

Chapman & Hall/CRC
Data Mining and Knowledge Discovery Series

DATA MINING

A Tutorial-Based Primer

SECOND EDITION

Richard J. Roiger



WITH VITALSOURCE®
EBOOK



CRC Press
Taylor & Francis Group

A CHAPMAN & HALL BOOK

DATA MINING

A Tutorial-Based Primer

SECOND EDITION

Chapman & Hall/CRC
Data Mining and Knowledge Discovery Series

SERIES EDITOR

Vipin Kumar

University of Minnesota

Department of Computer Science and Engineering

Minneapolis, Minnesota, U.S.A.

AIMS AND SCOPE

This series aims to capture new developments and applications in data mining and knowledge discovery, while summarizing the computational tools and techniques useful in data analysis. This series encourages the integration of mathematical, statistical, and computational methods and techniques through the publication of a broad range of textbooks, reference works, and handbooks. The inclusion of concrete examples and applications is highly encouraged. The scope of the series includes, but is not limited to, titles in the areas of data mining and knowledge discovery methods and applications, modeling, algorithms, theory and foundations, data and knowledge visualization, data mining systems and tools, and privacy and security issues.

PUBLISHED TITLES

ACCELERATING DISCOVERY: MINING UNSTRUCTURED INFORMATION FOR HYPOTHESIS GENERATION

Scott Spangler

ADVANCES IN MACHINE LEARNING AND DATA MINING FOR ASTRONOMY

Michael J. Way, Jeffrey D. Scargle, Kamal M. Ali, and Ashok N. Srivastava

BIOLOGICAL DATA MINING

Jake Y. Chen and Stefano Lonardi

COMPUTATIONAL BUSINESS ANALYTICS

Subrata Das

COMPUTATIONAL INTELLIGENT DATA ANALYSIS FOR SUSTAINABLE DEVELOPMENT

Ting Yu, Nitesh V. Chawla, and Simeon Simoff

COMPUTATIONAL METHODS OF FEATURE SELECTION

Huan Liu and Hiroshi Motoda

CONSTRAINED CLUSTERING: ADVANCES IN ALGORITHMS, THEORY, AND APPLICATIONS

Sugato Basu, Ian Davidson, and Kiri L. Wagstaff

CONTRAST DATA MINING: CONCEPTS, ALGORITHMS, AND APPLICATIONS

Guozhu Dong and James Bailey

DATA CLASSIFICATION: ALGORITHMS AND APPLICATIONS

Charu C. Aggarwal

DATA CLUSTERING: ALGORITHMS AND APPLICATIONS

Charu C. Aggarwal and Chandan K. Reddy

DATA CLUSTERING IN C++: AN OBJECT-ORIENTED APPROACH

Guojun Gan

DATA MINING: A TUTORIAL-BASED PRIMER, SECOND EDITION

Richard J. Roiger

DATA MINING FOR DESIGN AND MARKETING

Yukio Ohsawa and Katsutoshi Yada

DATA MINING WITH R: LEARNING WITH CASE STUDIES, SECOND EDITION

Luís Torgo

EVENT MINING: ALGORITHMS AND APPLICATIONS

Tao Li

FOUNDATIONS OF PREDICTIVE ANALYTICS

James Wu and Stephen Coggeshall

**GEOGRAPHIC DATA MINING AND KNOWLEDGE DISCOVERY,
SECOND EDITION**

Harvey J. Miller and Jiawei Han

GRAPH-BASED SOCIAL MEDIA ANALYSIS

Ioannis Pitas

HANDBOOK OF EDUCATIONAL DATA MINING

Cristóbal Romero, Sebastian Ventura, Mykola Pechenizkiy, and Ryan S.J.d. Baker

HEALTHCARE DATA ANALYTICS

Chandan K. Reddy and Charu C. Aggarwal

INFORMATION DISCOVERY ON ELECTRONIC HEALTH RECORDS

Vagelis Hristidis

INTELLIGENT TECHNOLOGIES FOR WEB APPLICATIONS

Priti Srinivas Sajja and Rajendra Akerkar

**INTRODUCTION TO PRIVACY-PRESERVING DATA PUBLISHING: CONCEPTS
AND TECHNIQUES**

Benjamin C. M. Fung, Ke Wang, Ada Wai-Chee Fu, and Philip S. Yu

**KNOWLEDGE DISCOVERY FOR COUNTERTERRORISM AND
LAW ENFORCEMENT**

David Skillicorn

KNOWLEDGE DISCOVERY FROM DATA STREAMS

João Gama

**MACHINE LEARNING AND KNOWLEDGE DISCOVERY FOR
ENGINEERING SYSTEMS HEALTH MANAGEMENT**

Ashok N. Srivastava and Jiawei Han

MINING SOFTWARE SPECIFICATIONS: METHODOLOGIES AND APPLICATIONS

David Lo, Siau-Cheng Khoo, Jiawei Han, and Chao Liu

MULTIMEDIA DATA MINING: A SYSTEMATIC INTRODUCTION TO CONCEPTS AND THEORY

Zhongfei Zhang and Ruofei Zhang

MUSIC DATA MINING

Tao Li, Mitsunori Ogihara, and George Tzanetakis

NEXT GENERATION OF DATA MINING

Hillol Kargupta, Jiawei Han, Philip S. Yu, Rajeev Motwani, and Vipin Kumar

RAPIDMINER: DATA MINING USE CASES AND BUSINESS ANALYTICS APPLICATIONS

Markus Hofmann and Ralf Klinkenberg

RELATIONAL DATA CLUSTERING: MODELS, ALGORITHMS, AND APPLICATIONS

Bo Long, Zhongfei Zhang, and Philip S. Yu

SERVICE-ORIENTED DISTRIBUTED KNOWLEDGE DISCOVERY

Domenico Talia and Paolo Trunfio

SPECTRAL FEATURE SELECTION FOR DATA MINING

Zheng Alan Zhao and Huan Liu

STATISTICAL DATA MINING USING SAS APPLICATIONS, SECOND EDITION

George Fernandez

SUPPORT VECTOR MACHINES: OPTIMIZATION BASED THEORY, ALGORITHMS, AND EXTENSIONS

Naiyang Deng, Yingjie Tian, and Chunhua Zhang

TEMPORAL DATA MINING

Theophano Mitsa

TEXT MINING: CLASSIFICATION, CLUSTERING, AND APPLICATIONS

Ashok N. Srivastava and Mehran Sahami

TEXT MINING AND VISUALIZATION: CASE STUDIES USING OPEN-SOURCE TOOLS

Markus Hofmann and Andrew Chisholm

THE TOP TEN ALGORITHMS IN DATA MINING

Xindong Wu and Vipin Kumar

UNDERSTANDING COMPLEX DATASETS: DATA MINING WITH MATRIX DECOMPOSITIONS

David Skillicorn

DATA MINING

A Tutorial-Based Primer

SECOND EDITION

Richard J. Roiger



CRC Press

Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

This book was previously published by Pearson Education, Inc.

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2017 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed on acid-free paper
Version Date: 20161025

International Standard Book Number-13: 978-1-4987-6397-4 (Pack - Book and Ebook)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

Contents

List of Figures, xvii

List of Tables, xxix

Preface, xxxi

Acknowledgments, xxxix

Author, xli

SECTION I Data Mining Fundamentals

CHAPTER 1 ■ Data Mining: A First View	3
CHAPTER OBJECTIVES	3
1.1 DATA SCIENCE, ANALYTICS, MINING, AND KNOWLEDGE DISCOVERY IN DATABASES	4
1.1.1 Data Science and Analytics	4
1.1.2 Data Mining	5
1.1.3 Data Science versus Knowledge Discovery in Databases	5
1.2 WHAT CAN COMPUTERS LEARN?	6
1.2.1 Three Concept Views	6
1.2.1.1 <i>The Classical View</i>	6
1.2.1.2 <i>The Probabilistic View</i>	7
1.2.1.3 <i>The Exemplar View</i>	7
1.2.2 Supervised Learning	8
1.2.3 Supervised Learning: A Decision Tree Example	9
1.2.4 Unsupervised Clustering	11
1.3 IS DATA MINING APPROPRIATE FOR MY PROBLEM?	14
1.3.1 Data Mining or Data Query?	14
1.3.2 Data Mining versus Data Query: An Example	15
1.4 DATA MINING OR KNOWLEDGE ENGINEERING?	16

1.5 A NEAREST NEIGHBOR APPROACH	18
1.6 A PROCESS MODEL FOR DATA MINING	19
1.6.1 Acquiring Data	20
1.6.1.1 <i>The Data Warehouse</i>	20
1.6.1.2 <i>Relational Databases and Flat Files</i>	21
1.6.1.3 <i>Distributed Data Access</i>	21
1.6.2 Data Preprocessing	21
1.6.3 Mining the Data	23
1.6.4 Interpreting the Results	23
1.6.5 Result Application	24
1.7 DATA MINING, BIG DATA, AND CLOUD COMPUTING	24
1.7.1 Hadoop	24
1.7.2 Cloud Computing	24
1.8 DATA MINING ETHICS	25
1.9 INTRINSIC VALUE AND CUSTOMER CHURN	26
1.10 CHAPTER SUMMARY	27
1.11 KEY TERMS	28
CHAPTER 2 ■ Data Mining: A Closer Look	33
CHAPTER OBJECTIVES	33
2.1 DATA MINING STRATEGIES	34
2.1.1 Classification	34
2.1.2 Estimation	35
2.1.3 Prediction	36
2.1.4 Unsupervised Clustering	39
2.1.5 Market Basket Analysis	40
2.2 SUPERVISED DATA MINING TECHNIQUES	41
2.2.1 The Credit Card Promotion Database	41
2.2.2 Rule-Based Techniques	42
2.2.3 Neural Networks	44
2.2.4 Statistical Regression	46
2.3 ASSOCIATION RULES	47
2.4 CLUSTERING TECHNIQUES	48
2.5 EVALUATING PERFORMANCE	49
2.5.1 Evaluating Supervised Learner Models	50
2.5.2 Two-Class Error Analysis	52

