNN similarity is described by **Algorithm 8.11**  and graphically illustrated by **Figure 8.27** . Each of the two black points has eight nearest neighbors, including each other. Four of those nearest neighbors—the points in gray—are shared. Thus, the shared nearest neighbor similarity between the two points is 4.

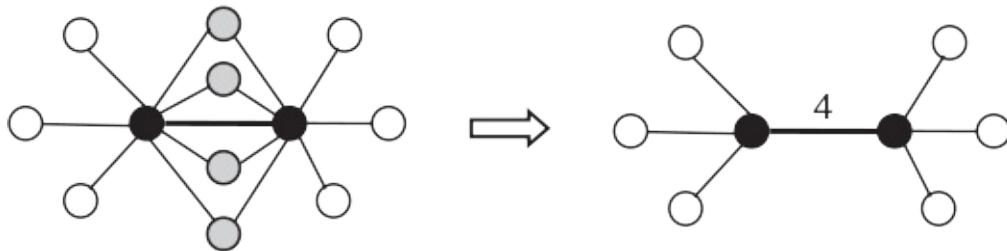


Figure 8.27.

Computation of SNN similarity between two points.

The similarity graph of the SNN similarities among objects is called the **SNN similarity graph**. Because many pairs of objects will have an SNN similarity of 0, this is a very sparse graph.

SNN Similarity versus Direct Similarity

SNN similarity is useful because it addresses some of the problems that occur with direct similarity. First, since it takes into account the context of an object by using the number of shared nearest neighbors, SNN similarity handles the situation in which an object happens to be relatively close to another object, but belongs to a different class. In such cases, the objects typically do not share many near neighbors and their SNN similarity is low.

SNN similarity also addresses problems with clusters of varying density. In a low-density region, the objects are farther apart than objects in denser regions. However, the SNN similarity of a pair of points only depends on the number of nearest neighbors two objects share, not how far these neighbors are from each object. Thus, SNN similarity performs an automatic scaling with respect to the density of the points.

8.4.7 The Jarvis-Patrick Clustering

Algorithm

Algorithm 8.12  expresses the Jarvis-Patrick clustering algorithm using the concepts of the last section. The JP clustering algorithm replaces the proximity between two points with the SNN similarity, which is calculated as described in **Algorithm 8.11** . A threshold is then used to sparsify this matrix of SNN similarities. In graph terms, an SNN similarity graph is created and sparsified. Clusters are simply the connected components of the SNN graph.

Algorithm 8.12 Jarvis-Patrick clustering algorithm.

- 1: Compute the SNN similarity graph.
- 2: Sparsify the SNN similarity graph by applying a similarity threshold.
- 3: Find the connected components (clusters) of the sparsified SNN similarity graph.

The storage requirements of the JP clustering algorithm are only $O(km)$ because it is not necessary to store the entire similarity matrix, even initially. The basic time complexity of JP clustering is $O(m^2)$, since the creation of the k -nearest neighbor list can require the computation of $O(m^2)$ proximities. However, for certain types of data, such as low-dimensional Euclidean data, special techniques, e.g., a k-d tree, can be used to more efficiently find the k -nearest neighbors without computing the entire similarity matrix. This can reduce the time complexity from $O(m^2)$ to $O(m \log m)$.

Example 8.13 (JP Clustering of a Two-

Dimensional Data Set).

We applied JP clustering to the “fish” data set shown in [Figure 8.28\(a\)](#) to find the clusters shown in [Figure 8.28\(b\)](#). The size of the nearest neighbor list was 20, and two points were placed in the same cluster if they shared at least 10 points. The different clusters are shown by the different markers and different shading. The points whose marker is an “x” were classified as noise by Jarvis-Patrick. They are mostly in the transition regions between clusters of different density.

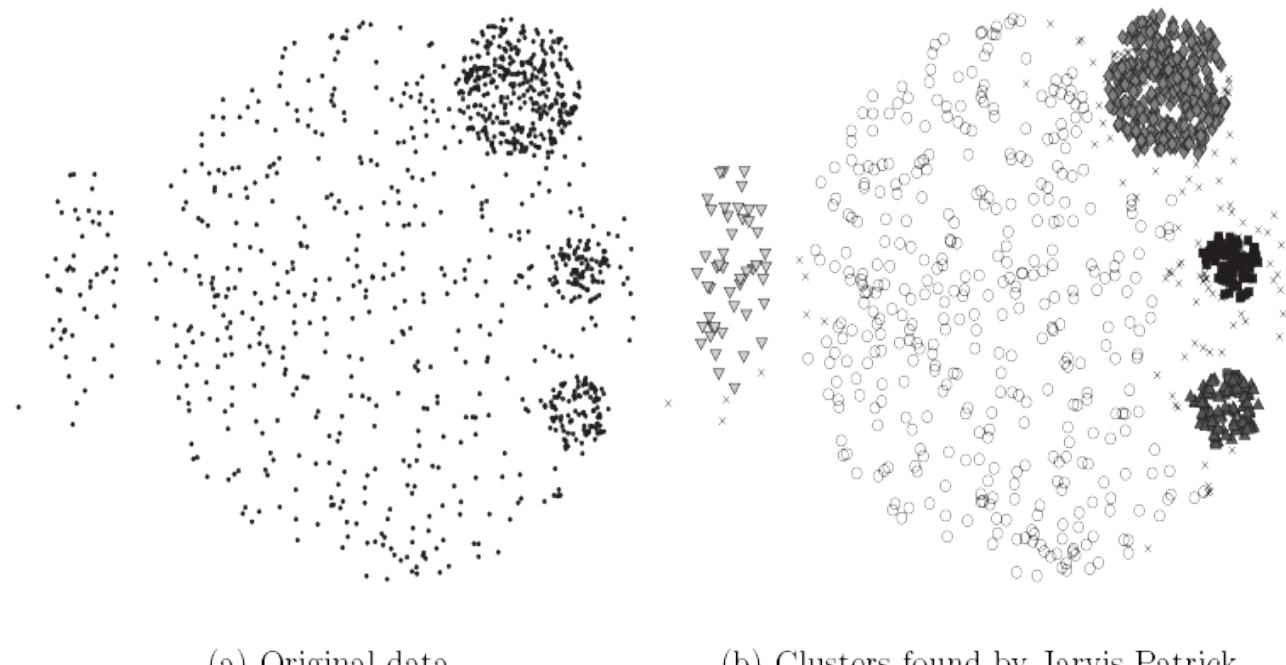


Figure 8.28.

Jarvis-Patrick clustering of a two-dimensional point set.

Strengths and Limitations

Because JP clustering is based on the notion of SNN similarity, it is good at dealing with noise and outliers and can handle clusters of different sizes,

shapes, and densities. The algorithm works well for high-dimensional data and is particularly good at finding tight clusters of strongly related objects.

However, JP clustering defines a cluster as a connected component in the SNN similarity graph. Thus, whether a set of objects is split into two clusters or left as one can depend on a single link. Hence, JP clustering is somewhat brittle; i.e., it can split true clusters or join clusters that should be kept separate.

Another potential limitation is that not all objects are clustered. However, these objects can be added to existing clusters, and in some cases, there is no requirement for a complete clustering. JP clustering has a basic time complexity of $O(m^2)$, which is the time required to compute the nearest neighbor list for a set of objects in the general case. In certain cases, e.g., low-dimensional data, special techniques can be used to reduce the time complexity for finding nearest neighbors to $O(m \log m)$. Finally, as with other clustering algorithms, choosing the best values for the parameters can be challenging.

8.4.8 SNN Density

As discussed in the introduction to this chapter, traditional Euclidean density becomes meaningless in high dimensions. This is true whether we take a grid-based view, such as that used by CLIQUE, a center-based view, such as that used by DBSCAN, or a kernel-density estimation approach, such as that used by DENCLUE. It is possible to use the center-based definition of density with a similarity measure that works well for high dimensions, e.g., cosine or Jaccard, but as described in [Section 8.4.6](#), such measures still have problems. However, because the SNN similarity measure reflects the local

configuration of the points in the data space, it is relatively insensitive to variations in density and the dimensionality of the space, and is a promising candidate for a new measure of density.

This section explains how to define a concept of SNN density by using SNN similarity and following the DBSCAN approach described in [Section 7.4](#). For clarity, the definitions of that section are repeated, with appropriate modification to account for the fact that we are using SNN similarity.

Core points. A point is a core point if the number of points within a given neighborhood around the point, as determined by SNN similarity and a supplied parameter Eps exceeds a certain threshold $MinPts$, which is also a supplied parameter.

Border points. A border point is a point that is not a core point, i.e., there are not enough points in its neighborhood for it to be a core point, but it falls within the neighborhood of a core point.

Noise points. A noise point is any point that is neither a core point nor a border point.

SNN density measures the degree to which a point is surrounded by similar points (with respect to nearest neighbors). Thus, points in regions of high and low density will typically have relatively high SNN density, while points in regions where there is a transition from low to high density—points that are between clusters—will tend to have low SNN density. Such an approach is well-suited for data sets in which there are wide variations in density, but clusters of low density are still interesting.

Example 8.14 (Core, Border, and Noise Points).

To make the preceding discussion of SNN density more concrete, we provide an example of how SNN density can be used to find core points and remove noise and outliers. There are 10,000 points in the 2D point data set shown in [Figure 8.29\(a\)](#). [Figures 8.29\(b–d\)](#) distinguish between these points based on their SNN density. [Figure 8.29\(b\)](#) shows the points with the highest SNN density, while [Figure 8.29\(c\)](#) shows points of intermediate SNN density, and [Figure 8.29\(d\)](#) shows figures of the lowest SNN density. From these figures, we see that the points that have high density (i.e., high connectivity in the SNN graph) are candidates for being representative or core points since they tend to be located well inside the cluster, while the points that have low connectivity are candidates for being noise points and outliers, as they are mostly in the regions surrounding the clusters.

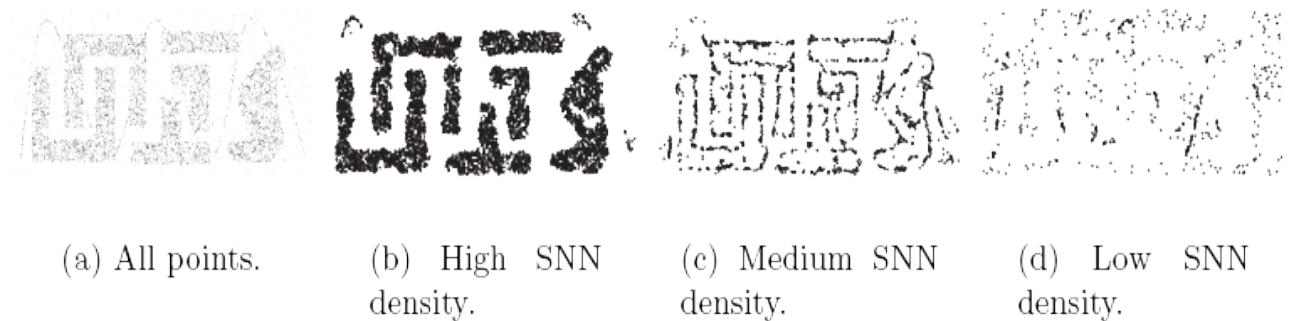


Figure 8.29.
SNN density of two-dimensional points.

8.4.9 SNN Density-Based Clustering

The SNN density defined above can be combined with the DBSCAN algorithm to create a new clustering algorithm. This algorithm is similar to the JP clustering algorithm in that it starts with the SNN similarity graph. However,

instead of using a threshold to sparsify the SNN similarity graph and then taking connected components as clusters, the SNN density-based clustering algorithm simply applies DBSCAN.

The SNN Density-based Clustering Algorithm

The steps of the SNN density-based clustering algorithm are shown in

Algorithm 8.13 

Algorithm 8.13 SNN density-based clustering algorithm.

- 1: Compute the SNN similarity graph.
- 2: Apply DBSCAN with user-specified parameters for Eps and $MinPts$.

The algorithm automatically determines the number of clusters in the data. Note that not all the points are clustered. The points that are discarded include noise and outliers, as well as points that are not strongly connected to a group of points. SNN density-based clustering finds clusters in which the points are strongly related to one another. Depending on the application, we might want to discard many of the points. For example, SNN density-based clustering is good for finding topics in groups of documents.

Example 8.15 (SNN Density-based Clustering of Time Series).

The SNN density-based clustering algorithm presented in this section is more flexible than Jarvis-Patrick clustering or DBSCAN. Unlike DBSCAN, it

can be used for high-dimensional data and situations in which the clusters have different densities. Unlike Jarvis-Patrick, which performs a simple thresholding and then takes the connected components as clusters, SNN density-based clustering uses a less brittle approach that relies on the concepts of SNN density and core points.

To demonstrate the capabilities of SNN density-based clustering on high-dimensional data, we applied it to monthly time series data of atmospheric pressure at various points on the Earth. More specifically, the data consists of the average monthly sea-level pressure (SLP) for a period of 41 years at each point on a 2.5° longitude-latitude grid. The SNN density-based clustering algorithm found the clusters (gray regions) indicated in [Figure 8.30](#). Note that these are clusters of time series of length 492 months, even though they are visualized as two-dimensional regions. The white areas are regions in which the pressure was not as uniform. The clusters near the poles are elongated because of the distortion of mapping a spherical surface to a rectangle.

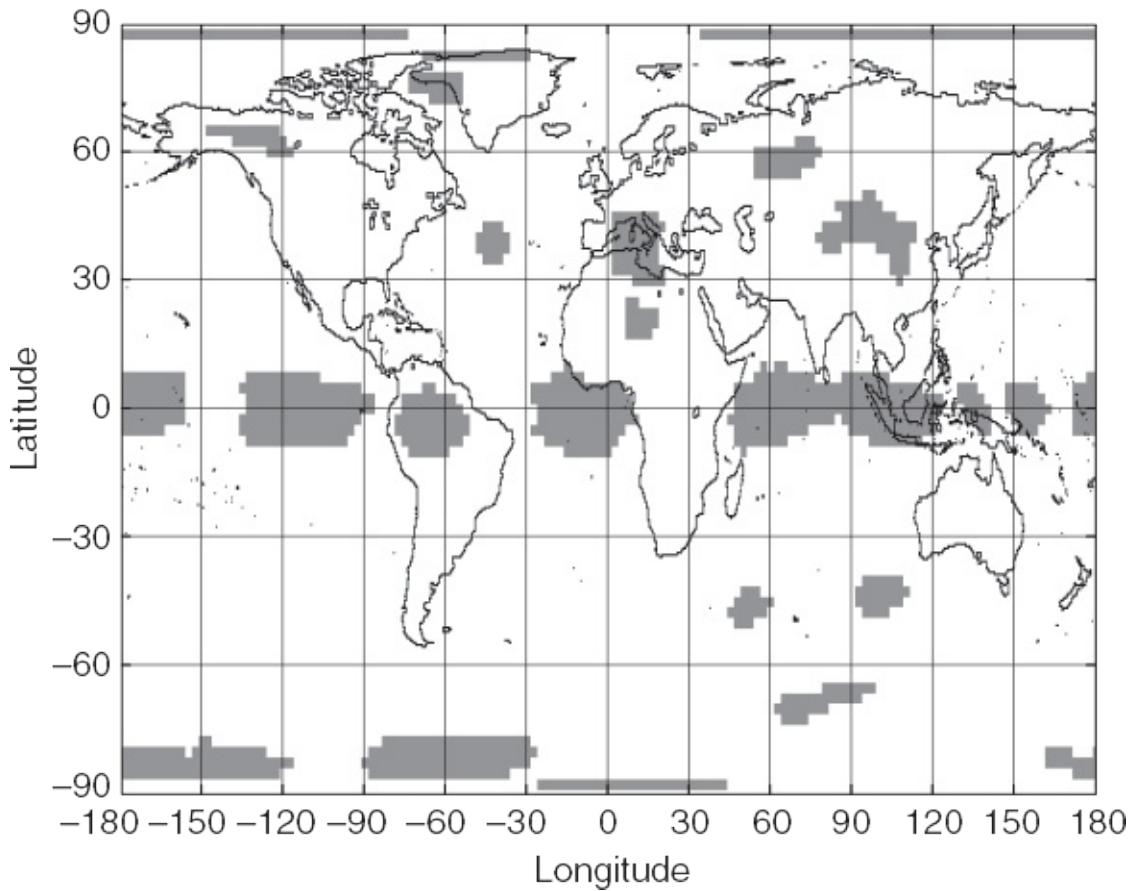


Figure 8.30.

Clusters of pressure time series found using SNN density-based clustering.

Using SLP, Earth scientists have defined time series, called **climate indices**, which are useful for capturing the behavior of phenomena involving the Earth's climate. For example, anomalies in climate indices are related to abnormally low or high precipitation or temperature in various parts of the world. Some of the clusters found by SNN density-based clustering have a strong connection to some of the climate indices known to Earth scientists.

Figure 8.31 shows the SNN density structure of the data from which the clusters were extracted. The density has been normalized to be on a scale

between 0 and 1. The density of a time series may seem like an unusual concept, but it measures the degree to which the time series and its nearest neighbors have the same nearest neighbors. Because each time series is associated with a location, it is possible to plot these densities on a two-dimensional plot. Because of temporal autocorrelation, these densities form meaningful patterns, e.g., it is possible to visually identify the clusters of [Figure 8.31](#).

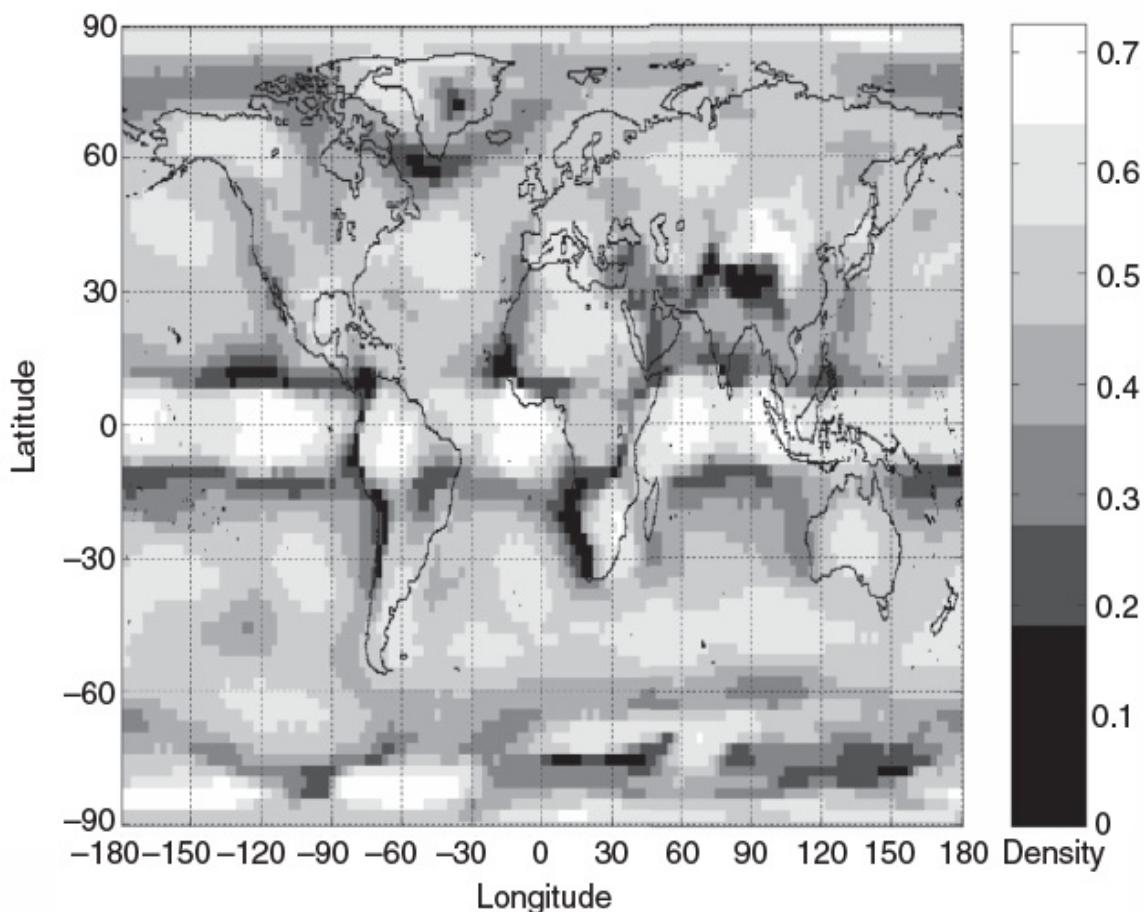


Figure 8.31.
SNN density of pressure time series.

Strengths and Limitations

The strengths and limitations of SNN density-based clustering are similar to those of JP clustering. However, the use of core points and SNN density adds considerable power and flexibility to this approach.

8.5 Scalable Clustering Algorithms

Even the best clustering algorithm is of little value if it takes an unacceptably long time to execute or requires too much memory. This section examines clustering techniques that place significant emphasis on scalability to the very large data sets that are becoming increasingly common. We start by discussing some general strategies for scalability, including approaches for reducing the number of proximity calculations, sampling the data, partitioning the data, and clustering a summarized representation of the data. We then discuss two specific examples of scalable clustering algorithms: CURE and BIRCH.

8.5.1 Scalability: General Issues and Approaches

The amount of storage required for many clustering algorithms is more than linear; e.g., with hierarchical clustering, memory requirements are usually $O(m^2)$, where m is the number of objects. For 10,000,000 objects, for example, the amount of memory required is proportional to 10^4 , a number still well beyond the capacities of current systems. Note that because of the requirement for random data access, many clustering algorithms cannot easily be modified to efficiently use secondary storage (disk), for which random data access is slow. Likewise, the amount of computation required for some clustering algorithms is more than linear. In the remainder of this section, we discuss a variety of techniques for reducing the amount of computation and

storage required by a clustering algorithm. CURE and BIRCH use some of these techniques.

Multidimensional or Spatial Access Methods Many techniques, such as K-means, Jarvis Patrick clustering, and DBSCAN, need to find the closest centroid, the nearest neighbors of a point, or all points within a specified distance. It is possible to use special techniques called multidimensional or spatial access methods to more efficiently perform these tasks, at least for low-dimensional data. These techniques, such as the k-d tree or R*-tree, typically produce a hierarchical partition of the data space that can be used to reduce the time required to find the nearest neighbors of a point. Note that grid-based clustering schemes also partition the data space.

Bounds on Proximities Another approach to avoiding proximity computations is to use bounds on proximities. For instance, when using Euclidean distance, it is possible to use the triangle inequality to avoid many distance calculations. To illustrate, at each stage of traditional K-means, it is necessary to evaluate whether a point should stay in its current cluster or be moved to a new cluster. If we know the distance between the centroids and the distance of a point to the (newly updated) centroid of the cluster to which it currently belongs, then we might be able to use the triangle inequality to avoid computing the distance of the point to any of the other centroids. See

Exercise 21  on [page 702](#).

Sampling Another approach to reducing the time complexity is to sample. In this approach, a sample of points is taken, these points are clustered, and then the remaining points are assigned to the existing clusters—typically to the closest cluster. If the number of points sampled is m , then the time complexity of an $O(m^2)$ algorithm is reduced to $O(m)$. A key problem with sampling, though, is that small clusters can be lost. When we discuss CURE,

we will provide a technique for investigating how frequently such problems occur.

Partitioning the Data Objects Another common approach to reducing time complexity is to use some efficient technique to partition the data into disjoint sets and then cluster these sets separately. The final set of clusters either is the union of these separate sets of clusters or is obtained by combining and/or refining the separate sets of clusters. We only discuss bisecting K-means ([Section 7.2.3](#)) in this section, although many other approaches based on partitioning are possible. One such approach will be described, when we describe CURE later on in this section.

If K-means is used to find K clusters, then the distance of each point to each cluster centroid is calculated at each iteration. When K is large, this can be very expensive. Bisecting K-means starts with the entire set of points and uses K-means to repeatedly bisect an existing cluster until we have obtained K clusters. At each step, the distance of points to two cluster centroids is computed. Except for the first step, in which the cluster being bisected consists of all the points, we only compute the distance of a subset of points to the two centroids being considered. Because of this fact, bisecting K-means can run significantly faster than regular K-means.

Summarization Another approach to clustering is to summarize the data, typically in a single pass, and then cluster the summarized data. In particular, the leader algorithm (see [Exercise 12](#)) either puts a data object in the closest cluster (if that cluster is sufficiently close) or starts a new cluster that contains the current object. This algorithm is linear in the number of objects and can be used to summarize the data so that other clustering techniques can be used. The BIRCH algorithm uses a similar concept.

Parallel and Distributed Computation If it is not possible to take advantage of the techniques described earlier, or if these approaches do not yield the desired accuracy or reduction in computation time, then other approaches are needed. A highly effective approach is to distribute the computation among multiple processors.

8.5.2 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a highly efficient clustering technique for data in Euclidean vector spaces, i.e., data for which averages make sense. BIRCH can efficiently cluster such data with one pass and can improve that clustering with additional passes. BIRCH can also deal effectively with outliers.

BIRCH is based on the notion of a Clustering Feature (CF) and a CF tree. The idea is that a cluster of data points (vectors) can be represented by a triple of numbers (N , LS , SS), where N is the number of points in the cluster, LS is the linear sum of the points, and SS is the sum of squares of the points. These are common statistical quantities that can be updated incrementally and that can be used to compute a number of important quantities, such as the centroid of a cluster and its variance (standard deviation). The variance is used as a measure of the diameter of a cluster.

These quantities can also be used to compute the distance between clusters. The simplest approach is to calculate an L 1 (city block) or L 2 (Euclidean) distance between centroids. We can also use the diameter (variance) of the merged cluster as a distance. A number of different distance measures for clusters are defined by BIRCH, but all can be computed using the summary statistics.

A CF tree is a height-balanced tree. Each interior node has entries of the form [CF i , child i] , where child i is a pointer to the i th child node. The space that each entry takes and the page size determine the number of entries in an interior node. The space of each entry is, in turn, determined by the number of attributes of each point.

Leaf nodes consist of a sequence of clustering features, CF i , where each clustering feature represents a number of points that have been previously scanned. Leaf nodes are subject to the restriction that each leaf node must have a diameter that is less than a parameterized threshold, T . The space that each entry takes, together with the page size, determines the number of entries in a leaf.

By adjusting the threshold parameter T , the height of the tree can be controlled. T controls the fineness of the clustering, i.e., the extent to which the data in the original set of data is reduced. The goal is to keep the CF tree in main memory by adjusting the T parameter as necessary.

A CF tree is built as the data is scanned. As each data point is encountered, the CF tree is traversed, starting from the root and choosing the closest node at each level. When the closest leaf cluster for the current data point is finally identified, a test is performed to see if adding the data item to the candidate cluster will result in a new cluster with a diameter greater than the given threshold, T . If not, then the data point is added to the candidate cluster by updating the CF information. The cluster information for all nodes from the leaf to the root is also updated.

If the new cluster has a diameter greater than T , then a new entry is created if the leaf node is not full. Otherwise the leaf node must be split. The two entries (clusters) that are farthest apart are selected as seeds and the remaining entries are distributed to one of the two new leaf nodes, based on which leaf

node contains the closest seed cluster. Once the leaf node has been split, the parent node is updated and split if necessary; i.e., if the parent node is full. This process may continue all the way to the root node.

BIRCH follows each split with a merge step. At the interior node where the split stops, the two closest entries are found. If these entries do not correspond to the two entries that just resulted from the split, then an attempt is made to merge these entries and their corresponding child nodes. This step is intended to increase space utilization and avoid problems with skewed data input order.

BIRCH also has a procedure for removing outliers. When the tree needs to be rebuilt because it has run out of memory, then outliers can optionally be written to disk. (An outlier is defined to be a node that has far fewer data points than average.) At certain points in the process, outliers are scanned to see if they can be absorbed back into the tree without causing the tree to grow in size. If so, they are reabsorbed. If not, they are deleted.

BIRCH consists of a number of phases beyond the initial creation of the CF tree. All the phases of BIRCH are described briefly in [Algorithm 8.14](#).

Algorithm 8.14 BIRCH.

- 1: **Load the data into memory by creating a CF tree that summarizes the data.**
- 2: **Build a smaller CF tree if it is necessary for phase 3.** T is increased, and then the leaf node entries (clusters) are reinserted. Since T has increased, some clusters will be merged.
- 3: **Perform global clustering.** Different forms of global clustering (clustering that uses the pairwise distances between

all the clusters) can be used. However, an agglomerative, hierarchical technique was selected. Because the clustering features store summary information that is important to certain kinds of clustering, the global clustering algorithm can be applied as if it were being applied to all the points in a cluster represented by the CF.

4: Redistribute the data points using the centroids of clusters discovered in step 3, and thus, discover a new set of clusters. This overcomes certain problems that can occur in the first phase of BIRCH. Because of page size constraints and the T parameter, points that should be in one cluster are sometimes split, and points that should be in different clusters are sometimes combined. Also, if the data set contains duplicate points, these points can sometimes be clustered differently, depending on the order in which they are encountered. By repeating this phase multiple times, the process converges to a locally optimal solution.

8.5.3 CURE

CURE (Clustering Using REpresentatives) is a clustering algorithm that uses a variety of different techniques to create an approach that can handle large data sets, outliers, and clusters with non-spherical shapes and non-uniform sizes. CURE represents a cluster by using multiple representative points from the cluster. These points will, in theory, capture the geometry and shape of the cluster. The first representative point is chosen to be the point farthest from the center of the cluster, while the remaining points are chosen so that they are farthest from all the previously chosen points. In this way, the

representative points are naturally relatively well distributed. The number of points chosen is a parameter, but it was found that a value of 10 or more worked well.

Once the representative points are chosen, they are shrunk toward the center by a factor, α . This helps moderate the effect of outliers, which are usually farther away from the center and thus, are shrunk more. For example, a representative point that was a distance of 10 units from the center would move by 3 units (for $\alpha=0.7$), while a representative point at a distance of 1 unit would only move 0.3 units.

CURE uses an agglomerative hierarchical scheme to perform the actual clustering. The distance between two clusters is the minimum distance between any two representative points (after they are shrunk toward their respective centers). While this scheme is not exactly like any other hierarchical scheme that we have seen, it is equivalent to centroid-based hierarchical clustering if $\alpha=0$, and roughly the same as single link hierarchical clustering if $\alpha=1$. Notice that while a hierarchical clustering scheme is used, the goal of CURE is to find a given number of clusters as specified by the user.

CURE takes advantage of certain characteristics of the hierarchical clustering process to eliminate outliers at two different points in the clustering process. First, if a cluster is growing slowly, then it may consist of outliers, since by definition, outliers are far from others and will not be merged with other points very often. In CURE, this first phase of outlier elimination typically occurs when the number of clusters is $1/3$ the original number of points. The second phase of outlier elimination occurs when the number of clusters is on the order of K , the number of desired clusters. At this point, small clusters are again eliminated.

Because the worst-case complexity of CURE is $O(m^2 \log m)$, it cannot be applied directly to large data sets. For this reason, CURE uses two techniques to speed up the clustering process. The first technique takes a random sample and performs hierarchical clustering on the sampled data points. This is followed by a final pass that assigns each remaining point in the data set to one of the clusters by choosing the cluster with the closest representative point. We discuss CURE's sampling approach in more detail later.

In some cases, the sample required for clustering is still too large and a second additional technique is required. In this situation, CURE partitions the sample data and then clusters the points in each partition. This preclustering step is then followed by a clustering of the intermediate clusters and a final pass that assigns each point in the data set to one of the clusters. CURE's partitioning scheme is also discussed in more detail later.

Algorithm 8.15 summarizes CURE. Note that K is the desired number of clusters, m is the number of points, p is the number of partitions, and q is the desired reduction of points in a partition, i.e., the number of clusters in a partition is m/pq . Therefore, the total number of clusters is m/q . For example, if $m=10,000$, $p=10$, and $q=100$, then each partition contains $10,000/10=1000$ points, and there would be $1000/100=10$ clusters in each partition and $10,000/100=100$ clusters overall.

Algorithm 8.15 CURE.

- 1: **Draw a random sample from the data set.** The CURE paper is notable for explicitly deriving a formula for what the size of this sample should be in order to guarantee, with high probability, that all clusters are represented by a minimum number of points.
- 2: **Partition the sample into p equal-sized partitions.**

- 3: Cluster the points in each partition into m pq clusters using CURE's hierarchical clustering algorithm to obtain a total of $m q$ clusters. Note that some outlier elimination occurs during this process.
- 4: Use CURE's hierarchical clustering algorithm to cluster the $m q$ clusters found in the previous step until only K clusters remain.
- 5: Eliminate outliers. This is the second phase of outlier elimination.
- 6: Assign all remaining data points to the nearest cluster to obtain a complete clustering.

Sampling in CURE

A key issue in using sampling is whether the sample is representative, that is, whether it captures the characteristics of interest. For clustering, the issue is whether we can find the same clusters in the sample as in the entire set of objects. Ideally, we would like the sample to contain some objects for each cluster and for there to be a separate cluster in the sample for those objects that belong to separate clusters in the entire data set.

A more concrete and attainable goal is to guarantee (with a high probability) that we have at least some points from each cluster. The number of points required for such a sample varies from one data set to another and depends on the number of objects and the sizes of the clusters. The creators of CURE derived a bound for the sample size that would be needed to ensure (with high probability) that we obtain at least a certain number of points from a cluster. Using the notation of this book, this bound is given by the following theorem.

Theorem 8.1.

Let f be a fraction, $0 \leq f \leq 1$. For cluster C_i of size m_i , we will obtain at least $f^* m_i$ objects from cluster C_i with a probability of $1 - \delta$, $0 \leq \delta \leq 1$, if our sample size s is given by the following:

$$s = fm + m m_i * \log(1/\delta) + m m_i \log(1/(2\delta)) + 2*f^* m_i * \log(1/\delta). \quad (8.21)$$

where m is the number of objects.

While this expression might look intimidating, it is reasonably easy to use. Suppose that there are 100,000 objects and that the goal is to have an 80% chance of obtaining 10% of the objects in cluster C_i , which has a size of 1000. In this case, $f=0.1, \delta=0.2, m=100,000, m_i=1000$, and thus $s=11,962$. If the goal is a 5% sample of C_i , which is 50 objects, then a sample size of 6440 will suffice.

Again, CURE uses sampling in the following way. First a sample is drawn, and then CURE is used to cluster this sample. After clusters have been found, each unclustered point is assigned to the closest cluster.

Partitioning

When sampling is not enough, CURE also uses a partitioning approach. The idea is to divide the points into p groups of size m/p and to use CURE to cluster each partition in order to reduce the number of objects by a factor of $q > 1$, where q can be roughly thought of as the average size of a cluster in a

partition. Overall, m/q clusters are produced. (Note that since CURE represents each cluster by a number of representative points, the reduction in the number of objects is not q .) This preclustering step is then followed by a final clustering of the m/q intermediate clusters to produce the desired number of clusters (K). Both clustering passes use CURE's hierarchical clustering algorithm and are followed by a final pass that assigns each point in the data set to one of the clusters.

The key issue is how p and q should be chosen. Algorithms such as CURE have a time complexity of $O(m^2)$ or higher, and furthermore, require that all the data be in main memory. We therefore want to choose p small enough so that an entire partition can be processed in main memory and in a 'reasonable' amount of time. At the current time, a typical desktop computer can perform a hierarchical clustering of a few thousand objects in a few seconds.

Another factor for choosing p , and also q , concerns the quality of the clustering. Specifically, the objective is to choose the values of p and q such that objects from the same underlying cluster end up in the same clusters eventually. To illustrate, suppose there are 1000 objects and a cluster of size 100. If we randomly generate 100 partitions, then each partition will, on average, have only one point from our cluster. These points will likely be put in clusters with points from other clusters or will be discarded as outliers. If we generate only 10 partitions of 100 objects, but q is 50, then the 10 points from each cluster (on average) will likely still be combined with points from other clusters, because there are only (on average) 10 points per cluster and we need to produce, for each partition, two clusters. To avoid this last problem, which concerns the proper choice of q , a suggested strategy is not to combine clusters if they are too dissimilar.

8.6 Which Clustering Algorithm?

A variety of factors need to be considered when deciding which type of clustering technique to use. Many, if not all, of these factors have been discussed to some extent in the current and previous chapters. Our goal in this section is to succinctly summarize these factors in a way that sheds some light on which clustering algorithm might be appropriate for a particular clustering task.

Type of Clustering For a clustering algorithm to be appropriate for a task, the type of clustering produced by the algorithm needs to match the type of clustering needed by the application. For some applications, such as creating a biological taxonomy, a hierarchy is preferred. In the case of clustering for summarization, a partitional clustering is typical. In yet other applications, both can prove useful.

Most clustering applications require a clustering of all (or almost all) of the objects. For instance, if clustering is used to organize a set of documents for browsing, then we would like most documents to belong to a group. However, if we wanted to find the strongest themes in a set of documents, then we might prefer to have a clustering scheme that produces only very cohesive clusters, even if many documents were left unclustered.

Finally, most applications of clustering assume that each object is assigned to one cluster (or one cluster on a level for hierarchical schemes). As we have seen, however, probabilistic and fuzzy schemes provide weights that indicate the degree or probability of membership in various clusters. Other techniques, such as DBSCAN and SNN density-based clustering, have the notion of core

points, which strongly belong to one cluster. Such concepts may be useful in certain applications.

Type of Cluster Another key aspect is whether the type of cluster matches the intended application. There are three commonly encountered types of clusters: prototype-, graph-, and density-based. Prototype-based clustering schemes, as well as some graph-based clustering schemes—complete link, centroid, and Ward's—tend to produce globular clusters in which each object is close to the cluster's prototype and/or to the other objects in the cluster. If, for example, we want to summarize the data to reduce its size and we want to do so with the minimum amount of error, then one of these types of techniques would be most appropriate. In contrast, density-based clustering techniques, as well as some graph-based clustering techniques, such as single link, tend to produce clusters that are not globular and thus contain many objects that are not very similar to one another. If clustering is used to segment a geographical area into contiguous regions based on the type of land cover, then one of these techniques is more suitable than a prototype-based scheme such as K-means.

Characteristics of Clusters Besides the general type of cluster, other cluster characteristics are important. If we want to find clusters in subspaces of the original data space, then we must choose an algorithm such as CLIQUE, which explicitly looks for such clusters. Similarly, if we are interested in enforcing spatial relationships between clusters, then SOM or some related approach would be appropriate. Also, clustering algorithms differ widely in their ability to handle clusters of varying shapes, sizes, and densities.

Characteristics of the Data Sets and Attributes As discussed in the introduction, the type of data set and attributes can dictate the type of algorithm to use. For instance, the K-means algorithm can only be used on data for which an appropriate proximity measure is available that allows

meaningful computation of a cluster centroid. For other clustering techniques, such as many agglomerative hierarchical approaches, the underlying nature of the data sets and attributes is less important as long as a proximity matrix can be created.

Noise and Outliers Noise and outliers are particularly important aspects of the data. We have tried to indicate the effect of noise and outliers on the various clustering algorithms that we have discussed. In practice, however, it can be difficult to evaluate the amount of noise in the data set or the number of outliers. More than that, what is noise or an outlier to one person might be interesting to another person. For example, if we are using clustering to segment an area into regions of different population density, we do not want to use a density-based clustering technique, such as DBSCAN, that assumes that regions or points with density lower than a global threshold are noise or outliers. As another example, hierarchical clustering schemes, such as CURE, often discard clusters of points that are growing slowly as such groups tend to represent outliers. However, in some applications we are most interested in relatively small clusters; e.g., in market segmentation, such groups might represent the most profitable customers.

Number of Data Objects We have considered how clustering is affected by the number of data objects in considerable detail in previous sections. We reiterate, however, that this factor often plays an important role in determining the type of clustering algorithm to be used. Suppose that we want to create a hierarchical clustering of a set of data, we are not interested in a complete hierarchy that extends all the way to individual objects, but only to the point at which we have split the data into a few hundred clusters. If the data is very large, we cannot directly apply an agglomerative hierarchical clustering technique. We could, however, use a divisive clustering technique, such as the minimum spanning tree (MST) algorithm, which is the divisive analog to single link, but this would only work if the data set is not too large. Bisecting K-

means would also work for many data sets, but if the data set is large enough that it cannot be contained completely in memory, then this scheme also runs into problems. In this situation, a technique such as BIRCH, which does not require that all data be in main memory, becomes more useful.

Number of Attributes We have also discussed the impact of dimensionality at some length. Again, the key point is to realize that an algorithm that works well in low or moderate dimensions may not work well in high dimensions. As in many other cases in which a clustering algorithm is inappropriately applied, the clustering algorithm will run and produce clusters, but the clusters will likely not represent the true structure of the data.

Cluster Description One aspect of clustering techniques that is often overlooked is how the resulting clusters are described. Prototype clusters are succinctly described by a small set of cluster prototypes. In the case of mixture models, the clusters are described in terms of small sets of parameters, such as the mean vector and the covariance matrix. This is also a very compact and understandable representation. For SOM, it is typically possible to visualize the relationships between clusters in a two-dimensional plot, such as that of [Figure 8.8](#). For graph- and density-based clustering approaches, however, clusters are typically described as sets of cluster members. Nonetheless, in CURE, clusters can be described by a (relatively) small set of representative points. Also, for grid-based clustering schemes, such as CLIQUE, more compact descriptions can be generated in terms of conditions on the attribute values that describe the grid cells in the cluster.

Algorithmic Considerations There are also important aspects of algorithms that need to be considered. Is the algorithm non-deterministic or order-dependent? Does the algorithm automatically determine the number of clusters? Is there a technique for determining the values of various parameters? Many clustering algorithms try to solve the clustering problem by

trying to optimize an objective function. Is the objective a good match for the application objective? If not, then even if the algorithm does a good job of finding a clustering that is optimal or close to optimal with respect to the objective function, the result is not meaningful. Also, most objective functions give preference to larger clusters at the expense of smaller clusters.

Summary The task of choosing the proper clustering algorithm involves considering all of these issues, and domain-specific issues as well. There is no formula for determining the proper technique. Nonetheless, a general knowledge of the types of clustering techniques that are available and consideration of the issues mentioned above, together with a focus on the intended application, should allow a data analyst to make an informed decision on which clustering approach (or approaches) to try.

8.7 Bibliographic Notes

An extensive discussion of fuzzy clustering, including a description of fuzzy c-means and formal derivations of the formulas presented in [Section 8.2.1](#), can be found in the book on fuzzy cluster analysis by [Höppner et al. \[595\]](#). While not discussed in this chapter, AutoClass by [Cheeseman et al. \[573\]](#) is one of the earliest and most prominent mixture-model clustering programs. An introduction to mixture models can be found in the tutorial of [Bilmes \[568\]](#), the book by [Mitchell \[606\]](#) (which also describes how the K-means algorithm can be derived from a mixture model approach), and the article by [Fraley and Raftery \[581\]](#). Mixture model is an example of a probabilistic clustering method, in which the clusters are represented as hidden variables in the model. More sophisticated probabilistic clustering methods such as latent Dirichlet allocation (LDA) [\[570\]](#) have been developed in recent years for domains such as text clustering.

Besides data exploration, SOM and its supervised learning variant, Learning Vector Quantization (LVQ), have been used for many tasks: image segmentation, organization of document files, and speech processing. Our discussion of SOM was cast in the terminology of prototype-based clustering. The book on SOM by [Kohonen et al. \[601\]](#) contains an extensive introduction to SOM that emphasizes its neural network origins, as well as a discussion of some of its variations and applications. One important SOM-related clustering development is the Generative Topographic Map (GTM) algorithm by [Bishop et al. \[569\]](#), which uses the EM algorithm to find Gaussian models satisfying two-dimensional topographic constraints.

The description of Chameleon can be found in the paper by [Karypis et al. \[599\]](#). Capabilities similar, although not identical to those of Chameleon have

been implemented in the CLUTO clustering package by [Karypis \[575\]](#). The METIS graph partitioning package by [Karypis and Kumar \[600\]](#) is used to perform graph partitioning in both programs, as well as in the OPOSSUM clustering algorithm by [Strehl and Ghosh \[616\]](#). A detailed discussion on spectral clustering can be found in the tutorial by von [Luxburg \[618\]](#). The spectral clustering method described in this chapter is based on an unnormalized graph Laplacian matrix and the ratio cut measure [\[590\]](#). Alternative formulations of spectral clustering have been developed using normalized graph Laplacian matrices for other evaluation measures [\[613\]](#).

The notion of SNN similarity was introduced by [Jarvis and Patrick \[596\]](#). A hierarchical clustering scheme based on a similar concept of mutual nearest neighbors was proposed by [Gowda and Krishna \[586\]](#). [Guha et al. \[589\]](#) created ROCK, a hierarchical graph-based clustering algorithm for clustering transaction data, which among other interesting features, also uses a notion of similarity based on shared neighbors that closely resembles the SNN similarity developed by Jarvis and Patrick. A description of the SNN density-based clustering technique can be found in the publications of Ertöz et al. [\[578, 579\]](#). SNN density-based clustering was used by [Steinbach et al. \[614\]](#) to find climate indices.

Examples of grid-based clustering algorithms are OptiGrid ([Hinneburg and Keim \[594\]](#)), the BANG clustering system ([Schikuta and Erhart \[611\]](#)), and WaveCluster ([Sheikholeslami et al. \[612\]](#)). The CLIQUE algorithm is described in the paper by [Agrawal et al. \[564\]](#). MAFIA ([Nagesh et al. \[608\]](#)) is a modification of CLIQUE whose goal is improved efficiency. [Kailing et al. \[598\]](#) have developed SUBCLU (density-connected SUBspace CLUstering), a subspace clustering algorithm based on DBSCAN. The DENCLUE algorithm was proposed by [Hinneburg and Keim \[593\]](#).

Our discussion of scalability was strongly influenced by the article of [Ghosh \[584\]](#). A wide-ranging discussion of specific techniques for clustering massive data sets can be found in the paper by [Murtagh \[607\]](#). CURE is work by [Guha et al. \[588\]](#), while details of BIRCH are in the paper by [Zhang et al. \[620\]](#). CLARANS ([Ng and Han \[609\]](#)) is an algorithm for scaling K-medoid clustering to larger databases. A discussion of scaling EM and K-means clustering to large data sets is provided by Bradley et al. [\[571, 572\]](#). A parallel implementation of K-means on the MapReduce framework has also been developed [\[621\]](#). In addition to K-means, other clustering algorithms that have been implemented on the MapReduce framework include [DBScan \[592\]](#), [spectral clustering \[574\]](#), and hierarchical clustering [\[617\]](#).

In addition to the approaches described in this chapter, there are many other clustering methods proposed in the literature. One class of methods that has become increasingly popular in recent years is based on non-negative matrix factorization (NMF) [\[602\]](#). The idea is an extension of the singular value decomposition (SVD) approach described in [Chapter 2](#), in which the data matrix is decomposed into a product of lower-rank matrices that represent the underlying components or clusters in the data. In NMF, additional constraints are imposed to ensure non-negativity in the elements of the component matrices. With different formulations and constraints, the NMF method can be shown to be equivalent to other clustering approaches, including K-means and spectral clustering [\[577, 603\]](#). Another popular class of methods utilizes the constraints provided by users to guide the clustering algorithm. Such algorithms are commonly known as constrained clustering or semi-supervised clustering [\[566, 567, 576, 619\]](#).

There are many aspects of clustering that we have not covered. Additional pointers are given in the books and surveys mentioned in the Bibliographic Notes of the previous chapter. Here, we mention four areas—omitting, unfortunately, many more. Clustering of transaction data ([Ganti et al. \[582\]](#),

Gibson et al. [585], Han et al. [591], and **Peters and Zaki [610]**) is an important area, as transaction data is common and of commercial importance. Streaming data is also becoming increasingly common and important as communications and sensor networks become pervasive. Two introductions to clustering for data streams are given in articles by **Barbará [565]** and **Guha et al. [587]**. Conceptual clustering (**Fisher and Langley [580]**, **Jonyer et al. [597]**, **Mishra et al. [605]**, **Michalski and Stepp [604]**, **Stepp and Michalski [615]**), which uses more complicated definitions of clusters that often correspond better to human notions of a cluster, is an area of clustering whose potential has perhaps not been fully realized. Finally, there has been a great deal of clustering work for data compression in the area of vector quantization. The book by **Gersho and Gray [583]** is a standard text in this area.

Bibliography

- [564] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 94–105, Seattle, Washington, June 1998. ACM Press.
- [565] D. Barbará. Requirements for clustering data streams. *SIGKDD Explorations Newsletter*, 3(2):23–27, 2002.
- [566] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *Proceedings of 19th International Conference on Machine Learning*, pages 19–26, 2002.
- [567] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. CRC Press, 2008.
- [568] J. Bilmes. A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report ICSITR-97-021, University of California at Berkeley, 1997.
- [569] C. M. Bishop, M. Svensen, and C. K. I. Williams. GTM: A principled alternative to the self-organizing map. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks – ICANN 1995*, pages 475–482, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

Networks—ICANN96. Intl. Conf., Proc., pages 165–170. Springer-Verlag, Berlin, Germany, 1996.

- [570] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.
- [571] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling Clustering Algorithms to Large Databases. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 9–15, New York City, August 1998. AAAI Press.
- [572] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling EM (Expectation Maximization) Clustering to Large Databases. Technical Report MSR-TR-98-35, Microsoft Research, October 1999.
- [573] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. AutoClass: a Bayesian classification system. In *Readings in knowledge acquisition and learning: automating the construction and improvement of expert systems*, pages 431–441. Morgan Kaufmann Publishers Inc., 1993.
- [574] W. Y. Chen, Y. Song, H. Bai, C. J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568586, 2011.
- [575] CLUTO 2.1.2: Software for Clustering High-Dimensional Datasets.
www.cs.umn.edu/~karypis, October 2016.

- [576] I. Davidson and S. Basu. A survey of clustering with instance level constraints. *ACM Transactions on Knowledge Discovery from Data*, 1:1–41, 2007.
- [577] C. Ding, X. He, and H. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proc of the SIAM International Conference on Data Mining*, page 606-610, 2005.
- [578] L. Ertöz, M. Steinbach, and V. Kumar. A New Shared Nearest Neighbor Clustering Algorithm and its Applications. In *Workshop on Clustering High Dimensional Data and its Applications, Proc. of Text Mine'01, First SIAM Intl. Conf. on Data Mining, Chicago, IL, USA*, 2001.
- [579] L. Ertöz, M. Steinbach, and V. Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *Proc. of the 2003 SIAM Intl. Conf. on Data Mining*, San Francisco, May 2003. SIAM.
- [580] D. Fisher and P. Langley. Conceptual clustering and its relation to numerical taxonomy. *Artificial Intelligence and Statistics*, pages 77–116, 1986.
- [581] C. Fraley and A. E. Raftery. How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis. *The Computer Journal*, 41(8):578–588, 1998.
- [582] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS–Clustering Categorical Data Using Summaries. In *Proc. of the 5th Intl. Conf. on*

Knowledge Discovery and Data Mining, pages 73–83. ACM Press, 1999.

[583] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*, volume 159 of *Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 1992.

[584] J. Ghosh. Scalable Clustering Methods for Data Mining. In N. Ye, editor, *Handbook of Data Mining*, pages 247–277. Lawrence Ealbaum Assoc, 2003.

[585] D. Gibson, J. M. Kleinberg, and P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. *VLDB Journal*, 8(3–4):222–236, 2000.

[586] K. C. Gowda and G. Krishna. Agglomerative Clustering Using the Concept of Mutual Nearest Neighborhood. *Pattern Recognition*, 10(2):105–112, 1978.

[587] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, May/June 2003.

[588] S. Guha, R. Rastogi, and K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 73–84. ACM Press, June 1998.

- [589] S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. In *Proc. of the 15th Intl. Conf. on Data Engineering*, pages 512–521. IEEE Computer Society, March 1999.
- [590] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. Computer-Aided Design*, 11(9):1074–1085, 1992.
- [591] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Hypergraph Based Clustering in High-Dimensional Data Sets: A Summary of Results. *IEEE Data Eng. Bulletin*, 21(1):15–22, 1998.
- [592] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan. MR-DBSCAN: an efficient parallel density-based clustering algorithm using MapReduce. In *Proc. of the IEEE International Conference on Parallel and Distributed Systems*, pages 473–480, 2011.
- [593] A. Hinneburg and D. A. Keim. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 58–65, New York City, August 1998. AAAI Press.
- [594] A. Hinneburg and D. A. Keim. Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In *Proc. of the 25th VLDB Conf.*, pages 506–517, Edinburgh, Scotland, UK, September 1999. Morgan Kaufmann.

- [595] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*. John Wiley & Sons, New York, July 2 1999.
- [596] R. A. Jarvis and E. A. Patrick. Clustering Using a Similarity Measure Based on Shared Nearest Neighbors. *IEEE Transactions on Computers*, C-22(11):1025–1034, 1973.
- [597] I. Jonyer, D. J. Cook, and L. B. Holder. Graph-based hierarchical conceptual clustering. *Journal of Machine Learning Research*, 2:19–43, 2002.
- [598] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-Connected Subspace Clustering for High-Dimensional Data. In *Proc. of the 2004 SIAM Intl. Conf. on Data Mining*, pages 428–439, Lake Buena Vista, Florida, April 2004. SIAM.
- [599] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. *IEEE Computer*, 32(8):68–75, August 1999.
- [600] G. Karypis and V. Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.
- [601] T. Kohonen, T. S. Huang, and M. R. Schroeder. *Self-Organizing Maps*. Springer-Verlag, December 2000.

- [602] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.
- [603] T. Li and C. H. Q. Ding. The Relationships Among Various Nonnegative Matrix Factorization Methods for Clustering. In *Proc of the IEEE International Conference on Data Mining*, pages 362–371, 2006.
- [604] R. S. Michalski and R. E. Stepp. Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4):396–409, 1983.
- [605] N. Mishra, D. Ron, and R. Swaminathan. A New Conceptual Clustering Framework. *Machine Learning Journal*, 56(1–3):115–151, July/August/September 2004.
- [606] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [607] F. Murtagh. Clustering massive data sets. In J. Abello, P. M. Pardalos, and M. G. C. Reisende, editors, *Handbook of Massive Data Sets*. Kluwer, 2000.
- [608] H. Nagesh, S. Goil, and A. Choudhary. Parallel Algorithms for Clustering High-Dimensional Large-Scale Datasets. In R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, and R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*, pages 335–356. Kluwer Academic Publishers, Dordrecht, Netherlands, October 2001.

- [609] R. T. Ng and J. Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.
- [610] M. Peters and M. J. Zaki. CLICKS: Clustering Categorical Data using K-partite Maximal Cliques. In *Proc. of the 21st Intl. Conf. on Data Engineering*, Tokyo, Japan, April 2005.
- [611] E. Schikuta and M. Erhart. The BANG-Clustering System: Grid-Based Data Analysis. In *Advances in Intelligent Data Analysis, Reasoning about Data, Second Intl. Symposium, IDA-97, London*, volume 1280 of *Lecture Notes in Computer Science*, pages 513–524. Springer, August 1997.
- [612] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. of the 24th VLDB Conf.*, pages 428–439, New York City, August 1998. Morgan Kaufmann.
- [613] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888 905, 2000.
- [614] M. Steinbach, P.-N. Tan, V. Kumar, S. Klooster, and C. Potter. Discovery of climate indices using clustering. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 446–455, New York, NY, USA, 2003. ACM Press.

- [615] R. E. Stepp and R. S. Michalski. Conceptual clustering of structured objects: A goal-oriented approach. *Artificial Intelligence*, 28(1):43–69, 1986.
- [616] A. Strehl and J. Ghosh. A Scalable Approach to Balanced, High-dimensional Clustering of Market-Baskets. In *Proc. of the 7th Intl. Conf. on High Performance Computing (HiPC 2000)*, volume 1970 of *Lecture Notes in Computer Science*, pages 525–536, Bangalore, India, December 2000. Springer.
- [617] T. Sun, C. Shu, F. Li, H. Yu, L. Ma, and Y. Fang. An efficient hierarchical clustering method for large datasets with map-reduce. In *Proc of the IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 494–499, 2009.
- [618] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, 2007.
- [619] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained K-means Clustering with Background Knowledge. In *Proceedings of 18th International Conference on Machine Learning*, pages 577–584, 2001.
- [620] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. of 1996 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 103–114, Montreal, Quebec, Canada, June 1996. ACM Press.

- [621] W. Zhao, H. Ma, and Q. He. Parallel K-Means Clustering based on MapReduce. In *Proc of the IEEE International Conference on Cloud Computing*, page 674-679, 2009.

8.8 Exercises

1. For sparse data, discuss why considering only the presence of non-zero values might give a more accurate view of the objects than considering the actual magnitudes of values. When would such an approach not be desirable?
2. Describe the change in the time complexity of K-means as the number of clusters to be found increases.
3. Consider a set of documents. Assume that all documents have been normalized to have unit length of 1. What is the “shape” of a cluster that consists of all documents whose cosine similarity to a centroid is greater than some specified constant? In other words, $\cos(d, c) \geq \delta$, where $0 < \delta \leq 1$.
4. Discuss the advantages and disadvantages of treating clustering as an optimization problem. Among other factors, consider efficiency, non-determinism, and whether an optimization-based approach captures all types of clusterings that are of interest.
5. What is the time and space complexity of fuzzy c-means? Of SOM? How do these complexities compare to those of K-means?
6. Traditional K-means has a number of limitations, such as sensitivity to outliers and difficulty in handling clusters of different sizes and densities, or with non-globular shapes. Comment on the ability of fuzzy c-means to handle these situations.
7. For the fuzzy c-means algorithm described in this book, the sum of the membership degree of any point over all clusters is 1. Instead, we could only require that the membership degree of a point in a cluster be between 0 and 1. What are the advantages and disadvantages of such an approach?

8. Explain the difference between likelihood and probability.
9. **Equation 8.12** gives the likelihood for a set of points from a Gaussian distribution as a function of the mean μ and the standard deviation σ . Show mathematically that the maximum likelihood estimate of μ and σ are the sample mean and the sample standard deviation, respectively.
10. We take a sample of adults and measure their heights. If we record the gender of each person, we can calculate the average height and the variance of the height, separately, for men and women. Suppose, however, that this information was not recorded. Would it be possible to still obtain this information? Explain.
11. Compare the membership weights and probabilities of **Figures 8.1** and **8.4**, which come, respectively, from applying fuzzy and EM clustering to the same set of data points. What differences do you detect, and how might you explain these differences?
12. **Figure 8.32** shows a clustering of a two-dimensional point data set with two clusters: The leftmost cluster, whose points are marked by asterisks, is somewhat diffuse, while the rightmost cluster, whose points are marked by circles, is compact. To the right of the compact cluster, there is a single point (marked by an arrow) that belongs to the diffuse cluster, whose center is farther away than that of the compact cluster. Explain why this is possible with EM clustering, but not K-means clustering.

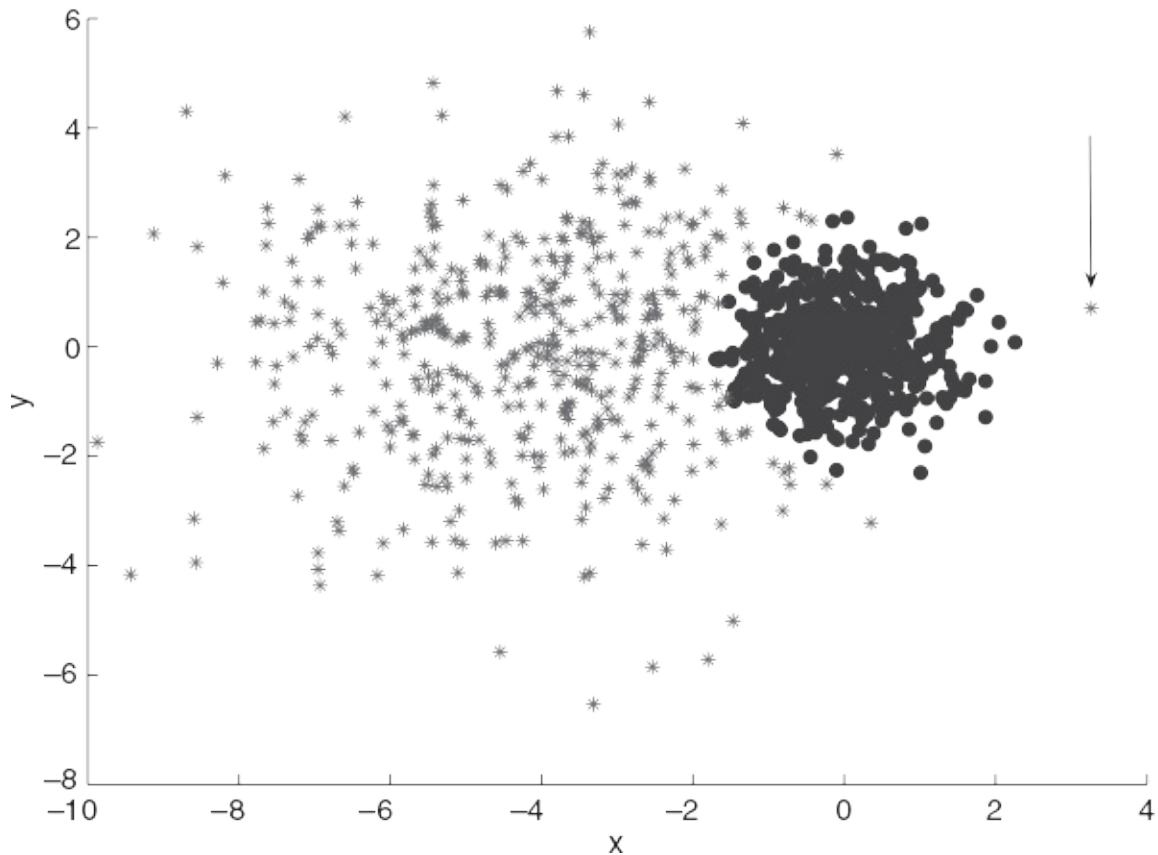


Figure 8.32.

Data set for [Exercise 12](#). EM clustering of a two-dimensional point set with two clusters of differing density.

13. Show that the MST clustering technique of [Section 8.4.2](#) produces the same clusters as single link. To avoid complications and special cases, assume that all the pairwise similarities are distinct.
14. One way to sparsify a proximity matrix is the following: For each object (row in the matrix), set all entries to 0 except for those corresponding to the objects k -nearest neighbors. However, the sparsified proximity matrix is typically not symmetric.
 - a. If object a is among the k -nearest neighbors of object b , why is b not guaranteed to be among the k -nearest neighbors of a ?

- b. Suggest at least two approaches that could be used to make the sparsified proximity matrix symmetric.
15. Give an example of a set of clusters in which merging based on the closeness of clusters leads to a more natural set of clusters than merging based on the strength of connection (interconnectedness) of clusters.
16. **Table 8.4** lists the two nearest neighbors of four points. Calculate the SNN similarity between each pair of points using the definition of SNN similarity defined in **Algorithm 8.11**.

Table 8.4. Two nearest neighbors of four points.

Point	First Neighbor	Second Neighbor
1	4	3
2	3	4
3	4	2
4	3	1

17. For the definition of SNN similarity provided by **Algorithm 8.11**, the calculation of SNN distance does not take into account the position of shared neighbors in the two nearest neighbor lists. In other words, it might be desirable to give higher similarity to two points that share the same nearest neighbors in the same or roughly the same order.
- Describe how you might modify the definition of SNN similarity to give higher similarity to points whose shared neighbors are in roughly the same order.
 - Discuss the advantages and disadvantages of such a modification.

18. Name at least one situation in which you would *not* want to use clustering based on SNN similarity or density.
19. Grid-clustering techniques are different from other clustering techniques in that they partition space instead of sets of points.
 - a. How does this affect such techniques in terms of the description of the resulting clusters and the types of clusters that can be found?
 - b. What kind of cluster can be found with grid-based clusters that cannot be found by other types of clustering approaches? (Hint: See [Exercise 20](#) in [Chapter 7](#), page 608.)
20. In CLIQUE, the threshold used to find cluster density remains constant, even as the number of dimensions increases. This is a potential problem because density drops as dimensionality increases; i.e., to find clusters in higher dimensions the threshold has to be set at a level that may well result in the merging of low-dimensional clusters. Comment on whether you feel this is truly a problem and, if so, how you might modify CLIQUE to address this problem.
21. Given a set of points in Euclidean space, which are being clustered using the K-means algorithm with Euclidean distance, the triangle inequality can be used in the assignment step to avoid calculating all the distances of each point to each cluster centroid. Provide a general discussion of how this might work.
22. Instead of using the formula derived in CURE—see [Equation 8.21](#)—we could run a Monte Carlo simulation to directly estimate the probability that a sample of size s would contain at least a certain fraction of the points from a cluster. Using a Monte Carlo simulation compute the probability that a sample of size s contains 50% of the elements of a cluster of size 100, where the total number of points is 1000, and where s can take the values 100, 200, or 500.

9 Anomaly Detection

*In anomaly detection, the goal is to find objects that do not conform to normal patterns or behavior. Often, anomalous objects are known as **outliers**, since, on a scatter plot of the data, they lie far away from other data points. Anomaly detection is also known as **deviation detection**, because anomalous objects have attribute values that deviate significantly from the expected or typical attribute values, or as **exception mining**, because anomalies are exceptional in some sense. In this chapter, we will mostly use the terms *anomaly* or *outlier*. There are a variety of anomaly detection approaches from several areas, including statistics, machine learning, and data mining. All try to capture the idea that an anomalous data object is unusual or in some way inconsistent with other objects.*

Alt1ough unusual objects or events are, by definition, relatively rare, their detection and analysis provides critical insights that are useful in a number of applications. The following examples illustrate applications for which anomalies are of considerable interest.

- **Fraud Detection.** *The purchasing behavior of someone who steals a credit card is often different from that of the original owner. Credit card companies attempt to detect a theft by looking for buying patterns that characterize theft or by noticing a change from typical behavior. Similar approaches are relevant in many domains such as detecting insurance claim fraud and insider trading.*
- **Intrusion Detection.** *Unfortunately, attacks on computer systems and computer networks are commonplace. While some of these attacks, such as those designed to disable or overwhelm computers and networks, are obvious, other attacks, such as those designed to secretly gather information, are difficult to detect. Many of these intrusions can only be detected by monitoring systems and networks for unusual behavior.*
- **Ecosystem Disturbances.** *The Earth's ecosystem has been experiencing rapid changes in the last few decades due to natural or anthropogenic reasons. This includes an increased propensity for extreme events, such as heat waves, droughts, and floods, which have a huge impact on the environment. Identifying such extreme events from sensor recordings and satellite images is important for understanding their origins and behavior, as well as for devising sustainable adaptation policies.*
- **Medicine and Public Health.** *For a particular patient, unusual symptoms or test results, such as an anomalous MRI scan, may indicate potential health problems. However, whether a particular test*

result is anomalous may depend on many other characteristics of the patient, such as age, sex, and genetic makeup. Furthermore, the categorization of a result as anomalous or not incurs a cost—unneeded additional tests if a patient is healthy and potential harm to the patient if a condition is left undiagnosed and untreated. The detection of emerging disease outbreaks, such as H1N1-influenza or SARS, which result in unusual and alarming test results in a series of patients, is also important for monitoring the spread of diseases and taking preventive actions.

- **Aviation Safety.** Since aircrafts are highly complex and dynamic systems, they are prone to accidents—often with drastic consequences—due to mechanical, environmental or human factors. To monitor the occurrence of such anomalies, most commercial airplanes are equipped with a large number of sensors to measure different flight parameters, such as information from the control system, the avionics and propulsion systems, and pilot actions. Identifying abnormal events in these sensor recordings (e.g., an anomalous sequence of pilot actions or an abnormally functioning aircraft component) can help prevent aircraft accidents and promote aviation safety.

Although much of the recent interest in anomaly detection is driven by applications in which anomalies are the focus, historically, anomaly detection (and

removal) has been viewed as a data preprocessing technique to eliminate erroneous data objects that may be recorded because of human error, a problem with the measuring device, or the presence of noise. Such anomalies provide no interesting information but only distort the analysis of normal objects. The identification and removal of such erroneous data objects is not the focus of this chapter. Instead, the emphasis is on detecting anomalous objects that are interesting in their own right.

9.1 Characteristics of Anomaly Detection Problems

Anomaly detection problems are quite diverse in nature as they appear in multiple application domains under different settings. This diversity in problem characteristics has resulted in a rich variety of anomaly detection methods that are useful in different situations. Before we discuss these methods, it will be useful to describe some of the key characteristics of anomaly detection problems that motivate the different styles of anomaly detection methods.

9.1.1 A Definition of an Anomaly

An important characteristic of an anomaly detection problem is the way an anomaly is defined. Since anomalies are rare occurrences that are not fully understood, they can be defined in different ways depending on the problem requirements. However, the following high-level definition of an anomaly encompasses most of the definitions commonly employed.

Definition 9.1.

An **anomaly** is an observation that doesn't fit the distribution of the data for normal instances, i.e., is unlikely under the distribution of the majority of instances.

We note the following points:

- This definition does not assume that the distribution is easy to express in terms of well-known statistical distributions. Indeed, the difficulty of doing so is the reason that many anomaly detection approaches use non-statistical approaches. Nonetheless, these approaches aim to find data objects that are not common.
- Conceptually, we can rank data objects according to the probability of seeing such an object or something more extreme. The lower the probability, the more likely the object is an anomaly. Often, the reciprocal of the probability is used as a ranking score. Again, this is only practical in some cases. Such approaches are discussed in [Section 9.3](#).
- There can be various causes of an anomaly: noise, the object comes from another distribution, e.g., a few grapefruit mixed with oranges, or the object is just a rare occurrence of data from the distribution, e.g., a 7 foot tall person. As mentioned, we are not interested in anomalies due to noise.

9.1.2 Nature of Data

The nature of the input data plays a key role in deciding the choice of a suitable anomaly detection technique. Some of the common characteristics of the input data include the number and types of attributes, and the representation used for describing every data instance.

Univariate or Multivariate

If the data contains a single attribute, the question of whether an object is anomalous depends on whether the object's value for that attribute is

anomalous. However, if the data objects are represented using many attributes, a data object may have anomalous values for some attributes but ordinary values for other attributes. Furthermore, an object may be anomalous even if none of its attribute values are individually anomalous. For example, it is common to have people who are two feet tall (children) or are 100 pounds in weight, but uncommon to have a two-foot tall person who weighs 100 pounds. Identifying an anomaly in a multivariate setting is thus challenging, particularly when the dimensionality of the data is high.

Record Data or Proximity Matrix

The most common approach for representing a data set is to use record data or its variants, e.g., a data matrix, where every data instance is described using the same set of attributes. However, for the purpose of anomaly detection, it is often sufficient to know how *different* an instance is in comparison to other instances. Hence, some anomaly detection methods work with a different representation of the input data known as a **proximity matrix**, where every entry in the matrix denotes the pairwise proximity (similarity or dissimilarity) between two instances. Note that a data matrix can always be converted to a proximity matrix by using an appropriate proximity measure. Also, a similarity matrix can be easily converted to a distance matrix using any of the transformations presented in [Section 2.4.1](#).

Availability of Labels

The label of a data instance denotes whether the instance is normal or anomalous. If we have a training set with labels for every data instance, then the problem of anomaly detection translates to a **supervised** learning (classification) problem. Classification techniques that address the so-called rare class problem are particularly relevant because anomalies are relatively rare with respect to normal objects. See [Section 4.11](#).

However, in most practical applications, we do not have a training set with accurate and representative labels of the normal and anomalous classes. Note that obtaining labels of the anomalous class is especially challenging because of their rarity. It is thus difficult for a human expert to catalog every type of anomaly since the properties of the anomalous class are often unknown. Hence, most anomaly detection problems are **unsupervised** in nature, i.e., the input data does not have any labels. All anomaly detection methods presented in this chapter operate in the unsupervised setting.

Note that in the absence of labels, it is challenging to differentiate anomalies from normal instances given an input data set. However, anomalies typically have some properties that techniques can take advantage of to make finding anomalies practical. Two key properties are the following:

Relatively Small in Number

Since anomalies are infrequent, most input data sets have a predominance of normal instances. The input data set is thus often used as an imperfect representation of the normal class in most anomaly detection techniques. However, the performance of such methods needs to be robust to the presence of outliers in the input data. Some anomaly detection methods also provide a mechanism to specify the expected number of outliers in the input data. Such methods can work with a larger number of anomalies in the data.

Sparsely Distributed

Anomalies, unlike normal objects, are often unrelated to each other and hence distributed sparsely in the space of attributes. Indeed, the successful operation of most anomaly detection methods depends on anomalies not being tightly clustered. However, some anomaly detection methods are

specifically designed to find clustered anomalies (see [Section 9.5.1](#)), which are assumed to either be small in size or distant from other instances.

9.1.3 How Anomaly Detection is Used

There are two different ways in which any generic anomaly detection method can be used. In the first approach, we are given an input data that contains both normal and anomalous instances, and are required to identify anomalies in this input data. All anomaly detection approaches presented in this chapter are able to operate in this setup. In the second approach, we are also provided with test instances (appearing one at a time) that need to be identified as anomalies. Most anomaly detection methods (with a few exceptions) are able to use the input data set to provide outputs on new test instances. Finding anomalies by finding anomalous clusters—[Section 9.5.1](#)—is one of the exceptions.

9.2 Characteristics of Anomaly Detection Methods

To cater to the diverse needs of anomaly detection problems, a number of techniques have been explored using concepts from different research disciplines. In this section, we provide a high-level description of some of the common characteristics of anomaly detection methods that are helpful in understanding their commonalities and differences.

Model-based vs. Model-free

Many approaches for anomaly detection use the input data to build *models* that can be used to identify whether a test instance is anomalous or not. Most **model-based** techniques for anomaly detection build a model of the normal class and identify anomalies that do not fit this model. For example, we can fit a Gaussian distribution to model the normal class and then identify anomalies that do not conform well to the learned distribution. The other type of model-based techniques learns a model of both the normal and anomalous classes, and identifies instances as anomalies if they are more likely to belong to the anomalous class. Although these approaches technically require representative labels from both classes, they often make assumptions about the nature of the anomalous class, e.g., that anomalies are rare and sparsely distributed, and thus can work even in an unsupervised setting.

In addition to identifying anomalies, model-based methods provide information about the nature the normal class and sometimes even the anomalous class. However, the assumptions they make about the properties of normal and

anomalous classes may not hold true in every problem. In contrast, **model-free** approaches do not explicitly characterize the distribution of the normal or anomalous classes. Instead, they directly identify instances as anomalies without learning models from the input data. For example, an instance can be identified as an anomaly if it is quite different from other instances in its neighborhood. Model-free approaches are often intuitive and simple to use.

Global vs. Local Perspective

An instance can be identified as an anomaly either by considering the global context, e.g., by building a model over all normal instances and using this global model for anomaly detection, or by considering the local perspective of every data instance. Specifically, an anomaly detection approach is termed **local** if its output on a given instance does not change if instances outside its local neighborhood are modified or removed. The difference between the global and local perspective can result in significant differences in the results of an anomaly detection method, because an object may seem unusual with respect to all objects globally, but not with respect to objects in its local neighborhood. For example, a person whose height is 6 feet 5 inches is unusually tall with respect to the general population, but not with respect to professional basketball players.

Label vs. Score

Different approaches for anomaly detection produce their outputs in different formats. The most basic type of output is a binary **anomaly label**: an object is either identified as an anomaly or as a normal instance. However, labels do not provide any information about the degree to which an instance is anomalous. Frequently, some of the detected anomalies are more extreme

than others, while some instances labeled as normal may be on the verge of being identified as anomalies.

Hence, many anomaly detection methods produce an **anomaly score** that indicates how strongly an instance is likely to be an anomaly. An anomaly score can easily be sorted and converted into ranks, so that an analyst can be provided with only the top-most scoring anomalies. Alternatively, a cutoff threshold can be applied to an anomaly score to obtain binary anomaly labels. The task of choosing the right threshold is often left to the discretion of the analyst. However, sometimes the scores have an associated meaning, e.g., statistical significance (see [Section 9.3](#)), which makes the analysis of anomalies easier and more interpretable.

In the following sections, we provide brief descriptions of six types of anomaly detection approaches. For each type, we will describe their basic idea, key features, and underlying assumptions using illustrative examples. At the end of every section, we also discuss their strengths and weakness in handling different aspects of anomaly detection problems. To follow common practice, we will use the terms outlier and anomaly interchangeably in the remainder of this chapter.

9.3 Statistical Approaches

Statistical approaches make use of probability distributions (e.g., the Gaussian distribution) to model the normal class. A key feature of such distributions is that they associate a probability value to every data instance, indicating how likely it is for the instance to be generated from the distribution. Anomalies are then identified as instances that are unlikely to be generated from the probability distribution of the normal class.

There are two types of models that can be used to represent the probability distribution of the normal class: parametric models and non-parametric models. While parametric models use well-known families of statistical distributions that require estimating parameters from the data, non-parametric models are more flexible and learn the distribution of the normal class directly from the available data. In the following, we discuss both of these types of models for anomaly detection.

9.3.1 Using Parametric Models

Some of the common types of parametric models that are widely used for describing many types of data sets, include the Gaussian distribution, the Poisson distribution, and the binomial distribution. They involve parameters that need to be learned from the data, e.g., a Gaussian distribution requires identifying the mean and variance parameters from the data.

Parametric models are quite effective in representing the behavior of the normal class, especially when the normal class is known to follow a specific

distribution. The anomaly scores computed by parametric models also have strong theoretical properties, which can be used for analyzing the anomaly scores and assessing their statistical significance. In the following, we discuss the use of the Gaussian distribution for modeling the normal class, in the univariate and multivariate settings.

Using the Univariate Gaussian Distribution

The Gaussian (normal) distribution is one of the most frequently used distributions in statistics, and we will use it to describe a simple approach to statistical outlier detection. The Gaussian distribution has two parameters, μ and σ , which are the mean and standard deviation, respectively, and is represented using the notation $N(\mu, \sigma)$. The probability density function $f(x)$ of a point x under the Gaussian distribution is given as

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (9.1)$$

Figure 9.1 shows the probability density function of $N(0,1)$. We can see that the $p(x)$ declines as x moves farther from the center of the distribution. We can thus use the distance of a point x from the origin as an anomaly score. As we will see later in **Section 9.3.4**, this distance value has an interpretation in terms of probability that can be used to assess the confidence in calling x as an outlier.

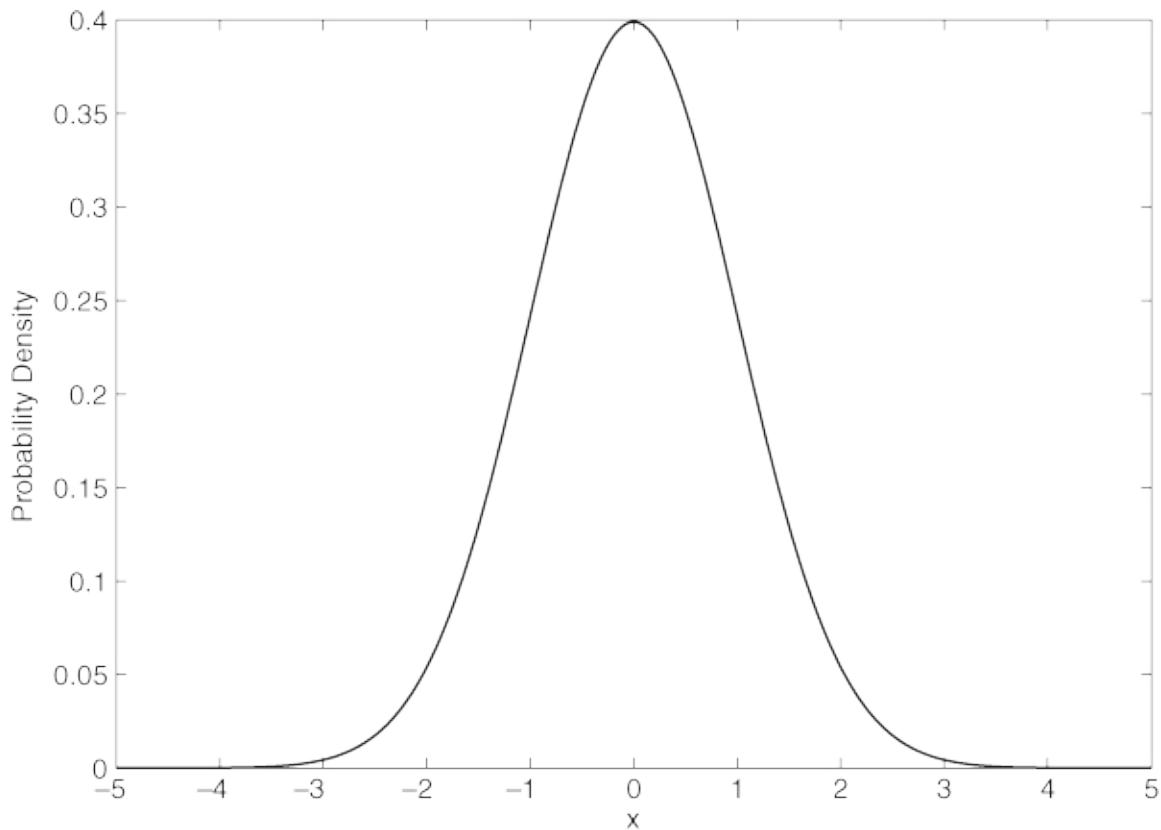


Figure 9.1.

Probability density function of a Gaussian distribution with a mean of 0 and a standard deviation of 1.

If the attribute of interest x follows a Gaussian distribution with mean μ and standard deviation σ , i.e., $N(\mu, \sigma)$, a common approach is to transform the attribute x to a new attribute z , which has a $N(0, 1)$ distribution. This can be done by using $z = (x - \mu) / \sigma$, which is called the z -score. Note that z^2 is directly related to the probability density of the point x in [Equation 9.1](#) since that equation can be rewritten as follows:

$$p(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (9.2)$$

The parameters μ and σ of the Gaussian distribution can be estimated from the training data of mostly normal instances, by using the sample mean \bar{x} as μ and the sample standard deviation s_x as σ . However, if we believe the

outliers are distorting the estimates of these parameters too much, more robust estimates of these quantities can be used—see Bibliographic Notes.

Using the Multivariate Gaussian Distribution

For a data set comprised of more than one continuous attribute, we can use a multivariate Gaussian distribution to model the normal class. A multivariate Gaussian distribution $N(\mu, \Sigma)$ involves two parameters, the mean vector μ and the covariance matrix Σ , which need to be estimated from the data. The probability density of a point x distributed as $N(\mu, \Sigma)$ is given by

$$f(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} e^{-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)}, \quad (9.3)$$

where p is the number of dimensions of x and $|\Sigma|$ denotes the determinant of the covariance matrix Σ .

In the case of a multivariate Gaussian distribution, the distance of a point x from the center μ cannot be directly used as a viable anomaly score. This is because a multivariate normal distribution is not symmetrical with respect to its center if there are correlations between the attributes. To illustrate this, [Figure 9.2](#) shows the probability density of a two-dimensional multivariate Gaussian distribution with mean of $(0,0)$ and a covariance matrix of

$$\Sigma = \begin{pmatrix} 1.00 & 0.75 \\ 0.75 & 3.00 \end{pmatrix}.$$

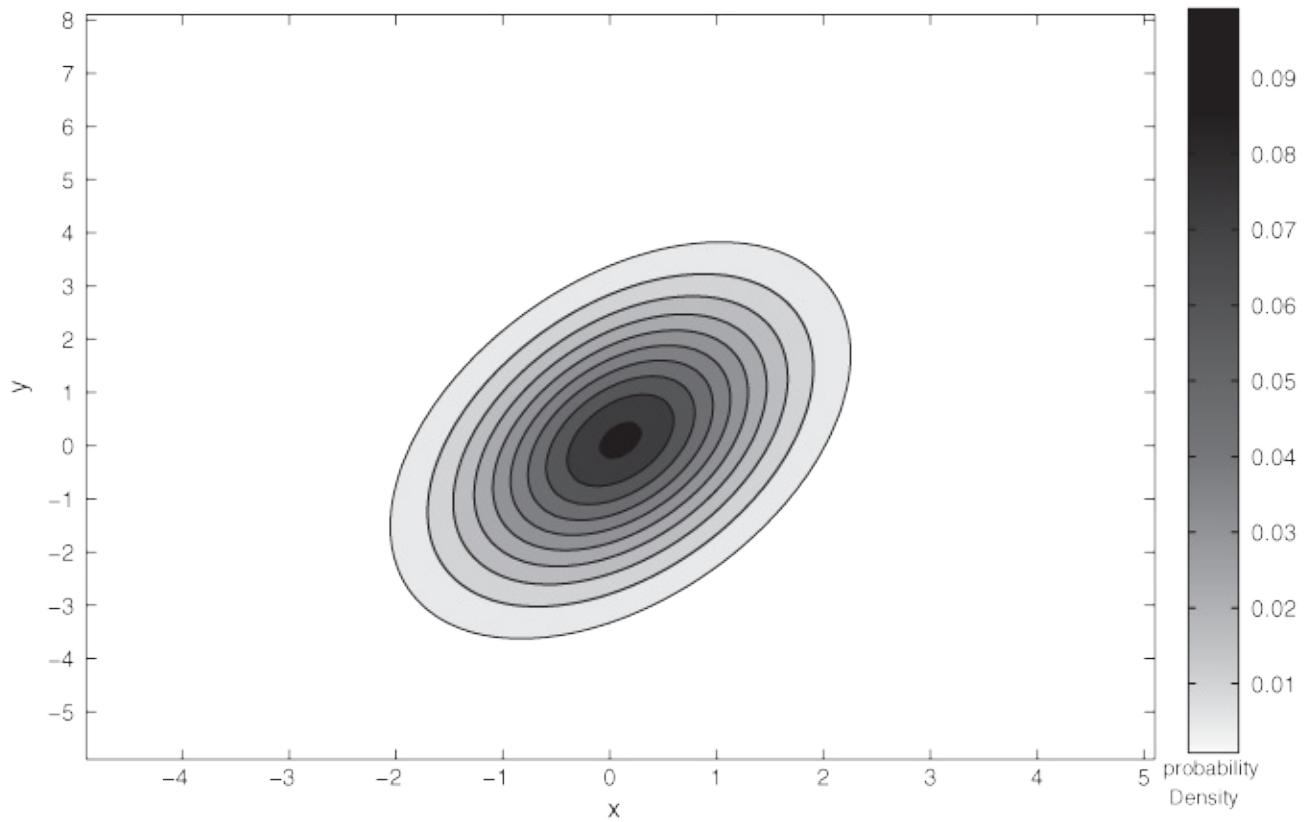


Figure 9.2.

Probability density of points for the Gaussian distribution used to generate the points of [Figure 9.3](#).

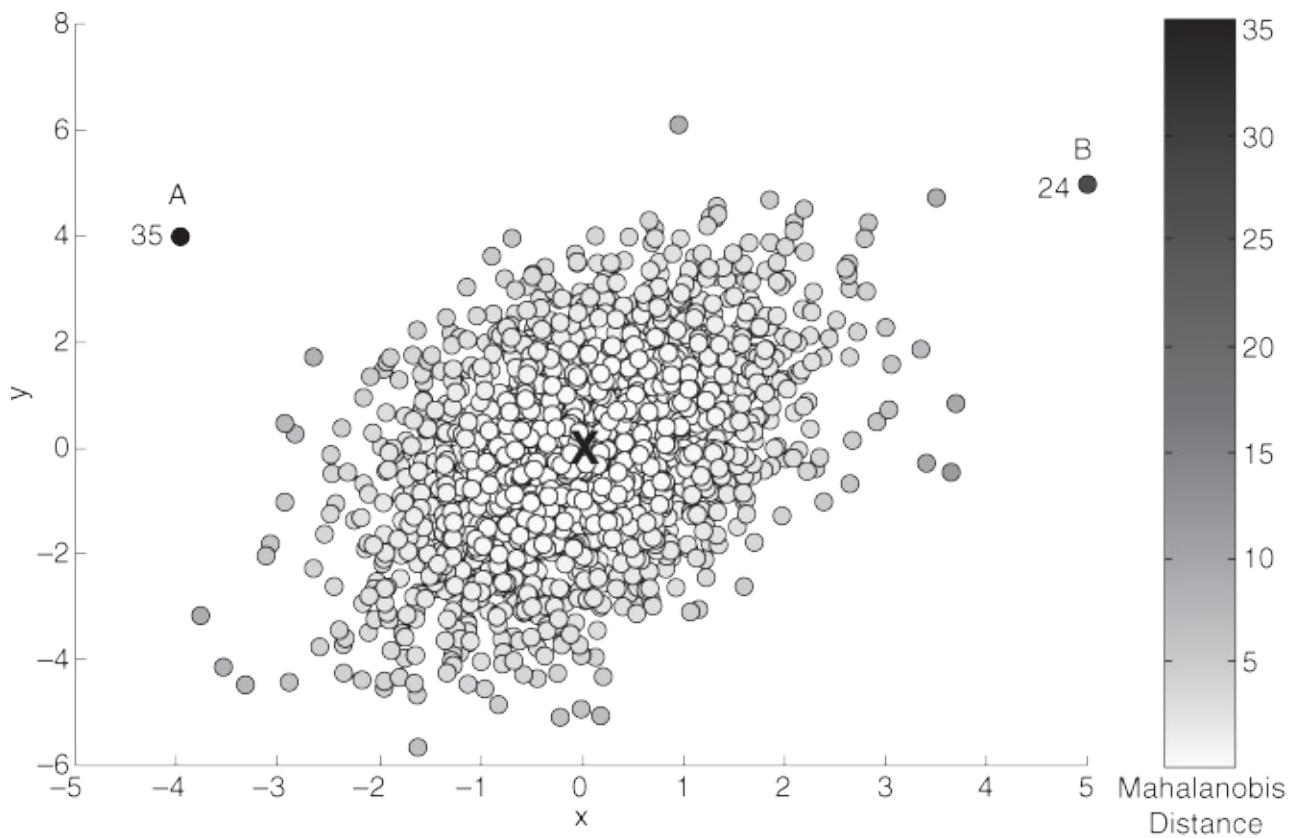


Figure 9.3.

Mahalanobis distance of points from the center of a two-dimensional set of 2002 points.

The probability density varies asymmetrically as we move outward from the center in different directions. To account for this fact, we need a distance measure that takes the shape of the data into consideration. The Mahalanobis distance is one such distance measure. (See [Equation 2.27](#) on page 96.) The Mahalanobis distance between a point \mathbf{x} and the mean of the data $\mathbf{x}_{\bar{}}$ is given by

$$\text{Mahalanobis}(\mathbf{x}, \mathbf{x}_{\bar{}}) = (\mathbf{x} - \mathbf{x}_{\bar{}})^T \mathbf{S}^{-1} (\mathbf{x} - \mathbf{x}_{\bar{}}), \quad (9.4)$$

where \mathbf{S} is the estimated covariance matrix of the data. Note that the Mahalanobis distance between \mathbf{x} and $\mathbf{x}_{\bar{}}$ is directly related to the probability

density of \mathbf{x} in [Equation 9.3](#), when \mathbf{x} — and \mathbf{S} are used as estimates of $\boldsymbol{\mu}$ and Σ , respectively. (See [Exercise 9](#) on page [751](#).)

Example 9.1 (Outliers in a Multivariate Normal Distribution).

[Figure 9.3](#) shows the Mahalanobis distance (from the mean of the distribution) for points in a two-dimensional data set. The points A (-4, 4) and B (5, 5) are outliers that were added to the data set, and their Mahalanobis distance is indicated in the figure. The other 2000 points of the data set were randomly generated using the distribution used for [Figure 9.2](#).

Both A and B have large Mahalanobis distances. However, even though A is closer to the center (the large black x at (0,0)) as measured by Euclidean distance, it is farther away than B in terms of the Mahalanobis distance because the Mahalanobis distance takes the shape of the distribution into account. In particular, point B has a Euclidean distance of 5.2 and a Mahalanobis distance of 24, while the point A has a Euclidean distance of 4.2 and a Mahalanobis distance of 35.

The above approaches assume that the normal class is generated from a single Gaussian distribution. Note that this may not always be the case, especially if there are multiple types of normal classes that have different means and variances. In such cases, we can use a Gaussian mixture model (as described in [Chapter 8.2.2](#)) to represent the normal class. For each point, the smallest Mahalanobis distance of the point to any of the Gaussian distributions is computed and used as the anomaly score. This approach is related to the clustering-based approaches for anomaly detection, which will be described in [Section 9.5](#).

9.3.2 Using Non-parametric Models

An alternative for modeling the distribution of the normal class is to use kernel density estimation-based techniques that employ kernel functions (described in [Section 8.3.3](#)) to approximate the density of the normal class from the available data. This results in the construction of a non-parametric probability distribution of the normal class, such that regions with a dense occurrence of normal instances have high probability and vice-versa. Note that kernel-based approaches do not assume that the data conforms to any known family of distributions but instead derive the distribution purely from the data. Having learned a probability density for the normal class using the kernel density approach, the anomaly score of an instance is computed as the inverse of its probability with respect to the learned density.

A simpler non-parametric approach to modeling the normal class is to build a histogram of the normal data. For example, if the data contains a single continuous attribute, then we can construct bins for different ranges of the attribute, using the equal-width discretization technique described in [Section 2.3.6](#). We can then check if a new test instance falls in any of the bins of the histogram. If it does not fall in any of the bins, we can identify it as an anomaly. Otherwise, we can use the inverse of the height (frequency) of the bin in which it falls as its anomaly score. This approach is known as the frequency-based or counting-based approach for anomaly detection.

A key step in using frequency-based approaches for anomaly detection is choosing the size of the bin used for constructing the histogram. A small bin size can falsely identify many normal instances as anomalous, since they might fall in empty or sparsely populated bins. On the other hand, if the bin size is too large, many anomalous instances may fall in heavily populated bins

and go unnoticed. Thus, choosing the right bin size is challenging, and often requires trying multiple size options or using expert knowledge.

9.3.3 Modeling Normal and Anomalous Classes

The statistical approaches described so far only model the distribution of the normal class but not the anomalous class. They assume that the training set predominantly has normal instances. However, if there are outliers present in the training data, which is common in most practical applications, the learning of the probability distributions corresponding to the normal class may be distorted, resulting in poor identification of anomalies.

Here, we present a statistical approach for anomaly detection that can tolerate a considerable fraction (λ) of outliers in the training set, provided that the outliers are uniformly distributed (and thus not clustered) in the attribute space. This approach makes use of a mixture modeling technique to learn the distribution of normal and anomalous classes. This approach is similar to the Expectation-Maximization (EM) based technique introduced in the context of clustering in [Chapter 8.2.2](#). Note that λ , the fraction of outliers, is like a prior.

The basic idea of this approach is to assume that instances are generated with probability λ from the anomalous class, which has uniform distribution, p_A , and with probability $1-\lambda$ from the normal class, which has the distribution, $f_M(\theta)$, where θ represents the parameters of the distribution. The approach for assigning training instances to the normal and anomaly classes can be described as follows. Initially, all the objects are assigned to

the normal class and the set of anomalous objects is empty. At every iteration of the EM algorithm, objects are transferred from the normal class to the anomaly class to improve the likelihood of the overall data. Let M_t and A_t be the set of normal and anomalous objects, respectively, at iteration t . The likelihood of the data set D , $\mathcal{L}_t(D)$, and its log-likelihood, $\log \mathcal{L}_t(D)$, are then given by the following equations:

$$\mathcal{L}_t(D) = \prod_{x_i \in D} P(x_i) = ((1-\lambda) |M_t| \prod_{x_i \in M_t} P_M(x_i, \theta_t))(\lambda |A_t| \prod_{x_i \in A_t} P_A(x_i))$$

$$\log \mathcal{L}_t(D) = |M_t| \log(1-\lambda) + \sum_{x_i \in M_t} \log P_M(x_i, \theta_t) + |A_t| \log \lambda + \sum_{x_i \in A_t} \log P_A(x_i)$$

where $|M_t|$ and $|A_t|$ are the number of objects in the normal and anomaly classes, respectively, and θ_t represents the parameters of the distribution of the normal class, which can be estimated using M_t . If the transfer of an object x from M_t to A_t results in a significant increase in the log-likelihood of the data (greater than a threshold c), then x is assigned to the set of outliers A_t . The set of outliers A_t keeps growing till we achieve the maximum likelihood of the data using M_t and A_t . This approach is summarized in [Algorithm 9.1](#).

Algorithm 9.1 Likelihood-based outlier detection.

- 1: Initialization: At time $t=0$, let M_t contain all the objects, while A_t is empty.
- 2: **for** each object x that belongs to M_t **do**
- 3: Move x from M_t to A_t to produce the new data sets A_{t+1} and M_{t+1} .

- 4: Compute the new log-likelihood of D , $\log \mathcal{L} t+1 (D)$
- 5: Compute the difference, $\Delta = \log \mathcal{L} t+1 (D) - \log \mathcal{L} t (D)$
- 6: **if** $\Delta > c$, where c is some threshold **then**
- 7: Classify x as an anomaly.
- 8: Increment t by one and use $M t+1$ and $A t+1$ in the next iteration.
- 9: **end if**
- 10: **end for**

Because the number of normal objects is large compared to the number of anomalies, the distribution of the normal objects may not change much when an object is moved to the set of anomalies. In that case, the contribution of each normal object to the overall likelihood of the normal objects will remain relatively constant. Furthermore, each object moved to the set of anomalies contributes a fixed amount to the likelihood of the anomalies. Thus, the overall change in the total likelihood of the data when an object is moved to the set of anomalies is roughly equal to the probability of the object under a uniform distribution (weighted by λ) minus the probability of the object under the distribution of the normal data objects (weighted by $1-\lambda$). Consequently, the set of anomalies will tend to consist of those objects that have significantly higher probability under a uniform distribution than under the distribution of the normal objects.

In the situation just discussed, the approach described by [Algorithm 9.1](#) is roughly equivalent to classifying objects with a low probability under the distribution of normal objects as outliers. For example, when applied to the points in [Figure 9.3](#), this technique would classify points A and B (and other points far from the mean) as outliers. However, if the distribution of the normal objects changes significantly as anomalies are removed or the distribution of the anomalies can be modeled in a more sophisticated manner, then the results produced by this approach will be different than the results of simply classifying low-probability objects as outliers. Also, this approach can work

even when the distribution of normal objects is multi-modal, e.g., by using a mixture of Gaussian distributions for $f_M(\theta)$. Also, conceptually, it should be possible to use this approach with distributions other than Gaussian.

9.3.4 Assessing Statistical Significance

Statistical approaches provide a way to assign a measure of confidence for the instances detected as anomalies. For example, since the anomaly scores computed by statistical approaches have a probabilistic meaning, we can apply a threshold to these scores with statistical guarantees. Alternatively, it is possible to define statistical tests (also termed as **discordancy tests**) that can identify the statistical significance of an instance being identified as an anomaly by a statistical approach. Many of these discordancy tests are highly specialized and assume a level of statistical knowledge beyond the scope of this text. Thus, we illustrate the basic ideas with a simple example that uses univariate Gaussian distributions, and refer the reader to the Bibliographic Notes for further pointers.

Consider the Gaussian distribution $N(0,1)$ shown in [Figure 9.1](#). As discussed previously in [Section 9.3.1](#), most of the probability density is centered around zero and there is little probability that an object (value) belonging to $N(0,1)$ will occur in the tails of the distribution. For instance, there is only a probability of 0.0027 that an object lies beyond the central area between ± 3 standard deviations. More generally, if c is a constant and x is the attribute value of an object, then the probability that $|x| \geq c$ decreases rapidly as c increases. Let $\alpha = \text{prob}(|x| \geq c)$. [Table 9.1](#) shows some sample values for c and the corresponding values for α when the distribution is $N(0,1)$. Note that a value that is more than 4 standard deviations from the mean is a one-in-a-thousand occurrence.

Table 9.1. Sample pairs (c,α) , $\alpha = \text{prob}(|x| \geq c)$ for a Gaussian distribution with mean 0 and standard deviation 1.

c	α for $N(0,1)$
1.00	0.3173
1.50	0.1336
2.00	0.0455
2.50	0.0124
3.00	0.0027
3.50	0.0005
4.00	0.0001

This interpretation of the distance of a point from the center can be used as the basis of a test to assess whether an object is an outlier, using the following definition.

Definition 9.2 (Outlier for a Single $N(0,1)$ Gaussian Attribute).

An object with attribute value x from a Gaussian distribution with mean of 0 and standard deviation 1 is an outlier if

$$|x| \geq c, \quad (9.7)$$

where c is a constant chosen so that $P(|x| \geq c) = \alpha$, where P represents probability.

To use this definition it is necessary to specify a value for α . From the viewpoint that unusual values (objects) indicate a value from a different distribution, α indicates the probability that we mistakenly classify a value from the given distribution as an outlier. From the viewpoint that an outlier is a rare value of a $N(0,1)$ distribution, α specifies the degree of rareness.

More generally, for a Gaussian distribution with mean μ and standard deviation σ , we can first compute the z score of x and then apply the above test on x . In practice, this works well when μ and σ are estimated from a large population. A more sophisticated statistical procedure (Grubbs' test), which takes into account the distortion of parameter estimates caused by outliers, is explored in [Exercise 7](#) on page [750](#).

The approach to outlier detection presented here is equivalent to testing data objects for statistical significance and classifying the statistically significant objects as anomalies. This is discussed in more detail in [Chapter 10](#).

9.3.5 Strengths and Weaknesses

Statistical approaches to outlier detection have a firm theoretical foundation and build on standard statistical techniques. When there is sufficient knowledge of the data and the type of test that should be applied, these approaches are statistically justifiable and can be very effective. They can also provide confidence intervals associated with the anomaly scores, which

can be very helpful in making decisions about test instances, e.g., determining thresholds on the anomaly score.

However, if the wrong model is chosen, then a normal instance can be erroneously identified as an outlier. For example, the data may be modeled as coming from a Gaussian distribution, but may actually come from a distribution that has a higher probability (than the Gaussian distribution) of having values far from the mean. Statistical distributions with this type of behavior are common in practice and are known as **heavy-tailed distributions**. Also, we note that while there are a wide variety of statistical outlier tests for single attributes, far fewer options are available for multivariate data, and these tests can perform poorly for high-dimensional data.

9.4 Proximity-based Approaches

Proximity-based methods identify anomalies as those instances that are most distant from the other objects. This relies on the assumption that normal instances are related and hence appear *close* to each other, while anomalies are different from the other instances and hence are relatively far from other instances. Since many of the proximity-based techniques are based on distances, they are also referred to as **distance-based outlier detection techniques**.

Proximity-based approaches are *model-free* anomaly detection techniques, since they do not construct an explicit model of the normal class for computing the anomaly score. They make use of the local perspective of every data instance to compute its anomaly score. They are more general than statistical approaches, since it is often easier to determine a meaningful proximity measure for a data set than to determine its statistical distribution. In the following, we present some of the basic proximity-based approaches for defining an anomaly score. Primarily, these techniques differ in the way they analyze the locality of a data instance.

9.4.1 Distance-based Anomaly Score

One of the simplest ways to define a proximity-based anomaly score of a data instance x is to use the distance to its k th nearest neighbor, $\text{dist}(x,k)$. If an instance x has many other instances located close to it (characteristic of the normal class), it will have a low value of $\text{dist}(x,k)$. On other hand, an

anomalous instance x will be quite distant from its k -neighboring instances and would thus have a high value of $\text{dist}(x,k)$.

Figure 9.4 shows a set of points in a two-dimensional space that have been shaded according to their distance to the k th nearest neighbor, $\text{dist}(x,k)$ (where $k=5$). Note that point C has been correctly assigned a high anomaly score, as it is located far away from other instances.

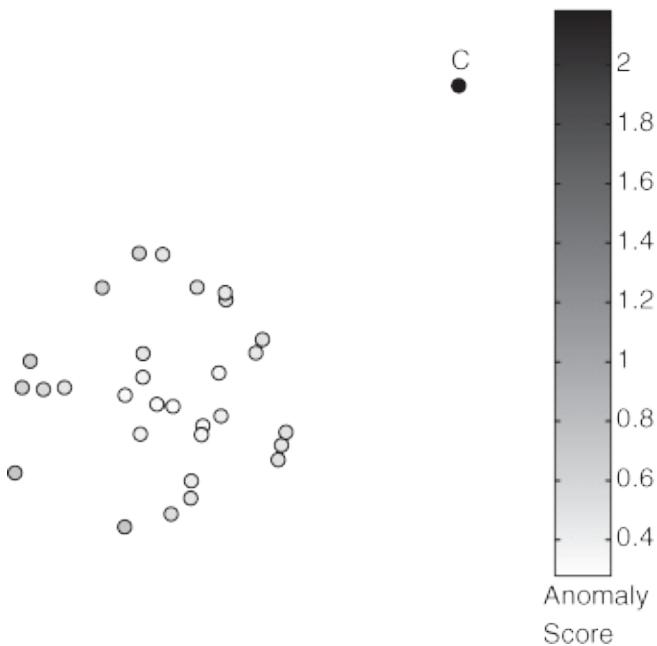


Figure 9.4.

Anomaly score based on the distance to fifth nearest neighbor.

Note that $\text{dist}(x,k)$ can be quite sensitive to the value of k . If k is too small, e.g., 1, then a small number of outliers located close to each other can show a low anomaly score. For example, **Figure 9.5** shows anomaly scores using $k=1$ for a set of normal points and two outliers that are located close to each other (shading reflects anomaly scores). Note that both C and its neighbor have a low anomaly score. If k is too large, then it is possible for all objects in a cluster that has fewer than k objects to become anomalies. For example, **Figure 9.6** shows a data set that has a small cluster of size 5 and a larger

cluster of size 30. For k=5 , the anomaly score of all points in the smaller cluster is very high.

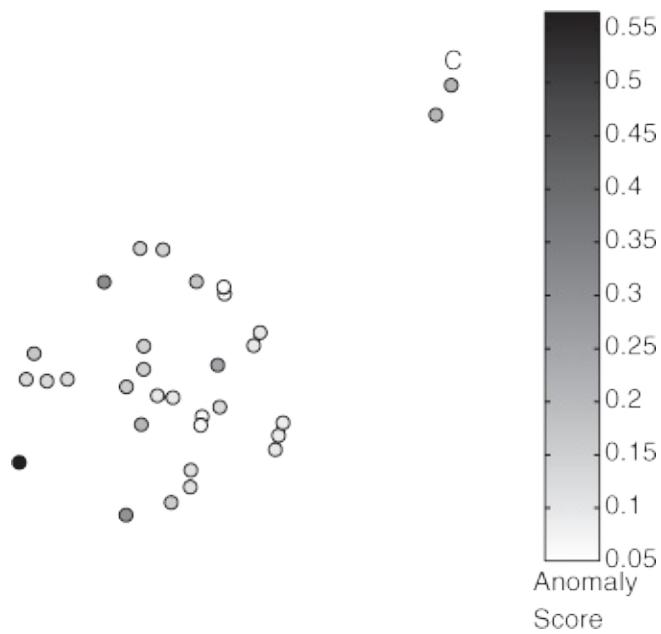


Figure 9.5.

Anomaly score based on the distance to the first nearest neighbor. Nearby outliers have low anomaly scores.

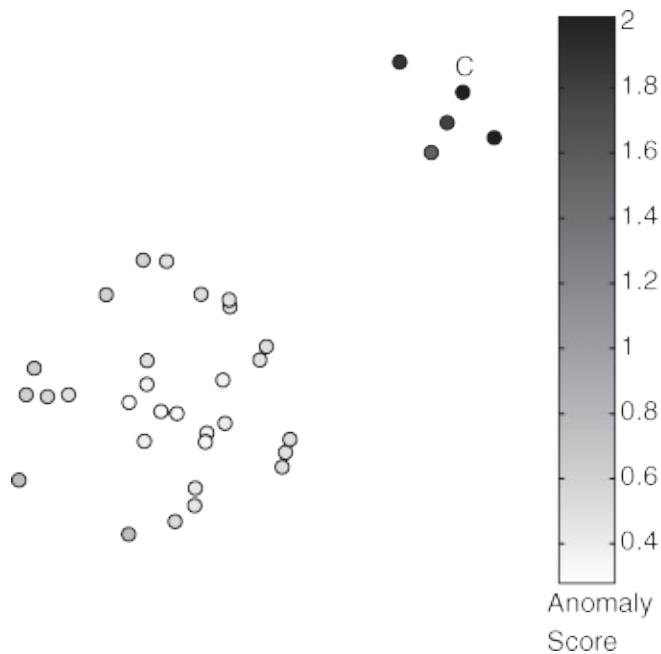


Figure 9.6.

Anomaly score based on distance to the fifth nearest neighbor. A small cluster becomes an outlier.

An alternative distance-based anomaly score that is more robust to the choice of k is the average distance to the first k -nearest neighbors, $\text{avg.dist}(x,k)$. Indeed, $\text{avg.dist}(x,k)$ is widely used in a number of applications as a reliable proximity-based anomaly score.

9.4.2 Density-based Anomaly Score

The density around an instance can be defined as $n/V(d)$, where n is the number of instances within a specified distance d from the instance, and $V(d)$ is the volume of the neighborhood. Since $V(d)$ is constant for a given d , the density around an instance is often represented using the number of instances n within a fixed distance d . This definition is similar to the one used by the DBSCAN clustering algorithm in [Section 7.4](#). From a density-based viewpoint, anomalies are instances that are in regions of low density. Hence, an anomaly will have a smaller number of instances within a distance d than a normal instance.

Similar to the trade-off in choosing the parameter k in distance-based measures, it is challenging to choose the parameter d in density-based measures. If d is too small, then many normal instances can incorrectly show low density values. If d is too large, then many anomalies may have densities that are similar to normal instances.

Note that the distance-based and density-based views of proximity are quite similar to each other. To realize this, consider the k -nearest neighbors of a data instance x , whose distance to the k th nearest neighbor is given by

$\text{dist}(x,k)$. In this approach, $\text{dist}(x,k)$ provides a measure of the density around x , using a different value of d for every instance. If $\text{dist}(x,k)$ is large, the density around x is small, and vice-versa. Distance-based and density-based anomaly scores thus follow an inverse relationship. This can be used to define the following measures of density that are based on the two distance measures, $\text{dist}(x,k)$ and $\text{avg.dist}(x,k)$:

$$\text{density}(x,k) = 1/\text{dist}(x,k), \text{ avg.density}(x,k) = 1/\text{avg.dist}(x,k).$$

9.4.3 Relative Density-based Anomaly Score

The above proximity-based approaches only consider the locality of an individual instance for computing its anomaly score. In scenarios where the data contains regions of varying densities, such methods would not be able to correctly identify anomalies, as the notion of a *normal* locality would change across regions.

To illustrate this, consider the set of two-dimensional points in [Figure 9.7](#). This figure has one rather loose cluster of points, another dense cluster of points, and two points, C and D, which are quite far from these two clusters. Assigning anomaly scores to points according to $\text{dist}(x,k)$ with $k=5$ correctly identifies point C to be an anomaly, but shows a low score for point D. In fact, the score for D is much lower than many points that are part of the loose cluster. To correctly identify anomalies in such data sets, we need a notion of density that is relative to the densities of neighboring instances. For example, point D in [Figure 9.7](#) has a higher absolute density than point A, but its density is lower relative to its nearest neighbors.

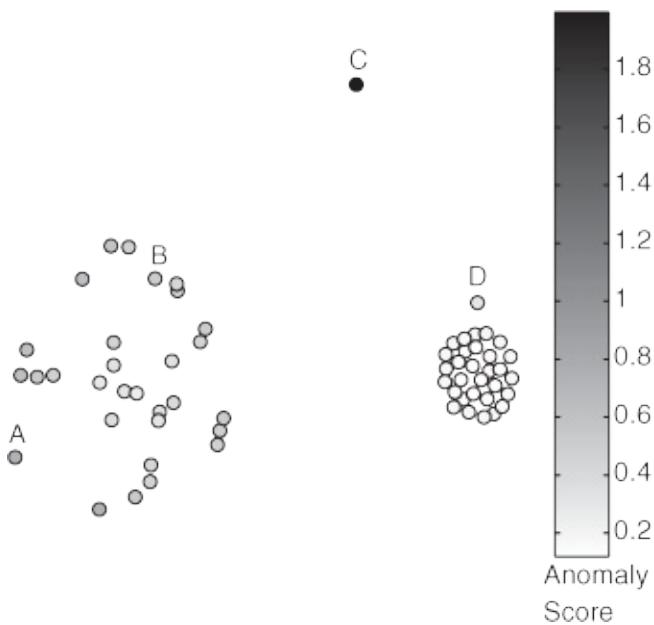


Figure 9.7.

Anomaly score based on the distance to the fifth nearest neighbor, when there are clusters of varying densities.

There are many ways to define the relative density of an instance. For a point x , One approach is to compute the ratio of the average density of its k -nearest neighbors, y_1 to y_k to the density of x , as follows:

$$\text{relative density}(x,k) = \sum_{i=1}^k \text{density}(y_i, k) / k \text{ density}(x, k). \quad (9.8)$$

The relative density of a point is high when the average density of points in its neighborhood is significantly higher than the density of the point.

Note that by replacing $\text{density}(x, k)$ with $\text{avg.density}(x, k)$ in the above equation, we can obtain a more robust measure of relative density. The above approach is similar to that used by the Local Outlier Factor (LOF) score, which is a widely-used measure for detecting anomalies using relative density. (See Bibliographic Notes.) However, LOF uses a somewhat different definition of density to achieve results that are more robust.

Example 9.2 (Relative Density Anomaly Detection).

Figure 9.8 shows the performance of the relative density-based anomaly detection method on the example data set used previously in **Figure 9.7**. The anomaly score of every point is computed using **Equation 9.8** (with $k=10$). The shading of every point represents its score, i.e., points with a higher score are darker. We have labeled points A, C, and D, which have the largest anomaly scores. Respectively, these points are the most extreme anomaly, the most extreme point with respect to the compact set of points, and the most extreme point in the loose set of points.

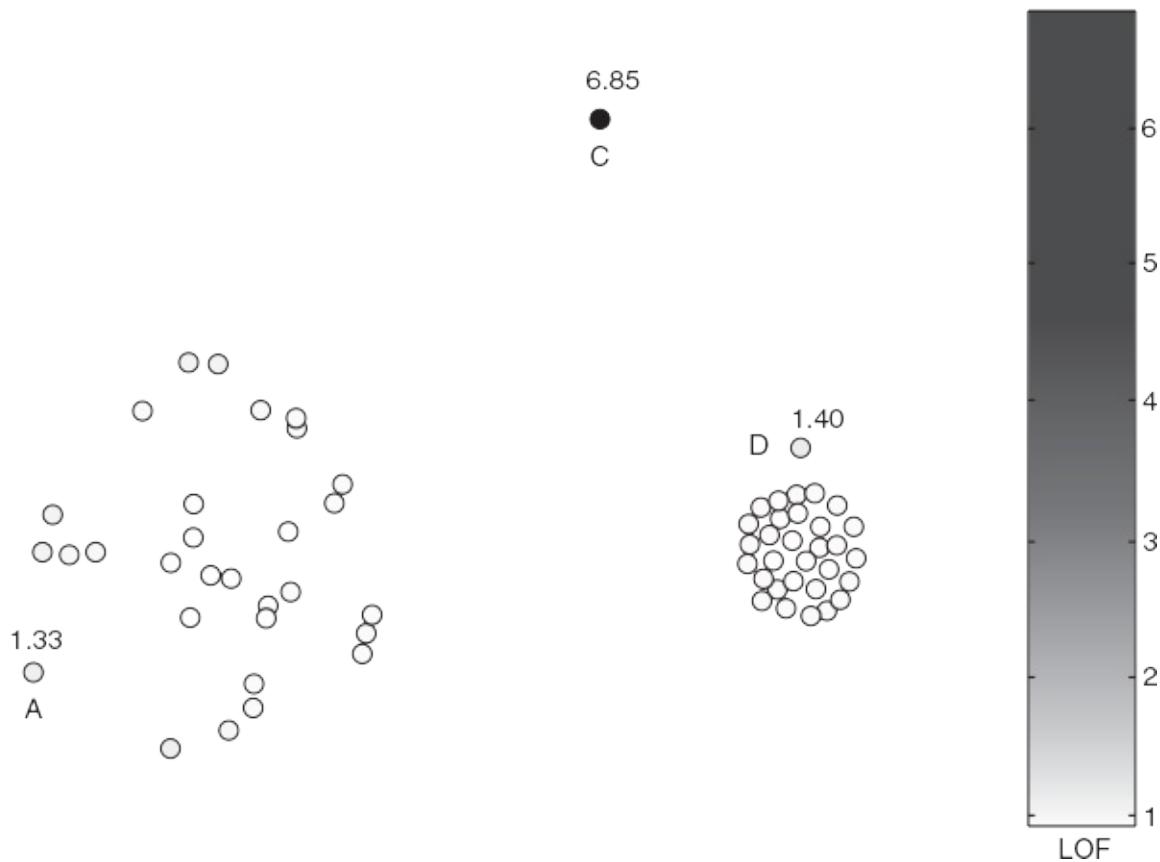


Figure 9.8.