2.5.3 Evaluating Numeric Output	53
2.5.4 Comparing Models by Measuring Lift	53
2.5.5 Unsupervised Model Evaluation	55
2.6 CHAPTER SUMMARY	56
2.7 KEY TERMS	57
CHAPTER 3 ■ Basic Data Mining Techniques	63
CHAPTER OBJECTIVES	63
3.1 DECISION TREES	64
3.1.1 An Algorithm for Building Decision Trees	64
3.1.2 Decision Trees for the Credit Card Promotion Database	70
3.1.3 Decision Tree Rules	73
3.1.4 Other Methods for Building Decision Trees	73
3.1.5 General Considerations	74
3.2 A BASIC COVERING RULE ALGORITHM	74
3.3 GENERATING ASSOCIATION RULES	80
3.3.1 Confidence and Support	80
3.3.2 Mining Association Rules: An Example	82
3.3.3 General Considerations	84
3.4 THE K-MEANS ALGORITHM	85
3.4.1 An Example Using K-means	86
3.4.2 General Considerations	89
3.5 GENETIC LEARNING	90
3.5.1 Genetic Algorithms and Supervised Learning	91
3.5.2 General Considerations	95
3.6 CHOOSING A DATA MINING TECHNIQUE	95
3.7 CHAPTER SUMMARY	97
3.8 KEY TERMS	98

SECTION II Tools for Knowledge Discovery

CHAPTER 4 ■ Weka—An Environment for Knowledge Discovery	105
CHAPTER OBJECTIVES	105
4.1 GETTING STARTED WITH WEKA	106
4.2 BUILDING DECISION TREES	109
4.3 GENERATING PRODUCTION RULES WITH PART	117

4.4 ATTRIBUTE SELECTION AND NEAREST NEIGHBOR CLASSIFICATION	122
4.5 ASSOCIATION RULES	127
4.6 COST/BENEFIT ANALYSIS, (OPTIONAL)	131
4.7 UNSUPERVISED CLUSTERING WITH THE K-MEANS ALGORITHM	137
4.8 CHAPTER SUMMARY	141
CHAPTER 5 ■ Knowledge Discovery with RapidMiner	145
CHAPTER OBJECTIVES	145
5.1 GETTING STARTED WITH RAPIDMINER	146
5.1.1 Installing RapidMiner	146
5.1.2 Navigating the Interface	146
5.1.3 A First Process Model	149
5.1.4 A Decision Tree for the Credit Card Promotion Database	156
5.1.5 Breakpoints	158
5.2 BUILDING DECISION TREES	159
5.2.1 Scenario 1: Using a Training and Test Set	160
5.2.2 Scenario 2: Adding a Subprocess	165
5.2.3 Scenario 3: Creating, Saving, and Applying the Final Model	167
5.2.3.1 <i>Saving a Model to an Output File</i>	167
5.2.3.2 <i>Reading and Applying a Model</i>	168
5.2.4 Scenario 4: Using Cross-Validation	168
5.3 GENERATING RULES	173
5.3.1 Scenario 1: Tree to Rules	173
5.3.2 Scenario 2: Rule Induction	176
5.3.3 Scenario 3: Subgroup Discovery	178
5.4 ASSOCIATION RULE LEARNING	181
5.4.1 Association Rules for the Credit Card Promotion Database	182
5.4.2 The Market Basket Analysis Template	183
5.5 UNSUPERVISED CLUSTERING WITH K-MEANS	187
5.6 ATTRIBUTE SELECTION AND NEAREST NEIGHBOR CLASSIFICATION	191
5.7 CHAPTER SUMMARY	194
CHAPTER 6 ■ The Knowledge Discovery Process	199
CHAPTER OBJECTIVES	199
6.1 A PROCESS MODEL FOR KNOWLEDGE DISCOVERY	199
6.2 GOAL IDENTIFICATION	201

6.3	CREATING A TARGET DATA SET	202
6.4	DATA PREPROCESSING	203
6.4.1	Noisy Data	203
6.4.1.1	<i>Locating Duplicate Records</i>	204
6.4.1.2	<i>Locating Incorrect Attribute Values</i>	204
6.4.1.3	<i>Data Smoothing</i>	204
6.4.1.4	<i>Detecting Outliers</i>	205
6.4.2	Missing Data	207
6.5	DATA TRANSFORMATION	208
6.5.1	Data Normalization	208
6.5.2	Data Type Conversion	209
6.5.3	Attribute and Instance Selection	209
6.5.3.1	<i>Wrapper and Filtering Techniques</i>	210
6.5.3.2	<i>More Attribute Selection Techniques</i>	211
6.5.3.3	<i>Genetic Learning for Attribute Selection</i>	211
6.5.3.4	<i>Creating Attributes</i>	212
6.5.3.5	<i>Instance Selection</i>	213
6.6	DATA MINING	214
6.7	INTERPRETATION AND EVALUATION	214
6.8	TAKING ACTION	215
6.9	THE CRISP-DM PROCESS MODEL	215
6.10	CHAPTER SUMMARY	216
6.11	KEY TERMS	216
<hr/>		
CHAPTER 7	■ Formal Evaluation Techniques	221
	CHAPTER OBJECTIVES	221
7.1	WHAT SHOULD BE EVALUATED?	222
7.2	TOOLS FOR EVALUATION	223
7.2.1	Single-Valued Summary Statistics	224
7.2.2	The Normal Distribution	225
7.2.3	Normal Distributions and Sample Means	226
7.2.4	A Classical Model for Hypothesis Testing	228
7.3	COMPUTING TEST SET CONFIDENCE INTERVALS	230
7.4	COMPARING SUPERVISED LEARNER MODELS	232
7.4.1	Comparing the Performance of Two Models	233
7.4.2	Comparing the Performance of Two or More Models	234

7.5 UNSUPERVISED EVALUATION TECHNIQUES	235
7.5.1 Unsupervised Clustering for Supervised Evaluation	235
7.5.2 Supervised Evaluation for Unsupervised Clustering	235
7.5.3 Additional Methods for Evaluating an Unsupervised Clustering	236
7.6 EVALUATING SUPERVISED MODELS WITH NUMERIC OUTPUT	236
7.7 COMPARING MODELS WITH RAPIDMINER	238
7.8 ATTRIBUTE EVALUATION FOR MIXED DATA TYPES	241
7.9 PARETO LIFT CHARTS	244
7.10 CHAPTER SUMMARY	247
7.11 KEY TERMS	248

SECTION III Building Neural Networks

CHAPTER 8 ■ Neural Networks	253
CHAPTER OBJECTIVES	253
8.1 FEED-FORWARD NEURAL NETWORKS	254
8.1.1 Neural Network Input Format	254
8.1.2 Neural Network Output Format	255
8.1.3 The Sigmoid Evaluation Function	256
8.2 NEURAL NETWORK TRAINING: A CONCEPTUAL VIEW	258
8.2.1 Supervised Learning with Feed-Forward Networks	258
8.2.1.1 <i>Training a Neural Network: Backpropagation Learning</i>	258
8.2.1.2 <i>Training a Neural Network: Genetic Learning</i>	259
8.2.2 Unsupervised Clustering with Self-Organizing Maps	259
8.3 NEURAL NETWORK EXPLANATION	260
8.4 GENERAL CONSIDERATIONS	262
8.5 NEURAL NETWORK TRAINING: A DETAILED VIEW	263
8.5.1 The Backpropagation Algorithm: An Example	263
8.5.2 Kohonen Self-Organizing Maps: An Example	266
8.6 CHAPTER SUMMARY	268
8.7 KEY TERMS	269
CHAPTER 9 ■ Building Neural Networks with Weka	271
CHAPTER OBJECTIVES	271
9.1 DATA SETS FOR BACKPROPAGATION LEARNING	272
9.1.1 The Exclusive-OR Function	272
9.1.2 The Satellite Image Data Set	273