Relative density (LOF) outlier scores for two-dimensional points of [Figure 9.7](#).

9.4.4 Strengths and Weaknesses

Proximity-based approaches are non-parametric in nature and hence are not restricted to any particular form of distribution of the normal and anomalous classes. They have a broad applicability over a wide range of anomaly detection problems where a reasonable proximity measure can be defined between instances. They are quite intuitive and visually appealing, since proximity-based anomalies can be interpreted visually when the data can be displayed in two- or three-dimensional scatter plots.

However, the effectiveness of proximity-based methods depends greatly on the choice of the distance measure. Defining distances in high-dimensional spaces can be challenging. In some cases, dimensionality reduction techniques can be used to map the instances into a lower dimensional feature space. Proximity-based methods can then be applied in the reduced space for detecting anomalies. Another challenge common to all proximity-based methods is their high computational complexity. Given n points, computing the anomaly score for every point requires considering all pair-wise distances, resulting in an $O(n^2)$ running time. For large data sets this can be too expensive, although specialized algorithms can be used to improve performance in some cases, e.g., with low-dimensional data sets. Choosing the right value of parameters (k or d) in proximity-based methods is also difficult and often requires domain expertise.

9.5 Clustering-based Approaches

Clustering-based methods for anomaly detection use clusters to represent the normal class. This relies on the assumption that normal instances appear close to each other and hence can be grouped into clusters. Anomalies are then identified as instances that do not fit well in the clustering of the normal class, or appear in small clusters that are far apart from the clusters of the normal class. Clustering-based methods can be categorized into two types: methods that consider small clusters as anomalies, and methods that define a point as anomalous if does not fit the clustering well, typically as measured by distance from a cluster center. We describe both types of clustering-based methods next.

9.5.1 Finding Anomalous Clusters

This approach assumes the presence of *clustered anomalies* in the data, where the anomalies appear in tight groups of small size. Clustered anomalies appear when the anomalies are being generated from the same anomalous class. For example, a network attack may have a common pattern in its occurrence, possibly because of a common attacker, who appears in similar ways in multiple instances.

Clusters of anomalies are generally small in size, since anomalies are rare in nature. They are also expected to be quite distant from the clusters of the normal class, since anomalies do not conform to normal patterns or behavior. Hence, a basic approach for detecting anomalous clusters is to cluster the

overall data and flag clusters that are either too small in size or too far from other clusters.

For instance, if we use a prototype-based method for clustering the overall data, e.g., using K-means, every cluster can be represented by its prototype, e.g., the centroid of the cluster. We can then treat every prototype as a point and straightforwardly identify clusters that are distant from the rest. As another example, if we are using hierarchical techniques such as MIN, MAX, or Group Average—see [Section 7.3](#)—then anomalies are often identified as those instances that are in small clusters or remain singletons even after almost all other points have been clustered.

9.5.2 Finding Anomalous Instances

From a clustering perspective, another way of describing an anomaly is as an instance that cannot be explained well by any of the normal clusters. Hence, a basic approach for anomaly detection is to first cluster all the data (comprised mainly of normal instances) and then assess the degree to which every instance belongs to its respective cluster. For example, if we use K-means clustering, the distance of an instance to its cluster centroid represents how strongly it belongs to the cluster. Instances that are quite distant from their respective cluster centroids can thus be identified as anomalies.

Although clustering-based methods for anomaly detection are quite intuitive and simple to use, there are a number of considerations that must be kept in mind while using them, as we discuss in the following.

Assessing the Extent to Which an Object

Belongs to a Cluster

For prototype-based clusters, there are several ways to assess the extent to which an instance belongs to a cluster. One method is to measure the distance of an instance from its cluster prototype and consider this as the anomaly score of the instance. However, if the clusters are of differing densities, then we can construct an anomaly score that measures the relative distance of an instance from the cluster prototype with respect to the distances of the other instances in the cluster. Another possibility, provided that the clusters can be accurately modeled in terms of Gaussian distributions, is to use the Mahalanobis distance.

For clustering techniques that have an objective function, we can assign an anomaly score to an instance that reflects the improvement in the objective function when that instance is eliminated from the overall data. However, such an approach is often computationally intensive. For that reason, the distance-based approaches of the previous paragraph are usually preferred.

Example 9.3 (Clustering-Based Example).

This example is based on the set of points shown in [Figure 9.7](#). Prototype-based clustering in this example uses the K-means algorithm, and the anomaly score of a point is computed in two ways: (1) by the point's distance from its closest centroid, and (2) by the point's relative distance from its closest centroid, where the relative distance is the ratio of the point's distance from the centroid to the median distance of all points in the cluster from the centroid. The latter approach is used to adjust for the large difference in density between compact and loose clusters.

The resulting anomaly scores are shown in [Figures 9.9](#) and [9.10](#). As before, the anomaly score, measured in this case by the distance or

relative distance, is indicated by the shading. We use two clusters in each case. The approach based on raw distance has problems with the differing densities of the clusters, e.g., D is not considered an outlier. For the approach based on relative distances, the points that have previously been identified as outliers using LOF (A, C, and D) also show up as anomalies here.

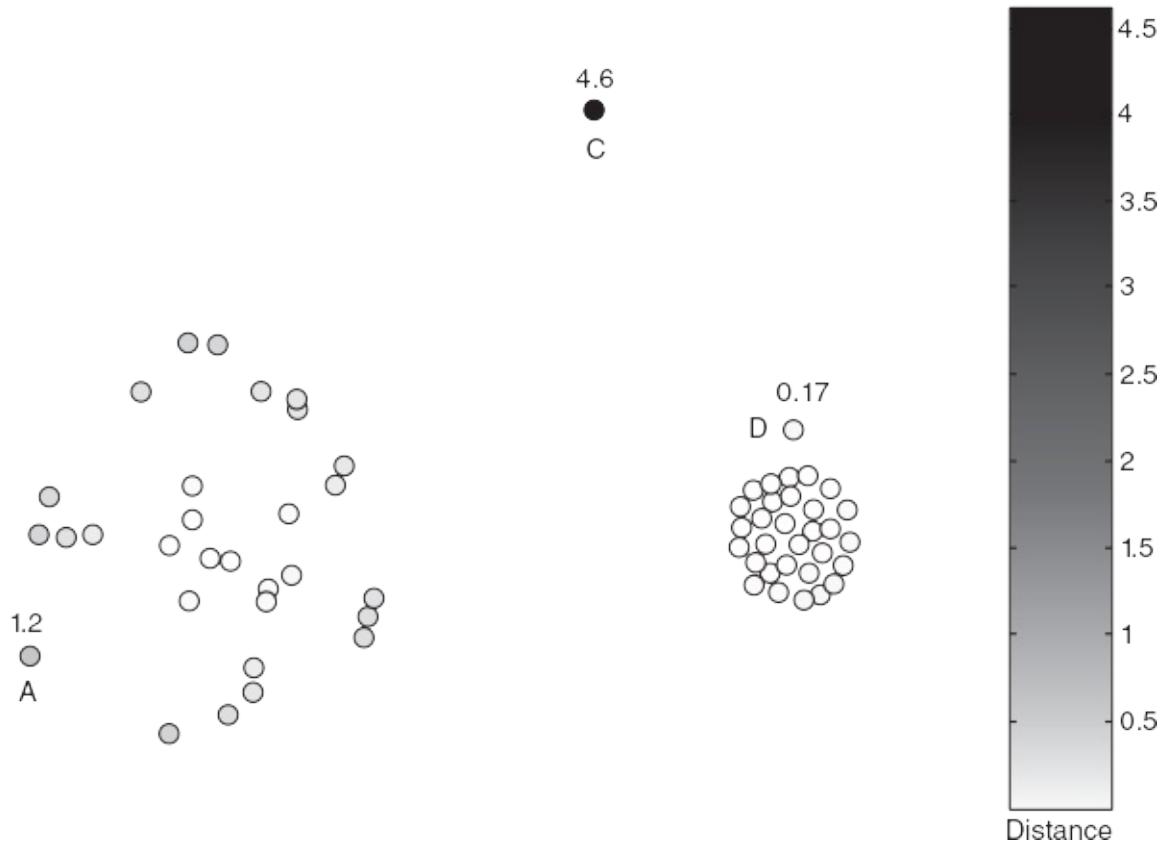


Figure 9.9.

Distance of points from closest centroid.

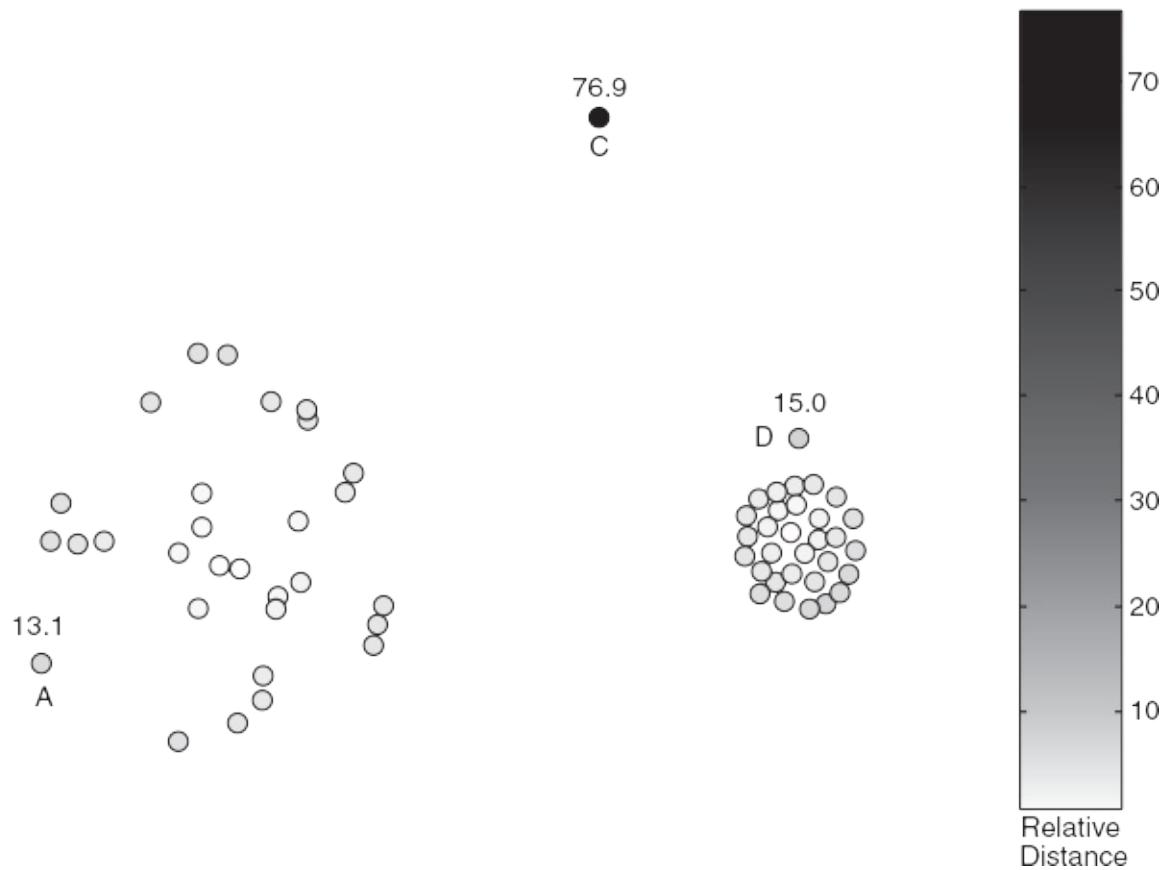


Figure 9.10.

Relative distance of points from closest centroid.

Impact of Outliers on the Initial Clustering

Clustering based schemes are often sensitive to the presence of outliers in the data. Hence, the presence of outliers can degrade the quality of clusters corresponding to the normal class since these clusters are discovered by clustering the overall data, which is comprised of normal and anomalous instances. To address this issue, the following approach can be used: instances are clustered, outliers, which are the points farthest from any cluster, are removed, and then the instances are clustered again. This approach is used at every iteration of the K-means algorithm. The K-means++ algorithm is an example of such an algorithm. While there is no guarantee that this approach will yield optimal results, it is easy to use.

A more sophisticated approach is to have a special group for instances that do not currently fit well in any cluster. This group represents potential outliers. As the clustering process proceeds, clusters change. Instances that no longer belong strongly to any cluster are added to the set of potential outliers, while instances currently in the outlier group are tested to see if they now strongly belong to a cluster and can be removed from the set of potential outliers. The instances remaining in the set at the end of the clustering are classified as outliers. Again, there is no guarantee of an optimal solution or even that this approach will work better than the simpler one described previously.

The Number of Clusters to Use

Clustering techniques such as K-means do not automatically determine the number of clusters. This is a problem when using clustering-based methods for anomaly detection, since whether an object is considered an anomaly or not may depend on the number of clusters. For instance, a group of 10 objects may be relatively close to one another, but may be included as part of a larger cluster if only a few large clusters are found. In that case, each of the 10 points could be regarded as an anomaly, even though they would have formed a cluster if a large enough number of clusters had been specified.

As with some of the other issues, there is no simple answer to this problem. One strategy is to repeat the analysis for different numbers of clusters. Another approach is to find a large number of small clusters. The idea is that (1) smaller clusters tend to be more cohesive and (2) if an object is an anomaly even when there are a large number of small clusters, then it is likely a true anomaly. The downside is that groups of anomalies may form small clusters and thus escape detection.

9.5.3 Strengths and Weaknesses

Clustering-based techniques can operate in an unsupervised setting as they do not require training data consisting of only normal instances. Along with identifying anomalies, the learned clusters of the normal class help in understanding the nature of the normal data. Some clustering techniques, such as K-means, have linear or near-linear time and space complexity and thus, an anomaly detection technique based on such algorithms can be highly efficient. However, the performance of clustering-based anomaly detection methods is heavily dependent upon the number of clusters used as well as the presence of outliers in the data. As discussed in [Chapters 7](#) and [8](#), each clustering algorithm is suitable only for a certain type of data; hence the clustering algorithm needs to be chosen carefully to effectively capture the cluster structure in the data.

9.6 Reconstruction-based Approaches

Reconstruction-based techniques rely on the assumption that the normal class resides in a space of lower dimensionality than the original space of attributes. In other words, there are patterns in the distribution of the normal class that can be captured using lower-dimensional representations, e.g., by using dimensionality reduction techniques.

To illustrate this, consider a data set of normal instances, where every instance is represented using p continuous attributes, x_1, \dots, x_p . If there is a hidden structure in the normal class, we can expect to approximate this data using fewer than p derived features. One common approach for deriving useful features from a data set is to use principal components analysis (PCA), as described in [Section 2.3.3](#). By applying PCA on the original data, we obtain p principal components, y_1, \dots, y_p , where every principal component is a linear combination of the original attributes. Each principal component captures the maximum amount of variation in the original data subject to the constraint that it must be orthogonal to the preceding principal components. Thus, the amount of variation captured decreases for each successive principal component, and hence, it is possible to approximate the original data using the top k principal components, y_1, \dots, y_k . Indeed, if there is a hidden structure in the normal class, we can expect to obtain a good approximation using a smaller number of features, $k < p$.

Once, we have derived a smaller set of k features, we can project any new data instance \mathbf{x} to its k -dimensional representation \mathbf{y} . Moreover, we can also re-project \mathbf{y} back to the original space of p attributes, resulting in a *reconstruction* of \mathbf{x} . Let us denote this reconstruction as \mathbf{x}^{\wedge} and the squared Euclidean distance between \mathbf{x} and \mathbf{x}^{\wedge} as the *reconstruction error*.

$$\text{Reconstruction Error}(x) = \|x - \hat{x}\|^2$$

Since the low-dimensional features are specifically learned to explain most of the variation in the normal data, we can expect the reconstruction error to be low for normal instances. However, the reconstruction error is high for anomalous instances, as they do not conform to the hidden structure of the normal class. The reconstruction error can thus be used as an effective anomaly detection score.

As an illustration of a reconstruction-based approach for anomaly detection, consider a two-dimensional data set of normal instances, shown as circles in [Figure 9.11](#). The black squares are anomalous instances. The solid black line shows the first principal component learned from this data, which corresponds to the direction of maximum variance of normal instances.

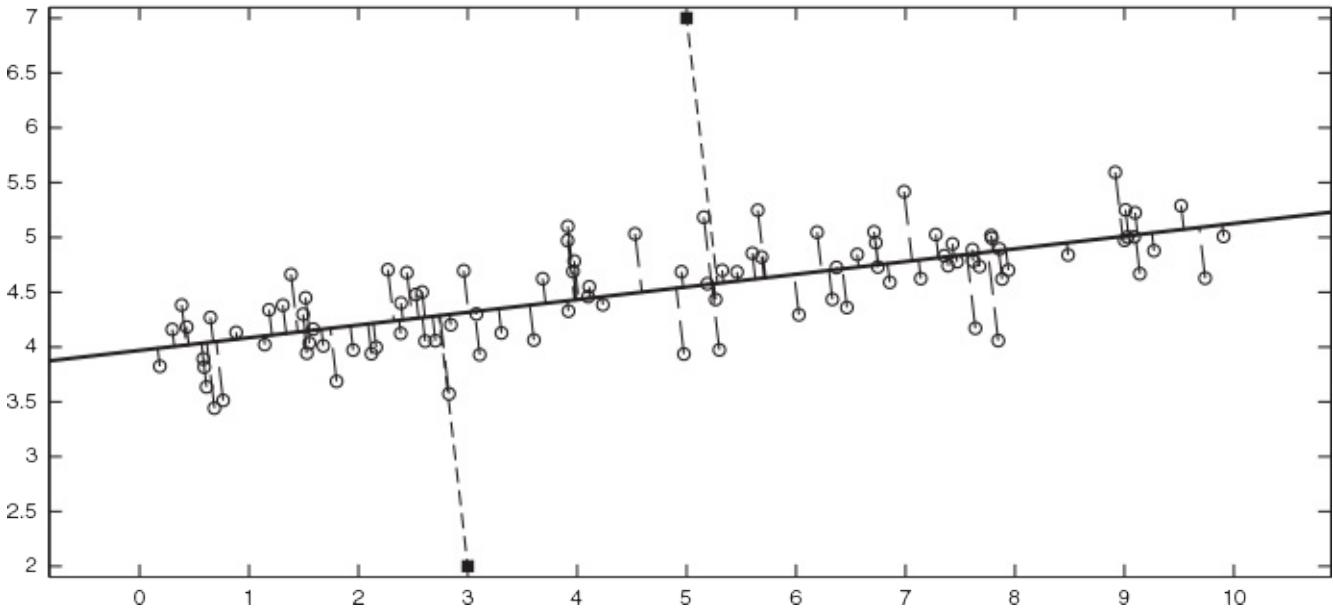


Figure 9.11.

Reconstruction of a two-dimensional data using a single principal component (shown as solid black line).

We can see that most of the normal instances are centered around this line.

This suggests that the first principal component provides a good approximation to the normal class using a lower-dimensional representation. Using this representation, we can project every data instance \mathbf{x} to a point on the line. This projection, \mathbf{x}^{\wedge} , serves as a reconstruction of the original instance using a single principal component.

The distance between \mathbf{x} and \mathbf{x}^{\wedge} , which corresponds to the reconstruction error of \mathbf{x} , is shown as dashed lines in [Figure 9.11](#). We can see that, since the first principal component has been learned to best fit the normal class, the reconstruction errors of the normal instances are quite small in value. However, the reconstruction errors for anomalous instances (shown as squares) are high, since they do not adhere to the structure of the normal class.

Although PCA provides a simple approach for capturing low-dimensional representations, it can only derive features that are linear combinations of the original attributes. When the normal class exhibits nonlinear patterns, it is difficult to capture them using PCA. In such scenarios, the use of an artificial neural network known as **autoencoder** provides one possible approach for nonlinear dimensionality reduction and reconstruction. As described in [Section 4.7](#), autoencoders are widely used in the context of deep learning to derive complex features from the training data in an unsupervised setting.

An autoencoder (also referred to as an autoassociator or a mirroring network) is a multi-layer neural network, where the number of input and output neurons is equal to the number of original attributes. [Figure 9.12](#) shows the general architecture of an autoencoder, which involves two basic steps, **encoding** and **decoding**. During encoding, a data instance \mathbf{x} is transformed to a low-dimensional representation \mathbf{y} , using a number of nonlinear transformations in the encoding layers. Notice that the number of neurons reduces at every encoding layer, so as to learn low-dimensional representations from the

original data. The learned representation \mathbf{y} is then mapped back to the original space of attributes using the decoding layers, resulting in a reconstruction of \mathbf{x} , denoted by \mathbf{x}^{\wedge} . The distance between \mathbf{x} and \mathbf{x}^{\wedge} (the reconstruction error) is then used as a measure of an anomaly score.

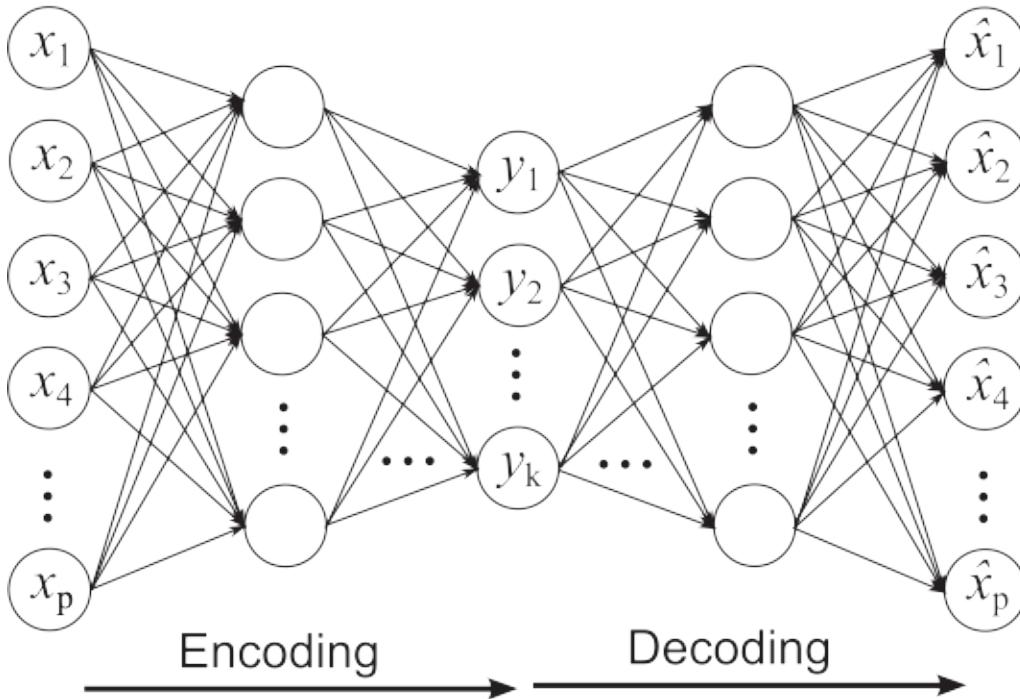


Figure 9.12.

A basic architecture of the autoencoder.

In order to learn an autoencoder from an input data set comprising primarily of normal instances, we can use the backpropagation techniques introduced in the context of artificial neural networks in [Section 4.7](#). The autoencoder scheme provides a powerful approach for learning complex and nonlinear representations of the normal class. A number of variants of the basic autoencoder scheme described above have also been explored to learn representations in different types of data sets. For example, the **denoising autoencoder** is able to robustly learn nonlinear representations from the training data, even in the presence of noise. For more details on the different types of autoencoders, see the Bibliographic Notes.

9.6.1 Strengths and Weaknesses

Reconstruction-based techniques provide a generic approach for modeling the normal class that does not require many assumptions about the distribution of normal instances. They are able to learn a rich variety of representations of the normal class by using a broad family of dimensionality reduction techniques. They can also be used in the presence of irrelevant attributes, since an attribute that does not share any relationship with the other attributes is likely to be ignored in the encoding step, as it would not be of much use in reconstructing the normal class. However, since the reconstruction error is computed by measuring the distance between \mathbf{x} and \mathbf{x}^* in the original space of attributes, performance can suffer when the number of attributes is large.

9.7 One-class Classification

One-class classification approaches learn a decision boundary in the attribute space that encloses all normal objects on one side of the boundary. [Figure 9.13](#) shows an example of a decision boundary in the one-class setting, where points belonging to one side of the boundary (shaded) belong to the normal class. This is in contrast to binary classification approaches introduced in [chapters 3](#) and [4](#) that learn boundaries to separate objects from two classes.

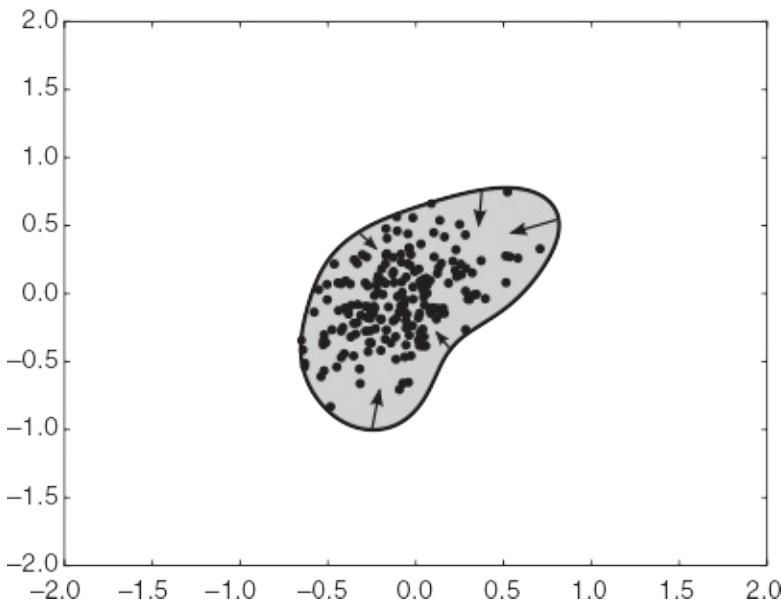


Figure 9.13.

The decision boundary of a one-class classification problem attempts to enclose the normal instances on the same side of the boundary.

One-class classification presents a unique perspective on anomaly detection, where, instead of learning the distribution of the normal class, the focus is on modeling the boundary of the normal class. From an operational standpoint, learning the boundary is indeed what we need to distinguish anomalies from

normal objects. In the words of Vladimir Vapnik, “One should solve the [classification] problem directly and never solve a more general problem [such as learning the distribution of the normal class] as an intermediate step.”

In this section, we present an SVM-based a one-class approach, known as **one-class SVM**, which only uses training instances from the normal class to learn its decision boundary. Contrast this with a normal SVM—see [Section 4.9](#)—which uses training instances from two classes. This involves the use of kernels and a novel “origin trick,” described as follows. (See [Section 2.4.7](#) for an introduction to kernel methods.)

9.7.1 Use of Kernels

In order to learn a nonlinear boundary that encloses the normal class, we transform the data to a higher dimensional space where the normal class can be separated using a linear hyperplane. This can be done by using a function ϕ that maps every data instance \mathbf{x} in the original space of attributes to a point $\phi(\mathbf{x})$ in the transformed high-dimensional space. (The choice of the mapping function will become clear later.) In the transformed space, the training instances can be separated using a linear hyperplane defined by parameters (\mathbf{w}, ρ) as follows.

$$\langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \rho,$$

where $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the inner product between vectors \mathbf{x} and \mathbf{y} . Ideally, we want a linear hyperplane that places all of the normal instances on one side. Hence, we want (\mathbf{w}, ρ) to be such that $\langle \mathbf{w}, \phi(\mathbf{x}) \rangle > \rho$ if \mathbf{x} belongs to the normal class, and $\langle \mathbf{w}, \phi(\mathbf{x}) \rangle < \rho$ if \mathbf{x} belongs to the anomaly class.

Let $\{x_1, x_2, \dots, x_n\}$ be the set of training instances belonging to the normal class. Similar to the use of kernels in SVMs (see [Chapter 4](#)), we define w as a linear combination of $\phi(x_i)$'s:

$$w = \sum_{i=1}^n \alpha_i \phi(x_i).$$

The separating hyperplane can then be described using α_i 's and ρ as follows.

$$\sum_{i=1}^n \alpha_i \langle \phi(x_i), \phi(x) \rangle = \rho,$$

Note that the above equation deals with inner products of $\phi(x)$ in the transformed space to describe the hyperplane. To compute such inner products, we can make use of kernel functions, $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle$, introduced in [Section 2.4.7](#). Note that kernel functions are extensively used for learning nonlinear boundaries in binary classification problems, e.g., using kernel-SVMs presented in [Chapter 4](#). However, learning nonlinear boundaries in the one-class setting is challenging in the absence of any information about the anomaly class during training. To overcome this challenge, one-class SVM uses the “origin trick” to learn the separating hyperplane, which works best with certain types of kernel functions. This approach can be briefly described as follows.

9.7.2 The Origin Trick

Consider the Gaussian kernel that is commonly used for learning nonlinear boundaries, which can be defined as

$$\kappa(x, y) = \exp(-\|x-y\|^2 / 2\sigma^2),$$

where $\|\cdot\|$ denotes the length of a vector and σ is a hyper-parameter. Before we use the Gaussian kernel to learn a separating hyperplane in the one-class setting, it will be worthwhile to first understand what the transformed space $\phi(x)$ of a Gaussian kernel looks like. There are two important properties of the transformed space of Gaussian kernels that are useful for understanding the intuition behind one-class SVMs:

1. *Every point is mapped to a hypersphere of unit radius.*

To realize this, consider the kernel function $\kappa(x, x)$ of a point x onto itself. Since $\|x-x\|_2 = 0$,

$$\kappa(x, x) = \langle \phi(x), \phi(x) \rangle = \|\phi(x)\|_2^2 = 1.$$

This implies that the length of $\phi(x)$ is equal to 1, and hence, $\phi(x)$ resides on a hypersphere of unit radius for all x .

2. *Every point is mapped to the same orthant in the transformed space.*

For any two points x and y , since $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle \geq 0$, the angle between $\phi(x)$ and $\phi(y)$ is always smaller than $\pi/2$. Hence, the mappings of all points lie in the same orthant (high-dimensional analogue of “quadrant”) in the transformed space.

For illustrative purposes, [Figure 9.14](#) shows a schematic visualization of the transformed space of Gaussian kernels, using the above two considerations. The black dots represent the mappings of training instances in the transformed space, which lie on a quarter arc of a circle with unit radius. In this view, the objective of one-class SVM is to learn a linear hyperplane that can separate the black dots from the mappings of anomalous instances, which would also reside on the same quarter arc. There are many possible hyperplanes that can achieve this task, two of which are shown in [Figure 9.14](#) as dashed lines. In order to choose the best hyperplane (shown as a bold line), we make use of the principle of structural risk minimization, discussed

in [Chapter 4](#) in the context of SVM. There are three main requirements that we seek in the optimal hyperplane defined by parameters (w, ρ) :

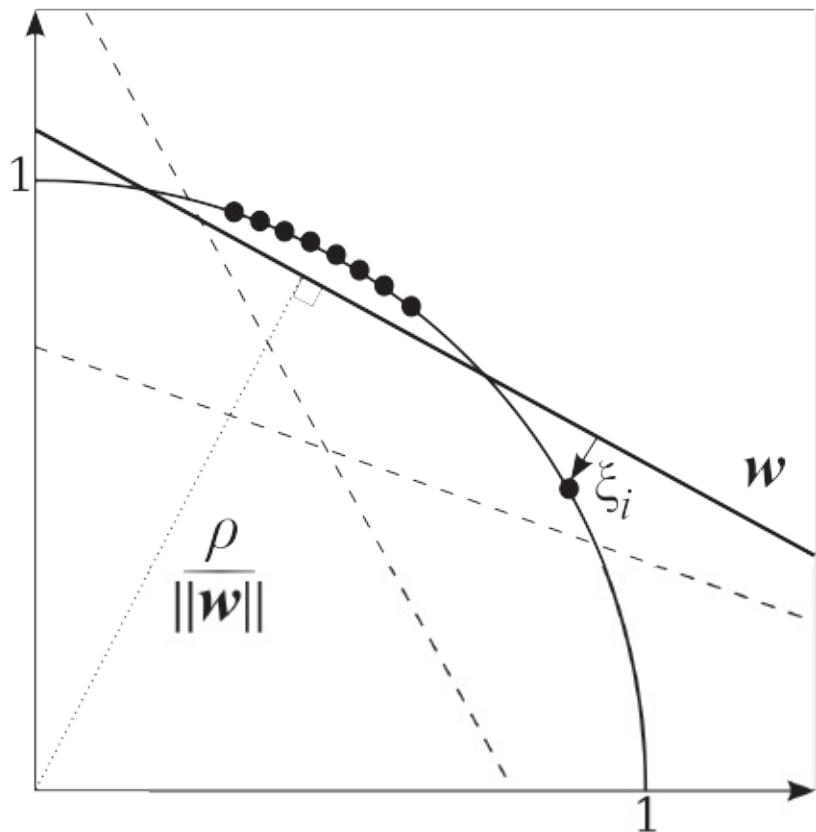


Figure 9.14.

Illustrating the concept of one-class SVM in the transformed space.

1. The hyperplane should have a large “margin” or a small value of $\|w\| / 2$. Having a large margin ensures that the model is simple and hence less susceptible to the phenomenon of overfitting.
2. The hyperplane should be as distant from the origin as possible. This ensures a tight representation of points on the upper side of the hyperplane (corresponding to the normal class). Notice from [Figure 9.14](#) that the distance of a hyperplane from the origin is essentially $\rho / \|w\|$. Hence, maximizing ρ translates to maximizing the distance of the hyperplane from the origin.

3. In the style of “soft-margin” SVMs, if some of the training instances lie on the opposite side of the hyperplane (corresponding to the anomaly class), then the distance of such points from the hyperplane should be minimized.

Note that it is important for an anomaly detection algorithm to be robust to a small number of outliers in the training set as that is quite common in real-world problems. An example of an anomalous training instance is shown in [Figure 9.14](#) as the lower-most black dot on the quarter arc. If a training instance x_i lies on the opposite side of the hyperplane (corresponding to the anomaly class), its distance from the hyperplane, as measured by its slack variable ξ_i , should be kept small. If x_i lies on the side corresponding to the normal class, then $\xi_i = 0$.

The above three requirements provide the foundation of the optimization objective of one-class SVM, which can be formally described as follows:

$$\min_{w, \rho, \xi} \frac{1}{2} \|w\|^2 - \rho + C \sum_{i=1}^n \xi_i, \quad (9.9)$$

$$\text{subject to } \langle w, \phi(x_i) \rangle \geq \rho - \xi_i, \quad \xi_i \geq 0,$$

where n is the number of training instances and $C \in (0, 1]$ is a hyper-parameter that maintains a trade-off between reducing the model complexity and improving the coverage of the decision boundary in keeping the training instances on the same side.

Notice the similarity of the above equation to the optimization objective of binary SVM, introduced in [Chapter 4](#). However, a key difference in one-class SVM is that the constraints are only defined for the normal class but not the anomaly class. At a first glance, this might seem to be a serious problem, because the hyperplane is held by constraints from one side (corresponding to the normal class) but is unconstrained from the other side. However, with

the help of the “origin trick,” one-class SVM is able to overcome this insufficiency by maximizing the distance of the hyperplane from the origin. From this perspective, the origin acts as a surrogate *second* class and the learned hyperplane attempts to separate the normal class from this second class in a manner similar to the way a binary SVM separates two classes.

Equation 9.7.2 is an instance of a quadratic programming problem (QPP) with linear inequality constraints, which is similar to the optimization problem of binary SVM. Hence, the optimization procedures discussed in [Chapter 4](#) for learning a binary SVM can be directly applied for solving Equation 9.7.2. The learned one-class SVM can then be applied on a test instance to identify if it belongs to the normal class or the anomaly class. Further, if a test instance is identified as an anomaly, its distance from the hyperplane can be seen as an estimate of its anomaly score.

The hyper-parameter v of one-class SVM has a special interpretation. It represents an upper bound on the fraction of training instances that can be tolerated as anomalies while learning the hyperplane. This means that nv represents the maximum number of training instances that can be placed on the other side of the hyperplane (corresponding to the anomaly class). A low value of v assumes that the training set has a smaller number of outliers, while a high value of v ensures that the learning of the hyperplane is robust to a large number of outliers during training.

[Figure 9.15](#) shows the learned decision boundary for an example training set of size 200 using $v=0.1$. We can see that the training data consists of mostly normal instances generated from a Gaussian distribution centered at $(0, 0)$. However, there are also some outliers in the input data that do not conform the distribution of the normal class. With $v=0.1$, the one-class SVM is able to place at most 20 training instances on the other side of the hyperplane (corresponding to the normal class). This results in a decision boundary that

robustly encloses the majority of normal instances. If we instead use $\nu=0.05$, we would only have the budget to tolerate at most 10 outliers in the training set, resulting in the decision boundary shown in [Figure 9.16\(a\)](#). We can see that this decision boundary assigns a much larger region to the normal class than is necessary. On the other hand, the decision boundary learned using $\nu=0.2$ is shown in [Figure 9.16\(b\)](#), which appears to be much more compact as it can tolerate up to 40 outliers in the training data. The choice of ν thus plays a crucial role in the learning of the decision boundary in one-class SVMs.

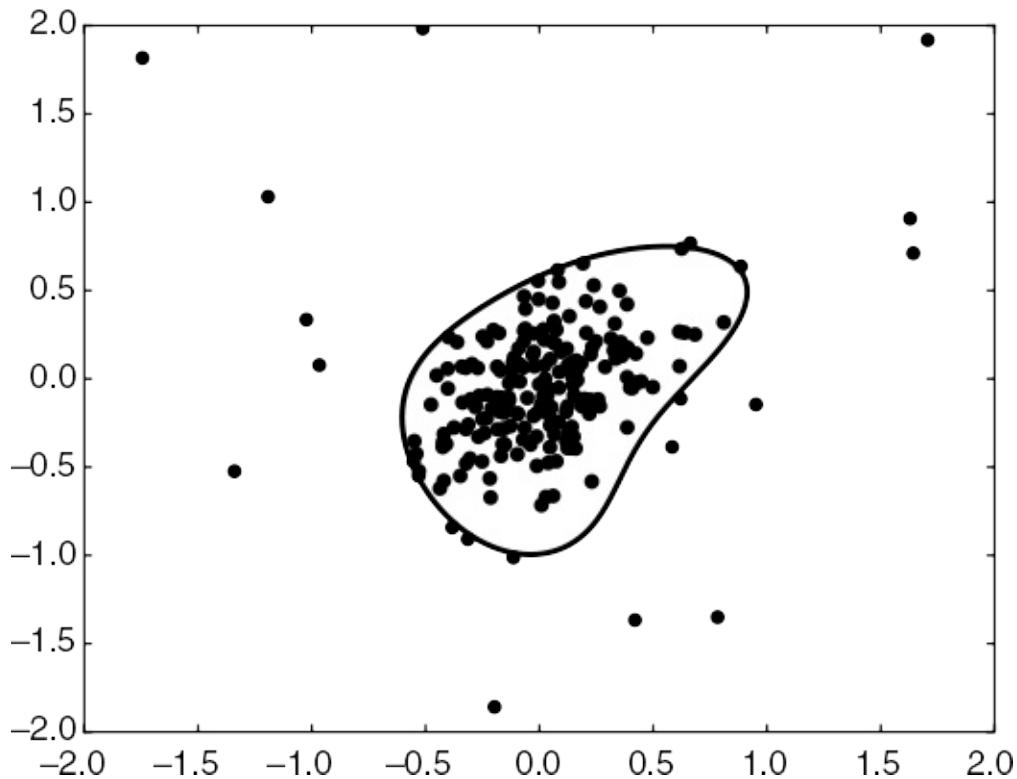


Figure 9.15.

Decision boundary of one-class SVM with $\nu = 0.1$.

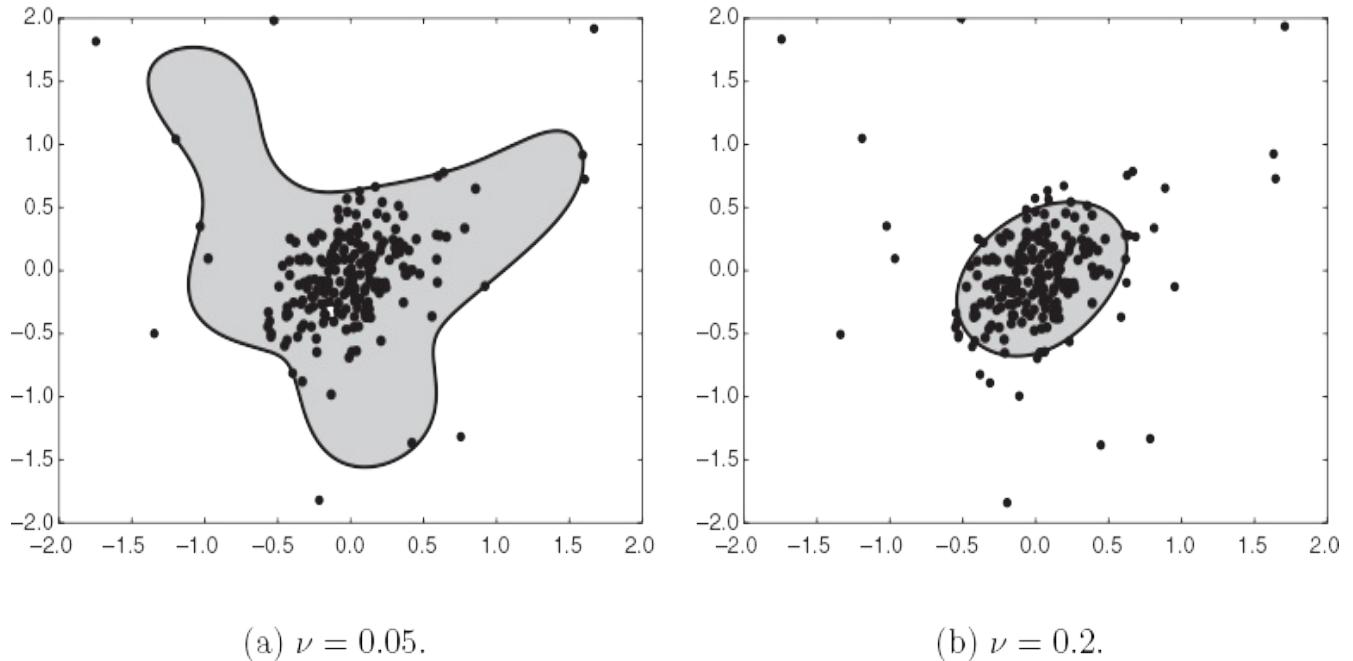


Figure 9.16.

Decision boundaries of one-class SVM for varying values of ν .

9.7.3 Strengths and Weaknesses

One-class SVMs leverage the principle of structural risk minimization in the learning of the decision boundary, which has strong theoretical foundations. They have the ability to strike a balance between the simplicity of the model and the effectiveness of the boundary in enclosing the distribution of the normal class. By using the hyper-parameter ν , they provide a built-in mechanism to avoid outliers in the training data, which is often common in real-world problems. However, as illustrated in [Figure 9.16](#), the choice of ν significantly impacts the properties of the learned decision boundary. Choosing the right value of ν is difficult, since the hyper-parameter selection techniques discussed in [Chapter 4](#) are only applicable in the multiclass setting, where it is possible to define validation error rates. Also, the use of a

Gaussian kernel requires a relatively large training size to effectively learn nonlinear decision boundaries in the attribute space. Further, like regular SVM, one-class SVM has a high computational cost. Hence, it is expensive to train, especially when the training set is large.

9.8 Information Theoretic Approaches

These approaches assume that the normal class can be represented using compact representations, also known as codes. Instead of explicitly learning such representations, the focus of information theoretic approaches is to quantify the amount of information required for encoding them. If the normal class shows some structure or pattern, we can expect to encode it using a small number of bits. Anomalies can then be identified as instances that introduce *irregularities* in the data, which increase the overall information content of the data set. This is an admissible definition of an anomaly in an operational setting, since anomalies are often associated with an element of *surprise*, as they do not conform to the patterns or behavior of the normal class.

There are a number of approaches for quantifying the information content (also referred to as complexity) of a data set. For example, if the data set contains a categorical variable, we can assess its information content using the entropy measure, described in [Section 2.3.6](#). For data sets with other types of attributes, other measures such as the Kolmogorov complexity can be used. Intuitively, the Kolmogorov complexity measures the complexity of a data set by the size of the smallest computer program (written in a pre-specified language) that can reproduce the original data. A more practical approach is to compress the data using standard compression techniques, and use the size of the resulting compressed file as a measure of the information content of the original data.

A basic information theoretic approach for anomaly detection can be described as follows. Let us denote the information content of a data set D as $\text{Info}(D)$. Consider computing the anomaly score of a data instance x in D . If we

remove x from D , we can measure the information content of the remaining data as $\text{Info}(D \setminus x)$. If x is indeed an anomaly, it would show a high value of

$$\text{Gain}(x) = \text{Info}(D) - \text{Info}(D \setminus x).$$

This happens because anomalies are expected to be surprising, and thus, their elimination should result in a substantial reduction in the information content. We can thus use $\text{Gain}(x)$ as a measure of anomaly score.

Typically, the reduction in information content is measured by eliminating a *subset* of instances (that are deemed anomalous) and not just a single instance. This is because most measures of information content are not sensitive to the elimination of a single instance, e.g., the size of a compressed data file does not change substantially by removing a single data entry. It is thus necessary to identify the smallest subset of instances X that show the largest value of $\text{Gain}(X)$ upon elimination. This is a non-trivial problem requiring exponential time complexity, although approximate solutions with linear time complexity have also been proposed. (See Bibliographic Notes.)

Example 9.4.

Given a survey report of the `weight` and `height` of a collection of participants, we want to identify those participants that have unusual heights and weights. Both `weight` and `height` can be represented as categorical variables that take three values: {low, medium, high}. **Table 9.2** shows the data for the weight and height information of 100 participants, which has an entropy of 2.08. We can see that there is a pattern in the height and weight distribution of normal participants, since most participants that have a high value of `weight` also have a high value of `height`, and vice-versa. However, there are 5 participants that have a high `weight` value but low `height` value, which is quite unusual. By

eliminating these 5 instances, the entropy of the resulting data set becomes 1.89 , resulting in a gain of $2.08 - 1.89 = 0.19$.

Table 9.2. Survey data of *weight* and *height* of 100 participants.

weight	height	Frequency
low	low	20
low	medium	15
medium	medium	40
high	high	20
high	low	5

9.8.1 Strengths and Weaknesses

Information theoretic approaches operate in the unsupervised setting, as they do not require a separate training set of normal-only instances. They do not make many assumptions about the structure of the normal class and are generic enough to be applied with data sets of varying types and properties. However, the performance of information theoretic approaches depends heavily on the choice of the measure used for capturing the information content of a data set. The measure should be suitably chosen so that it is sensitive to the elimination of a small number of instances. This is often a challenge, since compression techniques are often robust to small deviations, rendering them useful only when anomalies are large in numbers. Further, information theoretic approaches suffer from high computational cost, making them expensive to apply on large data sets.

9.9 Evaluation of Anomaly Detection

When class labels are available to distinguish between anomalies and normal data, then the effectiveness of an anomaly detection scheme can be evaluated by using measures of classification performance discussed in [Section 4.11](#). Since the anomalous class is usually much smaller than the normal class, measures such as precision, recall, and false positive rate are more appropriate than accuracy. In particular, the false positive rate, which is often referred to as the **false alarm rate**, often determines the practicality of the anomaly detection scheme since too many false alarms render an anomaly detection system useless.

If class labels are not available, then evaluation is challenging. For model-based approaches, the effectiveness of outlier detection can be judged with respect to the improvement in the goodness of fit of the model once anomalies are eliminated. Similarly for information theoretic approaches, the information gain gives a measure of the effectiveness. For reconstruction-based approaches, the reconstruction error provides a measure that can be used for evaluation.

The evaluation presented in the last paragraph is analogous to the unsupervised evaluation measures for cluster analysis, where measures—see [Section 7.5](#), such as the sum of the squared error (SSE) or the silhouette index, can be computed even when class labels are not present. Such measures were referred to as “internal” measures because they use only information present in the data set. The same is true of the anomaly evaluation measures mentioned in the last paragraph, i.e., they are internal measures. The key point is that the anomalies of interest for a particular application may not be those that an anomaly detection algorithm labels as

anomalies, just as the cluster labels produced by a clustering algorithm may not be consistent with the class labels provided externally. In practice, this means that selecting and tuning an anomaly detection approach based on feedback from the users of such a system.

A more general way to evaluate the results of anomaly detection is to look at the distribution of the anomaly scores. The techniques that we have discussed assume that only a relatively small fraction of the data consists of anomalies. Thus, the majority of anomaly scores should be relatively low, with a smaller fraction of scores toward the high end. (This assumes that a higher score indicates an instance is more anomalous.) Thus, by looking at the distribution of the scores via a histogram or density plot, we can assess whether the approach we are using generates scores that behave in a reasonable manner. We illustrate with an example.

Example 9.5. (Distribution of Anomaly Scores.).

[Figures 9.17](#) and [9.18](#) show the anomaly scores of two clusters of points. Both have 100 points, but the leftmost cluster is less dense. [Figure 9.17](#), which uses the average distance to the kth neighbor (average KNN dist), shows higher anomaly scores for the points in the less dense cluster. In contrast, [Figure 9.18](#), which uses the LOF for its anomaly scoring, shows similar scores between the two clusters.

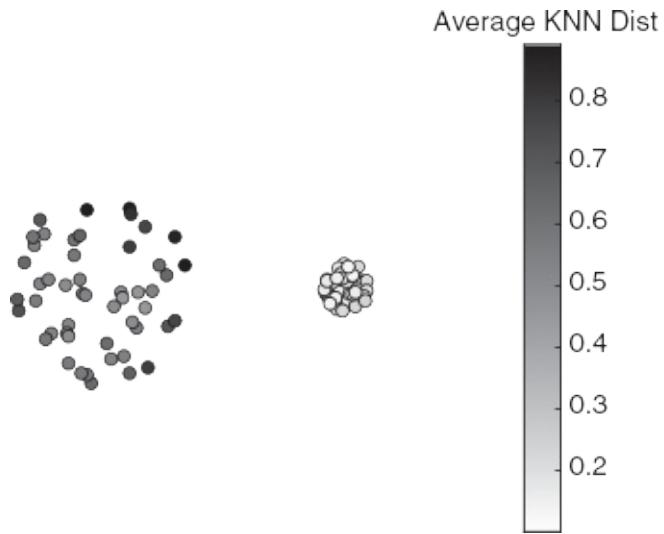


Figure 9.17.

Anomaly score based on average distance to fifth nearest neighbor.

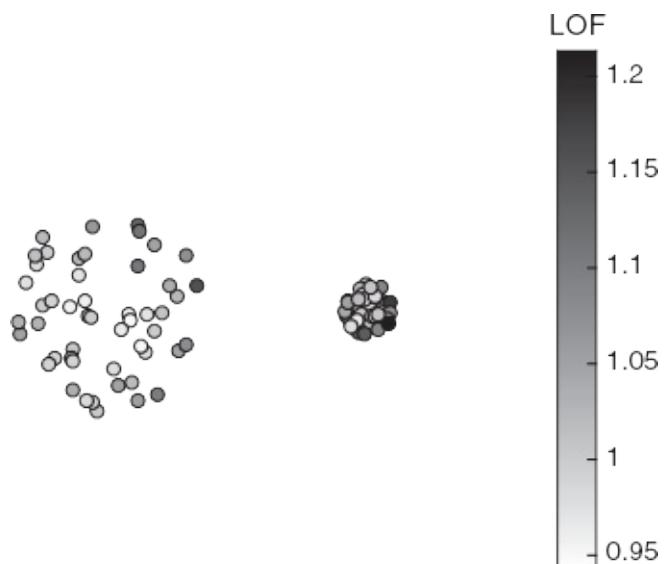


Figure 9.18.

Anomaly score based on LOF using five nearest neighbors.

The histograms of the average KNN dist and the LOF score are shown in [figures 9.19](#) and [9.20](#), respectively. The histogram of the LOF scores shows most points with similar anomaly scores and a few points with significantly larger values. The histogram of the average KNN dist shows a bimodal distribution.

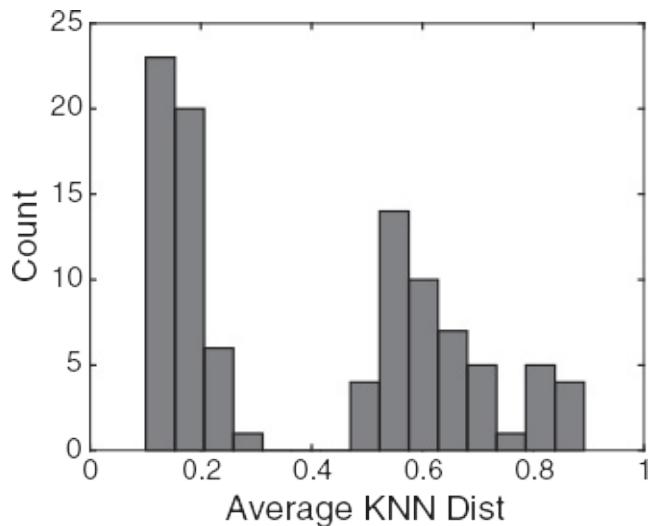


Figure 9.19.

Histogram of anomaly score based on average distance to the fifth nearest neighbor.

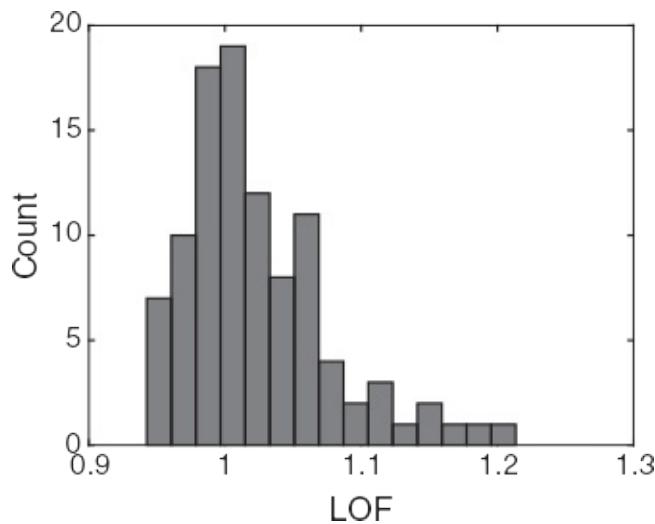


Figure 9.20.

Histogram of LOF anomaly score using five nearest neighbors.

The key point is that the distribution of anomaly scores should look similar to that of the LOF scores in this example. There may be one or more secondary peaks in the distribution as one moves to the right, but these secondary peaks should only contain a relatively small fraction of the

points, and not a large fraction of the points as with the average KNN dist approach.

9.10 Bibliographic Notes

Anomaly detection has a long history, particularly in statistics, where it is known as outlier detection. Relevant books on the topic are those of [Aggarwal \[623\]](#), [Barnett and Lewis \[627\]](#), [Hawkins \[648\]](#), and [Rousseeuw and Leroy \[683\]](#). The article by [Beckman and Cook \[629\]](#) provides a general overview of how statisticians look at the subject of outlier detection and provides a history of the subject dating back to comments by Bernoulli in 1777. Also see the related articles [\[630, 649\]](#). Another general article on outlier detection is the one by [Barnett \[626\]](#). Articles on finding outliers in multivariate data include those by [Davies and Gather \[639\]](#), [Gnanadesikan and Kettenring \[646\]](#), [Rocke and Woodruff \[681\]](#), [Rousseeuw and van Zomerenand \[685\]](#), and [Scott \[690\]](#). [Rosner \[682\]](#) provides a discussion of finding multiple outliers at the same time.

[Surveys by Chandola et al. \[633\]](#) and [Hodge and Austin \[651\]](#) provide extensive coverage of outlier detection methods, as does a recent book on the topic by [Aggarwal \[623\]](#). [Markou and Singh \[674, 675\]](#) give a two-part review of techniques for novelty detection that covers statistical and neural network techniques, respectively. [Pimento et al. \[678\]](#) is another review of novelty detection approaches, including many of the methods discussed in this chapter.

Statistical approaches for anomaly detection in the univariate case are well covered by the books in the first paragraph. [Shyu et al. \[692\]](#) use an approach based on principal components and the Mahalanobis distance to produce anomaly scores for multivariate data. An example of the kernel density approach for anomaly detection is given by [Schubert et al. \[688\]](#). The mixture model outlier approach discussed in [Section 9.3.3](#) is from [Eskin](#)

[641]. An approach based on the χ^2 measure is given by **Ye and Chen [695]**. Outlier detection based on geometric ideas, such as the depth of convex hulls, has been explored in papers by **Johnson et al. [654]**, **Liu et al. [673]**, and **Rousseeuw et al. [684]**.

The notion of a distance-based outlier and the fact that this definition can include many statistical definitions of an outlier was described by Knorr et al. [663–665]. **Ramaswamy et al. [680]** propose an efficient distance-based outlier detection procedure that gives each object an outlier score based on the distance of its k -nearest neighbor. Efficiency is achieved by partitioning the data using the first phase of BIRCH (Section 8.5.2). **Chaudhary et al. [634]** use k - d trees to improve the efficiency of outlier detection, while **Bay and Schwabacher [628]** use randomization and pruning to improve performance.

For relative density-based approaches, the best known technique is the local outlier factor (LOF) (Breunig et al. [631, 632]), which grew out of DBSCAN. Another locally aware anomaly detection algorithm is LOCI by **Papadimitriou et al. [677]**. A more recent view of the local approach is given by **Schubert et al. [689]**. Proximities can be viewed as a graph. The connectivity-based outlier factor (COF) by **Tang et al. [694]** is a graph-based approach to local outlier detection. A survey of graph based approaches is provided by **Akoglu et al. [625]**.

High dimensionality poses significant problems for distance- and density-based approaches. A discussion of outlier removal in high-dimensional space can be found in the papers by **Aggarwal and Yu [624]** and **Dunagan and Vempala [640]**. Zimek et al. provide a survey of anomaly detection approaches for high-dimensional numerical data [696].