9.2 MODELING THE EXCLUSIVE-OR FUNCTION: NUMERIC OUTPUT	274
9.3 MODELING THE EXCLUSIVE-OR FUNCTION: CATEGORICAL OUTPUT	280
9.4 MINING SATELLITE IMAGE DATA	282
9.5 UNSUPERVISED NEURAL NET CLUSTERING	287
9.6 CHAPTER SUMMARY	288
9.7 KEY TERMS	289
CHAPTER 10 • Building Neural Networks with RapidMiner	293
CHAPTER OBJECTIVES	293
10.1 MODELING THE EXCLUSIVE-OR FUNCTION	294
10.2 MINING SATELLITE IMAGE DATA	301
10.3 PREDICTING CUSTOMER CHURN	306
10.4 RAPIDMINER'S SELF-ORGANIZING MAP OPERATOR	311
10.5 CHAPTER SUMMARY	313
SECTION IV Advanced Data Mining Techniques	
CHAPTER 11 • Supervised Statistical Techniques	317
CHAPTER OBJECTIVES	317
11.1 NAÏVE BAYES CLASSIFIER	317
11.1.1 Naïve Bayes Classifier: An Example	318
11.1.2 Zero-Valued Attribute Counts	321
11.1.3 Missing Data	321
11.1.4 Numeric Data	322
11.1.5 Implementations of the Naïve Bayes Classifier	324
11.1.6 General Considerations	324
11.2 SUPPORT VECTOR MACHINES	324
11.2.1 Linearly Separable Classes	332
11.2.2 The Nonlinear Case	336
11.2.3 General Considerations	337
11.2.4 Implementations of Support Vector Machines	340
11.3 LINEAR REGRESSION ANALYSIS	340
11.3.1 Simple Linear Regression	344
11.3.2 Multiple Linear Regression	344
11.3.2.1 <i>Linear Regression—Weka</i>	344
11.3.2.2 <i>Linear Regression—RapidMiner</i>	345

11.4 REGRESSION TREES	349
11.5 LOGISTIC REGRESSION	350
11.5.1 Transforming the Linear Regression Model	350
11.5.2 The Logistic Regression Model	351
11.6 CHAPTER SUMMARY	352
11.7 KEY TERMS	352
<hr/> CHAPTER 12 ■ Unsupervised Clustering Techniques	<hr/> 357
CHAPTER OBJECTIVES	357
12.1 AGGLOMERATIVE CLUSTERING	358
12.1.1 Agglomerative Clustering: An Example	358
12.1.2 General Considerations	360
12.2 CONCEPTUAL CLUSTERING	360
12.2.1 Measuring Category Utility	361
12.2.2 Conceptual Clustering: An Example	362
12.2.3 General Considerations	364
12.3 EXPECTATION MAXIMIZATION	364
12.3.1 Implementations of the EM Algorithm	365
12.3.2 General Considerations	365
12.4 GENETIC ALGORITHMS AND UNSUPERVISED CLUSTERING	371
12.5 CHAPTER SUMMARY	374
12.6 KEY TERMS	374
<hr/> CHAPTER 13 ■ Specialized Techniques	<hr/> 377
CHAPTER OBJECTIVES	377
13.1 TIME-SERIES ANALYSIS	377
13.1.1 Stock Market Analytics	378
13.1.2 Time-Series Analysis—An Example	379
13.1.2.1 <i>Creating the Target Data Set—Numeric Output</i>	380
13.1.2.2 <i>Data Preprocessing and Transformation</i>	380
13.1.2.3 <i>Creating the Target Data Set—Categorical Output</i>	382
13.1.2.4 <i>Mining the Data—RapidMiner</i>	382
13.1.2.5 <i>Mining the Data—Weka</i>	387
13.1.2.6 <i>Interpretation, Evaluation, and Action</i>	390
13.1.3 General Considerations	390

13.2 MINING THE WEB	391
13.2.1 Web-Based Mining: General Issues	391
13.2.1.1 <i>Identifying the Goal</i>	391
13.2.2 Preparing the Data	392
13.2.2.1 <i>Mining the Data</i>	393
13.2.2.2 <i>Interpreting and Evaluating Results</i>	393
13.2.2.3 <i>Taking Action</i>	394
13.2.3 Data Mining for Website Evaluation	395
13.2.4 Data Mining for Personalization	395
13.2.5 Data Mining for Website Adaptation	396
13.2.6 PageRank and Link Analysis	396
13.2.7 Operators for Web-Based Mining	398
13.3 MINING TEXTUAL DATA	398
13.3.1 Analyzing Customer Reviews	399
13.4 TECHNIQUES FOR LARGE-SIZED, IMBALANCED, AND STREAMING DATA	404
13.4.1 Large-Sized Data	404
13.4.2 Dealing with Imbalanced Data	405
13.4.2.1 <i>Methods for Addressing Rarity</i>	406
13.4.2.2 <i>Receiver Operating Characteristics Curves</i>	406
13.4.3 Methods for Streaming Data	412
13.5 ENSEMBLE TECHNIQUES FOR IMPROVING PERFORMANCE	413
13.5.1 Bagging	413
13.5.2 Boosting	414
13.5.3 AdaBoost—An Example	414
13.6 CHAPTER SUMMARY	417
13.7 KEY TERMS	418
CHAPTER 14 ■ The Data Warehouse	423
CHAPTER OBJECTIVES	423
14.1 OPERATIONAL DATABASES	424
14.1.1 Data Modeling and Normalization	424
14.1.2 The Relational Model	425
14.2 DATA WAREHOUSE DESIGN	426
14.2.1 Entering Data into the Warehouse	427

14.2.2 Structuring the Data Warehouse: The Star Schema	429
14.2.2.1 <i>The Multidimensionality of the Star Schema</i>	430
14.2.2.2 <i>Additional Relational Schemas</i>	431
14.2.3 Decision Support: Analyzing the Warehouse Data	432
14.3 ONLINE ANALYTICAL PROCESSING	434
14.3.1 OLAP: An Example	435
14.3.2 General Considerations	438
14.4 EXCEL PIVOT TABLES FOR DATA ANALYTICS	438
14.5 CHAPTER SUMMARY	445
14.6 KEY TERMS	446

APPENDIX A—SOFTWARE AND DATA SETS FOR DATA MINING, 451

APPENDIX B—STATISTICS FOR PERFORMANCE EVALUATION, 455

BIBLIOGRAPHY, 461

INDEX, 465

List of Figures

Figure 1.1	A decision tree for the data in Table 1.1.	10
Figure 1.2	Data mining versus expert systems.	17
Figure 1.3	A process model for data mining.	20
Figure 1.4	A perfect positive correlation ($r = 1$).	22
Figure 1.5	A perfect negative correlation ($r = -1$).	23
Figure 1.6	Intrinsic versus actual customer value.	26
Figure 2.1	A hierarchy of data mining strategies.	34
Figure 2.2	A fully connected multilayer neural network.	45
Figure 2.3	An unsupervised clustering of the credit card database.	49
Figure 2.4	Targeted versus mass mailing.	54
Figure 3.1	A partial decision tree with root node = income range.	68
Figure 3.2	A partial decision tree with root node = credit card insurance.	69
Figure 3.3	A partial decision tree with root node = age.	70
Figure 3.4	A three-node decision tree for the credit card database.	71
Figure 3.5	A two-node decision tree for the credit card database.	71
Figure 3.6	Domain statistics for the credit card promotion database.	76
Figure 3.7	Class statistics for <i>life insurance promotion = yes</i> .	77
Figure 3.8	Class statistics for <i>life insurance promotion = no</i> .	78
Figure 3.9	Statistics for <i>life insurance promotion = yes</i> after removing five instances.	79
Figure 3.10	A coordinate mapping of the data in Table 3.6.	86
Figure 3.11	A K -means clustering of the data in Table 3.6 ($K = 2$).	89
Figure 3.12	Supervised genetic learning.	91

Figure 3.13	A crossover operation.	94
Figure 4.1	Weka GUI <i>Chooser</i> .	107
Figure 4.2	Explorer four graphical user interfaces (GUI's).	107
Figure 4.3	Weka install folder.	108
Figure 4.4	Sample data sets.	108
Figure 4.5	Instances of the contact-lenses file.	109
Figure 4.6	Loading the contact-lenses data set.	110
Figure 4.7	Navigating the <i>Explorer</i> interface.	111
Figure 4.8	A partial list of attribute filters.	112
Figure 4.9	Command line call for J48.	112
Figure 4.10	Parameter setting options for J48.	113
Figure 4.11	Decision tree for the contact-lenses data set.	113
Figure 4.12	Weka's tree visualizer.	114
Figure 4.13	Decision tree output for the contact-lenses data set.	115
Figure 4.14	Classifier output options.	116
Figure 4.15	Actual and predicted output.	116
Figure 4.16	Customer churn data.	118
Figure 4.17	A decision list for customer churn data.	118
Figure 4.18	Customer churn output generated by PART.	119
Figure 4.19	Loading the customer churn instances of unknown outcome.	120
Figure 4.20	Predicting customers likely to churn.	121
Figure 4.21	Nearest neighbor output for the spam data set.	123
Figure 4.22	Weka's attribute selection filter.	124
Figure 4.23	Options for the attribute selection filter.	124
Figure 4.24	Parameter settings for ranker.	125
Figure 4.25	Most predictive attributes for the spam data set.	126
Figure 4.26	IBk output after removing the 10 least predictive attributes.	126
Figure 4.27	Association rules for the contact-lenses data set.	128
Figure 4.28	Parameters for the Apriori algorithm.	129