Clustering and anomaly detection have a long relationship. In [Chapters 7](#) and [8](#), we considered techniques, such as BIRCH, CURE, DENCLUE, DBSCAN, and SNN density-based clustering, which specifically include techniques for handling anomalies. Statistical approaches that further discuss this relationship are described in papers by [Scott \[690\]](#) and [Hardin and Rocke \[647\]](#). The K-means-- algorithm, which can simultaneously handle clustering and outliers, was proposed by [Chawla and Gionis \[637\]](#).

Our discussion of reconstruction-based approaches focused on a neural network-based approach, i.e., the autoencoder. More broadly, a discussion of approaches in the area of neural networks can be found in papers by [Ghosh and Schwartzbard \[645\]](#), [Sykacek \[693\]](#), and [Hawkins et al. \[650\]](#), who discuss replicator networks. The one class SVM approach for anomaly detection was created by [Schölkopf et al. \[686\]](#) and improved by [Li et al. \[672\]](#). More generally, techniques for one class classification are surveyed in [\[662\]](#). The use of information measures in anomaly detection is described by [Lee and Xiang \[671\]](#).

In this chapter, we focused on unsupervised anomaly detection. Supervised anomaly detection falls into the category of rare class classification. Work on rare class detection includes the work of Joshi et al. [\[655–659\]](#). The rare class problem is also sometimes referred to as the imbalanced data set problem. Of relevance are an AAAI workshop ([Japkowicz \[653\]](#)), an ICML workshop ([Chawla et al. \[635\]](#)), and a special issue of SIGKDD Explorations ([Chawla et al. \[636\]](#)).

Evaluation of unsupervised anomaly detection approaches was discussed in [Section 9.9](#). See also the discussion in [Chapter 8](#) of the book by [Aggarwal \[623\]](#). In summary, evaluation approaches are quite limited. For supervised anomaly detection, an overview of current approaches for evaluation can be found in [Schubert et al. \[687\]](#).

In this chapter, we have focused on basic anomaly detection schemes. We have not considered schemes that take into account the spatial or temporal nature of the data. [Shekhar et al. \[691\]](#) provide a detailed discussion of the problem of spatial outliers and present a unified approach to spatial outlier detection. A discussion of the challenges for anomaly detection in climate data is provided by [Kawale et al. \[660\]](#).

The issue of outliers in time series was first considered in a statistically rigorous way by [Fox \[643\]](#). [Muirhead \[676\]](#) provides a discussion of different types of outliers in time series. [Abraham and Chuang \[622\]](#) propose a Bayesian approach to outliers in time series, while [Chen and Liu \[638\]](#) consider different types of outliers in time series and propose a technique to detect them and obtain good estimates of time series parameters. Work on finding deviant or surprising patterns in time series databases has been performed by [Jagadish et al. \[652\]](#) and [Keogh et al. \[661\]](#).

An important application area for anomaly detection is intrusion detection. Surveys of the applications of data mining to intrusion detection are given by [Lee and Stolfo \[669\]](#) and [Lazarevic et al. \[668\]](#). In a different paper, [Lazarevic et al. \[667\]](#) provide a comparison of anomaly detection routines specific to network intrusion. [Garcia et al. \[644\]](#) provide a recent survey of anomaly detection for network intrusion detection. A framework for using data mining techniques for intrusion detection is provided by [Lee et al. \[670\]](#). Clustering-based approaches in the area of intrusion detection include work by [Eskin et al. \[642\]](#), [Lane and Brodley \[666\]](#), and [Portnoy et al. \[679\]](#).

Bibliography

- [622] B. Abraham and A. Chuang. Outlier Detection and Time Series Modeling. *Technometrics*, 31(2):241–248, May 1989.
- [623] C. C. Aggarwal. *Outlier Analysis*. Springer Science & Business Media, 2013.
- [624] C. C. Aggarwal and P. S. Yu. Outlier Detection for High Dimensional Data. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 37–46, New York, NY, USA, 2001. ACM.
- [625] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015.
- [626] V. Barnett. The Study of Outliers: Purpose and Model. *Applied Statistics*, 27(3): 242–250, 1978.
- [627] V. Barnett and T. Lewis. *Outliers in Statistical Data*. Wiley Series in Probability and Statistics. John Wiley & Sons, 3rd edition, April 1994.
- [628] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. of the*

9th Intl. Conf. on Knowledge Discovery and Data Mining, pages 29–38.
ACM Press, 2003.

[629] R. J. Beckman and R. D. Cook. ‘Outlier.....s’. *Technometrics*, 25(2):119–149, May 1983.

[630] R. J. Beckman and R. D. Cook. [‘Outlier.....s’]: Response. *Technometrics*, 25(2): 161–163, May 1983.

[631] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. OPTICS-OF: Identifying Local Outliers. In *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, pages 262–270. Springer-Verlag, 1999.

[632] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *Proc. of 2000 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 93–104. ACM Press, 2000.

[633] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[634] A. Chaudhary, A. S. Szalay, and A. W. Moore. Very fast outlier detection in large multidimensional data sets. In *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, 2002.

[635] N. V. Chawla, N. Japkowicz, and A. Kolcz, editors. *Workshop on Learning from Imbalanced Data Sets II*, 20th Intl. Conf. on Machine

Learning, 2000. AAAI Press.

- [636] N. V. Chawla, N. Japkowicz, and A. Kolcz, editors. *SIGKDD Explorations Newsletter, Special issue on learning from imbalanced datasets*, volume 6(1), June 2004. ACM Press.
- [637] S. Chawla and A. Gionis. k-means-: A Unified Approach to Clustering and Outlier Detection. In *SDM*, pages 189–197. SIAM, 2013.
- [638] C. Chen and L.-M. Liu. Joint Estimation of Model Parameters and Outlier Effects in Time Series. *Journal of the American Statistical Association*, 88(421):284–297, March 1993.
- [639] L. Davies and U. Gather. The Identification of Multiple Outliers. *Journal of the American Statistical Association*, 88(423):782–792, September 1993.
- [640] J. Dunagan and S. Vempala. Optimal outlier removal in high-dimensional spaces. *Journal of Computer and System Sciences, Special Issue on STOC 2001*, 68(2):335– 373, March 2004.
- [641] E. Eskin. Anomaly Detection over Noisy Data using Learned Probability Distributions. In *Proc. of the 17th Intl. Conf. on Machine Learning*, pages 255–262, 2000.
- [642] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. J. Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of Data*

Mining in Computer Security, pages 78–100. Kluwer Academic, 2002.

[643] A. J. Fox. Outliers in Time Series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(3):350–363, 1972.

[644] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[645] A. Ghosh and A. Schwartzbard. A Study in Using Neural Networks for Anomaly and Misuse Detection. In *8th USENIX Security Symposium*, August 1999.

[646] R. Gnanadesikan and J. R. Kettenring. Robust Estimates, Residuals, and Outlier Detection with Multiresponse Data. *Biometrics*, 28(1):81–124, March 1972.

[647] J. Hardin and D. M. Rocke. Outlier Detection in the Multiple Cluster Setting using the Minimum Covariance Determinant Estimator. *Computational Statistics and Data Analysis*, 44:625–638, 2004.

[648] D. M. Hawkins. *Identification of Outliers*. Monographs on Applied Probability and Statistics. Chapman & Hall, May 1980.

[649] D. M. Hawkins. ‘[Outlier.....s]’: Discussion. *Technometrics*, 25(2):155–156, May 1983.

- [650] S. Hawkins, H. He, G. J. Williams, and R. A. Baxter. Outlier Detection Using Replicator Neural Networks. In *DaWaK 2000: Proc. of the 4th Intnl. Conf. on Data Warehousing and Knowledge Discovery*, pages 170–180. Springer-Verlag, 2002.
- [651] V. J. Hodge and J. Austin. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.
- [652] H. V. Jagadish, N. Koudas, and S. Muthukrishnan. Mining Deviants in a Time Series Database. In *Proc. of the 25th VLDB Conf.*, pages 102–113, 1999.
- [653] N. Japkowicz, editor. *Workshop on Learning from Imbalanced Data Sets I, Seventeenth National Conference on Artificial Intelligence, Published as Technical Report WS-00-05*, 2000. AAAI Press.
- [654] T. Johnson, I. Kwok, and R. T. Ng. Fast Computation of 2-Dimensional Depth Contours. In *KDD98*, pages 224–228, 1998.
- [655] M. V. Joshi. On Evaluating Performance of Classifiers for Rare Classes. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 641–644, 2002.
- [656] M. V. Joshi, R. C. Agarwal, and V. Kumar. Mining needle in a haystack: Classifying rare classes via two-phase rule induction. In *Proc. of 2001 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 91–102. ACM Press, 2001.

- [657] M. V. Joshi, R. C. Agarwal, and V. Kumar. Predicting rare classes: can boosting make any weak learner strong? In *Proc. of 2002 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 297–306. ACM Press, 2002.
- [658] M. V. Joshi, R. C. Agarwal, and V. Kumar. Predicting Rare Classes: Comparing Two-Phase Rule Induction to Cost-Sensitive Boosting. In *Proc. of the 6th European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 237–249. Springer-Verlag, 2002.
- [659] M. V. Joshi, V. Kumar, and R. C. Agarwal. Evaluating Boosting Algorithms to Classify Rare Classes: Comparison and Improvements. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 257–264, 2001.
- [660] J. Kawale, S. Chatterjee, A. Kumar, S. Liess, M. Steinbach, and V. Kumar. Anomaly construction in climate data: issues and challenges. In *NASA Conference on Intelligent Data Understanding CIDU*, 2011.
- [661] E. Keogh, S. Lonardi, and B. Chiu. Finding Surprising Patterns in a Time Series Database in Linear Time and Space. In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 2002.
- [662] S. S. Khan and M. G. Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(03):345–374, 2014.
- [663] E. M. Knorr and R. T. Ng. A Unified Notion of Outliers: Properties and Computation. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and*

Data Mining, pages 219–222, 1997.

- [664] E. M. Knorr and R. T. Ng. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *Proc. of the 24th VLDB Conf.*, pages 392–403, August 1998.
- [665] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3-4):237–253, 2000.
- [666] T. Lane and C. E. Brodley. An Application of Machine Learning to Anomaly Detection. In *Proc. 20th NIST-NCSC National Information Systems Security Conf.*, pages 366–380, 1997.
- [667] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In *Proc. of the 2003 SIAM Intl. Conf. on Data Mining*, 2003.
- [668] A. Lazarevic, V. Kumar, and J. Srivastava. Intrusion Detection: A Survey. In *Managing Cyber Threats: Issues, Approaches and Challenges*, pages 19–80. Kluwer Academic Publisher, 2005.
- [669] W. Lee and S. J. Stolfo. Data Mining Approaches for Intrusion Detection. In *7th USENIX Security Symposium*, pages 26–29, January 1998.
- [670] W. Lee, S. J. Stolfo, and K. W. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

- [671] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, pages 130–143, May 2001.
- [672] K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu. Improving one-class SVM for anomaly detection. In *Machine Learning and Cybernetics, 2003 International Conference on*, volume 5, pages 3077–3081. IEEE, 2003.
- [673] R. Y. Liu, J. M. Parelius, and K. Singh. Multivariate analysis by data depth: descriptive statistics, graphics and inference. *Annals of Statistics*, 27(3):783–858, 1999.
- [674] M. Markou and S. Singh. Novelty detection: A review—part 1: Statistical approaches. *Signal Processing*, 83(12):2481–2497, 2003.
- [675] M. Markou and S. Singh. Novelty detection: A review—part 2: Neural network based approaches. *Signal Processing*, 83(12):2499–2521, 2003.
- [676] C. R. Muirhead. Distinguishing Outlier Types in Time Series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(1):39–47, 1986.
- [677] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 315–326. IEEE, 2003.

- [678] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.
- [679] L. Portnoy, E. Eskin, and S. J. Stolfo. Intrusion detection with unlabeled data using clustering. In *In ACM Workshop on Data Mining Applied to Security*, 2001.
- [680] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proc. of 2000 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 427–438. ACM Press, 2000.
- [681] D. M. Rocke and D. L. Woodruff. Identification of Outliers in Multivariate Data. *Journal of the American Statistical Association*, 91(435):1047–1061, September 1996.
- [682] B. Rosner. On the Detection of Many Outliers. *Technometrics*, 17(3):221–227, 1975.
- [683] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. Wiley Series in Probability and Statistics. John Wiley & Sons, September 2003.
- [684] P. J. Rousseeuw, I. Ruts, and J. W. Tukey. The Bagplot: A Bivariate Boxplot. *The American Statistician*, 53(4):382–387, November 1999.
- [685] P. J. Rousseeuw and B. C. van Zomeren. Unmasking Multivariate Outliers and Leverage Points. *Journal of the American Statistical*

Association, 85(411):633–639, September 1990.

- [686] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt, et al. Support Vector Method for Novelty Detection. In *NIPS*, volume 12, pages 582–588, 1999.
- [687] E. Schubert, R. Wojdanowski, A. Zimek, and H.-P. Kriegel. On evaluation of outlier rankings and outlier scores. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012.
- [688] E. Schubert, A. Zimek, and H.-P. Kriegel. Generalized Outlier Detection with Flexible Kernel Density Estimates. In *SDM*, volume 14, pages 542–550. SIAM, 2014.
- [689] E. Schubert, A. Zimek, and H.-P. Kriegel. Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. *Data Mining and Knowledge Discovery*, 28(1):190–237, 2014.
- [690] D. W. Scott. Partial Mixture Estimation and Outlier Detection in Data and Regression. In M. Hubert, G. Pison, A. Struyf, and S. V. Aelst, editors, *Theory and Applications of Recent Robust Methods*, Statistics for Industry and Technology. Birkhauser, 2003.
- [691] S. Shekhar, C.-T. Lu, and P. Zhang. A Unified Approach to Detecting Spatial Outliers. *GeoInformatica*, 7(2):139–166, June 2003.

- [692] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang. A Novel Anomaly Detection Scheme Based on Principal Component Classifier. In *Proc. of the 2003 IEEE Intl. Conf. on Data Mining*, pages 353–365, 2003.
- [693] P. Sykacek. Equivalent error bars for neural network classifiers trained by bayesian inference. In *Proc. of the European Symposium on Artificial Neural Networks*, pages 121–126, 1997.
- [694] J. Tang, Z. Chen, A. W.-c. Fu, and D. Cheung. A robust outlier detection scheme for large data sets. In *In 6th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*. Citeseer, 2001.
- [695] N. Ye and Q. Chen. Chi-square Statistical Profiling for Anomaly Detection. In *Proc. of the 2000 IEEE Workshop on Information Assurance and Security*, pages 187–193, June 2000.
- [696] A. Zimek, E. Schubert, and H.-P. Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5(5):363– 387, 2012.

9.11 Exercises

1. Compare and contrast the different techniques for anomaly detection that were presented in [Section 9.2](#). In particular, try to identify circumstances in which the definitions of anomalies used in the different techniques might be equivalent or situations in which one might make sense, but another would not. Be sure to consider different types of data.
2. Consider the following definition of an anomaly: An anomaly is an object that is unusually influential in the creation of a data model.
 - a. Compare this definition to that of the standard model-based definition of an anomaly.
 - b. For what sizes of data sets (small, medium, or large) is this definition appropriate?
3. In one approach to anomaly detection, objects are represented as points in a multidimensional space, and the points are grouped into successive shells, where each shell represents a layer around a grouping of points, such as a convex hull. An object is an anomaly if it lies in one of the outer shells.
 - a. To which of the definitions of an anomaly in [Section 9.2](#) is this definition most closely related?
 - b. Name two problems with this definition of an anomaly.
4. Association analysis can be used to find anomalies as follows. Find strong association patterns, which involve some minimum number of objects. Anomalies are those objects that do not belong to any such patterns. To make this more concrete, we note that the hyperclique association pattern

discussed in [Section 5.8](#) is particularly suitable for such an approach. Specifically, given a user-selected h -confidence level, maximal hyperclique patterns of objects are found. All objects that do not appear in a maximal hyperclique pattern of at least size three are classified as outliers.

- a. Does this technique fall into any of the categories discussed in this chapter? If so, which one?
 - b. Name one potential strength and one potential weakness of this approach.
5. Discuss techniques for combining multiple anomaly detection techniques to improve the identification of anomalous objects. Consider both supervised and unsupervised cases.
6. Describe the potential time complexity of anomaly detection approaches based on the following approaches: model-based using clustering, proximity-based, and density. No knowledge of specific techniques is required. Rather, focus on the basic computational requirements of each approach, such as the time required to compute the density of each object.
7. The Grubbs' test, which is described by [Algorithm 9.2](#), is a more statistically sophisticated procedure for detecting outliers than that of [Definition 9.2](#). It is iterative and also takes into account the fact that the z-score does not have a normal distribution. This algorithm computes the z-score of each value based on the sample mean and standard deviation of the current set of values. The value with the largest magnitude z-score is discarded if its z-score is larger than g_c , the critical value of the test for an outlier at significance level α . This process is repeated until no objects are eliminated. Note that the sample mean, standard deviation, and g_c are updated at each iteration.

- a. What is the limit of the value $m-1mtc^2m-2+tc^2$ used for Grubbs' test as m approaches infinity? Use a significance level of 0.05.
- b. Describe, in words, the meaning of the previous result.

Algorithm 9.2 Grubbs' approach for outlier elimination.

- 1: Input the values and α
 $\{m$ is the number of values, α is a parameter, and tc is a value chosen so that $\alpha=P(x \geq tc)$ for a t distribution with $m-2$ degrees of freedom. $\}$
- 2: **repeat**
- 3: Compute the sample mean (\bar{x}) and standard deviation (s_x).
- 4: Compute a value g_c so that $P(|z| \geq g_c) = \alpha$.
 $(In terms of tc and m, g_c = m - 1mtc^2m - 2 + tc^2.)$
- 5: Compute the z-score of each value, i.e., $z = (x - \bar{x})/s_x$.
- 6: Let $g = \max|z|$ i.e., find the z-score of largest magnitude and call it g .
- 7: **if** $g > g_c$ **then**
- 8: Eliminate the value corresponding to g .
- 9: $m \leftarrow m - 1$
- 10: **end if**
- 11: **until** No objects are eliminated.

8. Many statistical tests for outliers were developed in an environment in which a few hundred observations was a large data set. We explore the limitations of such approaches.
- For a set of 1,000,000 values, how likely are we to have outliers according to the test that says a value is an outlier if it is more than three standard deviations from the average? (Assume a normal distribution.)
 - Does the approach that states an outlier is an object of unusually low probability need to be adjusted when dealing with large data sets? If so, how?
9. The probability density of a point \mathbf{x} with respect to a multivariate normal distribution having a mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is given by the equation
- $$f(\mathbf{x}) = \frac{1}{(2\pi)^m |\boldsymbol{\Sigma}|^{1/2}} e^{-(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})/2}. \quad (9.10)$$
- Using the sample mean $\bar{\mathbf{x}}$ and covariance matrix \mathbf{S} as estimates of the mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, respectively, show that the $\log(f(\mathbf{x}))$ is equal to the Mahalanobis distance between a data point \mathbf{x} and the sample mean $\bar{\mathbf{x}}$ plus a constant that does not depend on \mathbf{x} .
10. Compare the following two measures of the extent to which an object belongs to a cluster: (1) distance of an object from the centroid of its closest cluster and (2) the silhouette coefficient described in [Section 7.5.2](#).
11. Consider the (relative distance) K-means scheme for outlier detection described in [Section 9.5](#) and the accompanying figure, [Figure 9.10](#).
- The points at the bottom of the compact cluster shown in [Figure 9.10](#) have a somewhat higher outlier score than those points at the top of the compact cluster. Why?

- b. Suppose that we choose the number of clusters to be much larger, e.g., 10. Would the proposed technique still be effective in finding the most extreme outlier at the top of the figure? Why or why not?
- c. The use of relative distance adjusts for differences in density. Give an example of where such an approach might lead to the wrong conclusion.
12. If the probability that a normal object is classified as an anomaly is 0.01 and the probability that an anomalous object is classified as anomalous is 0.99, then what is the false alarm rate and detection rate if 99% of the objects are normal? (Use the definitions given below.)
- detection rate = $\frac{\text{number of anomalies detected}}{\text{total number of anomalies}}$ (9.11)
- false alarm rate = $\frac{\text{number of false anomalies}}{\text{number of objects classified}}$ (9.10)
13. When a comprehensive training set is available, a supervised anomaly detection technique can typically outperform an unsupervised anomaly technique when performance is evaluated using measures such as the detection and false alarm rate. However, in some cases, such as fraud detection, new types of anomalies are always developing. Performance can be evaluated according to the detection and false alarm rates, because it is usually possible to determine upon investigation whether an object (transaction) is anomalous. Discuss the relative merits of supervised and unsupervised anomaly detection under such conditions.
14. Consider a group of documents that has been selected from a much larger set of diverse documents so that the selected documents are as dissimilar from one another as possible. If we consider documents that are not highly related (connected, similar) to one another as being anomalous, then all of the documents that we have selected might be classified as anomalies. Is it

possible for a data set to consist only of anomalous objects or is this an abuse of the terminology?

15. Consider a set of points, where most points are in regions of low density, but a few points are in regions of high density. If we define an anomaly as a point in a region of low density, then most points will be classified as anomalies. Is this an appropriate use of the density-based definition of an anomaly or should the definition be modified in some way?

16. Consider a set of points that are uniformly distributed on the interval [0,1]. Is the statistical notion of an outlier as an infrequently observed value meaningful for this data?

17. An analyst applies an anomaly detection algorithm to a data set and finds a set of anomalies. Being curious, the analyst then applies the anomaly detection algorithm to the set of anomalies.

- a. Discuss the behavior of each of the anomaly detection techniques described in this chapter. (If possible, try this for real data sets and algorithms.)
- b. What do you think the behavior of an anomaly detection algorithm should be when applied to a set of anomalous objects?

10 Avoiding False Discoveries

The previous chapters have described the algorithms, concepts, and methodologies of four key areas of data mining: classification, association analysis, cluster analysis, and anomaly detection. A thorough understanding of this material provides the foundation required to start analyzing data in real-world situations. However, without careful consideration of some important issues in evaluating the performance of a data mining procedure, the results produced may not be meaningful or reproducible, i.e., the results may be false discoveries. The widespread nature of this problem has been reported by a number of high-profile publications in scientific fields, and is likewise, common in commerce and government. Hence, it is important to understand some of the common reasons for unreliable data mining results and how to avoid these false discoveries.

When a data mining algorithm is applied to a data set, it will dutifully produce clusters, patterns, predictive models, or a list of anomalies. However, any available data set is only a finite sample from the overall

population (distribution) of all instances, and there is often significant variability among instances within a population. Thus, the patterns and models discovered from a specific data set may not always capture the true nature of the population, i.e., allow accurate estimation or modeling of the properties of interest. Sometimes, the same algorithm will produce entirely different or inconsistent results when applied to another sample of data, thus indicating that the discovered results are spurious, e.g., not reproducible.

To produce valid (reliable and reproducible) results, it is important to ensure that a discovered pattern or relationship in the data is not an outcome of random chance (arising due to natural variability in the data samples), but rather, represents a significant effect. This often involves using statistical procedures, as will be described later. While ensuring the significance of a single result is demanding, the problem becomes more complex when we have multiple results that need to be evaluated simultaneously, such as the large numbers of itemsets typically discovered by a frequent pattern mining algorithm. In this case, many or even most of the results will represent false discoveries. This is also discussed in detail in this chapter.

The purpose of this chapter is to cover a few selected topics, knowledge of which is important for avoiding common data analysis problems and producing valid data mining results. Some of these topics have been

discussed in specific contexts earlier in the book, particularly in the evaluation sections of the preceding chapters. We will build upon these discussions to provide an indepth view of some standard procedures for avoiding false discoveries that are applicable across most areas of data mining. Many of these approaches were developed by statisticians for designed experiments, where the goal was to control external factors as much as possible. Currently, however, these approaches are often (perhaps even mostly) applied to observational data. A key goal of this chapter is to show how these techniques can be applied to typical data mining tasks to help ensure that the resulting models and patterns are valid.

10.1 Preliminaries: Statistical Testing

Before we discuss approaches for producing valid results in data mining problems, we first introduce the basic paradigm of statistical testing that is widely used for making inferences about the validity of results. A statistical test is a generic procedure for measuring the evidence for accepting or rejecting a hypothesis that the outcome (result) of an experiment or a data analysis procedure provides. For example, given the outcome of an experiment to study a new drug for a disease, we can test the evidence for the hypothesis that there is a measurable effect of the drug in treating the disease. As another example, given the outcome of a classifier on a test data set, we can test the evidence for the hypothesis that the classifier performs better than random guessing. In the following, we describe different frameworks for statistical testing.

10.1.1 Significance Testing

Suppose you want to hire a stockbroker who can make profitable decisions on your investments with a high success rate. You know of a stockbroker, Alice, who made profitable decisions for 7 of her last 10 stock picks. How confident would you be in offering her the job because of your assumption that a performance as good as Alice's is not likely due to random chance?

Questions of the above type can be answered using the basic tools of significance testing. Note that in any general problem of statistical testing, we are looking for some evidence in the outcome to validate a desired phenomenon, pattern, or relationship. For the problem of hiring a successful stockbroker, the desired phenomenon is that Alice indeed has knowledge of

how the stock prices vary and uses this knowledge to make 7 correct decisions out of 10. However, there is also a possibility that Alice's performance is no better than what might be obtained by randomly guessing on all 10 decisions. The primary goal of significance testing is to check whether there are sufficient evidence in the outcome to reject the default hypothesis (also called null hypothesis) that Alice's performance for making profitable stock decisions is no better than random.

Null Hypothesis

The **null hypothesis** is a general statement that the desired pattern or phenomenon of interest is not true and the observed outcome can be explained by the natural variability, e.g., by random chance. The null hypothesis is assumed to be true until there is sufficient evidence to indicate otherwise. It is commonly denoted as H_0 . Informally, if the result obtained from the data is *unlikely* under the null hypothesis, this provides evidence that our result is not just a result of natural variability in the data.

For example, the null hypothesis in the stockbroker problem could be that Alice is no better at making decisions than a person who performs random guessing. Rejecting this null hypothesis would imply that there is sufficient grounds to believe Alice's performance is better than random guessing. More generally, we are interested in the rejection of the null hypothesis since that typically implies an outcome that is not due to natural variability.

Since declaring the null hypothesis is the first step in the framework of significance testing, care must be taken to state it in a precise and complete manner so that the subsequent steps produce meaningful results. This is important because misstating or loosely stating the null hypothesis can yield misleading conclusions. A general approach is to begin with a statement of the desired result, e.g., a pattern captures an actual relationship between

variables, and take the null hypothesis to be the negation (opposite) of that statement, e.g., the pattern is due to natural variability in the data.

Test Statistic

To perform significance testing, we first need a way to quantify the evidence in the observed outcome with respect to the null hypothesis. This is achieved by using a **test statistic**, R , which typically summarizes every possible outcome as a numerical value. More specifically, the test statistic enables the computation of the probability of an outcome under the null hypothesis. For example, in the stockbroker problem, R could be the number of successful (profitable) decisions made in the last 10 decisions. In this way, the test statistic reduces an outcome consisting of 10 different decisions into a single numerical value, i.e., the count of successful decisions.

The test statistic is typically a count or real-valued quantity and measures how “extreme” an observed result is under the null hypothesis. Depending on the choice of the null hypothesis and the way the test statistic is designed, there can be different ways of defining what is “extreme” relative to the null hypothesis. For example, an observed test statistic, R_{obs} , can be considered extreme if it is greater than or equal to a certain value, R_H , smaller than or equal to a certain value, R_L , or outside a specified interval, $[R_L, R_H]$. The first two cases result in “one-sided tests” (right-tailed and left-tailed, respectively), while the last case results in a “two-sided test.”

Null Distribution

Having decided an appropriate test statistic for a problem, the next step in significance testing is to determine the distribution of the test statistic under

the null hypothesis. This is known as the **null distribution**, which can be formally described as follows.

Definition 10.1 (null distribution).

Given a test statistic, R , the distribution of R under the null hypothesis, H_0 , is called the **null distribution**, $P(R|H_0)$.

The null distribution can be determined in a number of ways. For example, we can use statistical assumptions about the behavior of R under H_0 to generate exact statistical models of the null distribution. We can also conduct experiments to produce samples from H_0 and then analyze these samples to approximate the null distribution. In general, the approach for determining the null distribution depends on the specific characteristics of the problem. We will discuss approaches for determining the null distribution in the context of data mining problems in [Section 10.2](#). We illustrate with an example of the null distribution for the stockbroker problem.

Example 10.1 (Null Distribution for Stockbroker Problem).

Consider the stockbroker problem where the test statistic, R , is the number of successes of a stockbroker in the last $N=100$ decisions. Under the null hypothesis that the stockbroker performs no better than random guessing, the probability of making a successful decision would be $p=0.5$. Assuming that the decisions on different days are independent of each other, the probability of obtaining an observed value of R , the total number of

successes in N decisions, under the null hypothesis can be modeled using the binomial distribution, which is described by the following equation:

$$P(R|H_0) = \binom{N}{R} p^R (1-p)^{N-R}.$$

Figure 10.1 shows the plot of this null distribution as a function of R for $N=100$.

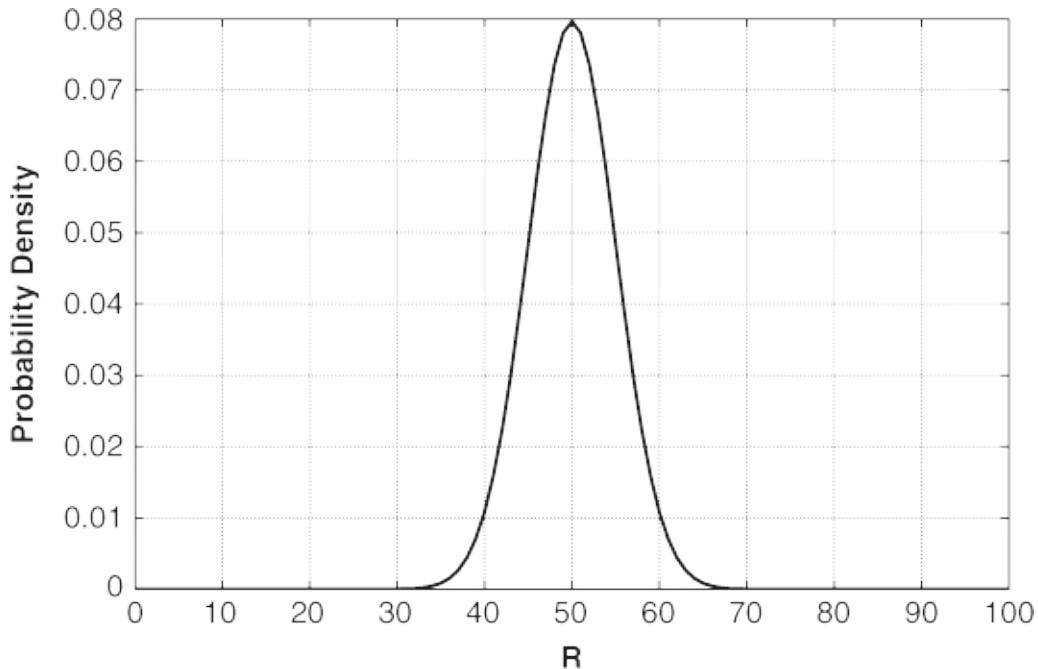


Figure 10.1.

Null distribution for the stockbroker problem with $N=100$.

The null distribution can be used to determine how *unlikely* is it to obtain the observed value of test statistic, R_{obs} , under the null hypothesis. In particular, the null hypothesis can be used to compute the probability of obtaining R_{obs} or “something more extreme” under the null hypothesis. This probability is called the *p-value*, which can be formally defined as follows:

Definition 10.2 (p-value).

The p-value of an observed test statistic, R_{obs} , is the probability of obtaining R_{obs} or something more extreme from the null distribution. Depending on how “more extreme” is defined for the test statistic, R , under the null hypothesis, H_0 , the p-value of R_{obs} can be written as follows:

$$p\text{-value}(R_{obs}) = \begin{cases} P(R \geq R_{obs} | H_0), & \text{for right-tailed tests.} \\ P(R \leq R_{obs} | H_0), & \text{for left-tailed tests.} \\ P(R \geq R_{obs} | H_0) + P(R \leq -|R_{obs}| | H_0), & \text{for two-sided tests.} \end{cases}$$

The reason that we account for “something more extreme” in the calculation of p-values is that the probability of any particular result is often 0 or close to 0. P-values thus capture the aggregate tail probabilities of the null distribution for test statistic values that are at least as extreme as R_{obs} . For the stockbroker problem, since larger values of the test statistic (count of successful decisions) would be considered more extreme under the null hypothesis, we would compute p-values using the right tail of the null distribution.

Example 10.2 (P-values are Tail Probabilities).

To illustrate the fact that p-values can be computed using the left tail, right tail, or both, consider an example where the null distribution has a Gaussian distribution with mean 0 and standard deviation 1, i.e., $N(0,1)$.

Figure 10.2 shows the test statistic values corresponding to a p-value of 0.05 for left-tailed, right tailed, or two-sided tests. (See shaded regions.)

We can see that the p-values correspond to the area in the tails of the null distribution. While a two-sided test has 0.025 probability in each of the tails, a one-sided test has all of its 0.05 probability in a single tail.

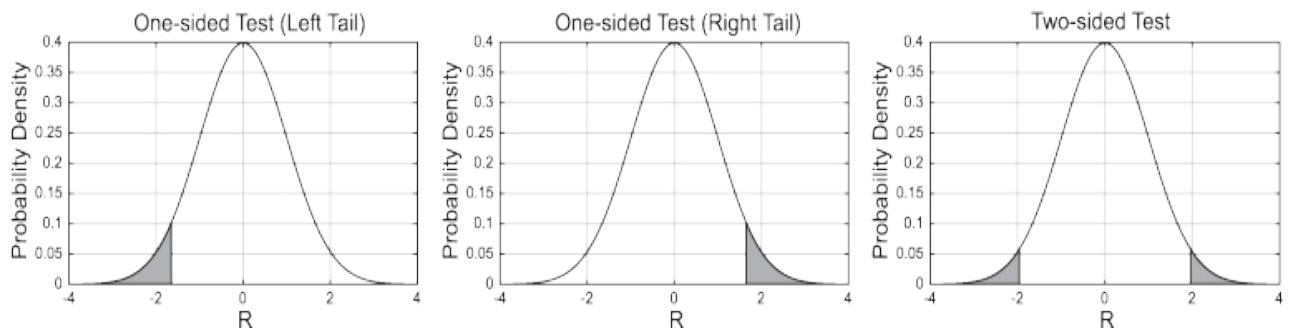


Figure 10.2.

Illustration of p-values as shaded regions for left-tailed, right-tailed, and two-sided tests.

Assessing Statistical Significance

P-values provide the necessary tool to assess the strength of the evidence in a result against the null hypothesis. The key idea is that if the p-value is low, then a result at least as extreme as the observed result is less likely to be obtained from H_0 . For example, if the p-value of a result is 0.01, then there is only a 1% chance of observing a result from the null hypothesis that is at least as extreme as the observed result.

A low p-value indicates smaller probabilities in the tails of the null distribution (for both one-sided and two-sided tests). This can provide sufficient evidence to believe that the observed result is a significant departure from the null hypothesis, thus convincing us to *reject H_0* . Formally, we often use a threshold on p-values (called the **level of significance**) and describe an observed p-value that is lower than this threshold as statistically significant.

Definition 10.3 (Statistically Significant Result).

Given a user-defined level of significance, α , a result is called statistically significant if it has a p-value lower than α .

Some common choices for the level of significance are 0.05 (5%) and 0.01 (1%). The p-value of a statistically significant result denotes the probability of falsely rejecting H_0 when H_0 is true. Hence, a low p-value provides higher confidence that the observed result is not likely to be consistent with H_0 , thus making it worthy of further investigation. This often means gathering additional data or conducting non-statistical verification, e.g., by performing experimental validation. (See Bibliographic Notes.) However, even when the p-value is low, there is always some chance (unless the p-value is 0) that H_0 is true and we have merely encountered a rare event.

It is important to keep in mind that a p-value is conditional probability, i.e., is computed under the assumption that H_0 is true. Consequently, a p-value is not the probability of H_0 , which may be likely or unlikely even if the test result is not significant. Thus, if a result is not significant, then it is not appropriate to say that we accept the null hypothesis. Instead, it is better to say that we fail to reject the null hypothesis. However, when the null hypothesis is known to be true most of the time, e.g., when testing for an effect or result that is rarely seen, it is common to say that we accept the null hypothesis. (See [Exercise 6](#).)

10.1.2 Hypothesis Testing

While significance testing was developed by the famous statistician Fisher as an actionable framework for statistical inference, its intended use is only limited to exploratory analyses of the null hypothesis in the preliminary stages of a study, e.g., to refine the null hypothesis or modify future experiments. One of the major limitations of significance testing is that it does not explicitly specify an **alternative hypothesis**, H_1 , which is typically that the statement we would like to establish as true, i.e., that a result is not spurious. Hence, significance testing can be used to reject the null hypothesis but is unsuitable for determining whether an observed result actually supports H_1 .

The framework of hypothesis testing, developed by the statisticians Neyman and Pearson, provides a more objective and rigorous approach for statistical testing, by explicitly defining both null and alternative hypotheses. Hence, apart from computing a p-value, i.e., the probability of falsely rejecting the null hypothesis when H_0 is true, we can also compute the probability of falsely saying a result is not significant when the alternative hypothesis is true. This allows hypothesis testing to provide a more detailed assessment of the evidence provided by an observed result.

In hypothesis testing, we first define both the null and alternative hypotheses (H_0 and H_1 , respectively) and choose a test statistic, R , that helps to differentiate the behavior of results under the null hypothesis from the behavior of results under the alternative hypothesis. As with significance testing, care must be taken to ensure that the null and alternative hypotheses are precisely and comprehensively defined. We then model the distribution of the test statistic under the null hypothesis, $P(R|H_0)$, as well as under the alternative hypothesis, $P(R|H_1)$. Similar to the null distribution, there can be many ways of generating the distribution of R under the alternative

hypothesis, e.g., by making statistical assumptions about the nature of H1 or by conducting experiments and analyzing samples from H1. In the following example, we concretely illustrate a simple approach for modeling $P(R|H_1)$ in the stockbroker problem.

Example 10.3 (Alternative Hypothesis for Stockbroker Problem).

In [Example 10.1](#), we saw that under the null hypothesis of random guessing, the probability of obtaining a success on any given day can be assumed to be $p=0.5$. There could be many alternative hypotheses for this problem, all of which would assume that the probability of success is greater than 0.5, i.e., $p>0.5$, thus representing a situation where the stockbroker performs better than random guessing. To be specific, assume $p=0.7$. The distribution of the test statistic (number of successes in $N=100$ decisions) under the alternative hypothesis would then be given by the following binomial distribution.

$$P(R|H_1) = \binom{N}{R} p^R (1-p)^{N-R}$$

[Figure 10.3](#) shows the plot of this distribution (dotted line) with respect to the null distribution (solid line). We can see that the alternative distribution is shifted toward the right. Notice that if a stockbroker has more than 60 successes, then this outcome will be more likely under H_1 than H_0 .

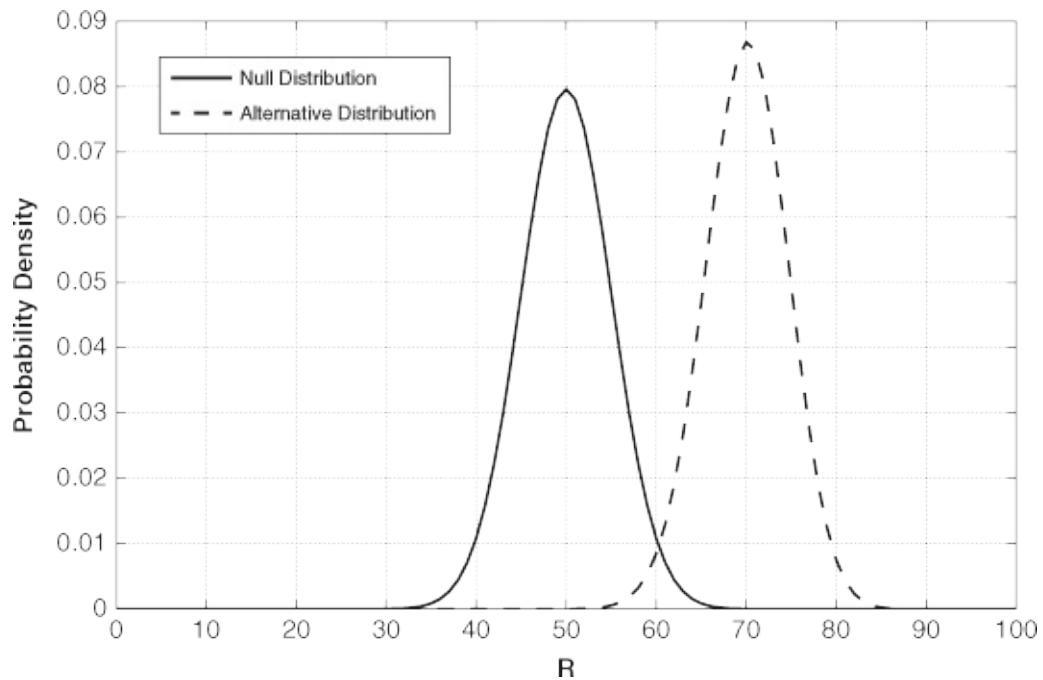


Figure 10.3.

Null and alternative distributions for the stockbroker problem with $N=100$.

Critical Region

Given the distributions of the test statistic under the null and alternative hypotheses, the framework of hypothesis testing decides if we should “reject” the null hypothesis or “not reject” the null hypothesis given the evidence provided by the test statistic computed from an observed result. This binary decision is typically made by specifying a range of possible values of the test statistic that are too extreme under H_0 . This set of values is called the **critical region**. If the observed test statistic, R_{obs} , falls in this region, then the null hypothesis is rejected. Otherwise, the null hypothesis is not rejected.

The critical region corresponds to the collection of extreme results whose probability of occurrence under the null hypothesis is less than a threshold. The critical region can either be in the left tail, right tail, or both left and right tails of the null distribution, depending on the type of statistical testing being used. The probability of the critical region under H_0 is called the **significance**

level, α . In other words, it is the probability of falsely rejecting the null hypothesis for results belonging to the critical region when H_0 is true. In most applications, a low value of α (e.g., 0.05 or 0.01) is specified by the user to define the critical region.

Rejecting the null hypothesis if the test statistic falls in the critical region is equivalent to evaluating the p-value of the test statistic and rejecting the null hypothesis if the p-value falls below a pre-specified threshold, α . Note that while every result has a different p-value, the significance level, α , in hypothesis testing is a fixed constant whose value is decided before any tests are performed.

Type I and Type II Errors

Up to this point, hypothesis testing may seem similar to significance testing, at least superficially. However, by considering both the null and alternative hypotheses, hypothesis testing allows us to look at two different types of errors, type I error and type II errors, as defined below.

Definition 10.4 (Type I Error).

A type I error is the error of incorrectly rejecting the null hypothesis for a result. The probability of incurring a type I error is called the type I error rate, α . It is equal to the probability of the critical region under H_0 , i.e., α is the same as the **significance level**. Formally,

$$\alpha = P(R \in \text{Critical Region} | H_0).$$

Definition 10.5 (Type II Error).

A type II Error is the error of falsely calling a result to be not significant when the alternative hypothesis is true. The probability of incurring a type II error is called the type II error rate, β , which is equal to the probability of observing test statistic values outside the critical region under H_1 , i.e.,

$$\beta = P(R \in \text{Critical Region} | H_1).$$

Note that deciding the critical region (specifying α) automatically determines the value of β for a particular test, given the distribution of the test statistic under the alternative hypothesis.

A closely related concept to the type II error rate is the **power** of the test, which is the probability of the critical region under H_1 , i.e., $1-\beta$. Power is an important characteristic of a test because it indicates how effective a test will be at correctly rejecting the null hypothesis. Low power means that many results that actually show the desired pattern or phenomenon will not be considered significant and thus will be missed. As a consequence, if the power of a test is low, then it may not be appropriate to ignore results that fall outside the critical region. Increasing the size of the critical regions to increase power and decrease type II errors will increase type I errors, and vice-versa. Hence, it is the balance between ensuring a low value of α and a high value of power that is at the core of hypothesis testing.

When the distribution of the test statistic under the null and alternative hypotheses depends on the number of samples used to estimate the test statistic, then increasing the number of samples helps obtain less variable estimates of the true null and alternative distributions. This reduces the chances of type I and type II errors. For example, evaluating a stockbroker on 100 decisions is more likely to give us an accurate estimate of their true success rate than evaluating the stockbroker on 10 decisions. The minimum number of samples required for ensuring a low value of α while having a high value of power is often determined by a statistical procedure called power analysis. (See Bibliographic Notes for more details.)

Example 10.4 (Classifying Medical Results).

Suppose the value of a blood test is used as the test statistic, R , to identify whether a patient has a certain disease or not. It is known that the value of this test statistic has a Gaussian distribution with mean 40 and standard deviation 5 for patients that do not have the disease. For patients having the disease, the test statistic has a mean of 60 and a standard deviation of 5. These distributions are shown in [Figure 10.4](#). H_0 is the null hypothesis that the patient doesn't have the disease, i.e., comes from the leftmost distribution shown in subfigures of [Figure 10.4](#). H_1 is the alternative hypothesis that the patient has the disease, i.e., comes from the rightmost distribution in the subfigures of [Figure 10.4](#).

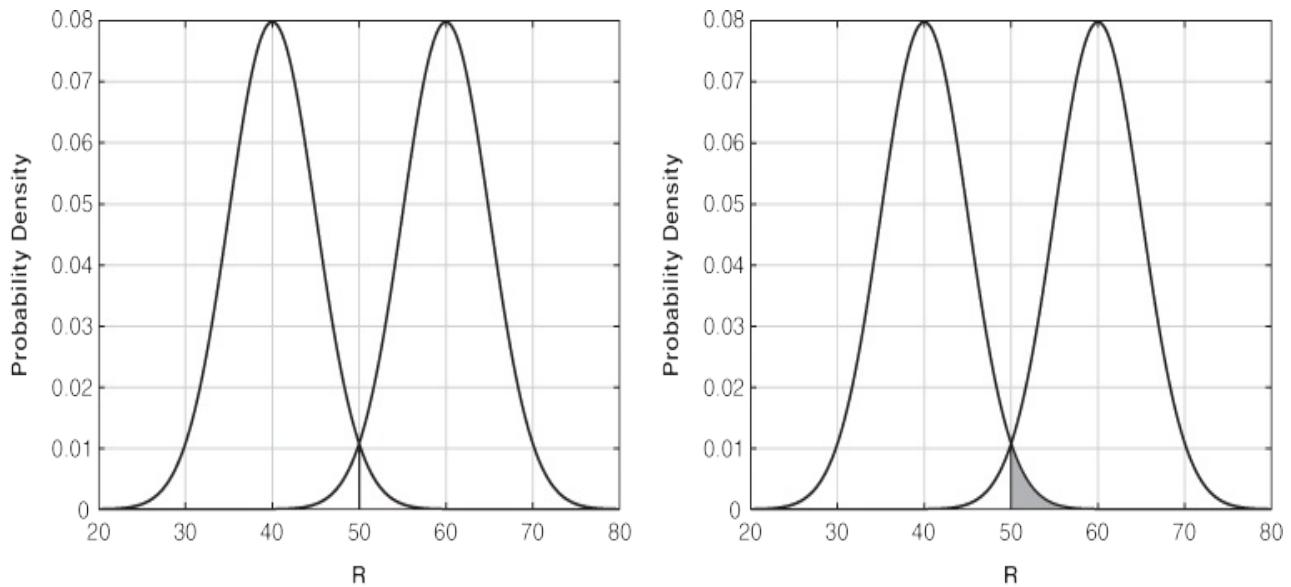


Figure 10.4.

Distribution of test statistic for the alternative hypothesis (rightmost density curve) and null hypothesis (leftmost density curve). Shaded region in right subfigure is α .

Suppose the critical region is chosen to be 50 and above, since a level of 50 is exactly halfway between means of the two distributions. The significance level, α , corresponding to this choice of critical region, shown as the shaded region in the right subfigure of [Figure 10.4](#), can then be calculated as follows:

$$\alpha = P(R \geq 50 | H_0) = P(R \geq 50 | R), R \sim N(\mu = 40, \sigma = 5) = \int_{50}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-(R-\mu)^2/2\sigma^2} dR = \int_{50}^{\infty} \frac{1}{\sqrt{2\pi}(5)} e^{-(R-40)^2/50} dR = 0.023$$

The type II error rate, β , for this choice of critical region can also be found to be equal to 0.023. (This is only because null and alternative hypotheses have the same distribution, except for their means, and the observed value is halfway between their means.) This is shown as the shaded region in the left subfigure of [Figure 10.5](#). The power is then equal to $1 - 0.023 = 0.977$, which is shown in the right subfigure of [Figure 10.5](#).

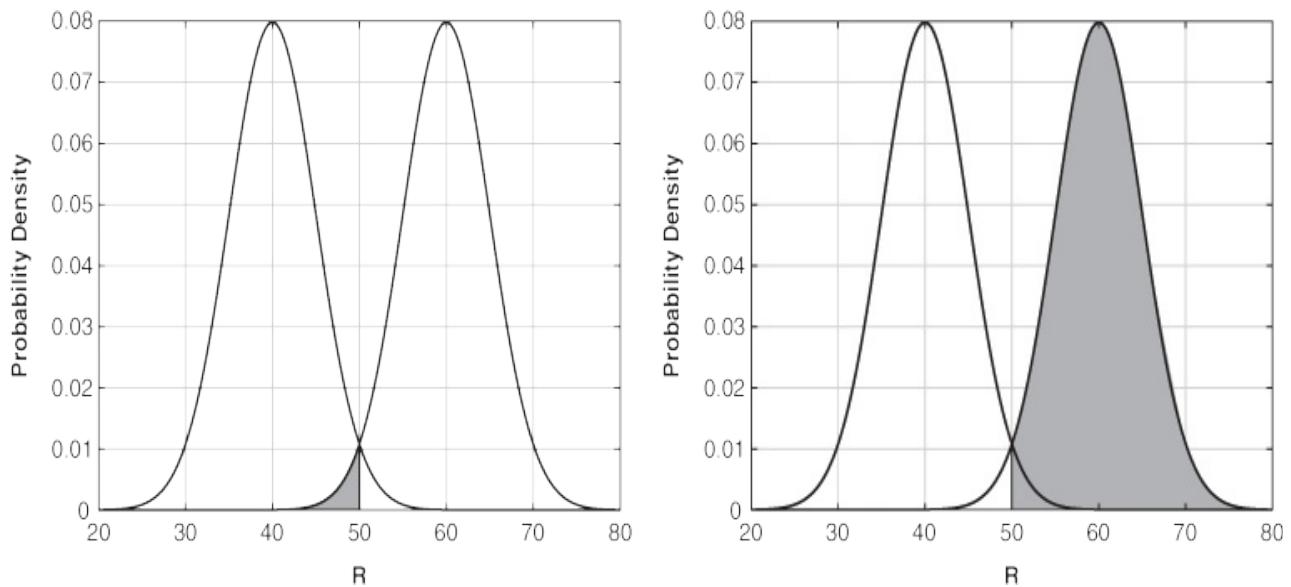


Figure 10.5.

Shaded region in left subfigure is β and shaded region in right subfigure is power.

If we use α of 0.05 instead of 0.023, the critical region would be slightly expanded to 48.22 and above. This would increase the power from 0.977 to 0.991, though at the cost of a higher value of α . On the other hand, decreasing α to 0.01 would decrease the power to 0.952.

Effect Size

Effect size brings in domain considerations by considering whether the observed result is significant from a domain point of view. For example, suppose that a new drug is found to lower blood pressure, but only by 1%. This difference will be statistically significant with a large enough test group, but the medical significance of an effect size of 1% is probably not worth the cost of the medicine and the potential for side effects. Thus, a consideration of effect size is critical since it can often happen that a result is statistically significant, but of no practical importance in the domain. This is particularly true for large data sets.

Definition 10.6 (effect size).

The effect size measures the magnitude of the effect or characteristic being evaluated, and is typically the magnitude of the test statistic.

In most problems there is a desired effect size that helps determines the null and alternative hypotheses. For the stockbroker problem, as illustrated in [Example 10.3](#), the desired effect size is the desired probability of success, 0.7. For the medical testing problem, which was just discussed in [Example 10.4](#), the effect size is the value of the threshold used to define the cutoff between normal patients and those with the disease. When comparing the means of two sets of observations (A and B), the effect size is the difference in the means, i.e., $\mu_A - \mu_B$ or the absolute difference, $|\mu_A - \mu_B|$.

The desired effect size impacts the choice of the critical region, and thus the significance level and power of the test. [Exercises 4](#) and [5](#) further explore some of these concepts.

10.1.3 Multiple Hypothesis Testing

The statistical testing frameworks discussed so far are designed to measure the evidence in a single result, i.e., whether the result belongs to the null hypothesis or the alternative hypothesis. However, many situations produce multiple results that need to be evaluated. For example, frequent pattern

mining typically produces many frequent itemsets from a given transaction data set, and we need to test every frequent itemset to determine whether there is a statistically significant association among its constituent items. The **multiple hypothesis testing** problem (also called the multiple testing problem or multiple comparison problem) addresses the statistical testing problem where multiple results are involved and a statistical test is performed on every result.

The simplest approach is to compute the p-value under the null hypothesis for each result independently of other results. If the p-value is significant for any result, then the null hypothesis is rejected for that result. However, this strategy will typically produce many erroneous results when the number of results to be tested is large. For example, even if something only has a 5% chance of happening for a single result, then it will happen, on average, 5 times out of a 100. Thus, our approach for hypothesis testing needs to be modified.

When working with multiple tests, we are interested in reporting the total number of errors committed on a collection of results (also referred to as a family of results). For example, if we have a collection of m results, we can count the total number of times a type I error is committed or a type II error is committed out of the m tests. The aggregate information of the performance across all tests can be summarized by the confusion matrix shown in [Table 10.1](#). In this table, a result that actually belongs to the null hypothesis is called a ‘negative’ while a result that actually belongs to the alternative hypothesis is called a ‘positive.’ This table is essentially the same as [Table 4.6](#), which was introduced in the context of evaluating classification performance in [Section 4.11.2](#).

Table 10.1. Confusion table in the context of multiple hypothesis testing.

	Declared significant (+)	Declared not significant (-)	Total
--	--------------------------	------------------------------	-------

	prediction)	prediction)	
H1 True (actual +)	True Positive (TP)	False Negative (FN) type II error	Positives (m1)
H0 True (actual -)	False Positive (FP) type I error	True Negative (TN)	Negatives (m0)
	Positive Predictions (Ppred)	Negative Predictions (Npred)	<i>m</i>

In most practical situations where we are performing multiple hypothesis testing, e.g., while using statistical tests to evaluate whether a collection of patterns, clusters, etc. are spurious, the required entries in [Table 10.1](#) are seldom available. (For classification, the table is available when reliable labels are available, in which case, many of the quantities of interest can be directly estimated. See [Section 10.3.2](#).) When entries are not available, we need to estimate them, or more typically, quantities derived from these entries. The following paragraphs of this section describe various approaches for doing this.

Family-wise Error Rate (FWER)

A useful error metric when dealing with a family of results is the **family-wise error rate (FWER)**, which is the probability of observing even a single false positive (type I error) in the entire set of m results. In particular,

$$\text{FEWR} = P(\text{FP} > 0).$$

If the FWER is lower than a certain threshold, say α , then the probability of observing any type I error among all the results is less than α .

The FWER thus measures the probability of observing a type I error in any or all of the m tests. Controlling the FWER, i.e., ensuring the FWER to be low, is useful in applications where a set of results is discarded if even a single test is erroneous (produces a type I error). For example, consider the problem of selecting a stockbroker described in [Example 10.3](#). In this case, the goal is to find, from a pool of applicants, a stockbroker that makes correct decisions at least 70% of the time. Even a single type I error can lead to an erroneous hiring decision. In such cases, estimating the FWER gives us a better picture of the performance of the entire set of results than the naïve approach of computing p-values separately for each result. The following example illustrates this concept in the context of the stockbroker problem.

Example 10.5 (Testing Multiple Stockbrokers).

Consider the problem of selecting successful stockbrokers from a pool of $m=50$ candidates. For every stockbroker, we perform a statistical test to check whether their performance (number of successful decisions in the last N decisions) is better than random guessing. If we use a significance level of $\alpha=0.05$ for every such test, the probability of making a type I error on any individual candidate is equal to 0.05. However, if we assume that the results are independent, the probability of observing even a single type I error in any of the 50 tests, i.e., the FWER, is given by

$$\text{FWER} = 1 - (1 - \alpha)^m = 1 - (1 - 0.05)^{50} = 0.923, \quad (10.1)$$

which is extremely high. Even though the probability of observing no false positives on a single test is quite high ($1 - \alpha = 0.95$), the probability of seeing no false positives across all tests ($0.95^{50} = 0.077$) diminishes by repeated multiplication. Hence, the FWER can be quite high when m is large, even if the type I error rate, α , is low.

Bonferroni Procedure

A number of procedures have been developed to ensure that the FWER of a set of results is lower than an acceptable threshold, α , which is often 0.05. These approaches, called FWER controlling procedures, basically try to adjust the p-value threshold which is used for every test, so that there is only a small chance of erroneously rejecting the null hypothesis in the presence of multiple tests. To illustrate this category of procedures, we describe the most conservative approach, which is the **Bonferroni procedure**.

Definition 10.7 (Bonferroni procedure).

If m results are to be tested so that the FWER is less than α , the Bonferroni procedure sets the significance level for every test to be $\alpha^* = \alpha/m$.

The intuition behind the Bonferroni procedure can be understood by observing the formula for FWER in [Equation 10.1](#), where the m tests are assumed to be independent of each other. Using a reduced significance level of α/m in [Equation 10.1](#) and applying the binomial theorem, we can see that the FWER is controlled below α as follows:

$$\text{FWER} = 1 - (1 - \alpha/m)^m = 1 - (1 + m(-\alpha/m) + (m^2)(-\alpha/m)^2/2! + \dots + (-\alpha/m)^m) = \alpha - (m^2)(-\alpha/m)^2/2! - (m^3)(-\alpha/m)^3/3! - \dots - (-\alpha/m)^m \leq \alpha.$$

While the above discussion was for the case where the tests are assumed to be independent, the Bonferroni approach guarantees no type I error in the m tests with a probability of $1-\alpha$, irrespective of whether the tests (results) are correlated or independent. We illustrate the importance of the Bonferroni procedure for controlling FWER using the following example.