Figure 4.29	The supermarket data set.	130
Figure 4.30	Instances of the supermarket data set.	130
Figure 4.31	Ten association rules for the supermarket data set.	131
Figure 4.32	A J48 classification of the credit card screening data set.	132
Figure 4.33	Invoking a cost/benefit analysis.	133
Figure 4.34	Cost/benefit output for the credit card screening data set.	133
Figure 4.35	Cost/benefit analysis set to match J48 classifier output.	134
Figure 4.36	Invoking a cost/benefit analysis.	135
Figure 4.37	Minimizing total cost.	135
Figure 4.38	Cutoff scores for credit card application acceptance.	136
Figure 4.39	Classes to clusters evaluation for simpleKmeans.	137
Figure 4.40	Include standard deviation values for simpleKmeans.	138
Figure 4.41	Classes to clusters output.	139
Figure 4.42	Partial list of attribute values for the <i>K</i> -means clustering in Figure 4.41.	140
Figure 4.43	Additional attribute values for the SimpleKMeans clustering in Figure 4.41.	140
Figure 5.1	An introduction to RapidMiner.	147
Figure 5.2	Creating a new blank process.	147
Figure 5.3	A new blank process with helpful pointers.	148
Figure 5.4	Creating and saving a process.	150
Figure 5.5	Importing the credit card promotion database.	150
Figure 5.6	Selecting the cells to import.	151
Figure 5.7	A list of allowable data types.	152
Figure 5.8	Changing the role of <i>Life Ins Promo</i> .	152
Figure 5.9	Storing a file in the data folder.	153
Figure 5.10	The credit card promotion database.	153
Figure 5.11	A successful file import.	154
Figure 5.12	Connecting the credit card promotion database to an output port.	154
Figure 5.13	Summary statistics for the credit card promotion database.	155
Figure 5.14	A bar graph for income range.	155

Figure 5.15	A scatterplot comparing age and life insurance promotion.	156
Figure 5.16	A decision tree process model.	157
Figure 5.17	A decision tree for the credit card promotion database.	158
Figure 5.18	A decision tree in descriptive form.	158
Figure 5.19	A list of operator options.	159
Figure 5.20	Customer churn—A training and test set scenario.	160
Figure 5.21	Removing instances of unknown outcome from the churn data set.	161
Figure 5.22	Partitioning the customer churn data.	162
Figure 5.23	The customer churn data set.	163
Figure 5.24	<i>Filter Examples</i> has removed all instances of unknown outcome.	163
Figure 5.25	A decision tree for the customer churn data set.	164
Figure 5.26	Output of the <i>Apply Model</i> operator.	164
Figure 5.27	A performance vector for the customer churn data set.	165
Figure 5.28	Adding a subprocess to the main process window.	166
Figure 5.29	A subprocess for data preprocessing.	167
Figure 5.30	Creating and saving a decision tree model.	168
Figure 5.31	Reading and applying a saved model.	169
Figure 5.32	An Excel file stores model predictions.	169
Figure 5.33	Testing a model using cross-validation.	170
Figure 5.34	A subprocess to read and filter customer churn data.	171
Figure 5.35	Nested subprocesses for cross-validation.	171
Figure 5.36	Performance vector for a decision tree tested using cross-validation.	172
Figure 5.37	Subprocess for the <i>Tree to Rules</i> operator.	174
Figure 5.38	Building a model with the <i>Tree to Rules</i> operator.	174
Figure 5.39	Rules generated by the <i>Tree to Rules</i> operator.	175
Figure 5.40	Performance vector for the customer churn data set.	175
Figure 5.41	A process design for rule induction.	176
Figure 5.42	Adding the <i>Discretize by Binning</i> operator.	177
Figure 5.43	Covering rules for customer churn data.	177

Figure 5.44	Performance vector for the covering rules of Figure 5.43.	178
Figure 5.45	Process design for subgroup discovery.	179
Figure 5.46	Subprocess design for subgroup discovery.	179
Figure 5.47	Rules generated by the <i>Subgroup Discovery</i> operator.	180
Figure 5.48	Ten rules identifying likely churn candidates.	181
Figure 5.49	Generating association rules for the credit card promotion database.	182
Figure 5.50	Preparing data for association rule generation.	183
Figure 5.51	Interface for listing association rules.	184
Figure 5.52	Association rules for the credit card promotion database.	184
Figure 5.53	<i>Market basket analysis</i> template.	185
Figure 5.54	The pivot operator rotates the example set.	186
Figure 5.55	Association rules for the <i>Market Basket Analysis</i> template.	186
Figure 5.56	Process design for clustering gamma-ray burst data.	188
Figure 5.57	A partial clustering of gamma-ray burst data.	189
Figure 5.58	Three clusters of gamma-ray burst data.	189
Figure 5.59	Decision tree illustrating a gamma-ray burst clustering.	190
Figure 5.60	A descriptive form of a decision tree showing a clustering of gamma-ray burst data.	190
Figure 5.61	Benchmark performance for nearest neighbor classification.	192
Figure 5.62	Main process design for nearest neighbor classification.	192
Figure 5.63	Subprocess for nearest neighbor classification.	193
Figure 5.64	Forward selection subprocess for nearest neighbor classification.	193
Figure 5.65	Performance vector when forward selection is used for choosing attributes.	194
Figure 5.66	Unsupervised clustering for attribute evaluation.	197
Figure 6.1	A seven-step KDD process model.	200
Figure 6.2	The Acme credit card database.	203
Figure 6.3	A process model for detecting outliers.	205
Figure 6.4	Two outlier instances from the diabetes patient data set.	206

Figure 6.5	Ten outlier instances from the diabetes patient data set.	207
Figure 7.1	Components for supervised learning.	222
Figure 7.2	A normal distribution.	225
Figure 7.3	Random samples from a population of 10 elements.	226
Figure 7.4	A process model for comparing three competing models.	239
Figure 7.5	Subprocess for comparing three competing models.	240
Figure 7.6	Cross-validation test for a decision tree with maximum depth = 5.	240
Figure 7.7	A matrix of <i>t</i> -test scores.	241
Figure 7.8	ANOVA comparing three competing models.	241
Figure 7.9	ANOVA operators for comparing nominal and numeric attributes.	242
Figure 7.10	The grouped ANOVA operator comparing class and maximum heart rate.	243
Figure 7.11	The ANOVA matrix operator for the cardiology patient data set.	243
Figure 7.12	A process model for creating a lift chart.	244
Figure 7.13	Preprocessing the customer churn data set.	245
Figure 7.14	Output of the Apply Model operator for the customer churn data set.	245
Figure 7.15	Performance vector for customer churn.	246
Figure 7.16	A Pareto lift chart for customer churn.	247
Figure 8.1	A fully connected feed-forward neural network.	254
Figure 8.2	The sigmoid evaluation function.	257
Figure 8.3	A 3×3 Kohonen network with two input-layer nodes.	260
Figure 8.4	Connections for two output-layer nodes.	266
Figure 9.1	Graph of the XOR function.	272
Figure 9.2	XOR training data.	273
Figure 9.3	Satellite image data.	274
Figure 9.4	Weka four graphical user interfaces (GUIs) for XOR training.	275
Figure 9.5	Backpropagation learning parameters.	276
Figure 9.6	Architecture for the XOR function.	278
Figure 9.7	XOR training output.	278

Figure 9.8	Network architecture with associated connection weights.	279
Figure 9.9	XOR network architecture without a hidden layer.	280
Figure 9.10	Confusion <i>matrix</i> for XOR without a hidden layer.	281
Figure 9.11	XOR with hidden layer and categorical output.	281
Figure 9.12	XOR confusion matrix and categorical output.	282
Figure 9.13	Satellite image data network architecture.	284
Figure 9.14	Confusion matrix for satellite image data.	284
Figure 9.15	Updated class assignment for instances 78 through 94 of the satellite image data set.	286
Figure 9.16	Initial classification for pixel instances 78 through 94 of the satellite image data set.	286
Figure 9.17	Parameter settings for Weka's <i>SelfOrganizingMap</i> .	287
Figure 9.18	Applying Weka's <i>SelfOrganizingMap</i> to the diabetes data set.	288
Figure 10.1	Statistics for the XOR function.	295
Figure 10.2	Main process for learning the XOR function.	295
Figure 10.3	Default settings for the <i>hidden layers</i> parameter.	296
Figure 10.4	A single hidden layer of five nodes.	297
Figure 10.5	Performance parameters for neural network learning.	298
Figure 10.6	Neural network architecture for the XOR function.	298
Figure 10.7	Hidden-to-output layer connection weights.	299
Figure 10.8	Prediction confidence values for the XOR function.	299
Figure 10.9	Performance vector for the XOR function.	300
Figure 10.10	Absolute error value for the XOR function.	300
Figure 10.11	Attribute declarations for the satellite image data set.	302
Figure 10.12	Main process for mining the satellite image data set.	302
Figure 10.13	Subprocesses for satellite image data set.	303
Figure 10.14	Network architecture for the satellite image data set.	304
Figure 10.15	Performance vector for the satellite image data set.	304
Figure 10.16	Removing correlated attributes from the satellite image data set.	305

Figure 10.17	Green and red have been removed from the satellite image data set.	305
Figure 10.18	Correlation matrix for the satellite image data set.	306
Figure 10.19	Neural network model for predicting customer churn.	307
Figure 10.20	Preprocessing the customer churn data.	308
Figure 10.21	Cross-validation subprocess for customer churn.	308
Figure 10.22	Performance vector for customer churn.	309
Figure 10.23	Process for creating and saving a neural network model.	309
Figure 10.24	Process model for reading and applying a neural network model.	310
Figure 10.25	Neural network output for predicting customer churn.	310
Figure 10.26	SOM process model for the cardiology patient data set.	312
Figure 10.27	Clustered instances of the cardiology patient data set.	312
Figure 11.1	RapidMiner's naïve Bayes operator.	325
Figure 11.2	Subprocess for applying naïve Bayes to customer churn data.	326
Figure 11.3	Naïve Bayes Distribution Table for customer churn data.	326
Figure 11.4	Naïve Bayes performance vector for customer churn data.	327
Figure 11.5	Life insurance promotion by gender.	328
Figure 11.6	Naïve Bayes model with output attribute = LifeInsPromo.	329
Figure 11.7	Predictions for the life insurance promotion.	329
Figure 11.8	Hyperplanes separating the circle and star classes.	330
Figure 11.9	Hyperplanes passing through their respective support vectors.	331
Figure 11.10	Maximal margin hyperplane separating the star and circle classes.	335
Figure 11.11	Loading the nine instances of Figure 11.8 into the Explorer.	338
Figure 11.12	Invoking SMO model.	339
Figure 11.13	Disabling data normalization/standardization.	339
Figure 11.14	The SMO-created MMH for the data shown in Figure 11.8.	340
Figure 11.15	Applying mySVM to the cardiology patient data set.	341
Figure 11.16	Normalized cardiology patient data.	342
Figure 11.17	Equation of the MMH for the cardiology patient data set.	342
Figure 11.18	Actual and predicted output for the cardiology patient data.	343

Figure 11.19	Performance vector for the cardiology patient data.	343
Figure 11.20	A linear regression model for the instances of Figure 11.8.	345
Figure 11.21	Main process window for applying RapidMiner's linear regression operator to the gamma-ray burst data set.	346
Figure 11.22	Subprocess windows for the Gamma Ray burst experiment.	346
Figure 11.23	Linear regression—actual and predicted output for the gamma-ray burst data set.	347
Figure 11.24	Summary statistics and the linear regression equation for the gamma-ray burst data set.	347
Figure 11.25	Scatterplot diagram showing the relationship between t90 and t50.	348
Figure 11.26	Performance vector resulting from the application of linear regression to the gamma-ray burst data set.	348
Figure 11.27	A generic model tree.	349
Figure 11.28	The logistic regression equation.	351
Figure 12.1	A Cobweb-created hierarchy.	363
Figure 12.2	Applying EM to the gamma-ray burst data set.	366
Figure 12.3	Removing correlated attributes from the gamma-ray burst data set.	367
Figure 12.4	An EM clustering of the gamma-ray burst data set.	367
Figure 12.5	Summary statistics for an EM clustering of the gamma-ray burst data set.	368
Figure 12.6	Decision tree representing a clustering of the gamma-ray burst data set.	368
Figure 12.7	The decision tree of Figure 12.6 in descriptive form.	369
Figure 12.8	Classes of the sensor data set.	370
Figure 12.9	Generic object editor allows us to specify the number of clusters.	370
Figure 12.10	Classes to clusters summary statistics.	371
Figure 12.11	Unsupervised genetic clustering.	372
Figure 13.1	A process model for extracting historical market data.	380
Figure 13.2	Historical data for XIV.	381
Figure 13.3	Time-series data with numeric output.	382
Figure 13.4	Time-series data with categorical output.	383
Figure 13.5	Time-series data for processing with RapidMiner.	383

Figure 13.6	A 3-month price chart for XIV.	384
Figure 13.7	A process model for time-series analysis with categorical output.	385
Figure 13.8	Predictions and confidence scores for time-series analysis.	385
Figure 13.9	Performance vector—time-series analysis for XIV.	386
Figure 13.10	Predicting the next-day closing price of XIV.	386
Figure 13.11	Time-series data formatted for Weka—categorical output.	387
Figure 13.12	Time-series data formatted for Weka—numeric output.	388
Figure 13.13	Time-series analysis with categorical output.	388
Figure 13.14	Time-series analysis with numeric output.	389
Figure 13.15	Cluster analysis using time-series data.	389
Figure 13.16	A generic Web usage data mining model.	392
Figure 13.17	Creating usage profiles from session data.	395
Figure 13.18	Hypertext link recommendations from usage profiles.	396
Figure 13.19	A page link structure.	397
Figure 13.20	A main process model for mining textual data.	401
Figure 13.21	A template to enter folder names used for textual data.	401
Figure 13.22	Class and folder names containing textual data.	402
Figure 13.23	Subprocess for tokenizing and stemming textual data.	402
Figure 13.24	A tokenized positive evaluation.	403
Figure 13.25	A tokenized and stemmed positive evaluation.	403
Figure 13.26	Textual data reduced to two dimensions.	403
Figure 13.27	Rules defining the three product evaluation classes.	404
Figure 13.28	An ROC graph for four competing models.	407
Figure 13.29	The PART algorithm applied to the spam data set.	408
Figure 13.30	An ROC curve created by applying PART to the spam data set.	409
Figure 13.31	Locating the true- and false-positive rate position in the ROC curve.	410
Figure 13.32	Confidence scores for predicted values with the spam data set.	410
Figure 13.33	Sorted confidence scores for the spam data set.	411
Figure 13.34	A main process for testing the AdaBoost operator.	415

Figure 13.35	Subprocess using a decision tree without AdaBoost.	415
Figure 13.36	Subprocess using AdaBoost, which builds several decision trees.	416
Figure 13.37	T-test results for testing AdaBoost.	416
Figure 13.38	Results of the ANOVA with the AdaBoost operator.	417
Figure 14.1	A simple entity-relationship diagram.	424
Figure 14.2	A data warehouse process model.	428
Figure 14.3	A star schema for credit card purchases.	429
Figure 14.4	Dimensions of the fact table shown in Figure 14.3.	431
Figure 14.5	A constellation schema for credit card purchases and promotions.	433
Figure 14.6	A multidimensional cube for credit card purchases.	435
Figure 14.7	A concept hierarchy for location.	436
Figure 14.8	Rolling up from months to quarters.	437
Figure 14.9	Creating a pivot table.	439
Figure 14.10	A blank <i>pivot table</i> .	440
Figure 14.11	A comparison of credit card insurance and income range.	440
Figure 14.12	A chart comparing credit card insurance and income range.	441
Figure 14.13	A credit card promotion cube.	442
Figure 14.14	A <i>pivot table</i> for the cube shown in Figure 14.13.	442
Figure 14.15	<i>Pivot table</i> position corresponding to the highlighted cell in Figure 14.13.	443
Figure 14.16	Drilling <i>down</i> into the cell highlighted in Figure 14.15.	444
Figure 14.17	Highlighting female customers with a slice operation.	444
Figure 14.18	A second approach for highlighting female customers.	445
Figure A.1	A successful installation.	452
Figure A.2	Locating and installing a package.	452
Figure A.3	List of installed packages.	452



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

List of Tables

Table 1.1	Hypothetical training data for disease diagnosis	9
Table 1.2	Data instances with an unknown classification	10
Table 1.3	Acme Investors Incorporated	12
Table 2.1	Cardiology patient data	37
Table 2.2	Typical and atypical instances from the cardiology domain	38
Table 2.3	Credit card promotion database	41
Table 2.4	Neural network training: actual and computed output	46
Table 2.5	A three-class confusion matrix	51
Table 2.6	A simple confusion matrix	52
Table 2.7	Two confusion matrices each showing a 10% error rate	52
Table 2.8	Two confusion matrices: no model and an ideal model	54
Table 2.9	Two confusion matrices for alternative models with lift equal to 2.25	55
Table 3.1	The credit card promotion database	67
Table 3.2	Training data instances following the path in Figure 3.4 to <i>credit card insurance = no</i>	72
Table 3.3	A subset of the credit card promotion database	82
Table 3.4	Single-item sets	83
Table 3.5	Two-item sets	83
Table 3.6	K -means input values	86
Table 3.7	Several applications of the K -means algorithm ($K = 2$)	89
Table 3.8	An initial population for supervised genetic learning	92
Table 3.9	Training data for genetic learning	93

Table 3.10	A second-generation population	94
Table 6.1	Initial population for genetic attribute selection	212
Table 7.1	A confusion matrix for the null hypothesis	229
Table 7.2	Absolute and squared error (output attribute = life insurance promotion)	237
Table 8.1	Initial weight values for the neural network shown in Figure 8.1	257
Table 8.2	A population of weight elements for the network in Figure 8.1	259
Table 9.1	Exclusive-OR function	272
Table 11.1	Data for Bayes classifier	318
Table 11.2	Counts and probabilities for attribute <i>gender</i>	318
Table 11.3	Addition of attribute <i>age</i> to the Bayes classifier data set	323
Table 12.1	Five instances from the credit card promotion database	358
Table 12.2	Agglomerative clustering: first iteration	358
Table 12.3	Agglomerative clustering: second iteration	359
Table 12.4	Data for conceptual clustering	364
Table 12.5	Instances for unsupervised genetic learning	373
Table 12.6	A first-generation population for unsupervised clustering	373
Table 13.1	Daily closing price for XIV, January 4, 2016 to January 8, 2016	381
Table 13.2	Daily closing price for XIV January 5, 2016 to January 11, 2016	381
Table 14.1	Relational table for <i>vehicle-type</i>	425
Table 14.2	Relational table for <i>customer</i>	426
Table 14.3	Join of Tables 14.1 and 14.2	426

Preface

Data mining is the process of finding interesting patterns in data. The objective of data mining is to use discovered patterns to help explain current behavior or to predict future outcomes. Several aspects of the data mining process can be studied. These include

- Data gathering and storage
- Data selection and preparation
- Model building and testing
- Interpreting and validating results
- Model application

A single book cannot concentrate on all areas of the data mining process. Although we furnish some detail about all aspects of data mining and knowledge discovery, our primary focus is centered on *model building and testing*, as well as on *interpreting and validating results*.