Example 10.6 (Bonferroni Procedure).

In the multiple stockbroker problem described in [Example 10.5](#), we analyze the effect of the Bonferroni procedure in controlling the FWER. The null distribution for an individual stockbroker can be modeled using the binomial distribution where $p=0.5$ and $N=100$. Given a set of m results simulated from the null distribution (assuming the results are independent), we compare the performance of two competing approaches: the naïve approach, which uses a significance level of $\alpha=0.05$, and the Bonferroni procedure, which uses a significance level of $\alpha^*=0.05/m$.

[Figure 10.6](#) shows the FWER of these two approaches as we vary the number of results, m . (We used 106 simulations.) We can see that the FWER of the Bonferroni procedure is always controlled to be α , while the FWER of the naïve approach shoots up rapidly and reaches a value close to 1 for m greater than 70. Hence, the Bonferroni procedure is preferred over the naïve approach when m is large and the FWER is the error metric we wish to control.

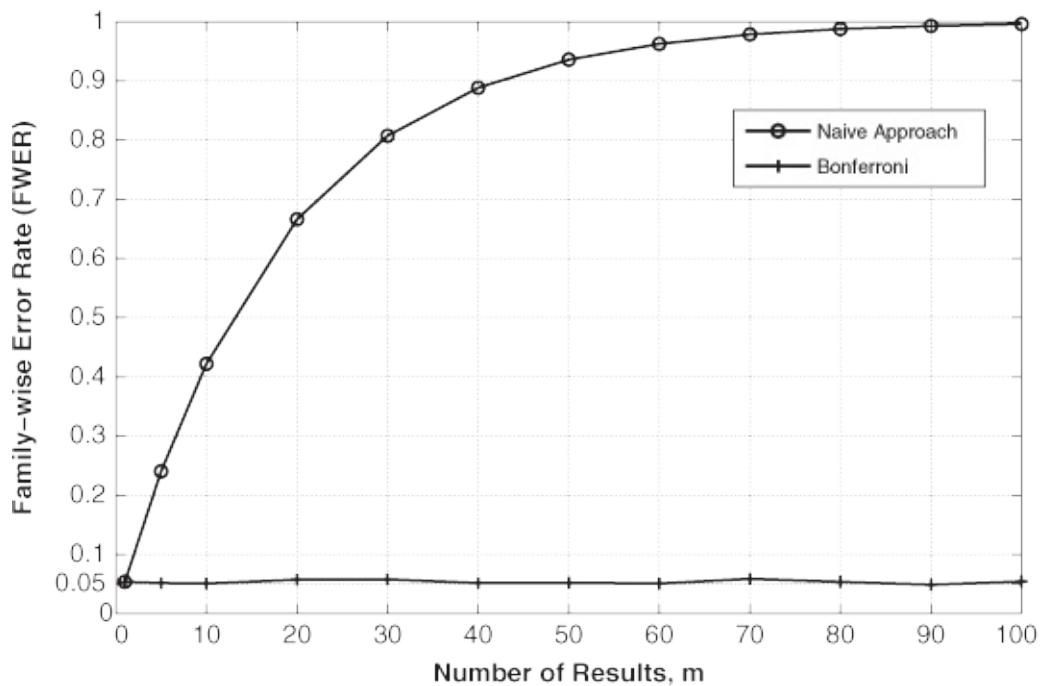


Figure 10.6.

The family wise error rate (FWER) curves for the naïve approach and the Bonferroni procedure as a function of the number of results, m .

The Bonferroni procedure is almost always overly conservative, i.e., it will eliminate some non-spurious results, especially when the number of results is large and the results may be correlated to each other, e.g., in frequent pattern mining. In the extreme case where all m results are perfectly correlated to each other (and hence identical), the Bonferroni procedure would still use a significance level of α/m even though a significance level of α would have sufficed. To address this limitation, a number of alternative FWER controlling procedures have been developed that are less conservative than Bonferroni when dealing with correlated results. (See Bibliographic Notes for more details.)

False discovery rate (FDR)

By definition, all FWER controlling procedures seek a low probability for obtaining any false positives, and thus, are not the appropriate tool when the goal is to allow some false positives in order to get more true positives. For example, in frequent pattern mining, we are interested in selecting frequent itemsets that show statistically significant associations (actual positives), while discarding the remaining ones. As another example, when testing for a serious disease, it is better to get more true positives (detect more actual cases of the disease) even if that means generating some false positives. In both cases, we are ready to tolerate a few false positives as long as we are able to achieve reasonable power for the detection of true positives.

The **false discovery rate** (FDR) provides an error metric to measure the rate of false positives, which are also called false discoveries. To compute FDR, we first define a variable, Q , that is equal to the number of false positives, FP , divided by the total number of results predicted as positive, P_{pred} . (See [Table 10.1](#).)

$$Q = \frac{FP}{P_{pred}} = \frac{FP}{TP + FP}, \quad \text{if } P_{pred} > 0, \quad \text{if } P_{pred} = 0$$

When we know FP , the number of false positives, as in classification, Q is essentially the false discovery rate, as defined in [Section 4.11.2](#), which introduced measures for evaluating classification performance under class imbalance. As such, Q is closely related to the precision. Specifically, $\text{precision} = 1 - \text{FDR} = 1 - Q$. However, in the context of statistical testing, when $P_{pred} = 0$, i.e., when no results are predicted as positive, $Q = 0$ by definition. However, in data mining, precision and thus FDR, as defined in [Section 4.11.2](#), are typically considered to be undefined in that situation.

In the cases where we do not know FP , it is not possible to use Q as the false discovery rate. Nonetheless, it is still possible to estimate the value of Q , on

average, i.e., to compute the expected value of Q and use that as our false discovery rate. Formally,

$$\text{FDR} = E(Q) \quad (10.2)$$

The FDR is a useful metric for ensuring that the rate of false positives is low, especially in cases where the positives are highly skewed, i.e., the number of actual positives in the collection of results, m_0 , is very small compared to the number of actual negatives, m_1 .

Benjamini-Hochberg Procedure

Statistical testing procedures that try to control the FDR are known as FDR controlling procedures. These procedures can typically ensure a low number of false positives (even when the positive class is relatively infrequent) while providing higher power than the more conservative FWER controlling procedures. A widely-used FDR controlling procedure is the **Benjamini-Hochberg (BH) procedure**, which sorts the results in increasing order of their p-values and uses a different significance level, $\alpha(i)$, for every result, R_i .

The basic idea behind the BH procedure is that if we have observed a large number of significant results that have a lower p-value than a given result, R_i , we can be less stringent while testing R_i and use a more relaxed significance level than α/m . [Algorithm 10.1](#) provides a summary of the BH procedure. The first step in this algorithm is to compute the p-values for every result and sort the results in increasing order of their p-values (steps 1 to 2). Thus, p_i would correspond to the i th smallest p-value. The significance level, α_i , for p_i is then computed using the following correction (line 3)

$$\alpha_i = i \times \alpha/m$$

Notice that the significance level for the smallest p-value, p_1 , is equal to α/m , which is same as the correction used in the Bonferroni procedure. Further, the significance level for the largest p-value, p_m , is equal to α , which is the significance level for a single test (without accounting for multiple hypothesis testing). In between these two p-values, the significance level grows linearly from α/m to α . Hence, the BH procedure can be viewed as striking a balance between the overly conservative Bonferroni procedure and the overly liberal naïve approach, thus resulting in higher power (finding more actual positives) without producing too many false positives. Let k be the largest index such that p_k is lower than its significance level, α_k (line 4). The BH procedure then declares the first k p-values as significant (lines 4 to 5). It can be shown that the FDR computed using the BH procedure is guaranteed to be smaller than α . In particular,

$$\text{FDR} \leq m_0 m \alpha \leq \alpha, \quad (10.3)$$

where m_0 is the number of actual negative results and m is the total number of results. (See [Table 10.1](#).)

Algorithm 10.1 Benjamini-Hochberg (BH)

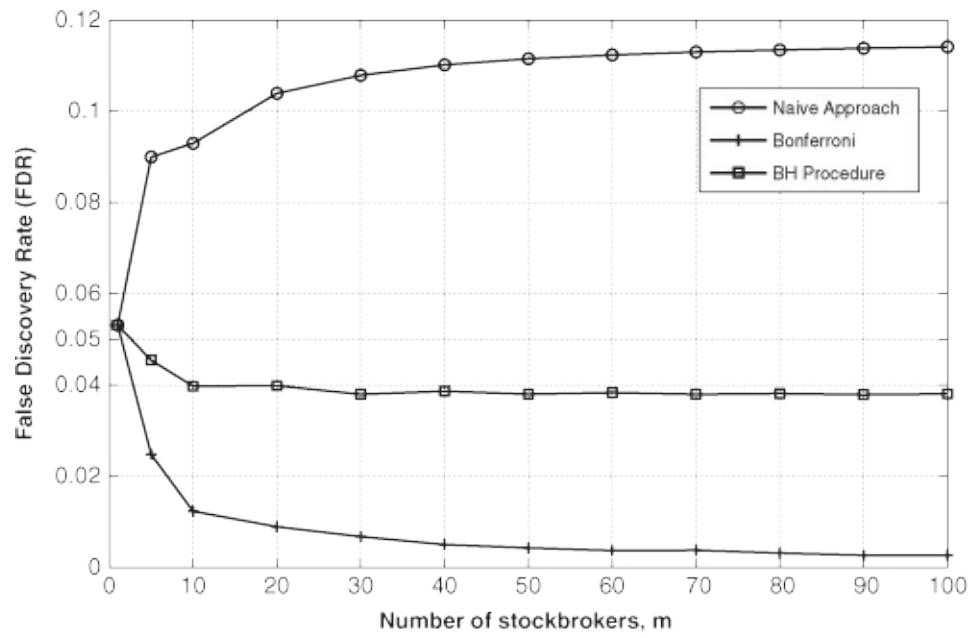
FDR algorithm.

- 1: Compute p-values for the m results.
- 2: Order the p-values from smallest to largest (p_1 to p_m).
- 3: Compute the significance level for p_i as $\alpha_i = i \times \alpha/m$.
- 4: Let k be the largest index such that $p_k \leq \alpha_k$.
- 5: Reject H_0 for all results corresponding to the first k p-values, $p_i, 1 \leq i \leq k$.

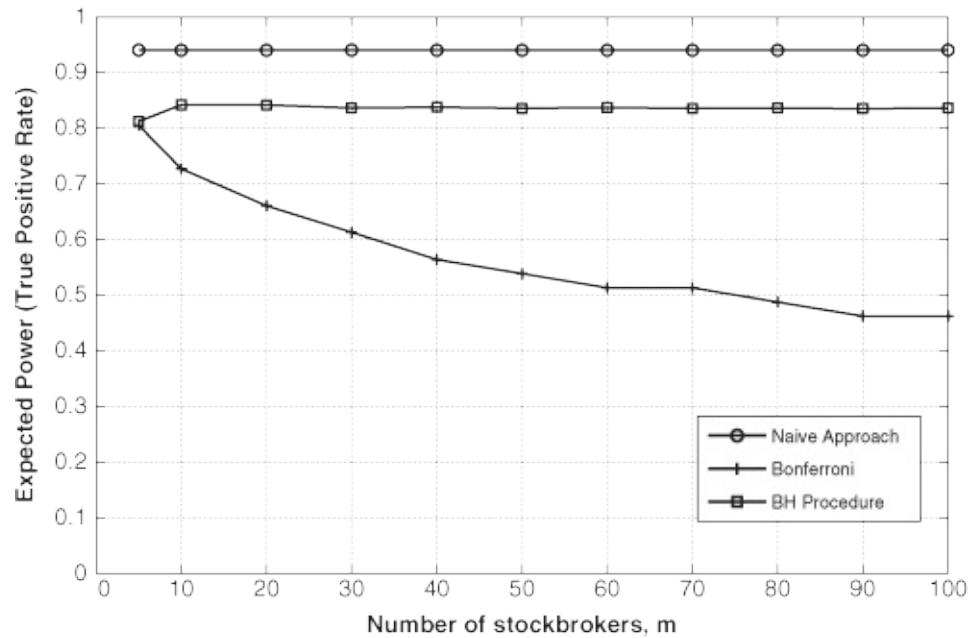
Example 10.7 (BH and Bonferroni procedure).

Consider the multiple stockbroker problem discussed in [Example 10.6](#), where instead of assuming all m stockbrokers to belong to the null distribution, we may have a small number of m_1 candidates who belong to an alternative distribution. The null distribution can be modeled by the binomial distribution with 0.5 probability of making a successful decision. The alternative distribution can be modeled by the binomial distribution with 0.55 probability of success, which is a slightly higher probability of success than that of random guessing. We consider $N=100$ decisions for both the null and alternative distributions.

We are interested in comparing the performance of the Bonferroni and BH procedures in detecting a large fraction of actual positives (stockbrokers that indeed perform better than random guessing) without incurring a lot of false positives. We ran 106 simulations of m stockbrokers where m_1 stockbrokers in each simulation belong to the alternative distribution while the rest belong to the null distribution. We chose $m_1=m/3$ to demonstrate the effect of a skewed positive class, which is quite common in most practical applications of multiple hypothesis testing. [Figure 10.7](#) shows the plots of FDR and expected power as we vary the number of stockbrokers in each simulation run, m , for three competing procedures: the naïve approach, the Bonferroni procedure, and the BH procedure. The choice of the threshold, α , in each of the three procedures was chosen to be 0.05.



(a) False Discovery Rate as function of m .



(b) Expected Power as function of m .

Figure 10.7.

Comparing the performance of multiple comparison procedures as we vary the number of results, m , and set $m_1=m/3$ results as positive. $\alpha=0.05$ for each of the three procedures.

We can see that the FDR of both the Bonferroni and the BH procedures are smaller than 0.05 for all values of m , but the FDR of the naïve approach is not controlled and is close to 0.1. This shows that the naïve approach is quite relaxed in calling a result to be positive, and thus incurs more false positives. However, it also generally shows a high power as many of the actual positives are indeed labeled as positive. On the other hand, the FDR of the Bonferroni is much smaller than 0.05 and is the lowest among all the three approaches. This is because the Bonferroni approach aims at controlling a more stringent error metric, i.e., the FWER, to be smaller than 0.05. However, it also has low power as it is conservative in calling an actual positive to be significant since its goal is to avoid any false positives.

The BH procedure makes a balance between being conservative and relaxed such that its FDR is always smaller than 0.05 but its expected power is quite high and comparable to the naïve approach. Hence, at the cost of a minor increase in the FDR over the Bonferroni procedure, it is able to achieve much higher power and thus obtain a better trade-off between minimizing type I errors and type II errors in multiple hypothesis testing problems. However, we emphasize, that FWER procedures, such as Bonferroni, and FDR controlling procedures, such as BH, are intended for two different tasks, and thus, the best procedure to use in any particular situation will vary depending on the goal of the analysis.

Equation 10.3  states that the FDR of the BH procedure is less than or equal to $m_0/m \times \alpha$, which is equal to α only when $m_0=m$, i.e., when there are no actual positives in the results. Hence, the BH procedure generally discovers a smaller number of true positives, i.e., has lower power, than it should be given a desired FDR of α . To address this limitation of the BH procedure, a number of statistical testing procedures have been developed to

provide tighter control over FDR, such as the positive FDR controlling procedures, and the local FDR controlling procedures. These techniques generally show better power than the BH procedure while ensuring a small number of false positives. (See Bibliographic Notes for more details.)

Note that some users of FDR controlling procedures assume that α should be chosen in the same way as for hypothesis (significance) testing or for FWER controlling procedures, which traditionally use $\alpha=0.05$ or $\alpha=0.01$. However, for FDR controlling procedures, α is the desired false discovery rate and is often chosen to have a value greater than 0.05, e.g., 0.20. The reason for this is that in many cases the person evaluating the results is willing to accept more false positives in order to get more true positives. This is especially true when few, if any, potential positive results are produced when α is set to a low value, such as 0.05 or 0.01. In the previous example, we chose α to be the same for all three techniques to keep the discussion simple.

10.1.4 Pitfalls in Statistical Testing

The statistical testing approaches presented above provide an effective framework for measuring the evidence in results. However, as with other data analysis techniques, using them incorrectly can often produce misleading conclusions. Much of the misunderstanding is centered on the use of p-values. In particular, p-values are commonly assigned additional meaning beyond what can be supported by the data and these procedures. In the following, we discuss some of the common pitfalls in statistical testing that should be avoided to produce valid results. Some of these describe p-values and their proper role while others identify common misinterpretations and misuses. (See Bibliographic Notes for more details.)

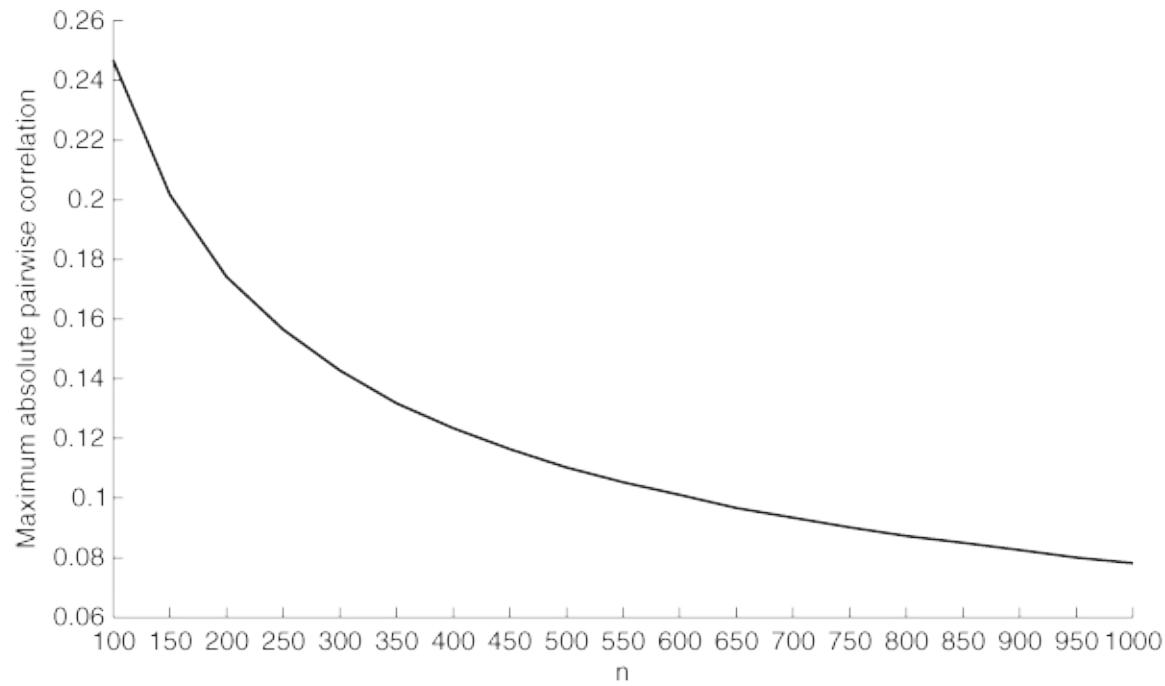
1. A *p*-value is not the probability that the null hypothesis is true. As described previously in **Definition 10.2**, the p-value is the conditional probability of observing a particular value of a test statistic, R , or something more extreme under the null hypothesis. Hence, we are *assuming* the null hypothesis is true in order to compute the p-value. A p-value does measure how compatible the observed result is with the null hypothesis.
2. Typically, there can be many hypotheses that explain a result that is found to be significant or non-significant under the null hypothesis. Note that a result that is declared non-significant, i.e., has a high p-value, was not necessarily generated from the null distribution. For example, if we model the null distribution using a Gaussian distribution with mean 0 and standard deviation 1, we will find an observed test statistic, $R_{obs}=1.5$, to be non-significant at a 5% level. However, the result could be from the alternative distribution, even if there is a low (but nonzero) probability of that event. Further, if we misspecified our null hypothesis, then the same observation could have easily come from another distribution, e.g., a Gaussian distribution with mean 1 and standard deviation 1, under which it is more likely. Hence, declaring a result to be non-significant does not amount to “accepting” the null hypothesis. Likewise, a significant result may be explained by many alternative hypothesis. Hence, rejecting the null hypothesis does not necessarily imply that we have accepted the alternative hypothesis. This is one of the reasons that p-values, or more generally the result of statistical testing, are not usually sufficient for making decisions. Factors external to the statistics, such as domain knowledge, must be applied as well.
3. A low p-value does not imply a useful effect size (magnitude of the test statistic) and vice versa. Recall that the effect size is the size of the test statistic for which the result is considered important in the domain of interest. Thus, effect size brings in domain considerations by

considering whether the observed result is significant from a domain point of view. For example, suppose that a new drug is found to lower blood pressure, but only by 1%. This difference may be statistically significant, but the medical significance of an effect size of 1% is probably not worth the cost of the medicine and the potential for side effects. In particular, a significant p-value may not have a large effect size and a non-significant p-value does not imply no effect size. Since p-values depend very strongly on the size of the data set, small p-values for big data applications are becoming increasingly common since even small effect sizes will show up as being statistically significant. Thus, it becomes critical to take effect size into consideration to avoid generating results that are statistically significant but not useful. In particular, even if a result is declared significant, we should ensure that its effect size is greater than a domain-specified threshold to be of practical importance.

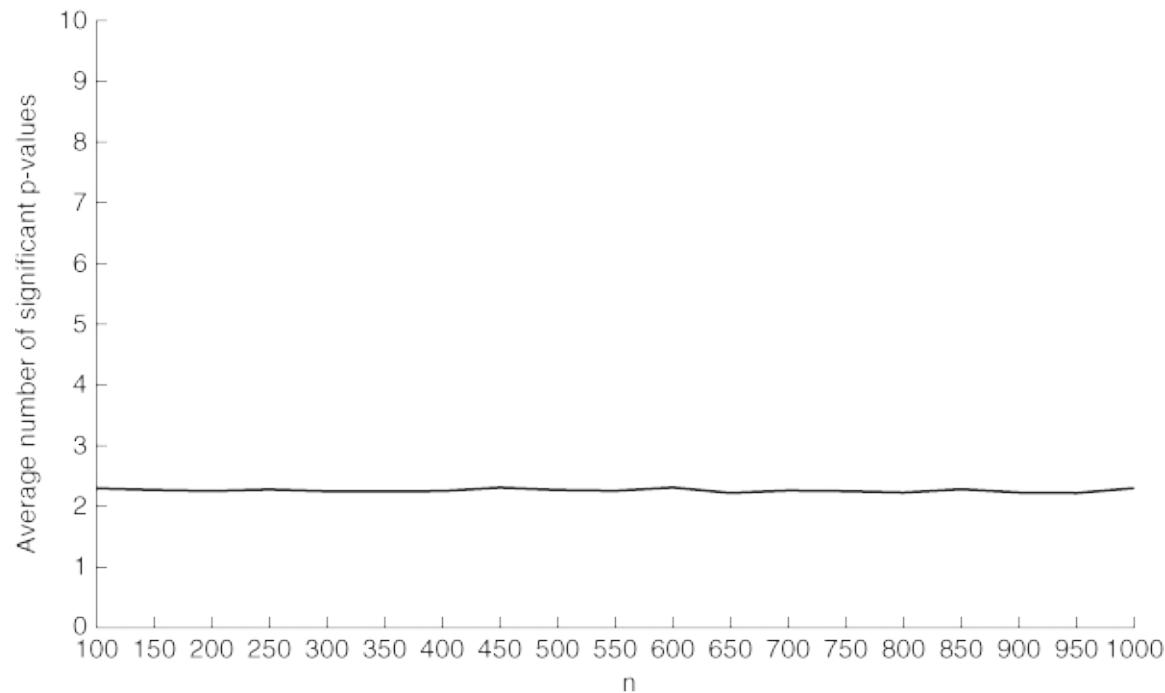
Example 10.8 (Significant p-values in random data).

To illustrate the point that we can obtain significantly low p-values even with small effect sizes, we consider the pairwise correlation of 10 random vectors that were generated from a Gaussian distribution of mean 0 and standard deviation 1. The null hypothesis is that the correlation of any two vectors is 0. [Figures 10.8a](#) and [10.8b](#) show that as the length of the vectors, n , increases, the maximum absolute pairwise correlation of any pair of vectors tends to 0, but the average number of pairwise correlations that have a p-value less than 0.05 remains constant at about 2.25. This shows that the number of significant pair-wise correlations does not decrease when n is large, although the effect size (maximum absolute correlation) is quite low.

This example also illustrates the importance of adjusting for multiple tests. There are 45 pairwise correlations and thus, on average, we would expect $0.05 \times 45 = 2.25$ significant correlations at the 5% level, as is shown in **Figure 10.8b**.



(a) Plot of maximum absolute pairwise correlation between 10 random vectors.



(b) Plot of average number of significant pairwise correlations

Figure 10.8.

Visualizing the effect of changing the vector length, n , on the correlations among 10 random vectors.

4. *It is unsound to analyze a result in multiple ways until we are able to declare it statistically significant.* This creates a multiple hypothesis testing problem and the p-values of individual results are no longer a good guide as to whether the null hypothesis should be rejected or not. (Such approaches are known as p-value hacking.) This can include cases where p-values are not explicitly used but the data is pre-processed or adjusted until a model is found that is acceptable to the investigator.

10.2 Modeling Null and Alternative Distributions

A primary requirement for conducting statistical testing is to know how the test statistic is distributed under the null hypothesis (and sometimes under the alternative hypothesis). In conventional problems of statistical testing, this consideration is kept in mind while designing the experimental setup for collecting data, so that we have enough data samples pertaining to the null and alternative hypotheses. For example, in order to test the effect of a new drug in curing a disease, experimental data is usually collected from two groups of subjects that are as similar as possible in all respects, except that one group is administered the drug while the control group is not. The data samples from the two groups then provide information to model the alternative and null distributions, respectively. There is an extensive body of work on experimental design that provides guidelines for conducting experiments and collecting data pertaining to the null and alternative hypotheses, so that they can be used later for statistical testing. However, such guidelines cannot be directly applied when dealing with observational data where the data is collected without a prior hypothesis in mind, as is common in many data mining problems.

Hence, a central objective when using statistical testing with observational data is to come up with an approach to model the distribution of the test statistic under the null and alternative hypotheses. In some cases, this can be done by making some statistical assumptions about the observations, e.g., that the data follows a known statistical distribution such as the normal, binomial, or hypergeometric distributions. For example, the instances in a data set may be generated by a single normal distribution whose mean and

variance can be estimated from the data set. Note that in almost all cases where a statistical model is used, the parameters of that model must be estimated from the data. Hence, any probabilities calculated using a statistical model may have some inherent error, with the magnitude of that error dependent on how well the chosen distribution fits the data and how well the parameters of the model can be estimated.

In some scenarios, it is difficult, or even impossible, to adequately model the behavior of the data with a known statistical distribution. An alternative method is to first generate sample data sets under the null or alternative hypotheses, and then model the distribution of the test statistic using the new data sets. For the alternative hypothesis, the new data sets must be similar to the current data set, but should reflect the natural variability inherent in the data. For the null hypothesis, these data sets should be as similar as possible to the original data set, but lack the structure or pattern of interest, e.g., a connection between attributes and values, cluster structure, or associations between attributes.

In the following, we describe some generic procedures for estimating the null distribution in the context of statistical testing for data mining problems. (Unfortunately, outside of using a known statistical distribution, there are not many widely used methods for generating the alternative distribution.) These procedures will serve as the building blocks of the specific approaches for statistical testing discussed in [Sections 10.3](#) to [10.6](#). Note that the exact details of the approach used for estimating the null distribution depends on the specific type of problem being studied and the nature of hypotheses being considered. However, at a high level, approaches involve generating completely new synthetic data sets or randomizing labels. In addition, we will discuss approaches for resampling existing instances, which can be useful for generating confidence intervals for various data mining results, such as the accuracy of a predictive model.

10.2.1 Generating Synthetic Data Sets

For analyses involving unlabeled data such as clustering and frequent pattern mining, the main approach for estimating a null distribution is to generate synthetic data sets, either by randomizing the order of attribute values or by generating new instances. The resultant data sets should be similar to the original data set in all manners except that they lack a pattern of interest, e.g., cluster structure or frequent patterns, whose significance has to be assessed.

For example, if we need to assess whether the items in a transaction data set are related to each other beyond whatever association occurs by random chance, we can generate synthetic transaction data sets by randomizing (permuting) the order of existing entries within rows and columns of the binary representation of a transaction data set. The goal is that the resulting data set should have properties similar to the original data set in terms of the number of times an item occurs in the transactions (i.e., the support of an item) and the number of items in every transaction (i.e., the length of a transaction), but have statistically independent items. These synthetic data sets can be processed to find association patterns and these results can be used to provide an estimate of the distribution of the test statistic of interest, e.g., the support or confidence of an itemset, under the null hypothesis. See [Section 10.4](#) for more details

If we need to evaluate whether the cluster structure in a data set is better than we might expect by random chance, we need to generate new instances that, when combined in a data set, lack cluster structure. The synthetic data sets can be clustered and used to estimate the null distribution of test statistic. For cluster analysis, the quantity of interest, i.e., the test statistic, is typically some measure of clustering goodness, such as SSE or the silhouette coefficient. See [Section 10.5](#).

Although the process of randomizing attributes may appear simple, executing this approach can be very challenging since a naïve attempt at generating synthetic data sets may omit important characteristics or structure of the original data and thus may yield an inadequate approximation of the true null distribution. For example, given a time series data, we need to ensure that the consecutive values in the randomized time series are similar to each other, since time series data typically exhibit temporal autocorrelation. Further, if the time series data have a yearly cycle (e.g., in climate data), we would need to ensure that such cyclic patterns are also preserved in the synthetically generated time series.

Specific techniques for generating synthetic data sets will be discussed in more detail in the context of association analysis and clustering in [Sections 10.4](#) and [10.5](#), respectively.

10.2.2 Randomizing Class Labels

When every data instance has an associated class label, a common approach for generating new data is to randomly permute the class labels, a process also referred to as permutation testing. This involves repeatedly shuffling (permuting) labels among data objects at random to generate a new data set that is identical to the old data set except for the label assignments. A classification model is built on each of these data sets and a test statistic calculated, e.g., classification accuracy. The resulting set of values—one for each permutation—can be used to provide a distribution for the statistic under the null hypothesis that the attributes in the data set have no relationship with the class labels. As will be described in [Section 10.3.1](#), this approach can be used to test how likely is it to achieve the classification performance of a learned classifier on a test set just by random chance. Although permuting the

labels is simple, it can result in inferior models of the null distribution. (See Bibliographic Notes.)

10.2.3 Resampling Instances

Ideally, we would like to have multiple samples from the underlying population of data instances so that we can assess the validity and generalizability of the models and patterns our data mining algorithms produce. One way to simulate such samples is to randomly sample instances from the original data to create synthetic collections of data instances—an approach called statistical resampling. For example, a common approach for generating new data sets is to use bootstrap sampling, where data instances are randomly selected with replacement such that the resultant data set is of the same size as the original set. For classification, an alternative to bootstrap sampling is k -fold cross-validation, where the data set is systematically split into subsets for k number of times. As we will see later in [Section 10.3.1](#), such statistical resampling approaches are used to compute distributions of measures of classification performance, such as accuracy, precision, and recall. Resampling approaches such as the bootstrap can also be used to estimate the distribution of the support of a frequent itemset. We can also use these distributions to produce confidence intervals for these measures.

10.2.4 Modeling the Distribution of the Test Statistic

Given multiple samples of data sets generated under the null hypothesis, we can compute the test statistic on every set of samples to obtain the null

distribution of the test statistic. This distribution can be used for providing estimates of the probabilities used in statistical testing procedures, such as p-values. One way to achieve this is to fit statistical models, e.g., the normal or the binomial distribution, on the test statistic values from the data sets generated under the null hypothesis. Alternatively, we can also make use of non-parametric approaches for computing p-values, given enough samples. For instance, we can count the fraction of times the test statistic generated under the null distribution exceeds (or takes “more extreme” values than) the test statistic of the observed result, and use this fraction as the p-value of the result.

10.3 Statistical Testing for Classification

There are a number of problems in classification that can be viewed from the perspective of statistical testing and thus can benefit from the techniques described previously in this chapter for avoiding false discoveries. In the following, we discuss some of these problems and the statistical testing approaches that can be used to address them. Note that approaches for comparing whether the performance of two models is significantly different is provided in [Section 3.9.2](#).

10.3.1 Evaluating Classification Performance

Suppose that a classifier applied to a test set shows an accuracy of $x\%$. In order to assess the validity of the classifier's results, it is important to understand how likely it is to obtain $x\%$ accuracy by random chance, i.e., when there is no relationship between the attributes in the data set and the class label. Also, if we choose a certain threshold for the classification accuracy to identify effective classifiers, then we would like to know how many times we can expect to falsely reject a good classifier that shows an accuracy lower than the threshold due to the natural variability in the data.

Such questions about the validity of a classifier's performance can be addressed by viewing this problem from the perspective of hypothesis testing

as follows. Consider a statistical testing setup where we learn a classifier on a training set and evaluate the learned classifier on a test set. The null hypothesis for this test is that the classifier is not able to learn a generalizable relationship between the attributes and the class labels from the given training set. The alternative hypothesis is that the classifier indeed learns a generalizable relationship between the attributes and the class labels from the training set. To evaluate whether an observed result belongs to the null hypothesis or the alternative hypothesis, we can use a measure of the classifier's performance on the test set, e.g., precision, recall, or accuracy, as the test statistic.

Randomization

In order to perform statistical testing using the above setup, we first need to generate new sample data sets under the null hypothesis that there are no non-random relationships between the attributes and class labels. This can be achieved by randomly permuting the class labels of the training data such that for every permutation of the labels, we produce a new training set where the attributes and class labels are unrelated to each other. We can then learn a classifier on every sample training set and apply the learned models on the test set to obtain a null distribution of the test statistic (classification performance). Then, for example, if we use accuracy as our test statistic, the observed value of accuracy for the model learned using original labels should be significantly higher than most or all of the accuracies generated by models learned over randomly permuted labels. However, note that a classifier may have a significant p-value but have an accuracy only slightly better than a random classifier, especially if the data set is large. Hence, it is important to take the effect size of the classifier (actual value of classification performance) into account along with information about its p-value.

Bootstrap and Cross-Validation

Another type of analysis relevant to predictive models, such as classification, is to model the distribution of various measures of classification performance. One way to estimate such distributions is to generate bootstrap samples from the labeled data (preserving the original labels) to create new training and test sets. The performance of a classification model trained and evaluated on a number of these bootstrapped data sets can then be used to generate a distribution for the measure of interest. Another way to create the alternative distribution would be to use the randomized cross-validation procedure (discussed in [Section 3.6.2](#)) where the process of randomly partitioning the labeled data into k folds is repeated multiple times.

Such resampling approaches can also help in estimating confidence intervals for measures of the true performance of the classifier trained over all possible instances. A confidence interval is an interval of parameter values in which an estimated parameter value is guaranteed to fall a certain percentage of times. The confidence level is the percentage of times the estimated parameter will fall within the interval. For example, given the distribution of a classifier's accuracy, we can estimate the interval of values that contains 95% of the distribution. This serves as the confidence interval of the classifier's true accuracy at the 95% confidence level. To quantify the inherent uncertainty in the result, confidence intervals are often reported along with point estimates of a model's output.

10.3.2 Binary Classification as Multiple Hypothesis Testing

The process of estimating the generalization performance of a binary classifier resembles the problem of multiple hypothesis testing discussed previously in [Section 10.1.2](#). In particular, every test instance belongs to the null hypothesis (negative class) or the alternative hypothesis (positive class). By applying a classification model on every test instance, we assign each instance to the positive or the negative class. The performance of a classification model on a set of results (results of classifying instances in a test set) can then be summarized by the familiar confusion matrix presented in [Table 10.1](#).

A unique aspect of binary classification that differentiates it from conventional problems of multiple hypothesis testing is the availability of ground truth labels on test instances. Hence, instead of making inferences using statistical assumptions (e.g., the distribution of the test statistic under the null and alternative hypothesis), we can directly compute error estimates for rejecting the null or alternative hypotheses using empirical methods, such as those presented in [Section 4.11.2](#). [Table 10.2](#) shows the correspondence between the error metrics used in statistical testing and evaluation measures used in classification problems.

Table 10.2. Correspondence between statistical testing concepts and classifier evaluation measures

Statistical Testing Concept	Classifier Evaluation Measure	Formula
Type I Error Rate, α	False Positive Rate	$\frac{FP}{FP + TN}$
Type II Error Rate, β	False Negative Rate	$\frac{FN}{TP + FN}$
Power, $1-\beta$	Recall	$\frac{TP}{TP + FN}$

While these error metrics can be readily computed with the help of labeled data, the reliability of such estimates depends on the accuracy of test labels which may not always be perfect. In such cases, it is important to quantify the uncertainty in the evaluation measures arising due to inaccuracies in the test labels. (See Bibliographic Notes for more details.) Further, when we apply a learned classification model on unlabeled instances, we can use statistical methods for quantifying the uncertainty in the classification outputs. For example, we can bootstrap the training set (as discussed in [Section 10.3.1](#)) to generate multiple classification models, and the distribution of their outputs on an unseen instance can be used to estimate the confidence interval of the output on that instance.

Although the above discussion was focused on assessing the quality of a classifier that produces binary outputs, statistical considerations can also be used to assess the quality of a classifier that produces real-valued outputs such as classification scores. The performance of a classifier across a range of score thresholds is generally analyzed with the help of Receiver Operating Characteristic curves, as discussed in [Section 4.11.4](#). The basic approach behind generating an ROC curve is to sort the predictions according to their score values and then plot the true positive rate and the false positive rate for every possible value of score threshold. Note that this approach bears some resemblance to the FDR controlling procedures described in [Section 10.1.3](#), where the top few ranking instances (with the lowest p-values) are labeled as positive by the classifier in order to maximize the FDR. However, in the presence of ground truth labels, we can empirically estimate measures of classification performance for different score thresholds without making use of any explicit statistical models or assumptions.

10.3.3 Multiple Hypothesis Testing in

Model Selection

The problem of multiple hypothesis testing plays a major role in the process of model selection, where even if a more complex model shows better performance than a simpler model, the difference in their performances may not be statistically significant. Specifically, from a statistical perspective, a model with a higher complexity offers a larger number of possible solutions that a learning algorithm can choose from, for a given classification problem. For example, having a larger number of attributes provides a larger set of candidate splitting criteria that a decision tree learning algorithm can choose to best fit the training data. However, when the training size is small and the number of candidate models are large, there is a higher chance of picking a spurious model. More generally, this version of the multiple hypothesis testing is known as **selective inference**. This problem arises in situations where the number of possible solutions for a given problem, such as building a predictive model, are numerous, but the number of tests to robustly determine the efficacy of a solution is quite small. Selective inference may lead to the model overfitting problem described in [Section 3.4](#).

How does the multiple comparison procedure relate to model overfitting? Many learning algorithms explore a set of independent alternatives, $\{\gamma_i\}$, and then choose an alternative, γ_{\max} , that maximizes a given criterion function. The algorithm will add γ_{\max} to the current model in order to improve its training error. This procedure is repeated until no further improvement is observed. As an example, during decision tree growing, multiple tests are performed to determine which attribute can best split the training data. The attribute that leads to the best split is chosen to extend the tree as long as the stopping criterion has not been satisfied.

Let T_0 be the initial decision tree and T_x be the new tree after inserting an internal node for attribute x . Consider the following stopping criterion for a decision tree classifier: x is added to the tree if the observed gain, $\Delta(T_0, T_x)$, is greater than some predefined threshold α . If there is only one attribute test condition to be evaluated, then we can avoid inserting spurious nodes by choosing a large enough value of α . However, in practice, there is more than one test condition available and the decision tree algorithm must choose the best splitting attribute x_{\max} from a set of candidates, $\{x_1, x_2, \dots, x_k\}$. The multiple comparison problem arises because the algorithm applies the following test, $\Delta(T_0, T_{x_{\max}}) > \alpha$ instead of $\Delta(T_0, T_x) > \alpha$, to decide whether a decision tree should be extended. Just as with the multiple stockbroker example, as the number of alternatives k increases, so does our chance of finding $\Delta(T_0, T_{x_{\max}}) > \alpha$. Unless the gain function Δ or threshold α is modified to account for k , the algorithm may inadvertently add spurious nodes with low predictive power to the tree, which leads to the model overfitting problem.

This effect becomes more pronounced when the number of training instances from which x_{\max} is chosen is small, because the variance of $\Delta(T_0, T_{x_{\max}})$ is higher when fewer training instances are available. As a result, the probability of finding $\Delta(T_0, T_{x_{\max}}) > \alpha$ increases when there are very few training instances. This often happens when the decision tree grows deeper, which in turn reduces the number of instances covered by the nodes and increases the likelihood of adding unnecessary nodes into the tree.

10.4 Statistical Testing for Association Analysis

Since problems in association analysis are usually unsupervised, i.e., we do not have access to ground truth labels to evaluate results, it is important to employ robust statistical testing approaches to ensure that the discovered results are statistically significant and not spurious. For example, in the discovery of frequent itemsets, we often use evaluation measures such as the support of an itemset to measure its interestingness. (The uncertainty in such evaluation measures can be quantified by using resampling methods, e.g., by bootstrapping the transactions and generating a distribution of the support of an itemset from the resulting data sets.) Given a suitable evaluation measure, we also need to specify a threshold on the measure to identify interesting patterns such as frequent itemsets. Although the choice of a relevant threshold is generally guided by domain considerations, it can also be informed with the help of statistical procedures, as we discuss in the following. To simplify this discussion, we assume that our transaction data set is represented as a sparse binary matrix, with 1's representing the presence of items and 0's representing their absence. (See [Section 2.1.2](#).)

Given a transaction data set, consider a result to be the discovery of a frequent k -itemset and the test statistic to be the support of the itemset or any other evaluation measure described in [Section 5.7](#). The null hypothesis for this result would be that the k items in an itemset are unrelated to each other. Given a collection of frequent itemsets, we could then apply multiple hypothesis testing methods such as the FWER or FDR controlling procedures to identify significant patterns with strongly associated items. However, the itemsets found by an association mining algorithm overlap in terms of the

items they contain. Hence, the multiple results in association analysis cannot be assumed to be independent of each other. For this reason, approaches such as the Bonferroni procedure may be overly conservative in calling a result significant, which leads to low power. Further, a transaction data set may have structure or characteristics, e.g., a subset of transactions containing a large number of items, which need to be accounted for when applying multiple hypothesis testing procedures.

Before we can apply statistical testing procedures for problems related to association analysis, we first need to estimate the distribution of the test statistic of an itemset under the null hypothesis of no association among the items. This can be done by either making use of statistical models or by performing randomization experiments. Both these categories of approaches are described in the following.

10.4.1 Using Statistical Models

Under the null hypothesis that the items are unrelated, we can model the support count of an itemset using statistical models of independence among items. For itemsets containing two independent items, we can use Fisher's exact test. For itemsets containing more than two items, we can use alternative tests of independence such as the chi-squared (χ^2) test. Both these approaches are illustrated in the following.

Using Fisher's Exact Test

Consider the problem of modeling the support count of a 2-itemset, $\{A, B\}$, under the null hypothesis that A and B occur independently of each other. We are given a data set with N transactions where A and B appear N_A and N_B

times, respectively. Assuming that A and B are independent, the probability of observing A and B together would then be given by

$$p_{AB} = p_A \times p_B = N_A N \times N_B N ,$$

where p_A and p_B are the probabilities of observing A and B individually, which are approximated by their support. The probability of not observing A and B together would then be equal to $(1 - p_{AB})$. Assuming that the N transactions are independent, we can consider modeling N_{AB} , the number of times A and B appear together, using the binomial distribution (introduced in [Section 10.1](#)) as follows:

$$P(N_{AB}=k) = (N k) (p_{AB})^k (1-p_{AB})^{N-k} .$$

However, the binomial distribution does not accurately model the support count of $\{A,B\}$ because it assigns, N_{AB} , the support count of $\{A,B\}$, a positive probability even when N_{AB} exceeds the individual support counts of A and B . More specifically, the binomial distribution represents the probability of observing an event (co-occurrence of A and B) when *sampling with replacement* with a fixed probability (p_{AB}). However, in reality, the probability of observing $\{A,B\}$ decreases if we have already sampled A and B a number of times, because the support counts of A and B are fixed.

Fisher's exact test was designed to handle the above situation where we perform *sampling without replacement* from a finite population of fixed size. This test is readily explained using the same terminology used in [Section 5.7](#), which dealt with the evaluation of association patterns. For easy reference, we reproduce the contingency table, [Table 5.6](#), which was used in that discussion. See [Table 10.3](#).

Table 10.3. A 2-way contingency table for variables A and B .

	B	B^-	
A	f_{11}	f_{10}	f_{1+}
A^-	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

We use the notation A^- (B^-) to indicate that A (B) is absent from a transaction. Each entry f_{ij} in this 2×2 table denotes a frequency count. For example, f_{11} is the number of times A and B appear together in the same transaction, while f_{01} is the number of transactions that contain B but not A . The row sum f_{1+} represents the support count for A , while the column sum f_{+1} represents the support count for B .

Note that if N , the number of transactions, and the supports of A (f_{1+}) and B (f_{+1}) are fixed, i.e., held constant, then f_{0+} and f_{+0} are fixed. This also implies that specifying the value for one of the entries, f_{11} , f_{10} , f_{01} , or f_{00} , completely specifies the rest of the entries in the table. In that case, Fisher's exact test gives us a simple formula for exactly computing the probability of any specific contingency table. Because of our intended application, we express the formula in terms of the support count, N_{AB} , for the 2-itemset $\{A,B\}$. Note that f_{11} is the observed support count of $\{A,B\}$.

$$P(N_{AB} = f_{11}) = (f_{1+} + f_{11})(f_{0+} + f_{+1} - f_{11})(N f_{+1}) \quad (10.4)$$

Example 10.9 (Fisher's Exact Test).

We illustrate the application of Fisher's exact test using the tea-coffee example described at the beginning of [Section 5.7.1](#). We are interested in modeling the null distribution of the support count of $\{\text{Tea}, \text{Coffee}\}$. As

described in [Section 5.7.1](#), the co-occurrence of *Tea* and *Coffee* can be summarized using the contingency table shown in [Table 10.4](#). We can see that the support count of *Coffee* is 800 and the support count of *Tea* is 200, out of a total of 1000 transactions.

Table 10.4. Beverage preferences among a group of 1000 people.

	<i>Coffee</i>	<i>Coffee</i> −	
<i>Tea</i>	150	50	200
<i>Tea</i> −	650	150	800
	800	200	1000

To model the null distribution of the support count of {*Tea,Coffee*} , we simply apply [Equation 10.4](#) from our discussion Fisher's exact test. This yields the following.

$$P(N_{AB} = f_{11}) = \frac{(200 f_{11})(800 800 - f_{11})(1000 800)}{1000!},$$

where N_{AB} is the support count of {*Tea, Coffee*}.

[Figure 10.9](#) shows a plot of the null distribution of the support count for {*Tea,Coffee*}. We can see that the largest probability for support count occurs when it is equal to 160. An intuitive explanation for this fact is that when *Tea* and *Coffee* are independent, the probability of observing *Tea* and *Coffee* together is equal to the product of their individual probabilities, i.e., $0.8 \times 0.2 = 0.16$. The expected support count of {*Tea,Coffee*} is thus equal to $0.16 \times 1000 = 160$. Support counts that are less than 160 indicate negative associations among items.

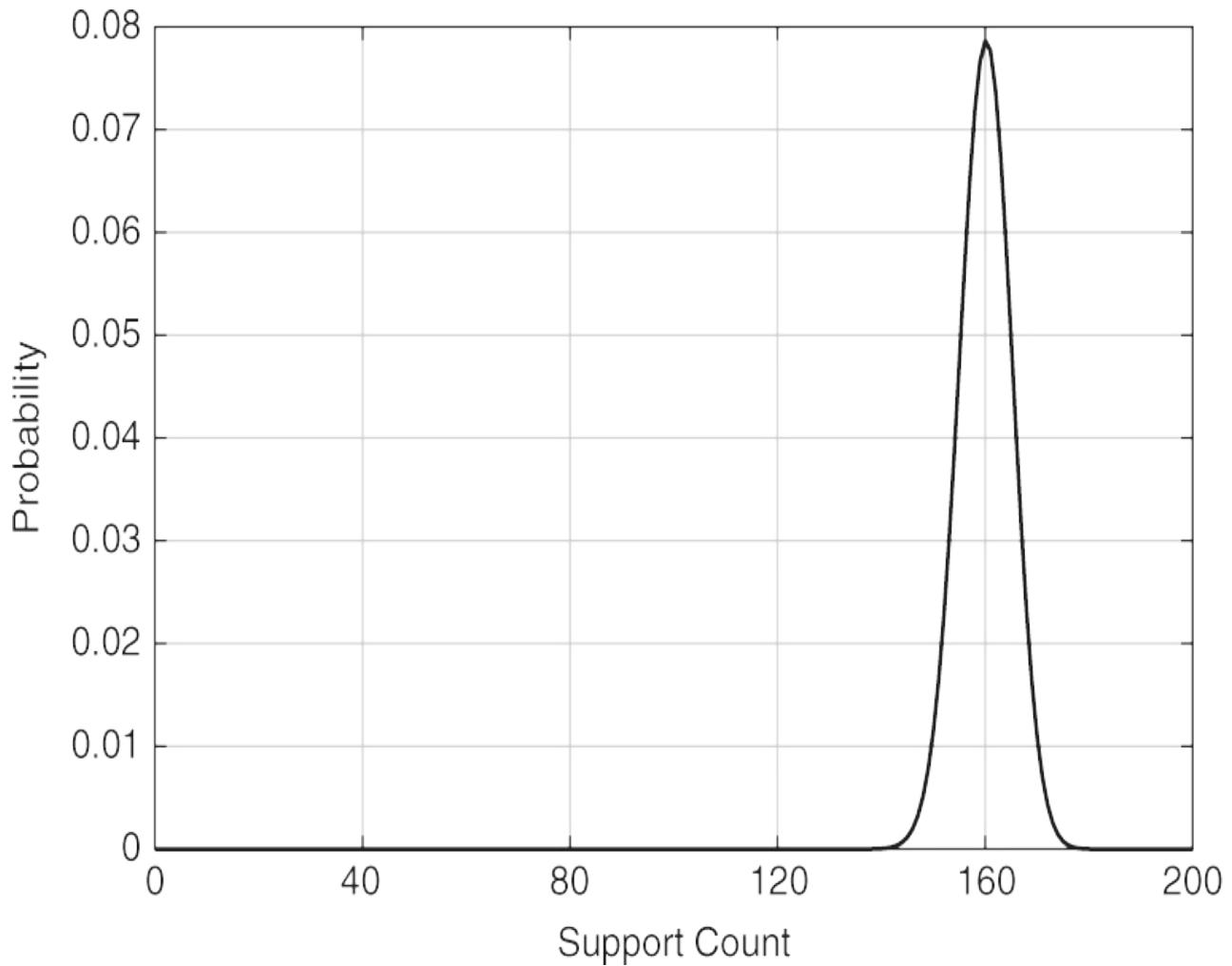


Figure 10.9.

Plot of the probability of support count given the independence of *Tea* and *Coffee*

Hence, the p-value of a support count of 150 can be calculated by summing the probabilities in the left tail of the null distribution for support count 150 and smaller. This yields a p-value of 0.032. This result is not conclusive, since a support count of 150 or less will occur roughly 3 times out of a 100, on average, if *Tea* and *Coffee* are independent. However, the low p-value tends to indicate that tea and coffee are related, albeit in a negative way, i.e., tea drinkers are less likely to drink coffee than those who don't drink tea. Note that this is just an example and does not necessarily reflect reality. Also note that this finding is consistent with our

previous analysis using alternative measures, such as the interest factor (lift measure), as described in [Section 5.7.1](#).

Although the discussion above was centered on the support measure, we can also model the null distribution of any other objective interestingness measure of a 2-itemset introduced in [Section 5.7](#), such as interest, odds ratio, cosine, or all-confidence. This is because all the entries of the contingency table can be uniquely determined by the support measure of the 2-itemset, given the number of transactions and the support counts of the two items. More specifically, the probabilities displayed in [Figure 10.9](#) are the probabilities of specific contingency tables corresponding to a specific value of support for the 2-itemset. For each of these tables, the values of any objective interestingness measure (of two items) can be calculated, and these values define the null distribution of the measure being considered. This approach can also be used to evaluate interestingness measures of association rules such as the confidence of $A \rightarrow B$, where A and B are itemsets.

Note that using Fisher's exact test is equivalent to using the hypergeometric distribution.

Using the Chi-Squared Test

The chi-squared (χ^2) test provides a generic but approximate approach for measuring the statistical independence among multiple items in an itemset. The basic idea behind the χ^2 test is to compute the expected value of every entry in a contingency table, such as the one shown in [Table 10.4](#), assuming that the items are statistically independent. The differences between the observed and expected values in the contingency table can then be used to compute a test statistic that follows the χ^2 distribution under the null hypothesis of no association between items.

Formally, consider a two-dimensional contingency table where the entry at the i th row and j th column is denoted by $O_{i,j}(i,j \in \{0,1\})$. (We use the notation $O_{i,j}$ instead of $f_{i,j}$ since the former is traditionally used to represent the “observed” value in discussions of the χ^2 statistic.) If the sum of all entries is equal to N , then we can compute the expected value at every entry as

$$E_{i,j} = N \times (\sum_i O_{i,j} / N) \times (\sum_j O_{i,j} / N). \quad (10.5)$$

This follows from the fact that the joint probability of observing independent events is equal to the product of the individual probabilities. When all items are statistically independent, $O_{i,j}$ would usually be close to $E_{i,j}$ for all values of i and j . Hence, the differences between $O_{i,j}$ and $E_{i,j}$ can be used to measure the deviation of the observed contingency table from the null hypothesis of no association. In particular, we can compute the following test statistic:

$$R = \sum_i \sum_j (O_{i,j} - E_{i,j})^2 / E_{i,j}. \quad (10.6)$$

Note that $R = 0$ only if $O_{i,j}$ and $E_{i,j}$ are equal for every value of i and j . It can be shown that the null distribution of R can be approximated by the χ^2 distribution with 1 degree of freedom when N is large. We can thus compute the p-value of an observed value of R using standard implementations of the χ^2 distribution.

While the above discussion was centered on the analysis of a two-dimensional contingency table involving two items, the χ^2 test can be readily extended to multi-dimensional contingency tables involving more than two items. For example, given a k -itemset $X = \{i_1, i_2, \dots, i_k\}$, we can construct a k -dimensional contingency table with observed entries represented as O_{i_1, i_2, \dots, i_k} ($i_1, i_2, \dots, i_k \in \{0, 1\}$). The expected values of the contingency table and the test statistic R could then be computed as follows

$$E_{i1,i2,\dots,ik} = N \times \prod_{j=1}^k (\sum_{ij} O_{i1,i2,\dots,ikN}). \quad (10.7)$$

$$R = \sum_{i1} \sum_{i2} \dots \sum_{ik} (O_{i1,i2,\dots,ik} - E_{i1,i2,\dots,ik})^2 / E_{i1,i2,\dots,ik}. \quad (10.8)$$

Under the null hypothesis that all k items in the itemset X are statistically independent, the distribution of R can again be approximated by a χ^2 distribution. However, the general formula for the degrees of freedom is $df = (\text{number of rows} - 1) \times (\text{number of columns} - 1)$. Thus, if we have a 4 by 3 contingency table, then $df = (4 - 1) \times (3 - 1) = 6$.

10.4.2 Using Randomization Methods

When it is difficult to model the null distribution of itemsets using statistical models, an alternative approach is to generate synthetic transaction data sets under the null hypothesis of no association among the items, with the same number of items and transactions as the original data. This involves randomly permuting the rows or columns in the original data such that the items in the resultant data are unrelated to each other. As discussed in [Section 10.2.1](#), we must ensure while randomizing the attributes that the resultant data sets are similar to the original data set in all respects except for the desired effect we are interested in evaluating, which is the association among items.

A basic structure we would like to preserve in the synthetic data sets is the support of every item in the original data. In particular, every item should appear in the same number of transactions in the synthetic data sets as in the original data set. One way to preserve this support structure of items is to randomly permute the entries in each column of the original data set independently of the other columns. This ensures that the items have the same support in the synthetically generated data sets but are independent of

each other. However, this may violate a different property of the original data that we would like to preserve, which is the length of every transaction (number of items in a transaction). This property can be preserved by randomly shuffling the rows, i.e., the row sums are preserved. However, a drawback of this approach is that the support of every item in the resultant data set may be different than the support of items in the original data set.

A randomization approach that can preserve both the supports and the transaction lengths of the original data is **swap randomization**. The basic idea is to pick a pair of ones in the original data set from two different rows and columns, say at (row k , column i) and (row l , column j), where $k \neq l$ and $i \neq j$. (See left table in [Figure 10.10](#).) These two entries define the diagonal of a rectangle of values in the binary transaction matrix. If the entries at opposite corners of the rectangle, i.e., (row k , column j) and (row l , column i), are zeros, then we can swap these zeros with the ones, as shown in [Figure 10.10](#). Note that by performing this swap, both the row sums and column sums are preserved while the association with other items is broken. This process continues until it is likely that the data set is significantly different from the original one. (An appropriate threshold for the number of swaps needs to be determined depending on the size and nature of the original data set.)

	col i	...	col j	
:	:	...	:	
row k	1	...	0	
:	:	...	:	
row l	0	...	1	
:	:	...	:	

⇒

	col i	...	col j	
:	:	...	:	
row k	0	...	1	
:	:	...	:	
row l	1	...	0	
:	:	...	:	

Figure 10.10.

Illustration of a swap for swap randomization.

Swap randomization has been shown to preserve the properties of transaction data sets more accurately than the other approaches mentioned. However, it is very computationally intensive, particularly for larger data sets, which can limit its application. Furthermore, apart from the support of items and transaction lengths, there may be other types of structure in the transaction data that swap randomization may not be able to preserve. For instance, there may be some known correlations among the items (due to domain considerations) that we would like to retain in the synthetic data sets while breaking the correlations among other items. A good example are data sets that record the presence or absence of a genetic variation at various locations on the genome (items) across multiple subjects (transactions). Items representing locations that are close on the genetic sequence are known to be highly correlated. This local structure of correlation may be lost in the synthetic data sets if we treat each column identically while randomizing. What is needed in this case is to keep the local correlation but to break correlation of areas that are further away.

After constructing synthetic data sets pertaining to the null hypothesis, we can generate the null distribution of the support of an itemset by observing its support in the synthetic data sets. This procedure can help in deciding support thresholds using statistical considerations so that the discovered frequent itemsets are statistically significant.

10.5 Statistical Testing for Cluster Analysis

The goodness of a clustering is typically evaluated with the help of cluster validity measures that either capture the cohesion or separation of clusters, such as the sum of squared errors (SSE), or make use of external labels such as entropy. In some cases, the minimum and maximum values of measures have intuitive interpretations that can be used to examine the goodness of a clustering. For instance, if we are given the true class labels of instances and we want our clustering to reflect the class structure, then a purity of 0 is bad, while a purity of 1 is good. Likewise, an entropy of 0 is good, as is an SSE of 0. However, in many cases, we are given intermediate values of cluster validity measures which are difficult to interpret directly without the help of domain considerations.

Statistical testing procedures provide a useful way of measuring the significance of a discovered clustering. In particular, we can consider the null hypothesis that there is no cluster structure among the instances and the clustering algorithm is producing a random partitioning of the data. The approach is to use the cluster validity measure as a test statistic. The distribution of that test statistic under the assumption that the data has no clustering structure is the null distribution. We can then test whether the validity measure actually observed for the data is significant. In the following, we consider two general cases: (1) the test statistic is an internal clustering validity index computed for unlabeled data, such as SSE or the silhouette coefficient, or (2) the test statistic is an external index, i.e., the cluster labels are to be compared against class labels, such as entropy or purity. These cluster validity measures are described in [Section 7.5](#).

10.5.1 Generating a Null Distribution for Internal Indices

Internal indices measure the goodness of a cluster only by reference to the data itself—see [Section 7.5.2](#). Furthermore, often the clustering is driven by an objective function, and in those cases, the measure of a clustering's goodness is provided by the objective function. Thus, most of the time, statistical evaluation of a clustering is not performed.

Another reason that such an evaluation is not performed is the difficulty in generating a null distribution. In particular, to get a meaningful null distribution for determining cluster structure, we need to create data with similar overall properties and characteristics as the data we have except that it has no cluster structure. But this can be difficult since data often has a complex structure, e.g., the dependencies among observations in time series data. Nonetheless, statistical testing can be useful if the difficulties can be overcome. We present a simple example to illustrate the approach.

Example 10.10 (Significance of SSE).

This example is based on K-means and the SSE. Suppose that we want a measure of how the well-separated clusters of [Figure 10.11a](#) compare with respect to random data. We generate many random (uniformly distributed) sets of 100 points having the same range of values along the two dimensions as the points in the three clusters, find three clusters in each data set using K-means, and accumulate the distribution of SSE values for these clusterings. By using this distribution of the SSE values, we can then estimate the probability of the SSE value for the original clusters. [Figure 10.11b](#) shows the histogram of the SSE from 500

random runs. The lowest SSE in the histogram is 0.0173. For the three clusters of [Figure 10.11a](#), the SSE is 0.0050. We could therefore conservatively claim that there is less than a 1% chance that a clustering such as that of [Figure 10.11a](#) could occur by chance.

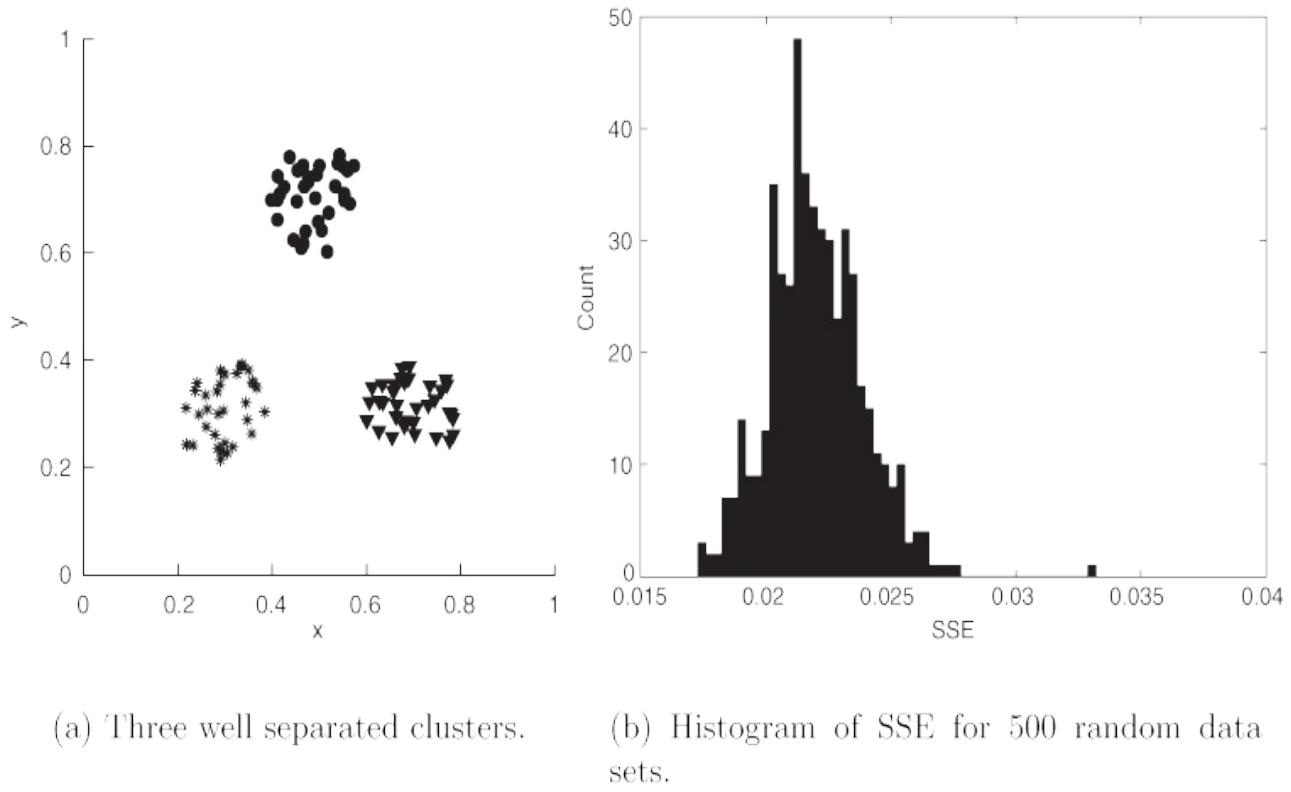


Figure 10.11.

Using randomization to evaluate the p-value for a clustering.

In the previous example, it was relatively straightforward to use randomization to evaluate the statistical significance of an internal cluster validity measure. In practice, domain evaluation is usually more important. For instance, a document clustering scheme could be evaluated by looking at the documents and judging whether the clusters make sense. More generally, a domain expert would evaluate the cluster for suitability to a desired application. Nonetheless, a statistical evaluation of clustering is sometimes necessary. A reference to an example for climate time series is provided in the Bibliographic Notes.

10.5.2 Generating a Null Distribution for External Indices

If external labels are used for evaluation, then a clustering is evaluated using a measure such as entropy or the Rand statistic—see [Section 7.5.7](#) which assesses how closely the cluster structure, as reflected in the cluster labels, matches the class labels. Some of these measures can be modeled with a statistical distribution, e.g., the adjusted Rand index, which is based on the multivariate hypergeometric distribution. If a measure has a well-known distribution, then this distribution can be used to compute a p-value.

However, randomization can also be used to generate a null distribution in this case, as follows.

1. Generate M randomized sets of labels, $L_1, \dots, L_i, \dots, L_M$
2. For each randomized set of labels, compute the value of the external index. Let m_i be the value of the external index obtained for the i th randomization. Let m_0 be the value of the external index for the original set of labels.
3. Assuming that a larger value of the external index is more desirable, define the p-value of m_0 to be the fraction of m_i for which $m_i > m_0$.

$$p\text{-value}(m_0) = |\{m_i : m_i > m_0\}|M \quad (10.9)$$

As with the case of unsupervised evaluation of a clustering, domain significance often assumes a dominant role. For example, consider clustering new articles into distinct groups as in [Example 7.15](#), where the articles belong to the classes: Entertainment, Financial, Foreign, Metro, National, and Sports. If we have the same number of clusters as the number of classes of

news articles, then an ideal clustering would have two characteristics. First, every cluster would contain *only* documents from one class, i.e., it would be pure. Second, every cluster would contain *all* of the documents from a particular class. An actual clustering of documents can be statistically significant, but still be quite poor in terms of purity and/or containing all the documents of a particular document class. Sometimes, such situations are still of interest, as we describe next.

10.5.3 Enrichment

In some cases involving labeled data, the goal of evaluating clusters is to find clusters that have more instances of a particular class than would be expected for a random clustering. When a cluster has more than the expected number of instances of a specific class, we say that the cluster is enriched in that class. This approach is commonly used in the analysis of bioinformatics data, such as gene expression data, but is applicable in many other areas as well. Furthermore, this approach can be used for any collection of groups, not just those created by clustering. We illustrate this approach with a simple example.

Example 10.11 (Enrichment of Neighborhoods of a City in Terms of Income Levels).

Assume that in a particular city there are 10 distinct neighborhoods, which correspond to clusters in our problem. Overall, there are 10,000 people in the city. Further, assume that there are 3 income levels, *Poor* (30%), *Medium* (50%), and *Wealthy* (20%). Finally, assume that one of the neighborhoods has 1,000 residents, 23% of whom fall into the *Wealthy* category. The question is whether this neighborhood has more wealthy

people than expected by random chance. The contingency table for this example is shown in [Table 10.5](#). We can analyze this table by using Fisher's exact test. (See [Example 10.9](#) in [Section 10.4.1](#).)

Table 10.5. Beverage preferences among a group of 1000 people.

	<i>In Neighborhood</i>	<i>In Neighborhood</i> ⁻	
<i>Wealthy</i>	230	1770	2,000
<i>Wealthy</i> ⁻	770	7,230	8,000
	20,000	1,000	10,000

Using Fisher's exact test, we find that the p-value for this result is 0.0076. This would seem to indicate that more wealthy people live in this neighborhood than would be expected by random chance at a significance level of 1%. However, several points need to be made. First, we may very well be testing every group against every neighborhood to look for enrichment. Thus, there would be 30 tests overall and the p-values should be adjusted for multiple comparisons. For instance, if we use the Bonferroni procedure, 0.0076 would not be a significant result since the significance threshold is now $0.01/30=0.0003$. Also, the odds ratio for this contingency table is only 1.22. Hence, even if the difference is significant, the actual magnitude of the difference doesn't seem very large, i.e., very far from an odds ratio of 1. In addition, note that multiplying all the entries of the table by 10 will greatly decrease the p-value ($\approx 10^{-9}$), but the odds ratio will remain the same. Despite these issues, enrichment can be a valuable tool and has yielded useful results for a variety of applications.

10.6 Statistical Testing for Anomaly Detection

Anomaly detection algorithms typically produce outputs in the form of class labels (when a classification model is trained over labeled anomalies) or anomaly scores. Statistical considerations can be used to ensure the validity of both these types of outputs as described in the following.

Supervised Anomaly Detection

If we have access to labeled anomalous instances, the problem of anomaly detection can be converted to a binary classification problem, where the negative class corresponds to the normal data instances, while the positive class corresponds to the anomalous instances. Statistical testing procedures discussed in [Section 10.3](#) for classification are directly relevant for avoiding false discoveries in supervised anomaly detection, albeit with the additional challenges of building a model for imbalanced classes (See [Section 4.11](#).) In particular, we need to ensure that the classification error metric used during statistical testing is sensitive to the imbalance among the classes and gives enough emphasis to the errors related to the rare anomaly class (false positives and false negatives). After learning a valid classification model, we can also use statistical methods to capture the uncertainty in the outputs of the model on unseen instances. For example, we can use resampling approaches such as the bootstrapping technique to learn multiple classification models from the training set, and the distribution of their labels produced on an unseen instance can be used to estimate confidence intervals of the true class label of the instance.

Unsupervised Anomaly Detection

Most unsupervised anomaly detection approaches produce an anomaly score on data instances to indicate how anomalous an instance is with respect to the normal class. It is then important to decide a suitable threshold on the anomaly score to identify instances that are significantly anomalous and hence are worthy of further investigation. The choice of a threshold is generally specified by the user based on domain considerations on what is acceptable as a significant departure from the normal behavior. Such decisions can also be reinforced with the help of statistical testing methods.

In particular, from a statistical perspective, we can consider every instance to be a result and its anomaly score to be the test statistic. The null hypothesis is that the instance belongs to the normal class while the alternative hypothesis is that the instance is significantly different from other points from the normal class and hence is an anomaly. Hence, given the null distribution of the anomaly score, we can compute the p-value of every result and use this information to determine statistically significant anomalies.

A prime requirement for performing statistical testing for anomaly detection is to obtain the distribution of anomaly scores for instances that belong to the normal class, as this is the null distribution. If the anomaly detection approach is based on statistical techniques (see [Section 9.3](#)), we have access to a statistical model for estimating the distribution of the normal class. In other cases, we can use randomization methods to generate synthetic data sets where the instances only belong to the normal class. For example, if it is possible to construct a model of the data without anomalies, then this model can be used to generate multiple samples of the data, and in turn, those samples can be used to create a distribution of the anomaly scores for instances that are normal. Unfortunately, just as for generating synthetic data for clustering, there is usually no easy way to construct random data sets that

look similar to the original data in all respects except that they contain only normal instances.

If anomaly detection is to be useful, however, then at some point, the results of the anomaly detection, particularly the top ranking anomalies, need to be evaluated by domain experts to assess the performance of the algorithm. If the anomalies produced by the algorithm do not agree with the expert assessment, this does not necessarily mean that the algorithm is not performing well. Instead, it may just mean that the definition of an anomaly being used by the expert and the algorithm differ. For instance, the expert may view certain aspects of the data as irrelevant, but the algorithm may be treating them as important. In such cases, these aspects of the data can be deemphasized to help refine the statistical testing procedures. Alternatively, there may be new types of anomalies that the expert is unfamiliar with, since anomalies are, by their very nature, supposed to be surprising.

Base Rate Fallacy

Consider an anomaly detection system that can accurately detect 99.9% of the fraudulent credit card transactions with a false alarm rate of only 0.01%. If a transaction is flagged as an anomaly by the system, how likely it is to be genuinely fraudulent? A common misconception is that the majority of the detected anomalies are fraudulent transactions given the high detection rate and low false alarm rate of the system. However, this can be misleading if the skew of the data is not taken into consideration. This problem is also known as *base rate fallacy* or *base rate neglect*.

To illustrate the problem, consider the contingency table shown in [Table 10.6](#). Let d be the detection rate (i.e., true positive rate) of the system and f be its false alarm rate, or to be more specific

Table 10.6. Contingency table for an anomaly detection system with detection rate d and false alarm rate f .

	Alarm	No Alarm	
Fraud	$d\alpha N$	$(1-d)\alpha N$	αN
No Fraud	$f(1-\alpha)N$	$(1-f)(1-\alpha)N$	$(1-\alpha)N$
	$d\alpha N + f(1-\alpha)N$	$(1-d)\alpha N + (1-f)(1-\alpha)N$	N

$$P(\text{Alarm}|\text{Fraud})=d \text{ and } P(\text{Alarm}|\text{Not Fraud})=f.$$

Our goal is to calculate the precision of the system, i.e., $P(\text{Fraud}|\text{Alarm})$. If the precision is high, then the majority of the alarms are indeed triggered by fraudulent transactions. Based on the information given in [Table 10.6](#), the precision of the system can be calculated as follows:

$$\text{Precision} = \frac{d\alpha N}{d\alpha N + f(1-\alpha)N} = \frac{d\alpha}{d\alpha + f(1-\alpha)}, \quad (10.10)$$

where α is the percentage of fraudulent transactions in the data. Since $d=0.999$ and $f=0.0001$, the precision of the system is

$$\text{Precision} = \frac{0.999\alpha}{0.999\alpha + 0.0001} = \frac{0.9989\alpha}{0.999\alpha + 0.0001} \quad (10.11)$$

If the data is not skewed, e.g., when $\alpha=0.5$, then its precision would be very high, 0.9999, so we can trust that the majority of the flagged transactions are fraudulent. However, if the data is highly skewed, e.g., when $\alpha=2\times10^{-5}$ (one in fifty thousand transactions), then the precision is only 0.167, which means that only about one in six alarms is a true anomalies.

The preceding example illustrates the importance of considering skewness of the data when choosing an appropriate anomaly detection system for a given application. If the event of interest occurs rarely, say, one in fifty thousand of the population, then even a system with 99.9% detection rate and 0.01% false alarm rate can still make 5 mistakes for every 6 anomalies flagged by the system. The precision of the system degrades significantly as the percentage of skewness in the data increases. The crux of this problem lies in the fact that detection rate and false alarm rate are metrics that are not sensitive to skewness in the class distribution, a problem that was first alluded to in [Section 4.11](#) during our discussion on the class imbalanced problem. The lesson here is that any evaluation of an anomaly detection system must take into account the degree of skewness in the data before deploying the system into practice.

10.7 Bibliographic Notes

Recently, there has been a growing body of literature that is concerned with the validity and reproducibility of research results. Perhaps the most well-known work in that area is the paper by Ioannidis [721], which asserts that most published research findings are false. There have been various critiques of this work, e.g., see Goodman and Greenland [717] and Ioannidis rebuttal [716, 722]. Regardless, concern about the validity and reproducibility of results has only continued to expand. A paper by Simmons et al. [742] states that almost any effect in psychology can be presented as statistically significant given current practice. The paper also suggests recommended changes in research practice and article review. A Nature survey by Baker [697] reported that more than 70% of researchers have tried and failed to replicate other researchers' results, and 50% have failed to replicate their own results. On a more positive note, Jager and Leek [724] looked at published medical research and although they identified a need for improvements, concluded that "our analysis suggests that the medical literature remains a reliable record of scientific progress." The recent book by Nate Silver [741] has discussed a number of predictive failures in various areas, including baseball, politics, and economics. Although numerous other studies and references can be cited in a number of areas, the key point is that there is a widespread perception, backed by a fair amount of evidence, that many current data analyses are not trustworthy and that there are various steps that can be taken to improve the situation [699, 723, 729]. Although this chapter has focused on statistical issues, many of the changes advocated, e.g., by Ioannidis in his original paper, are not statistical in nature.

The notion of significance testing was introduced by the prominent statistician Ronald Fisher [710, 734]. In response to perceived shortcomings, Neyman

and Pearson [735, 736] introduced hypothesis testing. The two approaches have often been merged in an approach known as null hypothesis statistical testing (NHST) [731], which has been the source of many problems [712, 720]. A number of p-value misconceptions are summarized in various recent papers, for example, those by Goodman [715], Nuzzo [737], and Gelman [711]. The American Statistical Association has recently issued a statement on p-values [751]. Papers that describe the Bayesian approach, as exemplified by the Bayes factor and prior odds, are Kass and Raftery [727] and Goodman and Sander [716]. A recent paper by Benjamin and large number of other prominent statisticians [699], uses such an approach to argue that 0.005, instead of 0.05, should be the default p-value threshold for statistical significance. More generally, the misinterpretation and misuse of p-values is not the only problem as some have noted [730]. Note that both Fisher's significance testing and the Neyman-Pearson hypothesis testing approaches were designed with statistically designed experiments in mind, but are often, perhaps mostly, applied to observational data. Indeed, most data being analyzed nowadays is observational data.

The seminal paper for the false discovery rate is by Benjamini and Hochberg [701]. The positive false discovery rate was proposed by Storey [743–745]. Efron has advocated the use of the local false discovery rate [704–707]. The work of Efron, Storey, Tibshirani, and others has been applied in a software package for analyzing microarray data: SAM: Significance Analysis of Microarrays [707, 746, 750]. More generally, most mathematical and statistical software has packages for computing FDR. In particular, see the FDRtool in R by Strimmer [748, 749] or the q-value routine [698, 747], which is available in Bioconductor, a well-known R package. A recent survey of past and current work in multiple hypothesis testing (multiple comparison) is given by Benjamini [700].

As discussed in [Section 10.2](#), resampling approaches, especially those based on the randomization / permutation and the bootstrap / cross-validation, are the main approach to modeling the null distribution or the distributions of evaluation metrics, and thus, computing evaluation measures of interest, such as p-values, false discovery rates, and confidence intervals. Discussion and references to the bootstrap and cross-validation are provided in the Bibliographic Notes of [Chapter 3](#). General resources on permutation / randomization include books by Edgington and [Onghena \[703\]](#), [Good \[714\]](#), and [Pesarin and Luigi \[740\]](#), as well as the articles by [Collingridge \[702\]](#), [Ernst \[709\]](#) and [Welch \[756\]](#). Although such techniques are widely used, there are limitations, such as those discussed in some detail by [Efron \[705\]](#). In this paper, Efron describes a Bayesian approach for estimating an empirical null distribution and using it to compute a “local” false discovery rate that is more accurate than approaches using a null distribution based on randomization or theoretical approaches.

As we have seen in the application specific sections, different areas of data analysis tend to use approaches specific to their problem. The permutation (randomization) of class labels described in [Section 10.3.1](#) is a straightforward and well-known technique in classification. The paper by Ojala and [Garigga \[738\]](#) examines this approach in more depth and presents an alternative randomization approach that can help identify, for a given data set, whether dependency among features is important in the classification performance. The paper by [Jensen and Cohen \[726\]](#) is a relevant reference for the discussion of multiple hypothesis testing in model selection. Clustering has relatively little work in terms of statistical validation since most users rely on measures of clustering goodness to evaluate outcomes. However, some useful resources are [Chapter 4](#) of Jain and Dubes’ clustering book [\[725\]](#) and the recent survey of clustering validity measures by Xiong and [Li. \[757\]](#). The swap randomization approach was introduced into association analysis by Gionis et al. [\[713\]](#). This paper has a number of references that trace the

origin of this approach in other areas, as well as references to other papers for the assessment of association patterns. This work was extended to real-valued matrices by Ojala et al. [739]. Another important resource for statistically sound association pattern discovery is the work of Webb [752–755]. Hämäläinen and Webb taught a tutorial in KDD 2014, Statistically Sound Pattern Discovery. Relevant publications by Hämäläinen include [719] and [718].

The design of experiments to reduce variability and increase power is a core component of statistics. There are a number of general books on the topic, e.g., the one by **Montgomery** [732], but many more specialized treatments of the topic are available for various domains. In recent years, A/B testing has emerged as a common tool of companies for comparing two alternatives, e.g., two web pages. A recent paper by **Kohavi et al.** [728] provides a survey and practical guide to A/B testing and some of its variants.

Much of the material presented in [sections 10.1](#) and [10.2](#) is covered in various statistics books and articles, many of which were mentioned previously. Additional reference material for significance and hypothesis testing can be found in introductory texts, although as mentioned above, these two approaches are not always clearly distinguished. The use of hypothesis testing is widespread in a number of domains, e.g., medicine, since the approach allow investigators to determine how many samples will be needed to achieve certain target values for the Type I error, power, and effect size. See, for example, **Ellis** [708] and **Murphy et al.** [733].

Bibliography

- [697] M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454, 2016.
- [698] D. Bass, A. Dabney, and D. Robinson. qvalue: Q-value estimation for false discovery rate control. R package, 2012.
- [699] D. J. Benjamin, J. Berger, M. Johannesson, B. A. Nosek, E.-J. Wagenmakers, R. Berk, K. Bollen, B. Brembs, L. Brown, C. Camerer, et al. Redefine statistical significance. *PsyArXiv*, 2017.
- [700] Y. Benjamini. Simultaneous and selective inference: current successes and future challenges. *Biometrical Journal*, 52(6):708–721, 2010.
- [701] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pages 289–300, 1995.
- [702] D. S. Collingridge. A primer on quantitized data analysis and permutation testing. *Journal of Mixed Methods Research*, 7(1):81–97, 2013.
- [703] E. Edgington and P. Onghena. *Randomization tests*. CRC Press, 2007.

- [704] B. Efron. *Local false discovery rates*. Division of Biostatistics, Stanford University, 2005.
- [705] B. Efron. Large-scale simultaneous hypothesis testing. *Journal of the American Statistical Association*, 2012.
- [706] B. Efron et al. Microarrays, empirical Bayes and the two-groups model. *Statistical science*, 23(1):1–22, 2008.
- [707] B. Efron, R. Tibshirani, J. D. Storey, and V. Tusher. Empirical Bayes analysis of a microarray experiment. *Journal of the American statistical association*, 96(456):1151– 1160, 2001.
- [708] P. D. Ellis. *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results*. Cambridge University Press, 2010.
- [709] M. D. Ernst et al. Permutation methods: a basis for exact inference. *Statistical Science*, 19(4):676–685, 2004.
- [710] R. A. Fisher. Statistical methods for research workers. In *Breakthroughs in Statistics*, pages 66–70. Springer, 1992 (originally, 1925).
- [711] A. Gelman. Commentary: P values and statistical practice. *Epidemiology*, 24(1):69–72, 2013.

- [712] G. Gigerenzer. Mindless statistics. *The Journal of Socio-Economics*, 33(5):587–606, 2004.
- [713] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(3):14, 2007.
- [714] P. Good. *Permutation tests: a practical guide to resampling methods for testing hypotheses*. Springer Science & Business Media, 2013.
- [715] S. Goodman. A dirty dozen: twelve p-value misconceptions. In *Seminars in hematology*, volume 45(13), pages 135–140. Elsevier, 2008.
- [716] S. Goodman and S. Greenland. Assessing the Unreliability of the Medical Literature: A response to Why Most Published Research Findings are False. *bepress*, 2007.
- [717] S. Goodman and S. Greenland. Why most published research findings are false: problems in the analysis. *PLoS Med*, 4(4):e168, 2007.
- [718] W. Härmäläinen. *Efficient search for statistically significant dependency rules in binary data*. PhD Thesis, Department of Computer Science, University of Helsinki, 2010.
- [719] W. Härmäläinen. Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. *Knowledge and information systems*, 32(2):383–414, 2012.

- [720] R. Hubbard. Alphabet Soup: Blurring the Distinctions Between α s and β 's in Psychological Research. *Theory & Psychology*, 14(3):295–327, 2004.
- [721] J. P. Ioannidis. Why most published research findings are false. *PLoS Med*, 2(8):e124, 2005.
- [722] J. P. Ioannidis. Why most published research findings are false: author's reply to Goodman and Greenland. *PLoS medicine*, 4(6):e215, 2007.
- [723] J. P. Ioannidis. How to make more published research true. *PLoS medicine*, 11(10): e1001747, 2014.
- [724] L. R. Jager and J. T. Leek. An estimate of the science-wise false discovery rate and application to the top medical literature. *Biostatistics*, 15(1):1–12, 2013.
- [725] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series. Prentice Hall, March 1988.
- [726] D. Jensen and P. R. Cohen. Multiple Comparisons in Induction Algorithms. *Machine Learning*, 38(3):309–338, March 2000.
- [727] R. E. Kass and A. E. Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.

- [728] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1168–1176. ACM, 2013.
- [729] D. Lakens, F. G. Adolfi, C. Albers, F. Anvari, M. A. Apps, S. E. Argamon, M. A. van Assen, T. Baguley, R. Becker, S. D. Benning, et al. Justify Your Alpha: A Response to Redefine Statistical Significance. *PsyArXiv*, 2017.
- [730] J. T. Leek and R. D. Peng. Statistics: P values are just the tip of the iceberg. *Nature*, 520(7549):612, 2015.
- [731] E. F. Lindquist. *Statistical analysis in educational research*. Houghton Mifflin, 1940.
- [732] D. C. Montgomery. *Design and analysis of experiments*. John Wiley & Sons, 2017.
- [733] K. R. Murphy, B. Myors, and A. Wolach. *Statistical power analysis: A simple and general model for traditional and modern hypothesis tests*. Routledge, 2014.
- [734] J. Neyman. RA Fisher (1890–1962): An Appreciation. *Science*, 156(3781):1456–1460, 1967.
- [735] J. Neyman and E. S. Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference: Part I. *Biometrika*, pages

175–240, 1928.

- [736] J. Neyman and E. S. Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference: Part II. *Biometrika*, pages 263–294, 1928.
- [737] R. Nuzzo. Scientific method: Statistical errors. *Nature News*, Feb. 12 2014.
- [738] M. Ojala and G. C. Garriga. Permutation tests for studying classifier performance. *Journal of Machine Learning Research*, 11(Jun):1833–1863, 2010.
- [739] M. Ojala, N. Vuokko, A. Kallio, N. Haiminen, and H. Mannila. Randomization of real-valued matrices for assessing the significance of data mining results. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 494–505. SIAM, 2008.
- [740] F. Pesarin and L. Salmaso. *Permutation tests for complex data: theory, applications and software*. John Wiley & Sons, 2010.
- [741] N. Silver. *The signal and the noise: Why so many predictions fail—but some don’t*. Penguin, 2012.
- [742] J. P. Simmons, L. D. Nelson, and U. Simonsohn. False-positive psychology undisclosed flexibility in data collection and analysis allows

presenting anything as significant. *Psychological science*, page 0956797611417632, 2011.

[743] J. D. Storey. A direct approach to false discovery rates. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(3):479–498, 2002.

[744] J. D. Storey. The positive false discovery rate: a Bayesian interpretation and the q-value. *Annals of statistics*, pages 2013–2035, 2003.

[745] J. D. Storey, J. E. Taylor, and D. Siegmund. Strong control, conservative point estimation and simultaneous conservative consistency of false discovery rates: a unified approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(1):187–205, 2004.

[746] J. D. Storey and R. Tibshirani. SAM: thresholding and false discovery rates for detecting differential gene expression in DNA microarrays. In *The analysis of gene expression data*, pages 272–290. Springer, 2003.

[747] J. D. Storey, W. Xiao, J. T. Leek, R. G. Tompkins, and R. W. Davis. Significance analysis of time course microarray experiments. *Proceedings of the National Academy of Sciences of the United States of America*, 102(36):12837–12842, 2005.

[748] K. Strimmer. fdrtool: a versatile R package for estimating local and tail area-based false discovery rates. *Bioinformatics*, 24(12):1461–1462, 2008.

- [749] K. Strimmer. A unified approach to false discovery rate estimation. *BMC bioinformatics*, 9(1):303, 2008.
- [750] V. G. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences*, 98 (9):5116–5121, 2001.
- [751] R. L. Wasserstein and N. A. Lazar. The ASA’s statement on p-values: context, process, and purpose. *The American Statistician*, 2016.
- [752] G. I. Webb. Discovering significant patterns. *Machine Learning*, 68(1):1–33, 2007.
- [753] G. I. Webb. Layered critical values: a powerful direct-adjustment approach to discovering significant patterns. *Machine Learning*, 71(2):307–323, 2008.
- [754] G. I. Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):3, 2010.
- [755] G. I. Webb and J. Vreeken. Efficient discovery of the most interesting associations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(3):15, 2014.
- [756] W. J. Welch. Construction of permutation tests. *Journal of the American Statistical Association*, 85(411):693–698, 1990.

[757] H. Xiong and Z. Li. Clustering Validation Measures. In C. C. Aggarwal and C. K. Reddy, editors, *Data Clustering: Algorithms and Applications*, pages 571–605. Chapman & Hall/CRC, 2013.

10.8 Exercises

1. Statistical testing proceeds in a manner analogous to the mathematical proof technique, proof by contradiction, which proves a statement by assuming it is false and then deriving a contradiction. Compare and contrast statistical testing and proof by contradiction.
2. Which of the following are suitable null hypotheses. If not, explain why.
 - a. *Comparing two groups* Consider comparing the average blood pressure of a group of subjects, both before and after they are placed on a low salt diet. In this case, the null hypothesis is that a low salt diet does reduce blood pressure, i.e., that the average blood pressure of the subjects is the same before and after the change in diet.
 - b. *Classification* Assume there are two classes, labeled + and −, where we are most interested in the positive class, e.g., the presence of a disease. H_0 is the statement that the class of an object is negative, i.e., that the patient does not have the disease.
 - c. *Association Analysis* For frequent patterns, the null hypothesis is that the items are independent and thus, any pattern that we detect is spurious.
 - d. *Clustering* The null hypothesis is that there is cluster structure in the data beyond what might occur at random.
 - e. *Anomaly Detection* Our assumption, H_0 , is that an object is not anomalous.
3. Consider once again the coffee-tea example, presented in **Example 10.9** . The following two tables are the same as the one presented in

Example 10.9 except that each entry has been divided by 10 (left table) or multiplied by 10 (right table).

Table 10.7. Beverage preferences among a group of 100 people (left) and 10,000 people (right).

	Coffee	Coffee ⁻		
Tea	15	5	20	
Tea ⁻	65	15	80	
	80	20	100	

	Coffee	Coffee ⁻		
Tea	1500	500	2000	
Tea ⁻	6500	1500	8000	
	8000	2000	10000	

- Compute the p-value of the observed support count for each table, i.e., for 15 and 1500. What pattern do you observe as the sample size increases?
 - Compute the odds ratio and interest factor for the two contingency tables presented in this problem and the original table of **Example 10.9** . (See **Section 5.7.1** for definitions of these two measures.) What pattern do you observe?
 - The odds ratio and interest factor are measures of effect size. Are these two effect sizes significant from a practical point of view?
 - What would you conclude about the relationship between p-values and effect size for this situation?
4. Consider the different combinations of effect size and p-value applied to an experiment where we want to determine the efficacy of a new drug.
- effect size small, p-value small
 - | effect size small, p-value large

-) effect size large, p-value small
-) effect size large, p-value large

Whether effect size is small or large depends on the domain, which in this case is medical. For this problem consider a small p-value to be less than 0.001, while a large p-value is above 0.05. Assume that the sample size is relatively large, e.g., thousands of patients with the condition that the drug hopes to treat.

- a. Which combination(s) would very likely be of interest?
- b. Which combinations(s) would very likely **not** be of interest?
- c. If the sample size were small, would that change your answers?

5. For Neyman-Pearson hypothesis testing, we need to balance the tradeoff between α , the probability of a type I error and power, i.e., $1-\beta$, where β is the probability of a type II error. Compute α , β , and the power for the cases given below, where we specify the null and alternative distributions and the accompanying critical region. All distributions are Gaussian with some specified mean u and standard deviation σ , i.e., $N(\mu, \sigma)$. Let T be the test statistic.

$H_0:N(0,1)$, $H_1:N(3,1)$, critical region: $T>2$.

$H_0:N(0,1)$, $H_1:N(3,1)$, critical region: $|T|>2$.

$H_0:N(-1,1)$, $H_1:N(3,1)$, critical region: $T>1$.

$H_0:N(-1,1)$, $H_1:N(3,1)$, critical region: $|T|>1$.

$H_0:N(-1,0.5)$, $H_1:N(3,0.5)$, critical region: $T>1$.

$H_0:N(-1,0.5)$, $H_1:N(3,0.5)$, critical region: $|T|>1$.

6. A p-value measures the probability of the result given that the null hypothesis is true. However, many people who calculate p-values have used it as the probability of the null hypothesis given the result, which is erroneous. A Bayesian approach to this problem is summarized by [Equation 10.12](#).

$$P(H_1|x_{\text{obs}})P(H_0|x_{\text{obs}}) = f(H_1|x_{\text{obs}})f(H_0|x_{\text{obs}}) \times P(H_1)P(H_0) \text{posterior odds} \frac{P(H_1)}{P(H_0)}$$

This approach computes the ratio of the probability of the alternative and null hypotheses (H_1 and H_0 , respectively) given the observed outcome, x_{obs} . In turn, this quantity is expressed as the product of two factors: the Bayes factor and the prior odds. The prior odds is the ratio of the probability of H_1 to the probability of H_0 based on prior information about how likely we believe each hypothesis is. Usually, the prior odds is estimated directly based on experience. For example, in drug testing in the laboratory, it may be known that most drugs do not produce potentially therapeutic effects. The Bayes factor is the ratio of the probability or probability density of the observed outcome, x_{obs} , under H_1 and H_0 . This quantity is computed and represents a measure of how much more or less likely the observed result is under the alternative hypothesis than the null hypothesis. Conceptually, the higher it is, the more we would tend to prefer the alternative to the null. The higher the Bayes factor, the stronger the evidence provided by the data for H_1 . More generally, this approach can be applied to assess the evidence for any hypothesis versus another. Thus, the roles of H_0 can be (and often are) reversed in [Equation 10.12](#).

- a. Suppose that the Bayes factor is 20, which is very strong, but the prior odds are 0.01. Would you be inclined to prefer the alternative or null hypothesis?
- b. Suppose the prior odds are 0.25, the null distribution is Gaussian with density given by $f_0(x)=N(0,2)$, and the alternative distribution is given by $f_1(x)=N(3,1)$. Compute the Bayes factor and posterior odds of H_1 for the

following values of x_{obs} : 2, 2.5, 3, 3.5, 4, 4.5, 5. Explain the pattern that you see in both quantities.

7. Consider the problem of determining whether a coin is a fair one, i.e., $P(\text{heads})=P(\text{tails})=0.5$, by flipping the coin 10 times. Use the binomial theorem and basic probability to answer the following questions.

A coin is flipped ten times and it comes up heads every time. What is the probability of getting 10 heads in a row and what would you conclude about whether the coin is fair?

Suppose 10,000 coins are each flipped 10 times in a row and the flips of 10 coins result in all heads, can you confidently say that these coins are not fair?

What can you conclude about results when evaluated individually versus in a group?

Suppose that you flip each coin 20 times and then evaluate 10,000 coins. Can you now confidently say that any coin which yields all heads is not fair?

8. **Algorithm 10.1** on 773 provides a method for calculating the false discovery rate using the method advocated by Benjamini and Hochberg. The description in the text is presented in terms of ordering the p-values and adjusting the significance level to assess whether a p-value is significant. Another way to interpret this method is in terms of ordering the p-values, smallest to largest, and computing adjusted" p-values, $p'i=pi*m/i$, where i identifies the i th smallest p-value and m is the number of p-values. The statistical significance is determined based on whether $p'i \leq \alpha$, where α is the desired false discovery rate.

- a. Compute the adjusted p-values for the p-values in [Table 10.8](#). Note that the adjusted p-values may not be monotonic. In that case, an adjusted p-value that is larger than its successor is changed to have the same value as its successor.

Table 10.8. Ordered Collection of p-values..

	1	2	3	4	5	6	7	8	9	10
original p-values	0.001	0.005	0.05	0.065	0.15	0.21	0.25	0.3	0.45	0.5

- b. If the desired FDR is 20%, i.e., $\alpha=0.20$, then for which p-values is H_0 rejected?
- c. Suppose that we use the Bonferroni procedure instead. For different values of α , namely 0.01, 0.05, and 0.10, compute the modified p-value threshold, $\alpha^*=\alpha/10$, that the Bonferroni procedure will use to evaluate p-values. Then determine, for each value of α^* , for which p-values, H_0 will be rejected. (If a p-value equals the threshold, it is rejected.)
9. The positive false discovery rate (pFDR) is similar to the false discovery rate defined in [Section 10.1.3](#) but assumes that the number of true positives is greater than 0. Calculation of the pFDR is similar to that of FDR, but requires an assumption on the value of m_0 , the number of results that satisfy the null hypothesis. The pFDR is less conservative than FDR, but more complicated to compute.

The positive false discovery rate also allows the definition of an FDR analogue of the p-value. The q-value is the expected fraction of hypotheses that will be false if the given hypothesis is accepted. Specifically, the q-value associated with a p-value is the expected proportion of false positives among all hypotheses that are more extreme, i.e., have a lower p-value. Thus, the q-

value associated with a p-value is the positive false discovery rate that would result if the p-value was used as the threshold for rejection.

Below we show 50 p-values, their Benjamini-Hochberg adjusted p-values, and their q-values.

p-values

0.0000	0.0000	0.0002	0.0004	0.0004	0.0010	0.0089	0.0089	0.0288	0.0479
0.0755	0.0755	0.0755	0.1136	0.1631	0.2244	0.2964	0.3768	0.3768	0.3768
0.4623	0.4623	0.4623	0.5491	0.5491	0.6331	0.7107	0.7107	0.7107	0.7107
0.7107	0.8371	0.9201	0.9470	0.9470	0.9660	0.9660	0.9660	0.9790	0.9928
0.9928	0.9928	0.9928	0.9960	0.9960	0.9989	0.9989	0.9995	0.9999	1.0000

BH adjusted p-values

q-Values

0.7592	0.7879	0.8032	0.8078	0.8078	0.8108	0.8108	0.8108	0.8129	0.8150
0.8150	0.8150	0.8150	0.8155	0.8155	0.8159	0.8159	0.8160	0.8161	0.8161

- a. How many p-values are considered significant using BH adjusted p-values and thresholds of 0.05, 0.10, 0.15, 0.20, 0.25, and 0.30?
- b. How many p-values are considered significant using q-values and thresholds of 0.05, 0.10, 0.15, 0.20, 0.25, and 0.30?
- c. Compare the two sets of results.

10. An alternative to the definitions of the false discovery rate discussed so far is the local false discover rate, which is based on modeling the observed values of the test statistic as a mixture of two distributions, where most of the observations come from the null distribution and some observations (the interesting ones) come from the alternative distribution. (See [Section 8.2.2](#) for more information on mixture models.) If Z is our test statistic, the density, $f(z)$ of Z , is given by the following:

$$f(z) = p_0 f_0(z) + p_1 f_1(z), \quad (10.13)$$

where p_0 is the probability an instance comes from the null distribution, $f_0(z)$ is the distribution of the p-values under the null hypothesis, p_1 is the probability an instance comes from the alternative distribution, and $f_1(z)$ is the distribution of p-values under the alternative hypothesis.

Using Bayes theorem, we can derive the probability of the null hypothesis for any value of z as follows.

$$p(H_0|z) = f(H_0 \text{ and } z) / f(z) = p_0 f_0(z) / f(z) \quad (10.14)$$

The quantity, $p(H_0|z)$, is the quantity that we would like to define as the local fdr. Since p_0 is often close to 1, the local false discovery rate, represented as fdr, all lowercase, is defined as the following:

$$fdr(z) = f_0(z)f(z) \quad (10.15)$$

This is a point estimate, not an interval estimate as with the standard FDR, which is based on p-value, and as such, it will vary with the value of the test statistic. Note that the local fdr has an easy interpretation, namely as the ratio of the density of observations from the null distribution to observations from both the null and alternative distributions. It also has the advantage of being interpretable directly as a real probability.

The challenge, of course, is in estimating the densities involved in [Equation 10.15](#), which are usually estimated empirically. We consider the following simple case, where we specify the distributions by Gaussian distributions. The null distribution is given by, $f_0(z) = N(0, 1)$, while the alternative distribution is given by $f_0(z) = N(3, 1)$. $p_0 = 0.999$ and $p_1 = 0.001$.

- a. Compute $p(H_0|z)$ for the following values of z : 2, 2.5, 3, 3.5, 4, 4.5, 5.
 - b. Compute the local fdr for the following values of z : 2, 2.5, 3, 3.5, 4, 4.5, 5.
 - c. How close are these two sets of values?
11. The following are two alternatives to swap randomization—presented in [Section 10.4.2](#)—for randomizing a binary matrix so that the number of 1's in any row and column are preserved. Examine each method and (i) verify that it does indeed preserve the number of 1's in any row and column, and (ii) identify the problem with the alternative approach.
- a. Randomly permute the order of the columns and rows. An example is shown in [Figure 10.12](#).

	i_1	i_2	i_3
t_1	1	1	0
t_2	1	0	1
t_3	1	1	1

	i_3	i_2	i_1
t_2	0	0	1
t_3	1	1	1
t_1	1	1	1

Figure 10.12.

A 3×3 matrix before and after randomizing the order of the rows and columns. The leftmost matrix is the original.

- b. **Figure 10.13** shows another approach to randomizing a binary matrix. This approach converts the binary matrix to a row-column representation, then randomly reassigns the columns to various entries, and finally converts the data back into the original binary matrix format.

	i_1	i_2	i_3	i_4
t_1	1	1	0	0
t_2	0	1	0	1
t_3	1	1	1	0
t_4	0	0	1	1

<i>row</i>	<i>col</i>	<i>row</i>	<i>col</i>
1	1	1	4
1	2	1	2
2	2	2	2
2	4	2	1
3	1	3	1
3	2	3	2
3	3	3	3
4	3	4	3
4	4	4	1

	i_1	i_2	i_3	i_4
t_1	0	1	0	1
t_2	1	1	0	0
t_3	0	1	1	1
t_4	1	0	1	0

Figure 10.13.

A 4×4 matrix before and after randomizing the entries. From right to left, the tables represent the following: The original binary data matrix, the matrix in row-column format, the row-column format after randomly permuting the entries in the *col* column, and the matrix reconstructed from the randomized row-column representation.

Author Index

- Abdulghani, A., **433**
- Abe, N., **345**
- Abello, J., **698**
- Abiteboul, S., **437**
- Abraham, B., **745**
- Adams, N. M., **430**
- Adolfi, F. G., **807**
- Adomavicius, G., **18**
- Aelst, S. V., **749**
- Afshar, R., **509**
- Agarwal, R. C., **342, 429, 747**
- Aggarwal, C., **18, 430, 507, 509**
- Aggarwal, C. C., **339, 429, 430, 600, 602, 745, 808**
- Agrawal, R., **18, 104, 183, 184, 430, 431, 435, 436, 507, 509, 696**
- Aha, D. W., **181, 339**
- Akaike, H., **181**
- Akoglu, L., **745**
- Aksehirli, E., **434**
- Albers, C., **807**
- Alcalá-Fdez, J., **508**
- Aldenderfer, M. S., **600**
- Alexandridis, M. G., **184**
- Ali, K., **430**
- Allen, D. M., **181**
- Allison, D. B., **181**
- Allwein, E. L., **340**
- Alsabti, K., **181**
- Altman, R. B., **19**
- Alvarez, J. L., **508**
- Amatriain, X., **18**
- Ambroise, C., **181**

- Anderberg, M. R., **102, 600**
- Anderson, T. W., 849
- Andrews, R., **340**
- Ankerst, M., **600**
- Antonie, M.-L., **507**
- Anvari, F., **807**
- Aone, C., **601**
- Apps, M. A., **807**
- Arabie, P., **600, 602**
- Argamon, S. E., **807**
- Arnold, A., **746**
- Arthur, D., **600**
- Atluri, G., **19, 508**
- Atwal, G. S., **103**
- Aumann, Y., **507**
- Austin, J., **747**
- Ayres, J., **507**

- Baguley, T., **807**
- Bai, H., **696**
- Baker, M., **806**
- Baker, R. J., **343**
- Bakiri, G., **341**
- Bakos, J., **438**
- Baldi, P., **340**
- Ball, G., **600**
- Bandyopadhyay, S., **103**
- Banerjee, A., **18, 19, 600, 696, 746**
- Barbará, D., **430, 696**
- Barnett, V., **745**
- Bass, D., **806**
- Bastide, Y., **435**
- Basu, S., **696**
- Batistakis, Y., **601**
- Baxter, R. A., **747**
- Bay, S. D., **430, 746**
- Bayardo, R., **430**
- Becker, R., **807**
- Beckman, R. J., **746**
- Belanche, L., **104**
- Belkin, M., 849
- Ben-David, S., **19**
- Bengio, S., **341**
- Bengio, Y., **340, 341, 343, 345**
- Benjamin, D. J., **806**
- Benjamini, Y., **430, 806**
- Bennett, K., **340**
- Benning, S. D., **807**
- Berger, J., **806**

- Berk, R., **806**
- Berkin, P., **600**
- Bernecker, T., **430**
- Berrar, D., **340**
- Berry, M. J. A., **18**
- Bertino, E., **20**
- Bertolami, R., **342**
- Bhaskar, R., **430**
- Bienenstock, E., **182**
- Bilmes, J., **696**
- Bins, M., **183**
- Bishop, C. M., **181, 340, 696**
- Blashfield, R. K., **600**
- Blei, D. M., **696**
- Blondel, M., **20, 184**
- Blum, A., **102, 340**
- Bobadilla, J., **18**
- Bock, H.-H., **600**
- Bock, H. H., **102**
- Boiteau, V., **600**
- Boley, D., **600, 602**
- Bollen, K., **806**
- Bolton, R. J., **430**
- Borg, I., **102**
- Borgwardt, K. M., **434**
- Boswell, R., **340**
- Bosworth, A., **103**
- Bottou, L., **340**
- Boulicaut, J.-F., **507**
- Bowyer, K. W., **340**
- Bradley, A. P., **340**

- Bradley, P. S., **18**, **438**, **600**, **696**
- Bradski, G., **182**
- Bratko, I., **183**
- Breiman, L., **181**, **340**
- Brembs, B., **806**
- Breslow, L. A., **181**
- Breunig, M. M., **600**, **746**
- Brin, S., **430**, **431**, **436**
- Brockwell, A., **20**
- Brodley, C. E., **184**, **747**
- Brown, L., **806**
- Brucher, M., **20**, **184**
- Bunke, H., **342**
- Buntine, W., **181**
- Burges, C. J. C., **340**
- Burget, L., **343**
- Burke, R., **18**
- Buturovic, L. J., **183**
- Bykowski, A., **507**

- Cai, C. H., **431**
- Cai, J., **438**
- Camerer, C., **806**
- Campbell, C., **340**
- Canny, J. F., **344**
- Cantú-Paz, E., **181**
- Cao, H., **431**
- Cardie, C., **699**
- Carreira-Perpinan, M. A., 849
- Carvalho, C., **435**
- Catanzaro, B., **182**
- Ceri, S., **434**
- Cernock`y, J., **343**
- Chakrabarti, S., **18**
- Chan, P. K., **19, 341**
- Chan, R., **431**
- Chandola, V., **746**
- Chang, E. Y., **696**
- Chang, L., **749**
- Charrad, M., **600**
- Chatterjee, S., **698, 747**
- Chaudhary, A., **746**
- Chaudhuri, S., **103**
- Chawla, N. V., **340, 746**
- Chawla, S., **746**
- Cheeseman, P., **696**
- Chen, B., **436**
- Chen, C., **746**
- Chen, M.-S., **18, 435, 509**
- Chen, Q., **431, 508, 749**
- Chen, S., **434**

- Chen, S.-C., **749**
- Chen, W. Y., **696**
- Chen, Z., **749**
- Cheng, C. H., **431**
- Cheng, H., **432, 507**
- Cheng, R., **437**
- Cherkassky, V., **18, 182, 340**
- Chervonenkis, A. Y., **184**
- Cheung, D., **749**
- Cheung, D. C., **431**
- Cheung, D. W., **431, 437**
- Chiu, B., **747**
- Chiu, J., **433**
- Choudhary, A., **434, 698**
- Chrisman, N. R., **102**
- Chu, C., **182**
- Chu, G., **808**
- Chuang, A., **745**
- Chui, C. K., **431**
- Chung, S. M., **433, 508**
- Clark, P., **340**
- Clifton, C., **18, 437**
- Clifton, D. A., **748**
- Clifton, L., **748**
- Coatney, M., **435, 508**
- Cochran, W. G., **102**
- Codd, E. F., **102**
- Codd, S. B., **102**
- Cohen, P. R., **183, 807**
- Cohen, W. W., **340**
- Collingridge, D. S., **806**

- Contreras, P., **600, 602**
- Cook, D. J., **698**
- Cook, R. D., **746**
- Cooley, R., **431**
- Cost, S., **340**
- Cotter, A., **182**
- Cournapeau, D., **20, 184**
- Courville, A., **340, 341**
- Courville, A. C., **341**
- Couto, J., **430**
- Cover, T. M., **102, 341**
- Cristianini, N., **104, 341**
- Cui, X., **181**

- Dabney, A., **806**
- Dash, M., **103**
- Datta, S., **19**
- Davidson, I., **696**
- Davies, L., **746**
- Davis, R. W., **808**
- Dayal, U., **431, 508, 602**
- Dean, J., 890
- Demmel, J. W., **102, 832**, 849
- Deng, A., **807**
- Desrosiers, C., **18**
- Dhillon, I. S., **600**
- Diaz-Verdejo, J., **746**
- Diday, E., **102**
- Diederich, J., **340**
- Dietterich, T. G., **341, 343**
- Ding, C., **437, 696**
- Ding, C. H. Q., **698**
- Dokas, P., **431**
- Domingos, P., **18, 182, 341**
- Dong, G., **431**
- Donoho, D. L., 849
- Doster, W., **184**
- Dougherty, J., **102**
- Doursat, R., **182**
- Drummond, C., **341**
- Dubes, R. C., **103, 601, 807**
- Dubourg, V., **20, 184**
- Duchesnay, E., **20, 184**
- Duda, R. O., **18, 182, 341, 600**
- Duda, W., **344**

- Dudoit, S., **182**
- Duin, R. P. W., **182, 344**
- DuMouchel, W., **431**
- Dunagan, J., **746**
- Dunham, M. H., **18, 341**
- Dunkel, B., **431**

- Edgington, E., **806**
- Edwards, D. D., **344**
- Efron, B., **182, 806**
- Elkan, C., **341, 601**
- Ellis, P. D., **806**
- Elomaa, T., **103**
- Erhan, D., **341**
- Erhart, M., **698**
- Erickson III, D. J., **103**
- Ernst, M. D., **806**
- Ertöz, L., **431, 696, 697, 748**
- Eskin, E., **746, 748**
- Esposito, F., **182**
- Ester, M., **600–602**
- Everitt, B. S., **601**
- Evfimievski, A. V., **431**
- Ezeife, C., **508**

- Fürnkranz, J., **341**
- Fabris, C. C., **431**
- Faghmous, J., **19**
- Faghmous, J. H., **18**
- Faloutsos, C., **20, 104, 748**, 849
- Fan, J., **697**
- Fan, W., **341**
- Fang, G., **432, 436**
- Fang, Y., **699**
- Fawcett, T., **344**
- Fayyad, U. M., **18, 103, 438, 600, 696**
- Feng, L., **432, 437**
- Feng, S., **697**
- Fernández, S., **342**
- Ferri, C., **341**
- Field, B., **432**
- Finucane, H. K., **104**
- Fisher, D., **601, 697**
- Fisher, N. I., **432**
- Fisher, R. A., **182, 806**
- Flach, P., **340**
- Flach, P. A., **341**
- Flannick, J., **507**
- Flynn, P. J., **601**
- Fodor, I. K., 849
- Fournier-Viger, P., **507**
- Fovino, I. N., **20**
- Fox, A. J., **746**
- Fraley, C., **697**
- Frank, E., **19, 20, 182, 345**
- Frasca, B., **807**

- Frawley, W., [435](#)
- Freeman, D., [696](#)
- Freitas, A. A., [431](#), [432](#)
- Freund, Y., [341](#)
- Friedman, J., [182](#), [342](#)
- Friedman, J. H., [19](#), [181](#), [432](#), [601](#)
- Fu, A., [431](#), [433](#)
- Fu, A. W.-c., [749](#)
- Fu, Y., [432](#), [508](#)
- Fukuda, T., [432](#), [434](#), [507](#)
- Fukunaga, K., [182](#), [341](#)
- Furuichi, E., [435](#)

- Gada, D., **103**
- Ganguly, A., **19**
- Ganguly, A. R., **18, 103**
- Ganti, V., **182, 697**
- Gao, X., **601**
- Gaohua Gu, F. H., **103**
- Garcia-Teodoro, P., **746**
- Garofalakis, M. N., **507**
- Garriga, G. C., **807**
- Gather, U., **746**
- Geatz, M., **20**
- Gehrke, J., **18, 19, 104, 182, 431, 507, 696, 697**
- Geiger, D., **341**
- Geisser, S., **182**
- Gelman, A., **806**
- Geman, S., **182**
- Gersema, E., **183**
- Gersho, A., **697**
- Ghazzali, N., **600**
- Ghemawat, S., 890
- Ghosh, A., **746**
- Ghosh, J., **600, 697, 699**
- Giannella, C., **19**
- Gibbons, P. B., **748**
- Gibson, D., **697**
- Gigerenzer, G., **806**
- Gionis, A., **746, 806**
- Glymour, C., **19**
- Gnanadesikan, R., **746**
- Goethals, B., **434**
- Goil, S., **698**

- Goldberg, A. B., **345**
- Golub, G. H., **832**
- Gomariz, A., **507**
- Good, P., **806**
- Goodfellow, I., **341**
- Goodfellow, I. J., **341**
- Goodman, R. M., **344**
- Goodman, S., **806**
- Gorfine, M., **103**
- Gowda, K. C., **697**
- Grama, A., **20**
- Gramfort, A., **20, 184**
- Graves, A., **342**
- Gray, J., **103**
- Gray, R. M., **697**
- Greenland, S., **806**
- Gries, D., 890
- Grimes, C., 849
- Grinstein, G. G., **18**
- Grisel, O., **20, 184**
- Groenen, P., **102**
- Grossman, R. L., **19, 698**
- Grossman, S. R., **104**
- Guan, Y., **600**
- Guestrin, C., **20**
- Guha, S., **19, 697**
- Gunopulos, D., **432, 437, 696**
- Guntzer, U., **432**
- Gupta, M., **432**
- Gupta, R., **432, 508**
- Gutiérrez, A., **18**

- Hagen, L., **697**
- Haibt, L., **344**
- Haight, R., **436**
- Haiminen, N., **807**
- Halic, M., **183**
- Halkidi, M., **601**
- Hall, D., **600**
- Hall, L. O., **340**
- Hall, M., **19, 182**
- Hamerly, G., **601**
- Hamilton, H. J., **432**
- Han, E., **432**
- Han, E.-H., **183, 342, 432, 508, 601, 697, 698**
- Han, J., **18, 19, 342, 430, 432–435, 437, 507–509, 601, 698**
- Hand, D. J., **19, 103, 342, 430**
- Hardin, J., **746**
- Hart, P. E., **18, 182, 341, 600**
- Hartigan, J., **601**
- Hastie, T., **19, 182, 342, 601**
- Hatonen, K., **437**
- Hawkins, D. M., **747**
- Hawkins, S., **747**
- He, H., **747**
- He, Q., **433, 699**
- He, X., **437, 696**
- He, Y., **437, 697**
- Hearst, M., **342**
- Heath, D., **182**
- Heckerman, D., **342**
- Heller, R., **103**
- Heller, Y., **103**