OBJECTIVES

I wrote the text to facilitate the following student learning goals:

- Understand what data mining is and how data mining can be employed to solve real problems
- Recognize whether a data mining solution is a feasible alternative for a specific problem
- Step through the knowledge discovery process and write a report about the results of a data mining session
- Know how to apply data mining software tools to solve real problems
- Apply basic statistical and nonstatistical techniques to evaluate the results of a data mining session

- Recognize several data mining strategies and know when each strategy is appropriate
- Develop a comprehensive understanding of how several data mining techniques build models to solve problems
- Develop a general awareness about the structure of a data warehouse and how a data warehouse can be used
- Understand what online analytical processing (OLAP) is and how it can be applied to analyze data

UPDATED CONTENT AND SOFTWARE CHANGES

The most obvious difference between the first and second editions of the text is the change in data mining software. Here is a short list of the major changes seen with the second edition:

- In Chapter 4, I introduce the *Waikato Environment for Knowledge Analysis* (Weka), an easy-to-use, publicly available tool for data mining. Weka contains a wealth of preprocessing and data mining techniques, graphical features, and visualization capabilities.
- Chapter 5 is all about data mining using RapidMiner Studio, a powerful open-source and code-free version of RapidMiner's commercial product. RapidMiner uses a drag and drop workflow paradigm for building models to solve complex problems. RapidMiner's intuitive user interface, visualization capabilities, and assortment of operators for preprocessing and mining data are second to none.
- This edition covers what are considered to be the top 10 data mining algorithms (Wu and Kumar, 2009). Nine of the algorithms are used in one or more tutorials.
- Tutorials have been added for attribute selection, dealing with imbalanced data, outlier analysis, time-series analysis, and mining textual data.
- Over 90% of the tutorials are presented using both Weka and RapidMiner. This allows readers maximum flexibility for their hands-on data mining experience.
- Selected new topics include
 - A brief introduction to big data and data analytics
 - Receiver operating characteristic (ROC) curves
 - Methods for handling large-sized, streaming, and imbalanced data
 - Extended coverage of textual data mining
 - Added techniques for attribute and outlier analysis

DATA SETS FOR DATA MINING

All data sets used for tutorials, illustrations, and end-of-chapter exercises are described in the text. The data sets come from several areas including business, health and medicine, and science. The data sets together with screenshots in PowerPoint and PDF format showing what you will see as you work through the tutorials can be downloaded from two locations:

- The CRC website: <https://www.crcpress.com/Data-Mining-A-Tutorial-Based-Primer-Second-Edition/Roiger/p/book/9781498763974>, under the Downloads tab
- <https://krypton.mnsu.edu/~sa7379bt>

INTENDED AUDIENCE

I developed most of the material for this book while teaching a one-semester data mining course open to students majoring or minoring in business or computer science. In writing this text, I directed my attention toward four groups of individuals:

- *Educators* in the areas of decision science, computer science, information systems, and information technology who wish to teach a unit, workshop, or entire course on data mining and knowledge discovery
- *Students* who want to learn about data mining and desire hands-on experience with a data mining tool
- *Business professionals* who need to understand how data mining can be applied to help solve their business problems
- *Applied researchers* who wish to add data mining methods to their problem-solving and analytics tool kit

CHAPTER FEATURES

I take the approach that model building is both an art and a science best understood from the perspective of learning by doing. My view is supported by several features found within the pages of the text. The following is a partial list of these features.

- *Simple, detailed examples.* I remove much of the mystery surrounding data mining by presenting simple, detailed examples of how the various data mining techniques build their models. Because of its tutorial nature, the text is appropriate as a self-study guide as well as a college-level textbook for a course about data mining and knowledge discovery.
- *Overall tutorial style.* All examples in Chapters 4, 5, 9, and 10 are tutorials. Selected sections in Chapters 6, 7, 11, 12, 13, and 14 offer easy-to-follow, step-by-step tutorials

for performing data analytics. All selected section tutorials are highlighted for easy differentiation from regular text.

- *Data sets for data mining.* A variety of data sets from business, medicine, and science are ready for data mining.
- *Key term definitions.* Each chapter introduces several key terms. A list of definitions for these terms is provided at the end of each chapter.
- *End-of-chapter exercises.* The end-of-chapter exercises reinforce the techniques and concepts found within each chapter. The exercises are grouped into one of three categories—review questions, data mining questions, and computational questions.
 - *Review questions* ask basic questions about the concepts and content found within each chapter. The questions are designed to help determine if the reader understands the major points conveyed in each chapter.
 - *Data mining questions* require the reader to use one or several data mining tools to perform data mining sessions.
 - *Computational questions* have a mathematical flavor in that they require the reader to perform one or several calculations. Many of the computational questions are appropriate for challenging the more advanced student.

CHAPTER CONTENT

The ordering of the chapters and the division of the book into separate parts is based on several years of experience in teaching courses on data mining. *Section I introduces material that is fundamental to understanding the data mining process.* The presentation is informal and easy to follow. Basic data mining concepts, strategies, and techniques are introduced. Students learn about the types of problems that can be solved with data mining.

Once the basic concepts are understood, *Section II provides the tools for knowledge discovery with detailed tutorials taking you through the knowledge discovery process.* The fact that data preprocessing is fundamental to successful data mining is emphasized. Also, special attention is given to formal data mining evaluation techniques.

Section III is all about neural networks. A conceptual and detailed presentation is offered for feed-forward networks trained with backpropagation learning and self-organizing maps for unsupervised clustering. Section III contains several tutorials for neural network learning with Weka and RapidMiner.

Section IV focuses on several specialized techniques. Topics of current interest such as time-series analysis, textual data mining, imbalanced and streaming data, as well as Web-based data mining are described.

Section I: Data Mining Fundamentals

- *Chapter 1* offers an overview of data analytics and all aspects of the data mining process. Special emphasis is placed on helping the student determine when data mining is an appropriate problem-solving strategy.
- *Chapter 2* presents a synopsis of several common data mining strategies and techniques. Basic methods for evaluating the outcome of a data mining session are described.
- *Chapter 3* details a decision tree algorithm, the Apriori algorithm for producing association rules, a covering rule algorithm, the *K*-means algorithm for unsupervised clustering, and supervised genetic learning. Tools are provided to help determine which data mining techniques should be used to solve specific problems.

Section II: Tools for Knowledge Discovery

- *Chapter 4* presents a tutorial introduction to Weka’s Explorer. Several tutorials provide a hands-on experience using the algorithms presented in the first three chapters.
- *Chapter 5* introduces RapidMiner Studio 7, an open-source, code-free version of RapidMiner’s commercial product. The chapter parallels the tutorials presented in Chapter 4 for building, testing, saving, and applying models.
- *Chapter 6* introduces the knowledge discovery in databases (KDD) process model as a formal methodology for solving problems with data mining. This chapter offers tutorials for outlier detection.
- *Chapter 7* describes formal statistical and nonstatistical methods for evaluating the outcome of a data mining session. The chapter illustrates how to create and read Pareto lift charts and how to apply RapidMiner’s ANOVA, *Grouped ANOVA*, and *T-Test* statistical operators for model evaluation.

Section III: Building Neural Networks

- *Chapter 8* presents two popular neural network models. A detailed explanation of neural network training is offered for the more technically inclined reader.
- *Chapter 9* offers tutorials for applying Weka’s MultilayerPerceptron neural network function for supervised learning and Weka’s SelfOrganizingMap for unsupervised clustering.
- *Chapter 10* presents tutorials on applying RapidMiner’s *Neural Network* operator for supervised learning, and *Self-Organizing Map* operator for unsupervised clustering.

Section IV: Advanced Data Mining Techniques

- *Chapter 11* details several supervised statistical techniques including naive Bayes classifier, support vector machines, linear regression, logistic regression, regression trees, and model trees. The chapter contains several examples and tutorials.
- *Chapter 12* presents several unsupervised clustering techniques including agglomerative clustering, hierarchical conceptual clustering, and expectation maximization (EM). Tutorials on using supervised learning for unsupervised cluster evaluation are presented.
- *Chapter 13* introduces techniques for performing time-series analysis, Web-based mining, and textual data mining. Methods for dealing with large-sized, imbalanced, and streaming data are offered. Bagging and boosting are described as methods for improving model performance. Tutorials and illustrations for time-series analysis, textual data mining, and ensemble learning are presented. A detailed example using receiver operator curves is offered.
- *Chapter 14* provides a gentle introduction to data warehouse design and OLAP. A tutorial on using Excel pivot tables for data analysis is included.