- Hernando, A., **18**
- Hernández-Orallo, J., **341**
- Herrera, F., **508**
- Hey, T., **19**
- Hidber, C., **432**
- Hilderman, R. J., **432**
- Hinneburg, A., **697**
- Hinton, G., **343**
- Hinton, G. E., **342–344**
- Hipp, J., **432**
- Ho, C.-T., **20**
- Hochberg, Y., **430, 806**
- Hodge, V. J., **747**
- Hofmann, H., **432**
- Holbrook, S. R., **437**
- Holder, L. B., **698**
- Holland, J., **344**
- Holmes, G., **19, 182**
- Holt, J. D., **433**
- Holte, R. C., **341, 342**
- Hong, J., **343**
- Hornick, M. F., **19**
- Houtsma, M., **433**
- Hsieh, M. J., **509**
- Hsu, M., **431, 508, 602**
- Hsu, W., **434**
- Hsueh, S., **433**
- Huang, H.-K., **748**
- Huang, T. S., **698**
- Huang, Y., **433**
- Hubbard, R., **807**

- Hubert, L., **600**, **602**
- Hubert, M., **749**
- Hulten, G., **18**
- Hung, E., **431**
- Hussain, F., **103**
- Hwang, S., **433**
- Hämäläinen, W., **807**
- Höppner, F., **697**

- Iba, W., **343**
- Imielinski, T., **430, 433**
- Inokuchi, A., **433, 508**
- Ioannidis, J. P., **807**
- Ioffe, S., **342**
- Irani, K. B., **103**

- Jagadish, H. V., **747**
- Jager, L. R., **807**
- Jain, A. K., **19, 103, 182, 601, 807**
- Jajodia, S., **430**
- Janardan, R., 849
- Japkowicz, N., **340, 342, 746, 747**
- Jardine, N., **601**
- Jaroszewicz, S., **433**
- Jarvis, R. A., **697**
- Jensen, D., **104, 183, 807**
- Jensen, F. V., **342**
- Jeudy, B., **507**
- Johannesson, M., **806**
- John, G. H., **103**
- Johnson, T., **747**
- Jolliffe, I. T., **103**, 849
- Jonyer, I., **698**
- Jordan, M. I., **342, 696**
- Joshi, A., **19**
- Joshi, M. V., **183, 342, 343, 508, 747**

- Kahng, A., **697**
- Kailing, K., **698**
- Kallio, A., **807**
- Kalpakis, K., **103**
- Kamath, C., **19, 181, 698**
- Kamber, M., **19, 342, 433, 601**
- Kantarciooglu, M., **18**
- Kantardzic, M., **19**
- Kao, B., **431**
- Karafiát, M., **343**
- Kargupta, H., **19**
- Karpatne, A., **19**
- Karypis, G., **18, 183, 342, 432, 433, 436, 508, 509, 601, 602, 697, 698**
- Kasif, S., **182, 183**
- Kass, G. V., **183**
- Kass, R. E., **807**
- Kaufman, L., **103, 601**
- Kawale, J., **747**
- Kegelmeyer, P., **19, 698**
- Kegelmeyer, W. P., **340**
- Keim, D. A., **697**
- Kelly, J., **696**
- Keogh, E., **747**
- Keogh, E. J., **103**
- Keshet, J., **182**
- Kettenring, J. R., **746**
- Keutzer, K., **182**
- Khan, S., **103**
- Khan, S. S., **747**
- Khardon, R., **432**
- Khoshgoftaar, T. M., **20**

- Khudanpur, S., **343**
- Kifer, D., **19**
- Kim, B., **183**
- Kim, S. K., **182**
- Kinney, J. B., **103**
- Kitagawa, H., **748**
- Kitsuregawa, M., **436**
- Kivinen, J., **343**
- Klawonn, F., **697**
- Kleinberg, J., **19**
- Kleinberg, J. M., **601, 697**
- Klemettinen, M., **433, 437**
- Klooster, S., **435, 436, 698**
- Knorr, E. M., **747**
- Kogan, J., **600**
- Kohavi, R., **102, 103, 183, 807**
- Kohonen, T., **698**
- Kolcz, A., **340, 746**
- Kong, E. B., **343**
- Koperski, K., **432**
- Kosters, W. A., **433**
- Koudas, N., **747**
- Koutra, D., **745**
- Kröger, P., **698**
- Kramer, S., **509**
- Krantz, D., **103–105**
- Kriegel, H., **430**
- Kriegel, H.-P., **600–602, 698, 746, 748, 749**
- Krishna, G., **697**
- Krizhevsky, A., **342–344**
- Krstajic, D., **183**

- Kruse, R., **697**
- Kruskal, J. B., **103**, 849
- Kröger, P., **601**
- Kubat, M., **343**
- Kuhara, S., **435**
- Kulkarni, S. R., **183**
- Kumar, A., **747**
- Kumar, V., **18**, **19**, **183**, **342–344**, **431**, **432**, **435–437**, **508**, **509**, **601**, **602**,
696–698, **746–748**, 849
- Kuok, C. M., **433**
- Kuramochi, M., **433**, **508**
- Kwok, I., **747**
- Kwong, W. W., **431**

- Lagani, V., **184**
- Lajoie, I., **345**
- Lakens, D., **807**
- Lakhal, L., **435**
- Lakshmanan, L. V. S., **434**
- Lambert, D., **19**
- Landau, S., **601**
- Lander, E. S., **104**
- Landeweerd, G., **183**
- Landgrebe, D., **183, 184**
- Lane, T., **747**
- Langford, J. C., **345**, 849
- Langley, P., **102, 343, 697**
- Larochelle, H., **345**
- Larsen, B., **601**
- Lavrac, N., **343**
- Lavrač, N., **434**
- Law, M. H. C., **19**
- Laxman, S., **430**
- Layman, A., **103**
- Lazar, N. A., **808**
- Lazarevic, A., **431, 748**
- Leahy, D. E., **183**
- LeCun, Y., **343**
- Lee, D. D., **698**
- Lee, P., **433**
- Lee, S. D., **431, 437**
- Lee, W., **433, 748**
- Lee, Y. W., **105**
- Leek, J. T., **807, 808**
- Leese, M., **601**

- Lent, B., **508**
- Leroy, A. M., **748**
- Lewis, D. D., **343**
- Lewis, T., **745**
- Li, F., **699**
- Li, J., **431**
- Li, K.-L., **748**
- Li, N., **433**
- Li, Q., 849
- Li, T., **698**
- Li, W., **105, 433, 438**
- Li, Y., **430**
- Li, Z., **601, 602, 808**
- Liao, W.-K., **434**
- Liess, S., **747**
- Lim, E., **433**
- Lin, C. J., **696**
- Lin, K.-I., **434**, 849
- Lin, M., **433**
- Lin, Y.-A., **182**
- Lindell, Y., **507**
- Lindgren, B. W., **103**
- Lindquist, E. F., **807**
- Ling, C. X., **343**
- Linoff, G., **18**
- Lipton, Z. C., **20**
- Liu, B., **434, 437, 509**
- Liu, H., **103, 104**
- Liu, J., **434**
- Liu, L.-M., **746**
- Liu, R. Y., **748**

- Liu, Y., [433](#), [434](#), [601](#)
- Livny, M., [699](#)
- Liwicki, M., [342](#)
- Llinares-López, F., [434](#)
- Lonardi, S., [747](#)
- Lu, C.-T., [749](#)
- Lu, H. J., [432](#), [435](#), [437](#)
- Lu, Y., [438](#)
- Luce, R. D., [103–105](#)
- Ludwig, J., [19](#)
- Lugosi, G., [183](#)
- Luo, C., [508](#)
- Luo, W., [697](#)

- Ma, D., **697**
- Ma, H., **699**
- Ma, L., **699**
- Ma, Y., **434**
- Mabroukeh, N. R., **508**
- Maciá-Fernández, G., **746**
- MacQueen, J., **601**
- Madden, M. G., **747**
- Madigan, D., **19**
- Malerba, D., **182**
- Maletic, J. I., **434**
- Malik, J., **698**
- Malik, J. M., **344**
- Mamoulis, N., **431**
- Manganaris, S., **430**
- Mangasarian, O., **343**
- Mannila, H., **19, 342, 432, 437, 508, 806, 807**
- Manzagol, P.-A., **341, 345**
- Mao, H., **697**
- Mao, J., **182**
- Maranell, G. M., **104**
- Marchiori, E., **433**
- Marcus, A., **434**
- Margineantu, D. D., **343**
- Markou, M., **748**
- Martin, D., **508**
- Masand, B., **431**
- Mata, J., **508**
- Matsuzawa, H., **434**
- Matwin, S., **343**
- McCullagh, P., **343**

- McCulloch, W. S., **343**
- McLachlan, G. J., **181**
- McVean, G., **104**
- Megiddo, N., **434**
- Mehta, M., **183, 184**
- Meilä, M., **602**
- Meira Jr., W., **20**
- Meo, R., **434**
- Merugu, S., **600**
- Meyer, G., **19**
- Meyerson, A., **19, 697**
- Michalski, R. S., **183, 343, 698, 699**
- Michel, V., **20, 184**
- Michie, D., **183, 184**
- Mielikäinen, T., **806**
- Mikolov, T., **343**
- Miller, H. J., **601**
- Miller, R. J., **434, 508**
- Milligan, G. W., **602**
- Mingers, J., **183**
- Mirkin, B., **602**
- Mirza, M., **341**
- Mishra, N., **19, 697, 698**
- Misra, J., 890
- Mitchell, T., **20, 183, 340, 343, 602, 698**
- Mitzenmacher, M., **104**
- Mobasher, B., **432, 697**
- Modha, D. S., **600**
- Moens, S., **434**
- Mok, K. W., **433, 748**
- Molina, L. C., **104**

- Montgomery, D. C., **807**
- Mooney, R., **696**
- Moore, A. W., **602, 746**
- Moret, B. M. E., **183**
- Morimoto, Y., **432, 434, 507**
- Morishita, S., **432, 507**
- Mortazavi-Asl, B., **435, 508**
- Mosteller, F., **104, 434**
- Motoda, H., **103, 104, 433, 508**
- Motwani, R., **19, 430, 431, 436, 437, 697**
- Mozetic, I., **343**
- Mueller, A., **434**
- Muggleton, S., **343**
- Muirhead, C. R., **748**
- Mulier, F., **18, 340**
- Mulier, F. M., **182**
- Mullainathan, S., **19**
- Murphy, K. P., **183**
- Murphy, K. R., **807**
- Murtagh, F., **600, 602, 698**
- Murthy, S. K., **183**
- Murty, M. N., **601**
- Muthukrishnan, S., **747**
- Myers, C. L., **508**
- Myneni, R., **435**
- Myors, B., **807**
- Müller, K.-R., **849**

- Nagesh, H., **698**
- Nakhaeizadeh, G., **432**
- Namburu, R., **19, 698**
- Naughton, J. F., **438**
- Navathe, S., **435, 509**
- Nebot, A., **104**
- Nelder, J. A., **343**
- Nelson, L. D., **808**
- Nemani, R., **435**
- Nestorov, S., **437**
- Neyman, J., **807**
- Ng, A. Y., **182, 696**
- Ng, R. T., **434, 698, 746, 747**
- Niblett, T., **183, 340**
- Nielsen, M. A., **343**
- Niknafs, A., **600**
- Nishio, S., **435**
- Niyogi, P., **849**
- Nobel, A. B., **434**
- Norvig, P., **344**
- Nosek, B. A., **806**
- Novak, P. K., **434**
- Nuzzo, R., **807**

- O'Callaghan, L., **19**, **697**
- Oates, T., **104**
- Oerlemans, A., **433**
- Ogihara, M., **105**, **438**
- Ohsuga, S., **438**
- Ojala, M., **807**
- Olken, F., **104**
- Olshen, R., **181**
- Olukotun, K., **182**
- Omiecinski, E., **435**, **509**
- Onghena, P., **806**
- Ono, T., **435**
- Orihara, M., **438**
- Ortega, F., **18**
- Osborne, J., **104**
- Ostrouchov, G., **103**
- others, **19**, **184**, **602**, **748**, **806**, **807**
- Ozden, B., **435**
- Ozgur, A., **435**, **748**

- Padmanabhan, B., **438, 509**
- Page, G. P., **181**
- Palit, I., **183**
- Palmer, C. R., **104**
- Pan, S. J., **343**
- Pandey, G., **432, 508**
- Pang, A., **434**
- Papadimitriou, S., **20, 748**
- Papaxanthos, L., **434**
- Pardalos, P. M., **698**
- Parelius, J. M., **748**
- Park, H., **849**
- Park, J. S., **435**
- Parr Rud, O., **20**
- Parthasarathy, S., **105, 435, 438, 508**
- Pasquier, N., **435**
- Passos, A., **20, 184**
- Patrick, E. A., **697**
- Pattipati, K. R., **184**
- Paulsen, S., **434**
- Pazzani, M., **341, 430**
- Pazzani, M. J., **103**
- Pearl, J., **341, 344**
- Pearson, E. S., **807**
- Pedregosa, F., **20, 184**
- Pei, J., **19, 432, 433, 435, 508**
- Pelleg, D., **602**
- Pellow, F., **103**
- Peng, R. D., **807**
- Perrot, M., **20, 184**
- Pesarin, F., **807**

- Peters, M., **698**
- Pfahringer, B., **19, 182**
- Piatetsky-Shapiro, G., **18, 20, 435**
- Pimentel, M. A., **748**
- Pirahesh, H., **103**
- Pison, G., **749**
- Pitts, W., **343**
- Platt, J. C., **748**
- Pohlmann, N., **807**
- Portnoy, L., **746, 748**
- Potter, C., **435, 436, 698**
- Powers, D. M., **344**
- Prasad, V. V. V., **429**
- Pregibon, D., **19, 20, 431**
- Prerau, M., **746**
- Prettenhofer, P., **20, 184**
- Prince, M., **344**
- Prins, J., **434**
- Protopopescu, V., **103**
- Provenza, L. P., **20**
- Provost, F. J., **104, 344**
- Psaila, G., **434**
- Pujol, J. M., **18**
- Puttagunta, V., **103**
- Pyle, D., **20**

- Quinlan, J. R., **184, 344**

- Raftery, A. E., **697, 807**
- Raghavan, P., **696, 697**
- Rakhshani, A., **184**
- Ramakrishnan, N., **20**
- Ramakrishnan, R., **18, 104, 182, 697, 699**
- Ramaswamy, S., **435, 748**
- Ramkumar, G. D., **435**
- Ramoni, M., **344**
- Ranka, S., **181, 435**
- Rao, N., **508**
- Rastogi, R., **507, 697, 748**
- Reddy, C. K., **183, 600, 602, 808**
- Redman, T. C., **104**
- Rehmsmeier, M., **344**
- Reichart, D., **103**
- Reina, C., **696**
- Reisende, M. G. C., **698**
- Renz, M., **430**
- Reshef, D., **104**
- Reshef, D. N., **104**
- Reshef, Y., **104**
- Reshef, Y. A., **104**
- Reutemann, P., **19, 182**
- Ribeiro, M. T., **20**
- Richter, L., **509**
- Riondato, M., **435**
- Riquelme, J. C., **508**
- Rissanen, J., **183**
- Rivest, R. L., **184**
- Robinson, D., **806**
- Rochester, N., **344**

- Rocke, D. M., **746, 748**
- Rogers, S., **699**
- Roiger, R., **20**
- Romesburg, C., **602**
- Ron, D., **698**
- Ronkainen, P., **437**
- Rosenblatt, F., **344**
- Rosenthal, A., **437**
- Rosete, A., **508**
- Rosner, B., **748**
- Rotem, D., **104**
- Rousseeuw, P. J., **103, 601, 748**
- Rousu, J., **103**
- Roweis, S. T., 849
- Ruckert, U., **509**
- Runkler, T., **697**
- Russell, S. J., **344**
- Ruts, I., **748**

- Sabeti, P., **104**
- Sabeti, P. C., **104**
- Sabripour, M., **181**
- Safavian, S. R., **184**
- Sahami, M., **102**
- Saigal, S., **103**
- Saito, T., **344**
- Salakhutdinov, R., **344**
- Salakhutdinov, R. R., **342**
- Salmaso, L., **807**
- Salzberg, S., **182, 183, 340**
- Samatova, N., **18, 19**
- Sander, J., **600–602, 746**
- Sarawagi, S., **435**
- Sarinnapakorn, K., **749**
- Satou, K., **435**
- Saul, L. K., **849**
- Savaresi, S. M., **602**
- Savasere, A., **435, 509**
- Saygin, Y., **20**
- Schölkopf, B., **344**
- Schafer, J., **20**
- Schaffer, C., **184**
- Schapire, R. E., **340, 341**
- Scheuermann, P., **436**
- Schikuta, E., **698**
- Schmidhuber, J., **342, 344**
- Schroeder, M. R., **698**
- Schroedl, S., **699**
- Schubert, E., **748, 749**
- Schuermann, J., **184**

- Schwabacher, M., **746**
- Schwartzbard, A., **746**
- Schwarz, G., **184**
- Schölkopf, B., **104, 748**, 849
- Scott, D. W., **749**
- Sebastiani, P., **344**
- Self, M., **696**
- Semeraro, G., **182**
- Sendhoff, B., **696**
- Seno, M., **436, 509**
- Settles, B., **344**
- Seung, H. S., **698**
- Shafer, J. C., **184, 429**
- Shasha, D. E., **20**
- Shawe-Taylor, J., **104, 341, 748**
- Sheikholeslami, G., **698**
- Shekhar, S., **18, 19, 433, 435, 437, 749**
- Shen, W., **509**
- Shen, Y., **431**
- Sheng, V. S., **343**
- Shi, J., **698**
- Shi, Z., **433**
- Shibayama, G., **435**
- Shim, K., **507, 697, 748**
- Shinghal, R., **433**
- Shintani, T., **436**
- Shu, C., **699**
- Shyu, M.-L., **749**
- Sibson, R., **601**
- Siebes, A. P. J. M., **432**
- Siegmund, D., **808**

- Silberschatz, A., **435, 436**
- Silva, V. d., 849
- Silver, N., **808**
- Silverstein, C., **430, 436**
- Simmons, J. P., **808**
- Simon, H., **696**
- Simon, N., **104**
- Simon, R., **184**
- Simonsohn, U., **808**
- Simovici, D., **433**
- Simpson, E.-H., **436**
- Singer, Y., **340**
- Singh, K., **748**
- Singh, L., **436**
- Singh, S., **20, 748**
- Singh, V., **181**
- Sivakumar, K., **19**
- Smalley, C. T., **102**
- Smith, A. D., **430**
- Smola, A. J., **104, 343, 344, 748**, 849
- Smyth, P., **18–20, 342, 344**
- Sneath, P. H. A., **104, 602**
- Soete, G. D., **600, 602**
- Sokal, R. R., **104, 602**
- Song, Y., **696**
- Soparkar, N., **431**
- Speed, T., **104**
- Spiegelhalter, D. J., **183**
- Spiliopoulou, M., **431**
- Späth, H., **602**
- Srebro, N., **182**

- Srikant, R., **18**, **104**, **430**, **431**, **434**, **436**, **507**, **509**
- Srivastava, J., **431**, **436**, **509**, **748**
- Srivastava, N., **342**, **344**
- Steinbach, M., **18**, **19**, **183**, **344**, **432**, **435–437**, **508**, **602**, **696–698**, **747**
- Stepp, R. E., **698**, **699**
- Stevens, S. S., **104**
- Stolfo, S. J., **341**, **433**, **746**, **748**
- Stone, C. J., **181**
- Stone, M., **184**
- Storey, J. D., **806**, **808**
- Stork, D. G., **18**, **182**, **341**, **600**
- Strang, G., **832**
- Strehl, A., **699**
- Strimmer, K., **808**
- Struyf, A., **749**
- Stutz, J., **696**
- Su, X., **20**
- Suen, C. Y., **185**
- Sugiyama, M., **434**
- Sun, S., **344**
- Sun, T., **699**
- Sun, Z., **430**
- Sundaram, N., **182**
- Suppes, P., **103–105**
- Sutskever, I., **342–344**
- Suzuki, E., **436**
- Svensen, M., **696**
- Swami, A., **430**, **433**, **508**
- Swaminathan, R., **698**
- Sykacek, P., **749**
- Szalay, A. S., **746**

- Szegedy, C., **342**

- Takagi, T., **435**
- Tan, C. L., **103**
- Tan, H., **697**
- Tan, P.-N., **344, 698**
- Tan, P. N., **183, 431, 435–437, 509**
- Tang, J., **749**
- Tang, S., **435**
- Tansley, S., **19**
- Tao, D., **345**
- Taouil, R., **435**
- Tarassenko, L., **748**
- Tatti, N., **437**
- Tax, D. M. J., **344**
- Tay, S. H., **437, 509**
- Taylor, C. C., **183**
- Taylor, J. E., **808**
- Taylor, W., **696**
- Tenenbaum, J. B., 849
- Teng, W. G., **509**
- Thakurta, A., **430**
- Theodoridis, Y., **20**
- Thirion, B., **20, 184**
- Thomas, J. A., **102**
- Thomas, S., **183, 435**
- Thompson, K., **343**
- Tian, S.-F., **748**
- Tibshirani, R., **19, 104, 182, 184, 342, 344, 601, 806, 808**
- Tibshirani, R. J., **184**
- Tickle, A., **340**
- Timmers, T., **183**
- Toivonen, H., **20, 105, 432, 437, 508**

- Tokuyama, T., **432, 434, 507**
- Tolle, K. M., **19**
- Tompkins, R. G., **808**
- Tong, H., **745**
- Torregrosa, A., **436**
- Tsamardinos, I., **184**
- Tsaparas, P., **806**
- Tseng, V. S., **507**
- Tsoukatos, I., **437**
- Tsur, S., **431, 435, 437**
- Tucakov, V., **747**
- Tukey, J. W., **104, 105, 748**
- Tung, A., **437, 601**
- Turnbaugh, P. J., **104**
- Tusher, V., **806**
- Tusher, V. G., **808**
- Tuzhilin, A., **18, 436, 438, 509**
- Tversky, A., **103–105**
- Tzvetkov, P., **509**

- Ullman, J., **431, 437**
- Uslaner, E. M., **103**
- Utgoff, P. E., **184**
- Uthurusamy, R., **18**

- Vaidya, J., **18, 437**
- Valiant, L., **184**
- van Assen, M. A., **807**
- van Rijsbergen, C. J., **344**
- van Zomeren, B. C., **748**
- Vanderplas, J., **20, 184**
- Vandin, F., **435**
- van der Laan, M. J., **182**
- Van Loan, C. F., **832**
- Vapnik, V., **345**
- Vapnik, V. N., **184**
- Varma, S., **184**
- Varoquaux, G., **20, 184**
- Vassilvitskii, S., **600**
- Vazirgiannis, M., **601**
- Velleman, P. F., **105**
- Vempala, S., **746**
- Venkatesh, S. S., **183**
- Venkatrao, M., **103**
- Verhein, F., **430**
- Verkamo, A. I., **508**
- Verma, T. S., **341**
- Verykios, V. S., **20**
- Vincent, P., **340, 341, 345**
- Virmani, A., **433**
- Vitter, J. S., 890
- von Luxburg, U., **699**
- von Seelen, W., **696**
- von der Malsburg, C., **696**
- Vorbruggen, J. C., **696**
- Vreeken, J., **808**

- Vu, Q., **436**
- Vuokko, N., **807**
- Vázquez, E., **746**

- Wagenmakers, E.-J., **806**
- Wagstaff, K., **696, 699**
- Wainwright, M., **342**
- Walker, T., **807**
- Wang, H., **184**
- Wang, J., **430, 509**
- Wang, J. T. L., **20**
- Wang, K., **437, 509**
- Wang, L., **437**
- Wang, Q., **19**
- Wang, Q. R., **185**
- Wang, R. Y., **105**
- Wang, W., **432, 434**
- Wang, Y. R., **105**
- Warde-Farley, D., **341**
- Washio, T., **433, 508**
- Wasserstein, R. L., **808**
- Webb, A. R., **20, 345**
- Webb, G. I., **434, 437, 509, 808**
- Weiss, G. M., **345**
- Weiss, R., **20, 184**
- Welch, W. J., **808**
- Werbos, P., **345**
- Widmer, G., **341**
- Widom, J., **508**
- Wierse, A., **18**
- Wilhelm, A. F. X., **432**
- Wilkinson, L., **105**
- Williams, C. K. I., **696**
- Williams, G. J., **747**
- Williamson, R. C., **343, 748**

- Wimmer, M., **600**
- Wish, M., 849
- Witten, I. H., **19, 20, 182, 345**
- Wojdanowski, R., **748**
- Wolach, A., **807**
- Wong, M. H., **433**
- Woodruff, D. L., **748**
- Wu, C.-W., **507**
- Wu, J., **601**
- Wu, N., **430**
- Wu, S., **601**
- Wu, X., **20, 344, 509**
- Wunsch, D., **602**

- Xiang, D., **748**
- Xiao, W., **808**
- Xin, D., **432**
- Xiong, H., **433, 436, 437, 601, 602, 808**, 849
- Xu, C., **345**
- Xu, R., **602**
- Xu, W., **748**
- Xu, X., **600–602**
- Xu, Y., **807**

- Yamamura, Y., **435**
- Yan, X., **19, 432, 437, 507, 509**
- Yang, C., **438**
- Yang, Q., **343, 431**
- Yang, Y., **185, 434, 508**
- Yao, Y. Y., **438**
- Ye, J., 849
- Ye, N., **104, 697, 749**
- Yesha, Y., **19**
- Yin, Y., **432**
- Yiu, T., **507**
- Yoda, K., **434**
- Yu, H., **436, 699**
- Yu, J. X., **432**
- Yu, L., **104**
- Yu, P. S., **18–20, 430, 435, 745**
- Yu, Y., **182**

- Zaïane, O. R., **432, 507**
- Zadrozny, B., **345**
- Zahn, C. T., **602**
- Zaki, M. J., **20, 105, 438, 509, 698**
- Zaniolo, C., **184**
- Zeng, C., **438**
- Zeng, L., **433**
- Zhang, A., **698**
- Zhang, B., **438, 602**
- Zhang, C., **509**
- Zhang, F., **438**
- Zhang, H., **438, 509**
- Zhang, J., **341**
- Zhang, M.-L., **345**
- Zhang, N., **19**
- Zhang, P., **749**
- Zhang, S., **509**
- Zhang, T., **699**
- Zhang, Y., **185, 437, 438**
- Zhang, Z., **438**
- Zhao, W., **699**
- Zhao, Y., **602**
- Zhong, N., **438**
- Zhou, Z.-H., **345**
- Zhu, H., **435**
- Zhu, X., **345**
- Ziad, M., **105**
- Zimek, A., **601, 748, 749**
- Züfle, A., **430**

Subject Index

- k -nearest neighbor graph, [657](#), [663](#), [664](#)

- accuracy, [119](#), [196](#)
- activation function, [251](#)
- AdaBoost, [306](#)
- aggregation, [51–52](#)
- anomaly detection
 - applications, [703–704](#)
 - clustering-based, [724–728](#)
 - example, [726](#)
 - impact of outliers, [726](#)
 - membership in a cluster, [725](#)
 - number of clusters, [728](#)
 - strengths and weaknesses, [728](#)
 - definition, [705–706](#)
 - definitions
 - distance-based, [719](#)
 - density-based, [720–724](#)
 - deviation detection, [703](#)
 - exception mining, [703](#)
 - outliers, [703](#)
 - proximity-based
 - distance-based, see [anomaly detection](#), [distance-based](#)
 - relative density, [722–723](#)
 - example, [723](#)
 - statistical, [710–719](#)
 - Gaussian, [710](#)
 - Grubbs, [751](#)
 - likelihood approach, [715](#)
 - multivariate, [712](#)
 - strengths and weaknesses, [718](#)
 - techniques, [708–709](#)
- Apriori

- algorithm, **364**
- principle, **363**
- association
 - analysis, **357**
 - categorical attributes, **451**
 - continuous attributes, **454**
 - indirect, **503**
 - pattern, **358**
 - rule, see **rule**
- attribute, **26–33**
 - definition of, **27**
 - number of values, **32**
 - type, **27–32**
 - asymmetric, **32–33**
 - binary, **32**
 - continuous, **30, 32**
 - discrete, **32**
 - general comments, **33–34**
 - interval, **29, 30**
 - nominal, **29, 30**
 - ordinal, **29, 30**
 - qualitative, **30**
 - quantitative, **30**
 - ratio, **29**
- avoiding false discoveries, **755–806**
 - considerations for anomaly detection, **800–803**
 - considerations for association analysis, **787**
 - randomization, **793–795**
 - considerations for classification, **783–787**
 - considerations for cluster analysis, **795–800**
 - generating a null distribution, **776–783**

- permutation test, **781**
- randomization, **781**
- hypothesis testing, see **hypothesis testing**
- multiple hypothesis testing, see **False Discovery Rate**
- problems with significance and hypothesis testing, **778**
- axon, **249**

- backpropagation, **258**
- bagging, see **classifier**
- Bayes
 - naive, see **classifier**
 - network, see **classifier**
 - theorem, **214**
- bias variance decomposition, **300**
- binarization, see **discretization, binarization, 452, 455**
- BIRCH, **684–686**
- Bonferroni Procedure), **768**
- boosting, see **classifier**
- Bregman divergence, **94–95**

- candidate
 - generation, **367, 368, 471, 487**
 - itemset, **362**
 - pruning, **368, 472, 493**
 - rule, **381**
 - sequence, **468**
- case, see **object**
- chameleon, **660–666**
 - algorithm, **664–665**
 - graph partitioning, **664, 665**
 - merging strategy, **662**
 - relative closeness, **663**
 - relative interconnectivity, **663**
 - self-similarity, **656, 661, 663–665**
 - strengths and limitations, **666**
- characteristic, see **attribute**
- city block distance, see **distance, city block**
- class
 - imbalance, **313**
- classification
 - class label, **114**
 - evaluation, **119**
- classifier
 - bagging, **302**
 - base, **296**
 - Bayesian belief, **227**
 - boosting, **305**
 - combination, **296**
 - decision tree, **119**
 - ensemble, **296**
 - logistic regression, **243**

- maximal margin, [278](#)
- naive-Bayes, [218](#)
- nearest neighbor, [208](#)
- neural networks, [249](#)
- perceptron, [250](#)
- probabilistic, [212](#)
- random forest, [310](#)
- Rote, [208](#)
- rule-based, [195](#)
- support vector machine, [276](#)
- unstable, [300](#)
- climate indices, [680](#)
- cluster analysis
 - algorithm characteristics, [619–620](#)
 - mapping to another domain, [620](#)
 - nondeterminism, [619](#)
 - optimization, [620](#)
 - order dependence, [619](#)
 - parameter selection, see [parameter selection](#)
 - scalability, see [scalability](#)
 - applications, [525–527](#)
 - as an optimization problem, [620](#)
 - chameleon, see [chameleon](#)
 - choosing an algorithm, [690–693](#)
 - cluster characteristics, [617–618](#)
 - data distribution, [618](#)
 - density, [618](#)
 - poorly separated, [618](#)
 - relationships, [618](#)
 - shape, [618](#)
 - size, [618](#)

- subspace, [618](#)
- cluster density, [618](#)
- cluster shape, [548](#), [618](#)
- cluster size, [618](#)
- data characteristics, [615–617](#)
 - attribute types, [617](#)
 - data types, [617](#)
 - high-dimensionality, [616](#)
 - mathematical properties, [617](#)
 - noise, [616](#)
 - outliers, [616](#)
 - scale, [617](#)
 - size, [616](#)
 - sparseness, [616](#)
- DBSCAN, see [DBSCAN](#)
- definition of, [525](#), [528](#)
- DENCLUE, see [DENCLUE](#)
- density-based clustering, [644–656](#)
- fuzzy clustering, see [fuzzy clustering](#)
- graph-based clustering, [656–681](#)
 - sparsification, [657–658](#)
- grid-based clustering, see [grid-based clustering](#)
- hierarchical, see [hierarchical clustering](#)
 - CURE, see [CURE](#), see [CURE](#)
 - minimum spanning tree, [658–659](#)
- Jarvis-Patrick, see [Jarvis-Patrick](#)
- K-means, see [K-means](#)
- mixture modes, see [mixture models](#)
- opossum, see [opossum](#)
- parameter selection, [567](#), [587](#), [619](#)
- prototype-based clustering, [621–644](#)

- seeshared nearest neighbor, density-based clustering, [679](#)
 - self-organizing maps, see [self-organizing maps](#)
 - spectral clustering, [666](#)
 - subspace clustering, see [subspace clustering](#)
 - subspace clusters, [618](#)
 - types of clusterings, [529–531](#)
 - complete, [531](#)
 - exclusive, [530](#)
 - fuzzy, [530](#)
 - hierarchical, [529](#)
 - overlapping, [530](#)
 - partial, [531](#)
 - partitional, [529](#)
 - types of clusters, [531–533](#)
 - conceptual, [533](#)
 - density-based, [532](#)
 - graph-based, [532](#)
 - prototype-based, [531](#)
 - well-separated, [531](#)
 - validation, see [cluster validation](#)
- cluster validation, [571–597](#)
 - assessment of measures, [594–596](#)
 - clustering tendency, [571, 588](#)
 - cohesion, [574–579](#)
 - cophenetic correlation, [586](#)
 - for individual clusters, [581](#)
 - for individual objects, [581](#)
 - hierarchical, [585, 594](#)
 - number of clusters, [587](#)
 - relative measures, [574](#)
 - separation, [574–578](#)

- silhouette coefficient, **581–582**
- supervised, **589–594**
 - classification measures, **590–592**
 - similarity measures, **592–594**
- supervised measures, **573**
- unsupervised, **574–589**
- unsupervised measures, **573**
- with proximity matrix, **582–585**
- codeword, **332**
- compaction factor, **400**
- concept hierarchy, **462**
- conditional independence, **229**
- confidence
 - factor, **196**
 - level, 857
 - measure, see **measure**
- confusion matrix, **118**
- constraint
 - maxgap, **475**
 - maxspan, **474**
 - mingap, **475**
 - timing, **473**
 - window size, **476**
- contingency table, **402**
- correlation
 - ϕ -coefficient, **406**
- coverage, **196**
- critical region, see **hypothesis testing, critical region**
- cross-validation, **165**
- CURE, **686–690**
 - algorithm, **686, 688**

- cluster feature, **684**
- clustering feature tree, **684**
- use of partitioning, **689–690**
- use of sampling, **688–689**
- curse of dimensionality, **292**

- dag, see **graph**
- data
 - attribute, see **attribute**
 - attribute types, **617**
 - cleaning, see **data quality**, **data cleaning**
 - distribution, **618**
 - high-dimensional, **616**
 - problems with similarity, **673**
 - market basket, **357**
 - mathematical properties, **617**
 - noise, **616**
 - object, see **object**
 - outliers, **616**
 - preprocessing, see **preprocessing**
 - quality, see **data quality**
 - scale, **617**
 - set, see **data set**
 - similarity, see **similarity**
 - size, **616**
 - sparse, **616**
 - transformations, see **transformations**
 - types, **617**
 - types of, **23**, **26–42**
- data quality, **23**, **42–50**
 - application issues, **49–50**
 - data documentation, **50**
 - relevance, **49**
 - timeliness, **49**
 - data cleaning, **42**
 - errors, **43–48**
 - accuracy, **45**

- artifacts, [44](#)
- bias, [45](#)
- collection, [43](#)
- duplicate data, [48](#)
- inconsistent values, [47–48](#)
- measurement, [43](#)
- missing values, [46–47](#)
- noise, [43–44](#)
- outliers, [46](#)
- precision, [45](#)
- significant digits, [45](#)
- data set, [26](#)
 - characteristics, [34–35](#)
 - dimensionality, [34](#)
 - resolution, [35](#)
 - sparsity, [34](#)
 - types of, [34–42](#)
 - graph-based, [37–38](#)
 - matrix, see **matrix**
 - ordered, [38–41](#)
 - record, [35–37](#)
 - sequence, [40](#)
 - sequential, [38](#)
 - spatial, [41](#)
 - temporal, [38](#)
 - time series, [39](#)
 - transaction, [36](#)
- DBSCAN, [565–569](#)
 - algorithm, [567](#)
 - comparison to K-means, [614–615](#)
 - complexity, [567](#)

- definition of density, **565**
- parameter selection, **567**
- types of points, **566**
 - border, **566**
 - core, **566**
 - noise, **566**
- decision
 - boundary, **146**
 - list, **198**
 - stump, **303**
 - tree, see **classifier**
- deduplication, **48**
- DENCLUE, **652–656**
 - algorithm, **653**
 - implementation issues, **654**
 - kernel density estimation, **654**
 - strengths and limitations, **654**
- dendrite, **249**
- density
 - center based, **565**
- dimension, see **attribute**
- dimensionality
 - curse, **57**
- dimensionality reduction, **56–58, 833–848**
 - factor analysis, **840–842**
 - FastMap, 845
 - ISOMAP, 845–847
 - issues, 847–848
 - Locally Linear Embedding, **842–844**
 - multidimensional scaling, 844–845
 - PCA, **58**

- SVD, [58](#)
- discretization, [63–69](#), [221](#)
 - association, see [association](#)
 - binarization, [64–65](#)
 - clustering, [456](#)
 - equal frequency, [456](#)
 - equal width, [456](#)
 - of binary attributes, see [discretization](#), [binarization](#)
 - of categorical variables, [68–69](#)
 - of continuous attributes, [65–68](#)
 - supervised, [66–68](#)
 - unsupervised, [65–66](#)
- dissimilarity, [76–78](#), [94–95](#)
 - choosing, [98–100](#)
 - definition of, [72](#)
 - distance, see [distance](#)
 - non-metric, [77](#)
 - transformations, [72–75](#)
- distance, [76–77](#)
 - city block, [76](#)
 - Euclidean, [76](#), [822](#)
 - Hamming, [332](#)
 - L1 norm, [76](#)
 - L2 norm, [76](#)
 - L^∞ , [76](#)
 - Lmax, [76](#)
 - Mahalanobis, [96](#)
 - Manhattan, [76](#)
 - metric, [77](#)
 - positivity, [77](#)
 - symmetry, [77](#)

- triangle inequality, [77](#)
- Minkowski, [76–77](#)
- supremum, [76](#)
- distribution
 - binomial, [162](#)
 - Gaussian, [162, 221](#)

- eager learner, see **learner**
- edge, **480**
- effect size, see **hypothesis testing, effect size**
- element, **466**
- EM algorithm, **631–637**
- ensemble method, see **classifier**
- entity, see **object**
- entropy, **67, 128**
 - use in discretization, see **discretization, supervised**
- error
 - error-correcting output coding, **331**
 - generalization, **156**
 - pessimistic, **158**
- error rate, **119**
- estimate error, **164**
- Euclidean distance, see **distance, Euclidean**
- evaluation
 - association, **401**
- exhaustive, **198**

- factor analysis, see **dimensionality reduction, factor analysis**
- False Discovery Rate, **778**
 - Benjamini-Hochberg FDR, **772**
- family-wise error rate, **768**
- FastMap, see **dimensionality reduction, FastMap**
- feature
 - irrelevant, **144**
- feature creation, **61–63**
 - feature extraction, **61–62**
 - mapping data to a new space, **62–63**
- feature extraction, see **feature creation, feature extraction**
- feature selection, **58–61**
 - architecture for, **59–60**
 - feature weighting, **61**
 - irrelevant features, **58**
 - redundant features, **58**
 - types of, **58–59**
 - embedded, **58**
 - filter, **59**
 - wrapper, **59**
- field, see **attribute**
- Fourier transform, **62**
- FP-growth, **393**
- FP-tree, see **tree**
- frequent subgraph, **479**
- fuzzy clustering, **621–626**
 - fuzzy c-means, **623–626**
 - algorithm, **623**
 - centroids, **624**
 - example, **626**
 - initialization, **624**

- SSE, [624](#)
- strengths and limitations, [626](#)
- weight update, [625](#)
- fuzzy sets, [622](#)
 - fuzzy psuedo-partition, [623](#)

- gain ratio, **135**
- generalization, see **rule**
- gini index, **128**
- graph, **480**
 - connected, **484**
 - directed acyclic, **462**
 - Laplacian, **667**
 - undirected, **484**
- grid-based clustering, **644–648**
 - algorithm, **645**
 - clusters, **646**
 - density, **645**
 - grid cells, **645**

- hierarchical clustering, **554–565**
 - agglomerative algorithm, **555**
 - centroid methods, **562**
 - cluster proximity, **555**
 - Lance-Williams formula, **562**
 - complete link, **555, 558–559**
 - complexity, **556**
 - group average, **555, 559–560**
 - inversions, **562**
 - MAX, see **complete link**
 - merging decisions, **564**
 - MIN, see **single link**
 - single link, **555, 558**
 - Ward's method, **561**
- high-dimensionality
 - seedata,high-dimensional, **616**
- holdout, **165**
- hypothesis
 - alternative, **459**, 858
 - null, **459**, 858
- hypothesis testing, **761**
 - critical region, **763**
 - effect size, **766**
 - power, **764**
 - Type I error, **763**
 - Type II error, **764**

- independence
 - conditional, [218](#)
- information gain
 - entropy-based, [131](#)
 - FOIL's, [201](#)
- interest, see [measure](#)
- ISOMAP, see [dimensionality reduction](#), [ISOMAP](#)
- isomorphism
 - definition, [481](#)
- item, see [attribute](#), [358](#)
 - competing, [494](#)
 - negative, [494](#)
- itemset, [359](#)
 - candidate, see [candidate](#)
 - closed, [386](#)
 - maximal, [384](#)

- Jarvis-Patrick, **676–678**
 - algorithm, **676**
 - example, **677**
 - strengths and limitations, **677**

- K-means, **534–553**
 - algorithm, **535–536**
 - bisecting, **547–548**
 - centroids, **537, 539**
 - choosing initial, **539–544**
 - comparison to dBSCAN, **614–615**
 - complexity, **544**
 - derivation, **549–553**
 - empty clusters, **544**
 - incremental, **546**
 - K-means++, **543–544**
 - limitations, **548–549**
 - objective functions, **537, 539**
 - outliers, **545**
 - reducing SEE, **545–546**
- kernel density estimation, **654**
- kernel function, **90–94**

- L1 norm, see [distance](#), [L1 norm](#)
- L2 norm, see [distance](#), [L2 norm](#)
- Lagrangian, [280](#)
- lazy learner, see [learner](#)
- learner
 - eager, [208](#), [211](#)
 - lazy, [208](#), [211](#)
- least squares, [831](#)
- leave-one-out, [167](#)
- lexicographic order, [371](#)
- linear algebra, [817–832](#)
 - matrix, see [matrix](#)
 - vector, see [vector](#)
- linear regression, [831](#)
- linear systems of equations, [831](#)
- linear transformation, see [matrix](#), [linear transformation](#)
- Locally Linear Embedding, see [dimensionality reduction](#), [Locally Linear Embedding](#)

- m-estimate, [224](#)
- majority voting, see [voting](#)
- Manhattan distance, see [distance](#), [Manhattan](#)
- margin
 - soft, [284](#)
- market basket data, see [data](#)
- matrix, [37](#), [823–829](#)
 - addition, [824–825](#)
 - column vector, [824](#)
 - confusion, see [confusion matrix](#)
 - definition, [823–824](#)
 - document-term, [37](#)
 - eigenvalue, [829](#)
 - eigenvalue decomposition, [829–830](#)
 - eigenvector, [829](#)
 - in data analysis, [831–832](#)
 - inverse, [828–829](#)
 - linear combination, [835](#)
 - linear transformations, [827–829](#)
 - column space, [828](#)
 - left nullspace, [828](#)
 - nullspace, [828](#)
 - projection, [827](#)
 - reflection, [827](#)
 - rotation, [827](#)
 - row space, [828](#)
 - scaling, [827](#)
 - multiplication, [825–827](#)
 - positive semidefinite, [835](#)
 - rank, [828](#)
 - row vector, [824](#)

- scalar multiplication, [825](#)
- singular value, [830](#)
- singular value decomposition, [830](#)
- singular vector, [830](#)
- sparse, [37](#)
- maxgap, see [constraint](#)
- maximum likelihood estimation, [629–631](#)
- maxspan, see [constraint](#)
- MDL, [160](#)
- mean, [222](#)
- measure
 - confidence, [360](#)
 - consistency, [408](#)
 - interest, [405](#)
 - IS, [406](#)
 - objective, [401](#)
 - properties, [409](#)
 - support, [360](#)
 - symmetric, [414](#)
- measurement, [27–32](#)
 - definition of, [27](#)
 - scale, [27
 - permissible transformations, \[30–31\]\(#\)
 - types, \[27–32\]\(#\)](#)
- metric
 - accuracy, [119](#)
- metrics
 - classification, [119](#)
- min-Apriori, [461](#)
- mingap, see [constraint](#)
- minimum description length, see [MDL](#)

- missing values, see **data quality, errors, missing values**
- mixture models, **627–637**
 - advantages and limitations, **637**
 - definition of, **627–629**
 - EM algorithm, **631–637**
 - maximum likelihood estimation, **629–631**
- model
 - comparison, **173**
 - descriptive, **116**
 - generalization, **118**
 - overfitting, **147**
 - predictive, **116**
 - selection, **156**
- model complexity
 - Occam's Razor
 - AIC, **157**
 - BIC, **157**
- monotonicity, **364**
- multiclass, **330**
- multidimensional scaling, see **dimensionality reduction, multidimensional scaling**
- multiple comparison, see **False Discovery Rate**
- multiple hypothesis testing, see **False Discovery Rate**
 - family-wise error rate, see **family-wise error rate**
- mutual exclusive, **198**
- mutual information, **88–89**

- nearest neighbor classifier, see **classifier**
- network
 - Bayesian, see **classifier**
 - multilayer, see **classifier**
 - neural, see **classifier**
- neuron, **249**
- node
 - internal, **120**
 - leaf, **120**
 - non-terminal, **120**
 - root, **120**
 - terminal, **120**
- noise, **211**
- null distribution, **758**
- null hypothesis, **757**

- object, [26](#)
- observation, see [object](#)
- Occam's razor, [157](#)
- OLAP, [51](#)
- opposum, [659–660](#)
 - algorithm, [660](#)
 - strengths and weaknesses, [660](#)
- outliers, see [data quality](#)
- overfitting, see [model](#), [149](#)

- p-value, [759](#)
- pattern
 - cross-support, [420](#)
 - hyperclique, [423](#)
 - infrequent, [493](#)
 - negative, [494](#)
 - negatively correlated, [495](#), [496](#)
 - sequential, see [sequential](#)
 - subgraph, see [subgraph](#)
- PCA, [833–836](#)
 - examples, [836](#)
 - mathematics, [834–835](#)
- perceptron, see [classifier](#)
- permutation test, [781](#)
- Piatesky-Shapiro
 - PS, [405](#)
- point, see [object](#)
- power, see [hypothesis testing](#), [power](#)
- Precision-Recall Curve, [328](#)
- precondition, [195](#)
- preprocessing, [23](#), [50–71](#)
 - aggregation, see [aggregation](#)
 - binarization, see [discretization](#), [binarization](#)
 - dimensionality reduction, [56](#)
 - discretization, see [discretization](#)
 - feature creation, see [feature creation](#)
 - feature selection, see [feature selection](#)
 - sampling, see [sampling](#)
 - transformations, see [transformations](#)
- proximity, [71–100](#)
 - choosing, [98–100](#)

- cluster, **555**
- definition of, **71**
- dissimilarity, see **dissimilarity**
- distance, see **distance**
- for simple attributes, **74–75**
- issues, **96–98**
 - attribute weights, **98**
 - combining proximities, **97–98**
 - correlation, **96**
 - standardization, **96**
- similarity, see **similarity**
- transformations, **72–74**
- pruning
 - post-pruning, **163**
 - prepruning, **162**

- random forest
 - see [classifier](#), **310**
- randomization, **781**
 - association patterns, **793–795**
- Receiver Operating Characteristic curve, see [ROC](#)
- record, see [object](#)
- reduced error pruning, **189, 346**
- regression
 - logistic, **243**
- ROC, **323**
- Rote classifier, see [classifier](#)
- rule
 - antecedent, **195**
 - association, **360**
 - candidate, see [candidate](#)
 - consequent, **195**
 - generalization, **458**
 - generation, **205, 362, 380, 458**
 - ordered, **198**
 - ordering, **206**
 - pruning, **202**
 - quantitative, **454**
 - discretization-based, **454**
 - non-discretization, **460**
 - statistics-based, **458**
 - redundant, **458**
 - specialization, **458**
 - validation, **459**
- rule set, **195**

- sample, see **object**
- sampling, **52–56, 314**
 - approaches, **53–54**
 - progressive, **55–56**
 - random, **53**
 - stratified, **54**
 - with replacement, **54**
 - without replacement, **54**
 - sample size, **54–55**
- scalability
 - clustering algorithms, **681–690**
 - BIRCH, **684–686**
 - CURE, **686–690**
 - general issues, **681–684**
- segmentation, **529**
- self-organizing maps, **637–644**
 - algorithm, **638–641**
 - applications, **643**
 - strengths and limitations, **643**
- sensitivity, **319**
- sequence
 - data sequence, **468**
 - definition, **466**
- sequential
 - pattern, **464**
 - pattern discovery, **468**
 - timing constraints, see **constraint**
- sequential covering, **199**
- shared nearest neighbor, **656**
 - density, **678–679**
 - density-based clustering, **679–681**

- algorithm, **680**
- example, **680**
- strengths and limitations, **681**
- principle, **657**
- similarity, **673–676**
 - computation, **675**
 - differences in density, **674**
 - versus direct similarity, **676**
- significance
 - level, 859
- significance testing, **761**
 - null distribution, see **null distribution**
 - null hypothesis, see **null hypothesis**
 - p-value, see **p-value**
 - statistical significance, see **statistical significance**
- similarity, **24, 78–85**
 - choosing, **98–100**
 - correlation, **83–85**
 - cosine, **81–82, 822**
 - definition of, **72**
 - differences, **85–88**
 - extended Jaccard, **83**
 - Jaccard, **80–81**
 - kernel function, **90–94**
 - mutual information, **88–89**
 - shared nearest neighbor, see **shared nearest neighbor, similarity**
 - simple matching coefficient, **80**
 - Tanimoto, **83**
 - transformations, **72–75**
- Simpson’s paradox, **416**
- soft splitting, **178**

- SOM, **618**, see **self-organizing maps**
- specialization, see **rule**
- split information, **135**
- statistical significance, **760**
- statistics
 - covarinace matrix, **834**
- subgraph
 - core, **487**
 - definition, **482**
 - pattern, **479**
 - support, see **support**
- subsequence, **467**
 - contiguous, **475**
- subspace clustering, **648–652**
 - CLIQUE, **650**
 - algorithm, **651**
 - monotonicity property, **651**
 - strengths and limitations, **652**
 - example, **648**
- subtree
 - replacement, **163**
- support
 - count, **359**
 - counting, **373, 473, 477, 493**
 - limitation, **402**
 - measure, see **measure**
 - pruning, **364**
 - sequence, **468**
 - subgraph, **483**
- support vector, **276**
- support vector machine, see **classifier**

- SVD, **838–840**
 - example, **838–840**
 - mathematics, **838**
- SVM, see **classifier**
 - nonlinear, **290**
- svm
 - non-separable, **284**
- synapse, **249**

- taxonomy, see **concept hierarchy**
- transaction, **358**
 - extended, **463**
 - width, **379**
- transformations, **69–71**
 - between similarity and dissimilarity, **72–74**
 - normalization, **70–71**
 - simple functions, **70**
 - standardization, **70–71**
- tree
 - conditional FP-tree, **398**
 - decision, see **classifier**
 - FP-tree, **394**
 - hash, **375**
 - oblique, **146**
- triangle inequality, **77**
- true positive, **319**
- Type I error, see **hypothesis testing**, **Type I error**
- Type II error, see **hypothesis testing**, **Type II error**

- underfitting, **149**
- universal approximator, **261**

- variable, see **attribute**
- variance, **222**
- vector, **817–823**
 - addition, **817–818**
 - column, see **matrix, column vector**
 - definition, **817**
 - dot product, **820–822**
 - in data analysis, **822–823**
 - linear independence, **821–822**
 - mean, **823**
 - multiplication by a scalar, **818–819**
 - norm, **820**
 - orthogonal, **819–821**
 - orthogonal projection, **821**
 - row, see **matrix, row vector**
 - space, **819–820**
 - basis, **819**
 - dimension, **819**
 - independent components, **819**
 - linear combination, **819**
 - span, **819**
- vector quantization, **527**
- vertex, **480**
- voting
 - distance-weighted, **210**
 - majority, **210**

- wavelet transform, [63](#)
- web crawler, [138](#)
- window size, see [constraint](#)

Copyright Permissions

Some figures and part of the text of Chapter 8 originally appeared in the article “Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data,” Levent Ertöz, Michael Steinbach, and Vipin Kumar, *Proceedings of the Third SIAM International Conference on Data Mining*, San Francisco, CA, May 1–3, 2003, SIAM. ©2003, SIAM.

Some figures and part of the text of Chapter 5 appeared in the article “Selecting the Right Objective Measure for Association Analysis,” Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava, *Information Systems*, 29(4), 293–313, 2004, Elsevier. ©2004, Elsevier.

Some of the figures and text of Chapters 8 appeared in the article “Discovery of Climate Indices Using Clustering,” Michael Steinbach, Pang-Ning Tan, Vipin Kumar, Steven Klooster, and Christopher Potter, *KDD ’03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 446–455, Washington, DC, August 2003, ACM. ©2003, ACM, INC. DOI = <http://doi.acm.org/10.1145/956750.956801>

Some of the figures (1-7,13) and text of Chapter 7 originally appeared in the chapter “The Challenge of Clustering High-Dimensional Data,” Levent Ertoz, Michael Steinbach, and Vipin Kumar in *New Directions in Statistical Physics, Econophysics, Bioinformatics, and Pattern Recognition*, 273–312, Editor, Luc Wille, Springer, ISBN 3-540-43182-9. ©2004, Springer-Verlag.

Some of the figures and text of Chapter 8 originally appeared in the article “Chameleon: Hierarchical Clustering Using Dynamic Modeling,” by George

Karypis, Eui-Hong (Sam) Han, and Vipin Kumar, IEEE Computer, Volume 32(8), 68-75, August, 1999, IEEE. ©1999, IEEE.