INSTRUCTOR SUPPLEMENTS

The following supplements are provided to help the instructor organize lectures and write examinations:

- *PowerPoint slides.* Each figure and table in the text is part of a PowerPoint presentation. These slides are also offered in PDF format.
 - A second set of slides containing the screenshots seen as you work through the tutorials in Chapters 4 through 14.
 - All RapidMiner processes used in the tutorials, demonstrations, and end-of-chapter exercises are readily available together with simple installation instructions.
- *Test questions.* Several test questions are provided for each chapter.
- *Answers to selected exercises.* Answers are given for most of the end-of-chapter exercises.
- *Lesson planner.* The lesson planner contains ideas for lecture format and points for discussion. The planner also provides suggestions for using selected end-of-chapter exercises in a laboratory setting.

Please note that these supplements are available to qualified instructors only. Contact your CRC sales representative or get help by visiting <https://www.crcpress.com/contactus> to access this material. Supplements will be updated as needed.

USING WEKA AND RAPIDMINER

Students are likely to benefit most by developing a working knowledge of both tools. This is best accomplished by students beginning their data mining experience with Weka's Explorer interface. The Explorer is easy to navigate and makes several of the more difficult preprocessing tasks transparent to the user. Missing data are automatically handled by most data mining algorithms, and data type conversions are automatic. The format for model evaluation, be it a training/test set scenario or cross-validation, is implemented with a simple click of the mouse. This transparency allows the beginning student to immediately experience the data mining process with a minimum of frustration. The Explorer is a great starting point but still supports the resources for a complete data mining experience.

Once students become familiar with the data mining process, they are ready to advance to Chapter 5 and RapidMiner. RapidMiner Studio's drag-and-drop workflow environment gives students complete control over their model building experience. Just a few of RapidMiner's features include an intuitive user interface, excellent graphics, and over 1500 operators for data visualization and preprocessing, data mining, and result evaluation. An interface for cloud computing as well as extensions for mining textual data, financial data, and the Web is supplemented by a large user community to help answer your data mining questions.

Here are a few suggestions when using both models:

- Cover the following sections to gain enough knowledge to understand the tutorials presented in later chapters.
 - If Weka is your choice, at a minimum, work through Sections 4.1, 4.2, and 4.7 of Chapter 4.
 - If you are focusing on RapidMiner, cover at least Sections 5.1 and 5.2 of Chapter 5.
- Here is a summary of the tutorials given in Chapters 6, 7, 11, 12, 13, and 14.
 - Chapter 6: RapidMiner is used to provide a tutorial on outlier analysis.
 - Chapter 7: Tutorials are presented using RapidMiner's *T-Test* and *ANOVA* operators for comparing model performance.
 - Chapter 11: Both models are used for tutorials highlighting naive Bayes classifier and support vector machines.
 - Chapter 12: RapidMiner and Weka are used to illustrate unsupervised clustering with the EM (Expectation Maximization) algorithm.
 - Chapter 13: Both RapidMiner and Weka are employed for time-series analysis. RapidMiner is used for a tutorial on textual data mining. Weka is employed for a tutorial on ROC curves. RapidMiner is used to give an example of ensemble learning.

- Chapter 14: Tutorials are given for creating simple and multidimensional MS Excel pivot tables.
- Chapter 9 is about neural networks using Weka. Chapter 10 employs RapidMiner to cover the same material. There are advantages to examining at least some of the material in both chapters. Weka’s neural network function is able to mine data having a numeric output attribute, and RapidMiner’s self-organizing map operator can perform dimensionality reduction as well as unsupervised clustering.

SUGGESTED COURSE OUTLINES

The text is appropriate for the undergraduate information systems or computer science student. It can also provide assistance for the graduate student who desires a working knowledge of data mining and knowledge discovery. The text is designed to be covered in a single semester.

A Data Mining Course for Information Systems Majors or Minors

Cover Chapters 1, 2, and 3 in detail. However, Section 3.5 of Chapter 3 may be omitted or lightly covered. Spend enough time on Chapters 4 and 5 for students to feel comfortable working with the software tools.

If your students lack a course in basic statistics, Sections 7.1 through 7.7 can be lightly covered. The tutorial material in Sections 7.8 through 7.10 is instructive. Section 8.5 can be excluded, but cover all of either Chapter 9 or Chapter 10. Cover topics from Chapters 11 through 13 as appropriate for your class. For Chapter 13, all students need some exposure to time-series analysis as well as Web-based and textual data mining.

A Data Mining Course for Undergraduate Computer Science Majors or Minors

Cover Chapters 1 through 13 in detail. Spend a day or two on the material in Chapter 14 to provide students with a basic understanding of online analytical processing and data warehouse design. Spend extra time covering material in Chapter 13. For a more intense course, the material in Appendix B, “Statistics for Performance Evaluation,” can be covered as part of the regular course.

A Data Mining Short Course

The undergraduate or graduate student interested in quickly developing a working knowledge of data mining should devote time to Chapters 1, 2, and 3, and Chapter 4 or 5. A working knowledge of neural networks can be obtained through the study of Chapter 8 (Sections 8.1 through 8.4) and Chapter 9 or Chapter 10.

Acknowledgments

I am indebted to my editor Randi Cohen for the confidence she placed in me and for allowing me the freedom to make critical decisions about the content of the text. I am very grateful to Dr. Mark Polczynski and found his constructive comments to be particularly helpful during revisions of the manuscript. Finally, I am most deeply indebted to my wife Suzanne for her extreme patience, helpful comments, and consistent support.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Author

Richard J. Roiger, PhD, is a professor emeritus at Minnesota State University, Mankato, where he taught and performed research in the Computer Information Science Department for 27 years. Dr. Roiger earned his PhD degree in computer and information sciences at the University of Minnesota. Dr. Roiger has presented conference papers and written several journal articles about topics in data mining and knowledge discovery. After retirement, Dr. Roiger continues to serve as a part-time faculty member teaching courses in data mining, artificial intelligence, and research methods. He is a board member of the Retired Education Association of Minnesota, where he serves as their financial advisor.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

I

Data Mining Fundamentals



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Data Mining

A First View

CHAPTER OBJECTIVES

- Define data mining and understand how data mining can be used to solve problems
- Understand that data analytics often uses data mining algorithms to build problem-solving models
- Understand that computers are best at learning concept definitions
- Know when data mining should be considered as a problem-solving strategy
- Understand that knowledge engineering and data mining use different means to accomplish similar goals
- Understand that supervised learning builds models by forming concept definitions from data containing predefined classes
- Know that unsupervised clustering builds models from data without the aid of pre-defined classes
- Recognize that classification models are able to classify new data of unknown origin
- Realize that data mining has been successfully applied to solve problems in several domains

This chapter offers an introduction to the fascinating world of data mining and knowledge discovery. You will learn about the basics of data mining and how data mining has been applied to solve real-world problems. In Section 1.1, the emphasis is on fundamental definitions and terminology. Section 1.2 discusses how computers are best at learning concept definitions and how concept definitions can be transformed into useful patterns.

Section 1.3 offers guidance in understanding the types of problems that may be appropriate for data mining. In Section 1.4, we explain how knowledge engineering builds models by extracting problem-solving knowledge from one or more human experts. In Section 1.5, we explore why using a simple table search may not be an appropriate data mining strategy. In Section 1.6, we offer a simple process model for data mining. Section 1.7 is about big data and cloud computing. Section 1.8 addresses the important issue of ethics in data mining. Section 1.9 looks at one of the most common business applications for data mining. We conclude this chapter, as well as all chapters in this book, with a short summary, key term definitions, and a set of exercises. Let's get started!

1.1 DATA SCIENCE, ANALYTICS, MINING, AND KNOWLEDGE DISCOVERY IN DATABASES

The era of *big data* is upon us—data so challenging that traditional data processing techniques are lacking. The International Data Corporation (IDC) predicts that by the year 2020, the digital universe will grow to 40,000 exabytes (1 exabyte = 10^{18} bytes). Considering that 5 exabytes represents all of the words ever spoken by humans, we can safely say that's a lot of data! Some of these data have the potential to provide useful knowledge—knowledge of the type that leads to better diagnostic tools, helps find disease cures, efficiently monitors and predicts the Earth's ecosystem, determines customer satisfaction, helps analyze political sentiment, detects fraudulent behavior, and even gives direction as to what stocks to buy and sell. But how do we extract this potentially useful knowledge from the mountains of data available to us? We turn to data science for help!

1.1.1 Data Science and Analytics

Data science or *data analytics* is defined as the process of extracting meaningful knowledge from data. Its methods come from several disciplines including computer science, mathematics, statistics, data warehousing, and distributed processing to name a few. A quick Internet search on data science leads to several terms with a similar definition, including *data analytics*, *predictive analytics*, *knowledge discovery from data*, *machine learning*, and *data mining*. To be brief, there is no lack of information depicting the similarities and differences between these terms. Clearly, this is to be expected, as confusing terminology is the rule rather than the exception with any evolving discipline.