Contents

1. INTRODUCTION TO DATA MINING
2. INTRODUCTION TO DATA MINING
3. Preface to the Second Edition
 - A. Overview
 - B. What is New in the Second Edition?
 - C. To the Instructor
 - D. Support Materials
4. Contents
5. 1 Introduction
 - A. 1.1 What Is Data Mining?
 - B. 1.2 Motivating Challenges
 - C. 1.3 The Origins of Data Mining
 - D. 1.4 Data Mining Tasks
 - E. 1.5 Scope and Organization of the Book
 - F. 1.6 Bibliographic Notes
 - G. Bibliography
 - H. 1.7 Exercises
6. 2 Data
 - A. 2.1 Types of Data
 1. 2.1.1 Attributes and Measurement
 - a. What Is an Attribute?
 - b. The Type of an Attribute
 - c. The Different Types of Attributes
 - d. Describing Attributes by the Number of Values
 - e. Asymmetric Attributes

- f. General Comments on Levels of Measurement
- 2. 2.1.2 Types of Data Sets
 - a. General Characteristics of Data Sets
 - a. Dimensionality
 - b. Distribution
 - c. Resolution
 - b. Record Data
 - a. Transaction or Market Basket Data
 - b. The Data Matrix
 - c. The Sparse Data Matrix
 - c. Graph-Based Data
 - a. Data with Relationships among Objects
 - b. Data with Objects That Are Graphs
 - d. Ordered Data
 - a. Sequential Transaction Data
 - b. Time Series Data
 - c. Sequence Data
 - d. Spatial and Spatio-Temporal Data
 - e. Handling Non-Record Data
- B. 2.2 Data Quality
 - 1. 2.2.1 Measurement and Data Collection Issues
 - a. Measurement and Data Collection Errors
 - b. Noise and Artifacts
 - c. Precision, Bias, and Accuracy
 - d. Outliers

- e. **Missing Values**
 - a. **Eliminate Data Objects or Attributes**
 - b. **Estimate Missing Values**
 - c. **Ignore the Missing Value during Analysis**
 - f. **Inconsistent Values**
 - g. **Duplicate Data**
2. **2.2.2 Issues Related to Applications**
- C. **2.3 Data Preprocessing**
- 1. **2.3.1 Aggregation**
 - 2. **2.3.2 Sampling**
 - a. **Sampling Approaches**
 - b. **Progressive Sampling**
 - 3. **2.3.3 Dimensionality Reduction**
 - a. **The Curse of Dimensionality**
 - b. **Linear Algebra Techniques for Dimensionality Reduction**
 - 4. **2.3.4 Feature Subset Selection**
 - a. **An Architecture for Feature Subset Selection**
 - b. **Feature Weighting**
 - 5. **2.3.5 Feature Creation**
 - a. **Feature Extraction**
 - b. **Mapping the Data to a New Space**
 - 6. **2.3.6 Discretization and Binarization**
 - a. **Binarization**
 - b. **Discretization of Continuous Attributes**

- a. **Unsupervised Discretization**
 - b. **Supervised Discretization**
 - c. **Categorical Attributes with Too Many Values**
7. **2.3.7 Variable Transformation**
- a. **Simple Functions**
 - b. **Normalization or Standardization**
- D. **2.4 Measures of Similarity and Dissimilarity**
- 1. **2.4.1 Basics**
 - a. **Definitions**
 - b. **Transformations**
 - 2. **2.4.2 Similarity and Dissimilarity between Simple Attributes**
 - 3. **2.4.3 Dissimilarities between Data Objects**
 - a. **Distances**
 - 4. **2.4.4 Similarities between Data Objects**
 - 5. **2.4.5 Examples of Proximity Measures**
 - a. **Similarity Measures for Binary Data**
 - a. **Simple Matching Coefficient**
 - b. **Jaccard Coefficient**
 - b. **Cosine Similarity**
 - c. **Extended Jaccard Coefficient (Tanimoto Coefficient)**
 - d. **Correlation**
 - e. **Differences Among Measures For Continuous Attributes**
 - 6. **2.4.6 Mutual Information**
 - 7. **2.4.7 Kernel Functions***

8. **2.4.8 Bregman Divergence***
 9. **2.4.9 Issues in Proximity Calculation**
 - a. **Standardization and Correlation for Distance Measures**
 - b. **Combining Similarities for Heterogeneous Attributes**
 - c. **Using Weights**
 10. **2.4.10 Selecting the Right Proximity Measure**
-
- E. **2.5 Bibliographic Notes**
 - F. **Bibliography**
 - G. **2.6 Exercises**
-
7. **3 Classification: Basic Concepts and Techniques**
 - A. **3.1 Basic Concepts**
 - B. **3.2 General Framework for Classification**
 - C. **3.3 Decision Tree Classifier**
 1. **3.3.1 A Basic Algorithm to Build a Decision Tree**
 - a. **Hunt's Algorithm**
 - b. **Design Issues of Decision Tree Induction**
 2. **3.3.2 Methods for Expressing Attribute Test Conditions**
 3. **3.3.3 Measures for Selecting an Attribute Test Condition**
 - a. **Impurity Measure for a Single Node**
 - b. **Collective Impurity of Child Nodes**
 - c. **Identifying the best attribute test condition**
 - d. **Splitting of Qualitative Attributes**
 - e. **Binary Splitting of Qualitative Attributes**
 - f. **Binary Splitting of Quantitative Attributes**
 - g. **Gain Ratio**
 4. **3.3.4 Algorithm for Decision Tree Induction**

5. **3.3.5 Example Application: Web Robot Detection**
6. **3.3.6 Characteristics of Decision Tree Classifiers**

D. 3.4 Model Overfitting

1. **3.4.1 Reasons for Model Overfitting**
 - a. **Limited Training Size**
 - b. **High Model Complexity**

E. 3.5 Model Selection

1. **3.5.1 Using a Validation Set**
2. **3.5.2 Incorporating Model Complexity**
 - a. **Estimating the Complexity of Decision Trees**
 - b. **Minimum Description Length Principle**
3. **3.5.3 Estimating Statistical Bounds**
4. **3.5.4 Model Selection for Decision Trees**

F. 3.6 Model Evaluation

1. **3.6.1 Holdout Method**
2. **3.6.2 Cross-Validation**

G. 3.7 Presence of Hyper-parameters

1. **3.7.1 Hyper-parameter Selection**
2. **3.7.2 Nested Cross-Validation**

H. 3.8 Pitfalls of Model Selection and Evaluation

1. **3.8.1 Overlap between Training and Test Sets**
2. **3.8.2 Use of Validation Error as Generalization Error**

I. 3.9 Model Comparison*

1. **3.9.1 Estimating the Confidence Interval for Accuracy**

2. 3.9.2 Comparing the Performance of Two Models

J. **3.10 Bibliographic Notes**

K. **Bibliography**

L. **3.11 Exercises**

8. 4 Classification: Alternative Techniques

A. **4.1 Types of Classifiers**

B. **4.2 Rule-Based Classifier**

1. **4.2.1 How a Rule-Based Classifier Works**

2. **4.2.2 Properties of a Rule Set**

3. **4.2.3 Direct Methods for Rule Extraction**

a. **Learn-One-Rule Function**

a. **Rule Pruning**

b. **Building the Rule Set**

b. **Instance Elimination**

4. **4.2.4 Indirect Methods for Rule Extraction**

5. **4.2.5 Characteristics of Rule-Based Classifiers**

C. **4.3 Nearest Neighbor Classifiers**

1. **4.3.1 Algorithm**

2. **4.3.2 Characteristics of Nearest Neighbor Classifiers**

D. **4.4 Naïve Bayes Classifier**

1. **4.4.1 Basics of Probability Theory**

a. **Bayes Theorem**

b. **Using Bayes Theorem for Classification**

2. **4.4.2 Naïve Bayes Assumption**

- a. **Conditional Independence**
- b. **How a Naïve Bayes Classifier Works**
 - Estimating Conditional Probabilities for Categorical Attributes**
 - Estimating Conditional Probabilities for Continuous Attributes**
- c. **Attributes**
- d. **Handling Zero Conditional Probabilities**
- e. **Characteristics of Naïve Bayes Classifiers**

E. 4.5 Bayesian Networks

- 1. **4.5.1 Graphical Representation**
 - a. **Conditional Independence**
 - b. **Joint Probability**
 - c. **Use of Hidden Variables**
- 2. **4.5.2 Inference and Learning**
 - a. **Variable Elimination**
 - b. **Sum-Product Algorithm for Trees**
 - c. **Generalizations for Non-Tree Graphs**
 - d. **Learning Model Parameters**
- 3. **4.5.3 Characteristics of Bayesian Networks**

F. 4.6 Logistic Regression

- 1. **4.6.1 Logistic Regression as a Generalized Linear Model**
- 2. **4.6.2 Learning Model Parameters**
- 3. **4.6.3 Characteristics of Logistic Regression**

G. 4.7 Artificial Neural Network (ANN)

- 1. **4.7.1 Perceptron**
 - a. **Learning the Perceptron**

2. **4.7.2 Multi-layer Neural Network**
 - a. **Learning Model Parameters**

3. **4.7.3 Characteristics of ANN**

H. **4.8 Deep Learning**

1. **4.8.1 Using Synergistic Loss Functions**
 - a. **Saturation of Outputs**
 - b. **Cross entropy loss function**
2. **4.8.2 Using Responsive Activation Functions**
 - a. **Vanishing Gradient Problem**
 - b. **Rectified Linear Units (ReLU)**
3. **4.8.3 Regularization**
 - a. **Dropout**
4. **4.8.4 Initialization of Model Parameters**
 - a. **Supervised Pretraining**
 - b. **Unsupervised Pretraining**
 - c. **Use of Autoencoders**
 - d. **Hybrid Pretraining**
5. **4.8.5 Characteristics of Deep Learning**

I. **4.9 Support Vector Machine (SVM)**

1. **4.9.1 Margin of a Separating Hyperplane**
 - a. **Rationale for Maximum Margin**
2. **4.9.2 Linear SVM**
 - a. **Learning Model Parameters**

- 3. **4.9.3 Soft-margin SVM**
 - a. **SVM as a Regularizer of Hinge Loss**
 - 4. **4.9.4 Nonlinear SVM**
 - a. **Attribute Transformation**
 - b. **Learning a Nonlinear SVM Model**
 - 5. **4.9.5 Characteristics of SVM**
- J. **4.10 Ensemble Methods**
- 1. **4.10.1 Rationale for Ensemble Method**
 - 2. **4.10.2 Methods for Constructing an Ensemble Classifier**
 - 3. **4.10.3 Bias-Variance Decomposition**
 - 4. **4.10.4 Bagging**
 - 5. **4.10.5 Boosting**
 - a. **AdaBoost**
- 6. **4.10.6 Random Forests**
 - 7. **4.10.7 Empirical Comparison among Ensemble Methods**
- K. **4.11 Class Imbalance Problem**
- 1. **4.11.1 Building Classifiers with Class Imbalance**
 - a. **Oversampling and Undersampling**
 - b. **Assigning Scores to Test Instances**
 - 2. **4.11.2 Evaluating Performance with Class Imbalance**
 - 3. **4.11.3 Finding an Optimal Score Threshold**
 - 4. **4.11.4 Aggregate Evaluation of Performance**
 - a. **ROC Curve**
 - b. **Precision-Recall Curve**

- L. **4.12 Multiclass Problem**
 - M. **4.13 Bibliographic Notes**
 - N. **Bibliography**
 - O. **4.14 Exercises**
9. **5 Association Analysis: Basic Concepts and Algorithms**
- A. **5.1 Preliminaries**
 - B. **5.2 Frequent Itemset Generation**
 - 1. **5.2.1 The *Apriori* Principle**
 - 2. **5.2.2 Frequent Itemset Generation in the *Apriori* Algorithm**
 - 3. **5.2.3 Candidate Generation and Pruning**
 - a. **Candidate Generation**
 - a. **Brute-Force Method**
 - b. **F_{k-1}xF₁ Method**
 - c. **F_{k-1}xF_{k-1} Method**
 - b. **Candidate Pruning**
 - 4. **5.2.4 Support Counting**
 - a. **Support Counting Using a Hash Tree***
 - 5. **5.2.5 Computational Complexity**
 - C. **5.3 Rule Generation**
 - 1. **5.3.1 Confidence-Based Pruning**
 - 2. **5.3.2 Rule Generation in *Apriori* Algorithm**
 - 3. **5.3.3 An Example: Congressional Voting Records**
 - D. **5.4 Compact Representation of Frequent Itemsets**
 - 1. **5.4.1 Maximal Frequent Itemsets**
 - 2. **5.4.2 Closed Itemsets**

- E. **5.5 Alternative Methods for Generating Frequent Itemsets***
- F. **5.6 FP-Growth Algorithm***
 - 1. **5.6.1 FP-Tree Representation**
 - 2. **5.6.2 Frequent Itemset Generation in FP-Growth Algorithm**
- G. **5.7 Evaluation of Association Patterns**
 - 1. **5.7.1 Objective Measures of Interestingness**
 - a. **Alternative Objective Interestingness Measures**
 - b. **Consistency among Objective Measures**
 - c. **Properties of Objective Measures**
 - a. **Inversion Property**
 - b. **Scaling Property**
 - c. **Null Addition Property**
 - d. **Asymmetric Interestingness Measures**
 - 2. **5.7.2 Measures beyond Pairs of Binary Variables**
 - 3. **5.7.3 Simpson's Paradox**
- H. **5.8 Effect of Skewed Support Distribution**
 - I. **5.9 Bibliographic Notes**
 - J. **Bibliography**
 - K. **5.10 Exercises**
- O. **6 Association Analysis: Advanced Concepts**
 - A. **6.1 Handling Categorical Attributes**
 - B. **6.2 Handling Continuous Attributes**
 - 1. **6.2.1 Discretization-Based Methods**
 - 2. **6.2.2 Statistics-Based Methods**
 - a. **Rule Generation**
 - b. **Rule Validation**

3. 6.2.3 Non-discretization Methods

C. 6.3 Handling a Concept Hierarchy

D. 6.4 Sequential Patterns

1. 6.4.1 Preliminaries

a. **Sequences**

b. **Subsequences**

2. 6.4.2 Sequential Pattern Discovery

3. 6.4.3 Timing Constraints*

a. **The maxspan Constraint**

b. **The mingap and maxgap Constraints**

c. **The Window Size Constraint**

4. 6.4.4 Alternative Counting Schemes*

E. 6.5 Subgraph Patterns

1. 6.5.1 Preliminaries

a. **Graphs**

b. **Graph Isomorphism**

c. **Subgraphs**

2. 6.5.2 Frequent Subgraph Mining

3. 6.5.3 Candidate Generation

4. 6.5.4 Candidate Pruning

5. 6.5.5 Support Counting

F. 6.6 Infrequent Patterns*

1. 6.6.1 Negative Patterns

2. 6.6.2 Negatively Correlated Patterns

6.6.3 Comparisons among Infrequent Patterns, Negative

- 3. Patterns, and Negatively Correlated Patterns
 - 4. 6.6.4 Techniques for Mining Interesting Infrequent Patterns
 - 5. 6.6.5 Techniques Based on Mining Negative Patterns
 - 6. 6.6.6 Techniques Based on Support Expectation
 - a. Support Expectation Based on Concept Hierarchy
 - b. Support Expectation Based on Indirect Association

G. 6.7 Bibliographic Notes

H. Bibliography

I. 6.8 Exercises

1. 7 Cluster Analysis: Basic Concepts and Algorithms

A. 7.1 Overview

1. **7.1.1 What Is Cluster Analysis?**
 2. **7.1.2 Different Types of Clusterings**
 3. **7.1.3 Different Types of Clusters**
 - a. **Road Map**

B. 7.2 K-means

1. **7.2.1 The Basic K-means Algorithm**
 - a. **Assigning Points to the Closest Centroid**
 - b. **Centroids and Objective Functions**
 - a. **Data in Euclidean Space**
 - b. **Document Data**
 - c. **The General Case**
 - c. **Choosing Initial Centroids**
 - a. **K-means++**
 - d. **Time and Space Complexity**

2. **7.2.2 K-means: Additional Issues**
 - a. **Handling Empty Clusters**
 - b. **Outliers**
 - c. **Reducing the SSE with Postprocessing**
 - d. **Updating Centroids Incrementally**
 3. **7.2.3 Bisecting K-means**
 4. **7.2.4 K-means and Different Types of Clusters**
 5. **7.2.5 Strengths and Weaknesses**
 6. **7.2.6 K-means as an Optimization Problem**
 - a. **Derivation of K-means as an Algorithm to Minimize the SSE**
 - b. **Derivation of K-means for SAE**
- C. **7.3 Agglomerative Hierarchical Clustering**
1. **7.3.1 Basic Agglomerative Hierarchical Clustering Algorithm**
 - a. **Defining Proximity between Clusters**
 - b. **Time and Space Complexity**
 2. **7.3.2 Specific Techniques**
 - a. **Sample Data**
 - b. **Single Link or MIN**
 - c. **Complete Link or MAX or CLIQUE**
 - d. **Group Average**
 - e. **Ward's Method and Centroid Methods**
 3. **7.3.3 The Lance-Williams Formula for Cluster Proximity**
 4. **7.3.4 Key Issues in Hierarchical Clustering**
 - a. **Lack of a Global Objective Function**
 - b. **Ability to Handle Different Cluster Sizes**
 - c. **Merging Decisions Are Final**

5. **7.3.5 Outliers**
6. **7.3.6 Strengths and Weaknesses**

D. **7.4 DBSCAN**

1. **7.4.1 Traditional Density: Center-Based Approach**
 - a. **Classification of Points According to Center-Based Density**
2. **7.4.2 The DBSCAN Algorithm**
 - a. **Time and Space Complexity**
 - b. **Selection of DBSCAN Parameters**
 - c. **Clusters of Varying Density**
 - d. **An Example**
3. **7.4.3 Strengths and Weaknesses**

E. **7.5 Cluster Evaluation**

1. **7.5.1 Overview**
7.5.2 Unsupervised Cluster Evaluation Using Cohesion and Separation
2. **Separation**
 - a. **Graph-Based View of Cohesion and Separation**
Relationship between Prototype-Based Cohesion and Graph-Based Cohesion
 - b. **Prototype-Based View of Cohesion and Separation**
Relationship of the Two Approaches to Prototype-Based Separation
 - c. **Graph-Based Cohesion**
Relationship between Cohesion and Separation
 - d. **Separation**
 - e. **Relationship between Cohesion and Separation**
 - f. **Relationship between Graph- and Centroid-Based Cohesion**
 - g. **Overall Measures of Cohesion and Separation**
 - h. **Evaluating Individual Clusters and Objects**
 - i. **The Silhouette Coefficient**

7.5.3 Unsupervised Cluster Evaluation Using the Proximity

3. Matrix

General Comments on Unsupervised Cluster Evaluation

- a. Measures**
- b. Measuring Cluster Validity via Correlation**
- c. Judging a Clustering Visually by Its Similarity Matrix**

4. 7.5.4 Unsupervised Evaluation of Hierarchical Clustering

5. 7.5.5 Determining the Correct Number of Clusters

6. 7.5.6 Clustering Tendency

7. 7.5.7 Supervised Measures of Cluster Validity

- a. Classification-Oriented Measures of Cluster Validity**
- b. Similarity-Oriented Measures of Cluster Validity**
- c. Cluster Validity for Hierarchical Clusterings**

8. 7.5.8 Assessing the Significance of Cluster Validity Measures

9. 7.5.9 Choosing a Cluster Validity Measure

F. 7.6 Bibliographic Notes

G. Bibliography

H. 7.7 Exercises

2. 8 Cluster Analysis: Additional Issues and Algorithms

A. 8.1 Characteristics of Data, Clusters, and Clustering Algorithms

1. 8.1.1 Example: Comparing K-means and DBSCAN

2. 8.1.2 Data Characteristics

3. 8.1.3 Cluster Characteristics

4. 8.1.4 General Characteristics of Clustering Algorithms

- a. Road Map**

B. 8.2 Prototype-Based Clustering

1. **8.2.1 Fuzzy Clustering**
 - a. **Fuzzy Sets**
 - b. **Fuzzy Clusters**
 - c. **Fuzzy c-means**
 - a. **Computing SSE**
 - b. **Initialization**
 - c. **Computing Centroids**
 - d. **Updating the Fuzzy Pseudo-partition**
 - d. **Strengths and Limitations**
 2. **8.2.2 Clustering Using Mixture Models**
 - a. **Mixture Models**
 - b. **Estimating Model Parameters Using Maximum Likelihood**
Estimating Mixture Model Parameters Using Maximum Likelihood
 - c. **Likelihood: The EM Algorithm**
Advantages and Limitations of Mixture Model Clustering
 - d. **Using the EM Algorithm**
 3. **8.2.3 Self-Organizing Maps (SOM)**
 - a. **The SOM Algorithm**
 - a. **Initialization**
 - b. **Selection of an Object**
 - c. **Assignment**
 - d. **Update**
 - e. **Termination**
 - b. **Applications**
 - c. **Strengths and Limitations**
- C. **8.3 Density-Based Clustering**

1. **8.3.1 Grid-Based Clustering**
 - a. **Defining Grid Cells**
 - b. **The Density of Grid Cells**
 - c. **Forming Clusters from Dense Grid Cells**
 - d. **Strengths and Limitations**
 2. **8.3.2 Subspace Clustering**
 - a. **CLIQUE**
 - b. **Strengths and Limitations of CLIQUE**
 - 8.3.3 DENCLUE: A Kernel-Based Scheme for Density-Based Clustering**
 3. **Clustering**
 - a. **Kernel Density Estimation**
 - b. **Implementation Issues**
 - c. **Strengths and Limitations of DENCLUE**
-
- D. **8.4 Graph-Based Clustering**
 1. **8.4.1 Sparsification**
 2. **8.4.2 Minimum Spanning Tree (MST) Clustering**
8.4.3 OPOSSUM: Optimal Partitioning of Sparse Similarities
 3. **Using METIS**
 - a. **Strengths and Weaknesses**
 - 8.4.4 Chameleon: Hierarchical Clustering with Dynamic Modeling**
 4. **Modeling**
 - a. **Deciding Which Clusters to Merge**
 - b. **Chameleon Algorithm**
 - a. **Sparsification**
 - c. **Graph Partitioning**
 - a. **Agglomerative Hierarchical Clustering**

- b. **Complexity**
 - d. **Strengths and Limitations**
 - 5. **8.4.5 Spectral Clustering**
 - Relationship between Spectral Clustering and Graph**
 - a. **Partitioning**
 - b. **Strengths and Limitations**
 - 6. **8.4.6 Shared Nearest Neighbor Similarity**
 - Problems with Traditional Similarity in High-Dimensional**
 - a. **Data**
 - b. **Problems with Differences in Density**
 - c. **SNN Similarity Computation**
 - d. **SNN Similarity versus Direct Similarity**
 - 7. **8.4.7 The Jarvis-Patrick Clustering Algorithm**
 - a. **Strengths and Limitations**
 - 8. **8.4.8 SNN Density**
 - 9. **8.4.9 SNN Density-Based Clustering**
 - a. **The SNN Density-based Clustering Algorithm**
 - b. **Strengths and Limitations**
- E. 8.5 Scalable Clustering Algorithms**
- 1. **8.5.1 Scalability: General Issues and Approaches**
 - 2. **8.5.2 BIRCH**
 - 3. **8.5.3 CURE**
 - a. **Sampling in CURE**
 - b. **Partitioning**

- F. **8.6 Which Clustering Algorithm?**
 - G. **8.7 Bibliographic Notes**
 - H. **Bibliography**
 - I. **8.8 Exercises**
3. **9 Anomaly Detection**
- A. **9.1 Characteristics of Anomaly Detection Problems**
 - 1. **9.1.1 A Definition of an Anomaly**
 - 2. **9.1.2 Nature of Data**
 - 3. **9.1.3 How Anomaly Detection is Used**
 - B. **9.2 Characteristics of Anomaly Detection Methods**
 - C. **9.3 Statistical Approaches**
 - 1. **9.3.1 Using Parametric Models**
 - a. **Using the Univariate Gaussian Distribution**
 - b. **Using the Multivariate Gaussian Distribution**
 - 2. **9.3.2 Using Non-parametric Models**
 - 3. **9.3.3 Modeling Normal and Anomalous Classes**
 - 4. **9.3.4 Assessing Statistical Significance**
 - 5. **9.3.5 Strengths and Weaknesses**
 - D. **9.4 Proximity-based Approaches**
 - 1. **9.4.1 Distance-based Anomaly Score**
 - 2. **9.4.2 Density-based Anomaly Score**
 - 3. **9.4.3 Relative Density-based Anomaly Score**
 - 4. **9.4.4 Strengths and Weaknesses**
 - E. **9.5 Clustering-based Approaches**
 - 1. **9.5.1 Finding Anomalous Clusters**
 - 2. **9.5.2 Finding Anomalous Instances**

Assessing the Extent to Which an Object Belongs to a

- a. **Cluster**
- b. **Impact of Outliers on the Initial Clustering**
- c. **The Number of Clusters to Use**

3. 9.5.3 Strengths and Weaknesses

F. 9.6 Reconstruction-based Approaches

1. 9.6.1 Strengths and Weaknesses

G. 9.7 One-class Classification

- 1. **9.7.1 Use of Kernels**
- 2. **9.7.2 The Origin Trick**
- 3. **9.7.3 Strengths and Weaknesses**

H. 9.8 Information Theoretic Approaches

1. 9.8.1 Strengths and Weaknesses

I. 9.9 Evaluation of Anomaly Detection

J. 9.10 Bibliographic Notes

K. Bibliography

L. 9.11 Exercises

4. 10 Avoiding False Discoveries

A. 10.1 Preliminaries: Statistical Testing

- 1. **10.1.1 Significance Testing**
 - a. **Null Hypothesis**
 - b. **Test Statistic**
 - c. **Null Distribution**
 - d. **Assessing Statistical Significance**

- 2. **10.1.2 Hypothesis Testing**
 - a. **Critical Region**
 - b. **Type I and Type II Errors**
 - c. **Effect Size**
 - 3. **10.1.3 Multiple Hypothesis Testing**
 - a. **Family-wise Error Rate (FWER)**
 - b. **Bonferroni Procedure**
 - c. **False discovery rate (FDR)**
 - d. **Benjamini-Hochberg Procedure**
 - 4. **10.1.4 Pitfalls in Statistical Testing**
- B. **10.2 Modeling Null and Alternative Distributions**
- 1. **10.2.1 Generating Synthetic Data Sets**
 - 2. **10.2.2 Randomizing Class Labels**
 - 3. **10.2.3 Resampling Instances**
 - 4. **10.2.4 Modeling the Distribution of the Test Statistic**
- C. **10.3 Statistical Testing for Classification**
- 1. **10.3.1 Evaluating Classification Performance**
 - 2. **10.3.2 Binary Classification as Multiple Hypothesis Testing**
 - 3. **10.3.3 Multiple Hypothesis Testing in Model Selection**
- D. **10.4 Statistical Testing for Association Analysis**
- 1. **10.4.1 Using Statistical Models**
 - a. **Using Fisher's Exact Test**
 - b. **Using the Chi-Squared Test**
 - 2. **10.4.2 Using Randomization Methods**

- E. **10.5 Statistical Testing for Cluster Analysis**
 - 1. **10.5.1 Generating a Null Distribution for Internal Indices**
 - 2. **10.5.2 Generating a Null Distribution for External Indices**
 - 3. **10.5.3 Enrichment**
 - F. **10.6 Statistical Testing for Anomaly Detection**
 - G. **10.7 Bibliographic Notes**
 - H. **Bibliography**
 - I. **10.8 Exercises**
-
- 5. **Author Index**
 - 6. **Subject Index**
 - 7. **Copyright Permissions**

List of Illustrations

- 1. **Figure 1.1.**
- 2. **Figure 1.2.**
- 3. **Figure 1.3.**
- 4. **Figure 1.4.**
- 5. **Figure 2.1.**
- 6. **Figure 2.2.**
- 7. **Figure 2.3.**
- 8. **Figure 2.4.**
- 9. **Figure 2.5.**
- 10. **Figure 2.6.**
- 11. **Figure 2.7.**
- 12. **Figure 2.8.**
- 13. **Figure 2.9.**

14. [Figure 2.10.](#)
15. [Figure 2.11.](#)
16. [Figure 2.12.](#)
17. [Figure 2.13.](#)
18. [Figure 2.14.](#)
19. [Figure 2.15.](#)
20. [Figure 2.16.](#)
21. [Figure 2.17.](#)
22. [Figure 2.19.](#)
23. [Figure 2.20.](#)
24. [Figure 2.21.](#)
25. [Figure 2.22.](#)
26. [Figure 3.1.](#)
27. [Figure 3.2.](#)
28. [Figure 3.3.](#)
29. [Figure 3.4.](#)
30. [Figure 3.5.](#)
31. [Figure 3.6.](#)
32. [Figure 3.7.](#)
33. [Figure 3.8.](#)
34. [Figure 3.9.](#)
35. [Figure 3.10.](#)
36. [Figure 3.11.](#)
37. [Figure 3.12.](#)
38. [Figure 3.13.](#)
39. [Figure 3.14.](#)
40. [Figure 3.15.](#)
41. [Figure 3.16.](#)
42. [Figure 3.17.](#)
43. [Figure 3.18.](#)
44. [Figure 3.19.](#)

45. [Figure 3.20.](#)
46. [Figure 3.21.](#)
47. [Figure 3.22.](#)
48. [Figure 3.23.](#)
49. [Figure 3.24.](#)
50. [Figure 3.25.](#)
51. [Figure 3.26.](#)
52. [Figure 3.27.](#)
53. [Figure 3.28.](#)
54. [Figure 3.29.](#)
55. [Figure 3.30.](#)
56. [Figure 3.31.](#)
57. [Figure 3.32.](#)
58. [Figure 3.33.](#)
59. [Figure 3.34.](#)
60. [Figure 3.35.](#)
61. [Figure 3.36.](#)
62. [Figure 3.37.](#)
63. [Figure 4.1.](#)
64. [Figure 4.2.](#)
65. [Figure 4.3.](#)
66. [Figure 4.4.](#)
67. [Figure 4.5.](#)
68. [Figure 4.6.](#)
69. [Figure 4.7.](#)
70. [Figure 4.8.](#)
71. [Figure 4.9.](#)
72. [Figure 4.10.](#)
73. [Figure 4.11.](#)
74. [Figure 4.12.](#)
75. [Figure 4.13.](#)

76. [Figure 4.14.](#)
77. [Figure 4.15.](#)
78. [Figure 4.16.](#)
79. [Figure 4.17.](#)
80. [Figure 4.18.](#)
81. [Figure 4.19.](#)
82. [Figure 4.20.](#)
83. [Figure 4.21.](#)
84. [Figure 4.22.](#)
85. [Figure 4.23.](#)
86. [Figure 4.24.](#)
87. [Figure 4.25.](#)
88. [Figure 4.26.](#)
89. [Figure 4.27.](#)
90. [Figure 4.28.](#)
91. [Figure 4.29.](#)
92. [Figure 4.30.](#)
93. [Figure 4.31.](#)
94. [Figure 4.32.](#)
95. [Figure 4.33.](#)
96. [Figure 4.34.](#)
97. [Figure 4.35.](#)
98. [Figure 4.36.](#)
99. [Figure 4.37.](#)
100. [Figure 4.38.](#)
101. [Figure 4.39.](#)
102. [Figure 4.40.](#)
103. [Figure 4.41.](#)
104. [Figure 4.42.](#)
105. [Figure 4.43.](#)
106. [Figure 4.44.](#)

107. [Figure 4.45.](#)
108. [Figure 4.46.](#)
109. [Figure 4.47.](#)
110. [Figure 4.48.](#)
111. [Figure 4.49.](#)
112. [Figure 4.50.](#)
113. [Figure 4.51.](#)
114. [Figure 4.52.](#)
115. [Figure 4.53.](#)
116. [Figure 4.54.](#)
117. [Figure 4.55.](#)
118. [Figure 4.56.](#)
119. [Figure 4.57.](#)
120. [Figure 4.58.](#)
121. [Figure 4.59.](#)
122. [Figure 5.1.](#)
123. [Figure 5.2.](#)
124. [Figure 5.3.](#)
125. [Figure 5.4.](#)
126. [Figure 5.5.](#)
127. [Figure 5.6.](#)
128. [Figure 5.7.](#)
129. [Figure 5.8.](#)
130. [Figure 5.9.](#)
131. [Figure 5.10.](#)
132. [Figure 5.11.](#)
133. [Figure 5.12.](#)
134. [Figure 5.13.](#)
135. [Figure 5.14.](#)
136. [Figure 5.15.](#)
137. [Figure 5.16.](#)

- 138. [Figure 5.17.](#)
- 139. [Figure 5.18.](#)
- 140. [Figure 5.19.](#)
- 141. [Figure 5.20.](#)
- 142. [Figure 5.21.](#)
- 143. [Figure 5.22.](#)
- 144. [Figure 5.23.](#)
- 145. [Figure 5.24.](#)
- 146. [Figure 5.25.](#)
- 147. [Figure 5.26.](#)
- 148. [Figure 5.27.](#)
- 149. [Figure 5.28.](#)
- 150. [Figure 5.29.](#)
- 151. [Figure 5.30.](#)
- 152. [Figure 5.31.](#)
- 153. [Figure 5.32.](#)
- 154. [Figure 5.33.](#)
- 155. [Figure 5.34.](#)
- 156. [Figure 6.1.](#)
- 157. [Figure 6.2.](#)
- 158. [Figure 6.3.](#)
- 159. [Figure 6.4.](#)
- 160. [Figure 6.5.](#)
- 161. [Figure 6.6.](#)
- 162. [Figure 6.7.](#)
- 163. [Figure 6.8.](#)
- 164. [Figure 6.9.](#)
- 165. [Figure 6.10.](#)
- 166. [Figure 6.11.](#)
- 167. [Figure 6.12.](#)
- 168. [Figure 6.13.](#)

- 169. [Figure 6.14.](#)
- 170. [Figure 6.15.](#)
- 171. [Figure 6.16.](#)
- 172. [Figure 6.17.](#)
- 173. [Figure 6.18.](#)
- 174. [Figure 6.19.](#)
- 175. [Figure 6.20.](#)
- 176. [Figure 6.21.](#)
- 177. [Figure 6.22.](#)
- 178. [Figure 6.23.](#)
- 179. [Figure 6.24.](#)
- 180. [Figure 6.25.](#)
- 181. [Figure 6.26.](#)
- 182. [Figure 6.27.](#)
- 183. [Figure 6.28.](#)
- 184. [Figure 6.29.](#)
- 185. [Figure 7.1.](#)
- 186. [Figure 7.2.](#)
- 187. [Figure 7.3.](#)
- 188. [Figure 7.4.](#)
- 189. [Figure 7.5.](#)
- 190. [Figure 7.6.](#)
- 191. [Figure 7.7.](#)
- 192. [Figure 7.8.](#)
- 193. [Figure 7.9.](#)
- 194. [Figure 7.10.](#)
- 195. [Figure 7.11.](#)
- 196. [Figure 7.12.](#)
- 197. [Figure 7.13.](#)
- 198. [Figure 7.14.](#)
- 199. [Figure 7.15.](#)

- 200. [Figure 7.16.](#)
- 201. [Figure 7.17.](#)
- 202. [Figure 7.18.](#)
- 203. [Figure 7.19.](#)
- 204. [Figure 7.20.](#)
- 205. [Figure 7.21.](#)
- 206. [Figure 7.22.](#)
- 207. [Figure 7.23.](#)
- 208. [Figure 7.24.](#)
- 209. [Figure 7.25.](#)
- 210. [Figure 7.26.](#)
- 211. [Figure 7.27.](#)
- 212. [Figure 7.28.](#)
- 213. [Figure 7.29.](#)
- 214. [Figure 7.30.](#)
- 215. [Figure 7.31.](#)
- 216. [Figure 7.32.](#)
- 217. [Figure 7.33.](#)
- 218. [Figure 7.34.](#)
- 219. [Figure 7.35.](#)
- 220. [Figure 7.36.](#)
- 221. [Figure 7.37.](#)
- 222. [Figure 7.38.](#)
- 223. [Figure 7.39.](#)
- 224. [Figure 7.40.](#)
- 225. [Figure 7.41.](#)
- 226. [Figure 8.1.](#)
- 227. [Figure 8.2.](#)
- 228. [Figure 8.3.](#)
- 229. [Figure 8.4.](#)
- 230. [Figure 8.5.](#)

- 231. [Figure 8.6.](#)
- 232. [Figure 8.7.](#)
- 233. [Figure 8.8.](#)
- 234. [Figure 8.9.](#)
- 235. [Figure 8.10.](#)
- 236. [Figure 8.11.](#)
- 237. [Figure 8.12.](#)
- 238. [Figure 8.13.](#)
- 239. [Figure 8.14.](#)
- 240. [Figure 8.15.](#)
- 241. [Figure 8.16.](#)
- 242. [Figure 8.17.](#)
- 243. [Figure 8.18.](#)
- 244. [Figure 8.19.](#)
- 245. [Figure 8.20.](#)
- 246. [Figure 8.21.](#)
- 247. [Figure 8.22.](#)
- 248. [Figure 8.23.](#)
- 249. [Figure 8.24.](#)
- 250. [Figure 8.25.](#)
- 251. [Figure 8.26.](#)
- 252. [Figure 8.27.](#)
- 253. [Figure 8.28.](#)
- 254. [Figure 8.29.](#)
- 255. [Figure 8.30.](#)
- 256. [Figure 8.31.](#)
- 257. [Figure 8.32.](#)
- 258. [Figure 9.1.](#)
- 259. [Figure 9.2.](#)
- 260. [Figure 9.3.](#)
- 261. [Figure 9.4.](#)

- 262. [Figure 9.5.](#)
- 263. [Figure 9.6.](#)
- 264. [Figure 9.7.](#)
- 265. [Figure 9.8.](#)
- 266. [Figure 9.9.](#)
- 267. [Figure 9.10.](#)
- 268. [Figure 9.11.](#)
- 269. [Figure 9.12.](#)
- 270. [Figure 9.13.](#)
- 271. [Figure 9.14.](#)
- 272. [Figure 9.15.](#)
- 273. [Figure 9.16.](#)
- 274. [Figure 9.17.](#)
- 275. [Figure 9.18.](#)
- 276. [Figure 9.19.](#)
- 277. [Figure 9.20.](#)
- 278. [Figure 10.1.](#)
- 279. [Figure 10.2.](#)
- 280. [Figure 10.3.](#)
- 281. [Figure 10.4.](#)
- 282. [Figure 10.5.](#)
- 283. [Figure 10.6.](#)
- 284. [Figure 10.7.](#)
- 285. [Figure 10.8.](#)
- 286. [Figure 10.9.](#)
- 287. [Figure 10.10.](#)
- 288. [Figure 10.11.](#)
- 289. [Figure 10.12.](#)
- 290. [Figure 10.13.](#)

List of Tables

1. **Table 1.1.** Market basket data.
2. **Table 1.2.** Collection of news articles.
3. **Table 2.1.** A sample data set containing student information.
4. **Table 2.2.** Different attribute types.
5. **Table 2.3.** Transformations that define attribute levels.
6. **Table 2.4.** Data set containing information about customer purchases.
7. **Table 2.5.** Conversion of a categorical attribute to three binary attributes.
8. **Table 2.6.** Conversion of a categorical attribute to five asymmetric binary attributes.
9. **Table 2.7.** Similarity and dissimilarity for simple attributes
10. **Table 2.8.** x and y coordinates of four points.
11. **Table 2.9.** Euclidean distance matrix for Table 2.8.
12. **Table 2.10.** L₁ distance matrix for Table 2.8.
13. **Table 2.11.** L_∞ distance matrix for Table 2.8.
14. **Table 2.12.** Properties of cosine, correlation, and Minkowski distance measures.
15. **Table 2.13.** Similarity between (x, y), (x, ys), and (x, yt).
16. **Table 2.14.** Entropy for x
17. **Table 2.15.** Entropy for y
18. **Table 2.16.** Joint entropy for x and y
19. **Table 3.1.** Examples of classification tasks.
20. **Table 3.2.** A sample data for the vertebrate classification problem.
21. **Table 3.3.** A sample data for the loan borrower classification problem.
22. **Table 3.4.** Confusion matrix for a binary classification problem.
23. **Table 3.5.** Data set for Exercise 2.

24. **Table 3.6.** Data set for Exercise 3.
25. **Table 3.7.** Comparing the test accuracy of decision trees T10 and T100.
26. **Table 3.8.** Comparing the accuracy of various classification methods.
27. **Table 4.1.** Example of a rule set for the vertebrate classification problem.
28. **Table 4.2.** The vertebrate data set.
29. **Table 4.3.** Example of a mutually exclusive and exhaustive rule set.
30. **Table 4.4.** Example of data set used to construct an ensemble of bagging classifiers.
31. **Table 4.5.** Comparing the accuracy of a decision tree classifier against three ensemble methods.
32. **Table 4.6.** A confusion matrix for a binary classification problem in which the classes are not equally important.
33. **Table 4.7.** Entries of the confusion matrix in terms of the TPR, TNR, skew, α , and total number of instances, N.
34. **Table 4.8.** Comparison of various rule-based classifiers.
35. **Table 4.9.** Data set for Exercise 7.
36. **Table 4.10.** Data set for Exercise 8.
37. **Table 4.11.** Data set for Exercise 10.
38. **Table 4.12.** Data set for Exercise 12.
39. **Table 4.13.** Posterior probabilities for Exercise 16.
40. **Table 5.1.** An example of market basket transactions.
41. **Table 5.2.** A binary 0/1 representation of market basket data.
42. **Table 5.3.** List of binary attributes from the 1984 United States Congressional Voting Records. Source: The UCI machine learning repository.
43. **Table 5.4.** Association rules extracted from the 1984 United States Congressional Voting Records.

44. **Table 5.5.** A transaction data set for mining closed itemsets.
45. **Table 5.6.** A 2-way contingency table for variables A and B.
46. **Table 5.7.** Beverage preferences among a group of 1000 people.
47. **Table 5.8.** Information about people who drink tea and people who use honey in their beverage.
48. **Table 5.9.** Examples of objective measures for the itemset {A, B}.
49. **Table 5.10.** Example of contingency tables.
50. **Table 5.11.** Rankings of contingency tables using the measures given in Table 5.9.
51. **Table 5.12.** Contingency tables for the pairs {p,q} and {r,s}.
52. **Table 5.13.** The grade-gender example. (a) Sample data of size 100.
53. **Table 5.14.** An example demonstrating the effect of null addition.
54. **Table 5.15.** Properties of symmetric measures.
55. **Table 5.16.** Example of a three-dimensional contingency table.
56. **Table 5.17.** A two-way contingency table between the sale of high-definition television and exercise machine.
57. **Table 5.18.** Example of a three-way contingency table.
58. **Table 5.19.** Grouping the items in the census data set based on their support values.
59. **Table 5.20.** Example of market basket transactions.
60. **Table 5.21.** Market basket transactions.
61. **Table 5.22.** Example of market basket transactions.
62. **Table 5.23.** Example of market basket transactions.
63. **Table 5.24.** A Contingency Table.
64. **Table 5.25.** Contingency tables for Exercise 20.
65. **Table 6.1.** Internet survey data with categorical attributes.
66. **Table 6.2.** Internet survey data after binarizing categorical and symmetric binary attributes.
67. **Table 6.3.** Internet survey data with continuous attributes.
68. **Table 6.4.** Internet survey data after binarizing categorical and continuous attributes.

69. **Table 6.5.** A breakdown of Internet users who participated in online chat according to their age group.
70. **Table 6.6.** Document-word matrix.
71. **Table 6.7.** Examples illustrating the concept of a subsequence.
72. **Table 6.8.** Graph representation of entities in various application domains.
73. **Table 6.9.** A two-way contingency table for the association rule $X \rightarrow Y$.
74. **Table 6.10.** Traffic accident data set.
75. **Table 6.11.** Data set for Exercise 2.
76. **Table 6.12.** Data set for Exercise 3.
77. **Table 6.13.** Data set for Exercise 4.
78. **Table 6.14.** Data set for Exercise 6.
79. **Table 6.15.** Example of market basket transactions.
80. **Table 6.16.** Example of event sequences generated by various sensors.
81. **Table 6.17.** Example of event sequence data for Exercise 14.
82. **Table 6.18.** Example of numeric data set.
83. **Table 7.1.** Table of notation.
84. **Table 7.2.** K-means: Common choices for proximity, centroids, and objective functions.
85. **Table 7.3.** xy-coordinates of six points.
86. **Table 7.4.** Euclidean distance matrix for six points.
87. **Table 7.5.** Table of Lance-Williams coefficients for common hierarchical clustering approaches.
88. **Table 7.6.** Table of graph-based cluster evaluation measures.
89. **Table 7.7.** Cophenetic distance matrix for single link and data in **Table 2.14** on page 90.
90. **Table 7.8.** Cophenetic correlation coefficient for data of **Table 2.14** and four agglomerative hierarchical clustering techniques.
91. **Table 7.9.** K-means clustering results for the LA Times document

data set.

92. **Table 7.10.** Ideal cluster similarity matrix.
93. **Table 7.11.** Class similarity matrix.
94. **Table 7.12.** Two-way contingency table for determining whether pairs of objects are in the same class and same cluster.
95. **Table 7.13.** Similarity matrix for Exercise 16.
96. **Table 7.14.** Confusion matrix for Exercise 21.
97. **Table 7.15.** Table of cluster labels for Exercise 24.
98. **Table 7.16.** Similarity matrix for Exercise 24.
99. **Table 8.1.** First few iterations of the EM algorithm for the simple example.
100. **Table 8.2.** Point counts for grid cells.
101. **Table 8.3.** Similarity among documents in different sections of a newspaper.
102. **Table 8.4.** Two nearest neighbors of four points.
103. **Table 9.1.** Sample pairs $(c,\alpha), \alpha = \text{prob}(|x| \geq c)$ for a Gaussian distribution with mean 0 and standard deviation 1.
104. **Table 9.2.** Survey data of weight and height of 100 participants.
105. **Table 10.1.** Confusion table in the context of multiple hypothesis testing.
106. **Table 10.2.** Correspondence between statistical testing concepts and classifier evaluation measures
107. **Table 10.3.** A 2-way contingency table for variables A and B.
108. **Table 10.4.** Beverage preferences among a group of 1000 people.
109. **Table 10.5.** Beverage preferences among a group of 1000 people.
110. **Table 10.6.** Contingency table for an anomaly detection system with detection rate d and false alarm rate f.
111. **Table 10.7.** Beverage preferences among a group of 100 people (left) and 10,000 people (right).
112. **Table 10.8.** Ordered Collection of p-values..

Landmarks

1. **Frontmatter**
2. **Start of Content**
3. **backmatter**
4. **List of Illustrations**
5. **List of Tables**

1. **i**
2. **ii**
3. **iii**
4. **iv**
5. **v**
6. **vi**
7. **vii**
8. **viii**
9. **ix**
10. **x**
11. **xi**
12. **xii**
13. **xiii**
14. **xiv**
15. **xv**
16. **xvi**
17. **xvii**
18. **xviii**
19. **xix**
20. **xx**
21. **1**
22. **2**

- 23. **3**
- 24. **4**
- 25. **5**
- 26. **6**
- 27. **7**
- 28. **8**
- 29. **9**
- 30. **10**
- 31. **11**
- 32. **12**
- 33. **13**
- 34. **14**
- 35. **15**
- 36. **16**
- 37. **17**
- 38. **18**
- 39. **19**
- 40. **20**
- 41. **21**
- 42. **22**
- 43. **23**
- 44. **24**
- 45. **25**
- 46. **26**
- 47. **27**
- 48. **28**
- 49. **29**
- 50. **30**
- 51. **31**
- 52. **32**
- 53. **33**

- 54. **34**
- 55. **35**
- 56. **36**
- 57. **37**
- 58. **38**
- 59. **39**
- 60. **40**
- 61. **41**
- 62. **42**
- 63. **43**
- 64. **44**
- 65. **45**
- 66. **46**
- 67. **47**
- 68. **48**
- 69. **49**
- 70. **50**
- 71. **51**
- 72. **52**
- 73. **53**
- 74. **54**
- 75. **55**
- 76. **56**
- 77. **57**
- 78. **58**
- 79. **59**
- 80. **60**
- 81. **61**
- 82. **62**
- 83. **63**
- 84. **64**

- 85. **65**
- 86. **66**
- 87. **67**
- 88. **68**
- 89. **69**
- 90. **70**
- 91. **71**
- 92. **72**
- 93. **73**
- 94. **74**
- 95. **75**
- 96. **76**
- 97. **77**
- 98. **78**
- 99. **79**
- 100. **80**
- 101. **81**
- 102. **82**
- 103. **83**
- 104. **84**
- 105. **85**
- 106. **86**
- 107. **87**
- 108. **88**
- 109. **89**
- 110. **90**
- 111. **91**
- 112. **92**
- 113. **93**
- 114. **94**
- 115. **95**

- 116. **96**
- 117. **97**
- 118. **98**
- 119. **99**
- 120. **100**
- 121. **101**
- 122. **102**
- 123. **103**
- 124. **104**
- 125. **105**
- 126. **106**
- 127. **107**
- 128. **108**
- 129. **109**
- 130. **110**
- 131. **111**
- 132. **112**
- 133. **113**
- 134. **114**
- 135. **115**
- 136. **116**
- 137. **117**
- 138. **118**
- 139. **119**
- 140. **120**
- 141. **121**
- 142. **122**
- 143. **123**
- 144. **124**
- 145. **125**
- 146. **126**

- 147. **127**
- 148. **128**
- 149. **129**
- 150. **130**
- 151. **131**
- 152. **132**
- 153. **133**
- 154. **134**
- 155. **135**
- 156. **136**
- 157. **137**
- 158. **138**
- 159. **139**
- 160. **140**
- 161. **141**
- 162. **142**
- 163. **143**
- 164. **144**
- 165. **145**
- 166. **146**
- 167. **147**
- 168. **148**
- 169. **149**
- 170. **150**
- 171. **151**
- 172. **152**
- 173. **153**
- 174. **154**
- 175. **155**
- 176. **156**
- 177. **157**

- 178. **158**
- 179. **159**
- 180. **160**
- 181. **161**
- 182. **162**
- 183. **163**
- 184. **164**
- 185. **165**
- 186. **166**
- 187. **167**
- 188. **168**
- 189. **169**
- 190. **170**
- 191. **171**
- 192. **172**
- 193. **173**
- 194. **174**
- 195. **175**
- 196. **176**
- 197. **177**
- 198. **178**
- 199. **179**
- 200. **180**
- 201. **181**
- 202. **182**
- 203. **183**
- 204. **184**
- 205. **185**
- 206. **186**
- 207. **187**
- 208. **188**

- 209. **189**
- 210. **190**
- 211. **191**
- 212. **192**
- 213. **193**
- 214. **194**
- 215. **195**
- 216. **196**
- 217. **197**
- 218. **198**
- 219. **199**
- 220. **200**
- 221. **201**
- 222. **202**
- 223. **203**
- 224. **204**
- 225. **205**
- 226. **206**
- 227. **207**
- 228. **208**
- 229. **209**
- 230. **210**
- 231. **211**
- 232. **212**
- 233. **213**
- 234. **214**
- 235. **215**
- 236. **216**
- 237. **217**
- 238. **218**
- 239. **219**

- 240. **220**
- 241. **221**
- 242. **222**
- 243. **223**
- 244. **224**
- 245. **225**
- 246. **226**
- 247. **227**
- 248. **228**
- 249. **229**
- 250. **230**
- 251. **231**
- 252. **232**
- 253. **233**
- 254. **234**
- 255. **235**
- 256. **236**
- 257. **237**
- 258. **238**
- 259. **239**
- 260. **240**
- 261. **241**
- 262. **242**
- 263. **243**
- 264. **244**
- 265. **245**
- 266. **246**
- 267. **247**
- 268. **248**
- 269. **249**
- 270. **250**

- 271. **251**
- 272. **252**
- 273. **253**
- 274. **254**
- 275. **255**
- 276. **256**
- 277. **257**
- 278. **258**
- 279. **259**
- 280. **260**
- 281. **261**
- 282. **262**
- 283. **263**
- 284. **264**
- 285. **265**
- 286. **266**
- 287. **267**
- 288. **268**
- 289. **269**
- 290. **270**
- 291. **271**
- 292. **272**
- 293. **273**
- 294. **274**
- 295. **275**
- 296. **276**
- 297. **277**
- 298. **278**
- 299. **279**
- 300. **280**
- 301. **281**

- 302. **282**
- 303. **283**
- 304. **284**
- 305. **285**
- 306. **286**
- 307. **287**
- 308. **288**
- 309. **289**
- 310. **290**
- 311. **291**
- 312. **292**
- 313. **293**
- 314. **294**
- 315. **295**
- 316. **296**
- 317. **297**
- 318. **298**
- 319. **299**
- 320. **300**
- 321. **301**
- 322. **302**
- 323. **303**
- 324. **304**
- 325. **305**
- 326. **306**
- 327. **307**
- 328. **308**
- 329. **309**
- 330. **310**
- 331. **311**
- 332. **312**

- 333. **313**
- 334. **314**
- 335. **315**
- 336. **316**
- 337. **317**
- 338. **318**
- 339. **319**
- 340. **320**
- 341. **321**
- 342. **322**
- 343. **323**
- 344. **324**
- 345. **325**
- 346. **326**
- 347. **327**
- 348. **328**
- 349. **329**
- 350. **330**
- 351. **331**
- 352. **332**
- 353. **333**
- 354. **334**
- 355. **335**
- 356. **336**
- 357. **337**
- 358. **338**
- 359. **339**
- 360. **340**
- 361. **341**
- 362. **342**
- 363. **343**

- 364. **344**
- 365. **345**
- 366. **346**
- 367. **347**
- 368. **348**
- 369. **349**
- 370. **350**
- 371. **351**
- 372. **352**
- 373. **353**
- 374. **354**
- 375. **355**
- 376. **356**
- 377. **357**
- 378. **358**
- 379. **359**
- 380. **360**
- 381. **361**
- 382. **362**
- 383. **363**
- 384. **364**
- 385. **365**
- 386. **366**
- 387. **367**
- 388. **368**
- 389. **369**
- 390. **370**
- 391. **371**
- 392. **372**
- 393. **373**
- 394. **374**

- 395. **375**
- 396. **376**
- 397. **377**
- 398. **378**
- 399. **379**
- 400. **380**
- 401. **381**
- 402. **382**
- 403. **383**
- 404. **384**
- 405. **385**
- 406. **386**
- 407. **387**
- 408. **388**
- 409. **389**
- 410. **390**
- 411. **391**
- 412. **392**
- 413. **393**
- 414. **394**
- 415. **395**
- 416. **396**
- 417. **397**
- 418. **398**
- 419. **399**
- 420. **400**
- 421. **401**
- 422. **402**
- 423. **403**
- 424. **404**
- 425. **405**

- 426. **406**
- 427. **407**
- 428. **408**
- 429. **409**
- 430. **410**
- 431. **411**
- 432. **412**
- 433. **413**
- 434. **414**
- 435. **415**
- 436. **416**
- 437. **417**
- 438. **418**
- 439. **419**
- 440. **420**
- 441. **421**
- 442. **422**
- 443. **423**
- 444. **424**
- 445. **425**
- 446. **426**
- 447. **427**
- 448. **428**
- 449. **429**
- 450. **430**
- 451. **431**
- 452. **432**
- 453. **433**
- 454. **434**
- 455. **435**
- 456. **436**

- 457. **437**
- 458. **438**
- 459. **439**
- 460. **440**
- 461. **441**
- 462. **442**
- 463. **443**
- 464. **444**
- 465. **445**
- 466. **446**
- 467. **447**
- 468. **448**
- 469. **449**
- 470. **450**
- 471. **451**
- 472. **452**
- 473. **453**
- 474. **454**
- 475. **455**
- 476. **456**
- 477. **457**
- 478. **458**
- 479. **459**
- 480. **460**
- 481. **461**
- 482. **462**
- 483. **463**
- 484. **464**
- 485. **465**
- 486. **466**
- 487. **467**

- 488. **468**
- 489. **469**
- 490. **470**
- 491. **471**
- 492. **472**
- 493. **473**
- 494. **474**
- 495. **475**
- 496. **476**
- 497. **477**
- 498. **478**
- 499. **479**
- 500. **480**
- 501. **481**
- 502. **482**
- 503. **483**
- 504. **484**
- 505. **485**
- 506. **486**
- 507. **487**
- 508. **488**
- 509. **489**
- 510. **490**
- 511. **491**
- 512. **492**
- 513. **493**
- 514. **494**
- 515. **495**
- 516. **496**
- 517. **497**
- 518. **498**

- 519. **499**
- 520. **500**
- 521. **501**
- 522. **502**
- 523. **503**
- 524. **504**
- 525. **505**
- 526. **506**
- 527. **507**
- 528. **508**
- 529. **509**
- 530. **510**
- 531. **511**
- 532. **512**
- 533. **513**
- 534. **514**
- 535. **515**
- 536. **516**
- 537. **517**
- 538. **518**
- 539. **519**
- 540. **520**
- 541. **521**
- 542. **522**
- 543. **523**
- 544. **524**
- 545. **525**
- 546. **526**
- 547. **527**
- 548. **528**
- 549. **529**

- 550. **530**
- 551. **531**
- 552. **532**
- 553. **533**
- 554. **534**
- 555. **535**
- 556. **536**
- 557. **537**
- 558. **538**
- 559. **539**
- 560. **540**
- 561. **541**
- 562. **542**
- 563. **543**
- 564. **544**
- 565. **545**
- 566. **546**
- 567. **547**
- 568. **548**
- 569. **549**
- 570. **550**
- 571. **551**
- 572. **552**
- 573. **553**
- 574. **554**
- 575. **555**
- 576. **556**
- 577. **557**
- 578. **558**
- 579. **559**
- 580. **560**

- 581. **561**
- 582. **562**
- 583. **563**
- 584. **564**
- 585. **565**
- 586. **566**
- 587. **567**
- 588. **568**
- 589. **569**
- 590. **570**
- 591. **571**
- 592. **572**
- 593. **573**
- 594. **574**
- 595. **575**
- 596. **576**
- 597. **577**
- 598. **578**
- 599. **579**
- 600. **580**
- 601. **581**
- 602. **582**
- 603. **583**
- 604. **584**
- 605. **585**
- 606. **586**
- 607. **587**
- 608. **588**
- 609. **589**
- 610. **590**
- 611. **591**

- 612. **592**
- 613. **593**
- 614. **594**
- 615. **595**
- 616. **596**
- 617. **597**
- 618. **598**
- 619. **599**
- 620. **600**
- 621. **601**
- 622. **602**
- 623. **603**
- 624. **604**
- 625. **605**
- 626. **606**
- 627. **607**
- 628. **608**
- 629. **609**
- 630. **610**
- 631. **611**
- 632. **612**
- 633. **613**
- 634. **614**
- 635. **615**
- 636. **616**
- 637. **617**
- 638. **618**
- 639. **619**
- 640. **620**
- 641. **621**
- 642. **622**

- 643. **623**
- 644. **624**
- 645. **625**
- 646. **626**
- 647. **627**
- 648. **628**
- 649. **629**
- 650. **630**
- 651. **631**
- 652. **632**
- 653. **633**
- 654. **634**
- 655. **635**
- 656. **636**
- 657. **637**
- 658. **638**
- 659. **639**
- 660. **640**
- 661. **641**
- 662. **642**
- 663. **643**
- 664. **644**
- 665. **645**
- 666. **646**
- 667. **647**
- 668. **648**
- 669. **649**
- 670. **650**
- 671. **651**
- 672. **652**
- 673. **653**

- 674. **654**
- 675. **655**
- 676. **656**
- 677. **657**
- 678. **658**
- 679. **659**
- 680. **660**
- 681. **661**
- 682. **662**
- 683. **663**
- 684. **664**
- 685. **665**
- 686. **666**
- 687. **667**
- 688. **668**
- 689. **669**
- 690. **670**
- 691. **671**
- 692. **672**
- 693. **673**
- 694. **674**
- 695. **675**
- 696. **676**
- 697. **677**
- 698. **678**
- 699. **679**
- 700. **680**
- 701. **681**
- 702. **682**
- 703. **683**
- 704. **684**

- 705. **685**
- 706. **686**
- 707. **687**
- 708. **688**
- 709. **689**
- 710. **690**
- 711. **691**
- 712. **692**
- 713. **693**
- 714. **694**
- 715. **695**
- 716. **696**
- 717. **697**
- 718. **698**
- 719. **699**
- 720. **700**
- 721. **701**
- 722. **702**
- 723. **703**
- 724. **704**
- 725. **705**
- 726. **706**
- 727. **707**
- 728. **708**
- 729. **709**
- 730. **710**
- 731. **711**
- 732. **712**
- 733. **713**
- 734. **714**
- 735. **715**

- 736. **716**
- 737. **717**
- 738. **718**
- 739. **719**
- 740. **720**
- 741. **721**
- 742. **722**
- 743. **723**
- 744. **724**
- 745. **725**
- 746. **726**
- 747. **727**
- 748. **728**
- 749. **729**
- 750. **730**
- 751. **731**
- 752. **732**
- 753. **733**
- 754. **734**
- 755. **735**
- 756. **736**
- 757. **737**
- 758. **738**
- 759. **739**
- 760. **740**
- 761. **741**
- 762. **742**
- 763. **743**
- 764. **744**
- 765. **745**
- 766. **746**

- 767. **747**
- 768. **748**
- 769. **749**
- 770. **750**
- 771. **751**
- 772. **752**
- 773. **753**
- 774. **754**
- 775. **755**
- 776. **756**
- 777. **757**
- 778. **758**
- 779. **759**
- 780. **760**
- 781. **761**
- 782. **762**
- 783. **763**
- 784. **764**
- 785. **765**
- 786. **766**
- 787. **767**
- 788. **768**
- 789. **769**
- 790. **770**
- 791. **771**
- 792. **772**
- 793. **773**
- 794. **774**
- 795. **775**
- 796. **776**
- 797. **777**

- 798. **778**
- 799. **779**
- 800. **780**
- 801. **781**
- 802. **782**
- 803. **783**
- 804. **784**
- 805. **785**
- 806. **786**
- 807. **787**
- 808. **788**
- 809. **789**
- 810. **790**
- 811. **791**
- 812. **792**
- 813. **793**
- 814. **794**
- 815. **795**
- 816. **796**
- 817. **797**
- 818. **798**
- 819. **799**
- 820. **800**
- 821. **801**
- 822. **802**
- 823. **803**
- 824. **804**
- 825. **805**
- 826. **806**
- 827. **807**
- 828. **808**

- 829. **809**
- 830. **810**
- 831. **811**
- 832. **812**
- 833. **813**
- 834. **814**
- 835. **815**
- 836. **816**
- 837. **817**
- 838. **818**
- 839. **819**
- 840. **820**
- 841. **821**
- 842. **822**
- 843. **823**
- 844. **824**
- 845. **825**
- 846. **826**
- 847. **827**
- 848. **828**
- 849. **829**
- 850. **830**
- 851. **831**
- 852. **832**
- 853. **833**
- 854. **834**
- 855. **835**
- 856. **836**
- 857. **837**
- 858. **838**
- 859. **839**

860. **840**

861. **841**

862. **842**

863. **843**