Data science and data analytics are often equated and are recognized as the broader terms among those listed. Several universities offer master's degree programs featuring one of the two titles. We use both terms but favor *data analytics* as it implies actionable behavior.

The data analytics process produces models for solving problems. The basic steps of this process include acquiring data, preprocessing data, modeling, testing, reporting results, and repeating as necessary. Data to be acquired may be housed in an Excel spreadsheet but more likely will come from a data warehouse made up of historical data or streaming in from a computer network. Model building takes place only after acquired data are read and

preprocessed as needed. But how are the models built, and what do the models look like? These questions are often answered with the help of data mining!

1.1.2 Data Mining

We define *data mining* as the process of finding interesting structure in data. The structure may take many forms, including a set of rules, a graph or network, a tree, one or several equations, and more. The structure can be part of a complex visual dashboard or as simple as a list of political candidates and an associated number representing voter sentiment based on Twitter feeds.

A data mining session uses one or several algorithms for the purpose of identifying interesting trends and patterns within data. The knowledge gained from a data mining session is a generalized model of the data. The ultimate goal is to apply what has been discovered to new situations.

Several data mining techniques exist. However, all data mining methods use *induction-based learning*. Induction-based learning is the process of forming general concept definitions by observing specific examples of concepts to be learned.

In the next chapter, we show examples of the structures produced by data mining techniques without detailing the algorithms used to build them. Beyond Chapter 2, we describe the most common algorithms used to create these structures, and show you how to build models using these algorithms with the help of two well-known knowledge discovery tools.

1.1.3 Data Science versus Knowledge Discovery in Databases

Data mining finds its origins in the areas of statistics, mathematics, machine learning, artificial intelligence, and business. The term first appeared in the academic community around 1995. The phrase *knowledge discovery in databases* (KDD) was coined in 1989 to emphasize that knowledge can be derived from data-driven discovery and is frequently used interchangeably with *data mining*.

Technically, KDD is the application of the scientific method to data mining. In addition to performing data mining, a typical KDD process model includes a methodology for extracting and preparing data as well as making decisions about actions to be taken once data mining has taken place. When a particular application involves the analysis of large volumes of data stored in several locations, data extraction and preparation become the most time-consuming parts of the discovery process.

The primary goal of both data science and KDD is to extract useful knowledge from data. However, there are two clear distinctions. First, data science is often associated with large volumes of data, whereas KDD has its origins in the database community. Secondly, data mining is a required step within the KDD process model. Although data mining is often seen in data science applications, it is not required. The only requirement is that the models used for the analytics process come from data.

Finally, along with data science, we have seen the advent of big data, distributed data mining, and cloud computing. Through these changes, data mining remains a mainstay within the process of turning data into knowledge. Whether the acquired data come from

an Excel spreadsheet or a set of nodes distributed among several computers, the fundamental data mining algorithms and techniques you are about to study endure!

1.2 WHAT CAN COMPUTERS LEARN?

Data mining is about learning. Learning is a complex process. Computers are good at some types of learning but not others. Four levels of learning can be differentiated (Merril and Tennyson, 1977):

- *Facts.* A fact is a simple statement of truth.
- *Concepts.* A concept is a set of objects, symbols, or events grouped together because they share certain characteristics.
- *Procedures.* A procedure is a step-by-step course of action to achieve a goal. We use procedures in our everyday functions as well as in the solution of difficult problems.
- *Principles.* Principles represent the highest level of learning. Principles are general truths or laws that are basic to other truths.

Concepts are the output of a data mining session. The data mining tool dictates the form of learned concepts. As stated previously, common concept structures include trees, rules, networks, and mathematical equations. Tree structures and production rules are easy for humans to interpret and understand. Networks and mathematical equations are black-box concept structures in that the knowledge they contain is not easily understood. We will examine these and other data mining structures throughout the text. First, we will take a look at three common concept views.

1.2.1 Three Concept Views

Concepts can be viewed from different perspectives. An understanding of each view will help you categorize the data mining techniques discussed in this text. Let's take a moment to define and illustrate each view.

1.2.1.1 *The Classical View*

The *classical view* attests that all concepts have definite defining properties. These properties determine if an individual item is an example of a particular concept. The classical-view definition of a concept is crisp and leaves no room for misinterpretation. This view supports all examples of a particular concept as being equally representative of the concept. Here is a rule that employs a classical-view definition of a good credit risk for an unsecured loan:

IF Annual Income $\geq \$45,000$ and Years at Current Position ≥ 5 and Owns Home = True
THEN Good Credit Risk = True

The classical view states that all three rule conditions must be met for the applicant to be considered a good credit risk.

1.2.1.2 The Probabilistic View

The *probabilistic* view does not require concept representations to have defining properties. The probabilistic view holds that concepts are represented by properties that are probable of concept members. The assumption is that people store and recall concepts as generalizations created from individual exemplar (instance) observations. A probabilistic-view definition of a good credit risk might look like this:

- The mean annual income for individuals who consistently make loan payments on time is \$45,000.
- Most individuals who are good credit risks have been working for the same company for at least 5 years.
- The majority of good credit risks own their own home.

This definition offers general guidelines about the characteristics representative of a good credit risk. Unlike the classical-view definition, this definition cannot be directly applied to achieve an answer about whether a specific person should be given an unsecured loan. However, the definition can help with the decision-making process by associating a probability of membership with a specific classification. For example, a homeowner with an annual income of \$39,000 employed at the same position for 4 years might be classified as a good credit risk with a probability of 0.85.

1.2.1.3 The Exemplar View

Like the probabilistic view, the exemplar view does not require concept representations to have defining properties. The *exemplar view* states that a given instance is determined to be an example of a particular concept if the instance is similar enough to a set of one or more known examples of the concept. The view attests that people store and recall likely concept exemplars that are then used to classify new instances. Consider the loan applicant described in the previous paragraph. The applicant would be classified as a good credit risk if the applicant were similar enough to one or more of the stored instances representing good-credit-risk candidates. Here is a possible list of exemplars considered to be good credit risks:

- Exemplar #1:

Annual income = \$42,000

Number of years at current position = 6

Homeowner

- Exemplar #2:

Annual income = \$62,000

Number of years at current position = 16

Renter

- Exemplar #3:

Annual income = \$38,000

Number of years at current position = 12

Homeowner

As with the probabilistic view, the exemplar view can associate a probability of concept membership with each classification.

1.2.2 Supervised Learning

As we have seen, concepts can be studied from at least three points of view. In addition, concept definitions can be formed in several ways. Supervised learning is probably the best-understood concept-learning method and the most widely used technique for data mining.

When we are young, we use induction to form basic concept definitions. We see instances of concepts representing animals, plants, building structures, and the like. We hear the labels given to individual instances and choose what we believe to be the defining concept features (attributes, e.g., legs, leaves, windows) and form our own classification models (if leaves, then plant). Later, we use the models we have developed to help us identify objects of similar structure (has leaves, so is a plant). The name for this type of learning is induction-based supervised concept learning or just *supervised learning*.

The purpose of supervised learning is twofold. First, we use supervised learning to build classification models from sets of data containing examples and nonexamples (has legs, so is not a plant) of the concepts to be learned. Each example or nonexample is known as an *instance* of data. Second, once a classification model has been constructed, the model is used to determine the classification of newly presented instances of unknown origin. It is worth noting that, although model creation is inductive, applying the model to classify new instances of unknown origin is a deductive process.

To more clearly illustrate the idea of supervised learning, consider the hypothetical data set shown in Table 1.1. The data set is very small and is relevant for illustrative purposes only. The table data are displayed in *attribute-value format* where the first row shows names for the attributes whose values are contained in the table. The attributes *sore throat*, *fever*, *swollen glands*, *congestion*, and *headache* are possible symptoms experienced by individuals who have a particular affliction (a *strep throat*, a *cold*, or an *allergy*). These attributes are known as *input attributes* and are used to create a model to represent the data. *Diagnosis* is the attribute whose value we wish to predict. *Diagnosis* is known as the *class* or *output attribute*.

TABLE 1.1 Hypothetical Training Data for Disease Diagnosis

Patient ID	Sore Throat	Fever	Swollen Glands	Congestion	Headache	Diagnosis
1	Yes	Yes	Yes	Yes	Yes	Strep throat
2	No	No	No	Yes	Yes	Allergy
3	Yes	Yes	No	Yes	No	Cold
4	Yes	No	Yes	No	No	Strep throat
5	No	Yes	No	Yes	No	Cold
6	No	No	No	Yes	No	Allergy
7	No	No	Yes	No	No	Strep throat
8	Yes	No	No	Yes	Yes	Allergy
9	No	Yes	No	Yes	Yes	Cold
10	Yes	Yes	No	Yes	Yes	Cold

Starting with the second row of the table, each remaining row is an *instance* of data. An individual row shows the symptoms and affliction of a single patient. For example, the patient with ID = 1 has a sore throat, fever, swollen glands, congestion, and a headache. The patient has been diagnosed as having strep throat.

Suppose we wish to develop a generalized model to represent the data shown in Table 1.1. Even though this data set is small, it would be difficult for us to develop a general representation unless we knew something about the relative importance of the individual attributes and possible relationships among the attributes. Fortunately, an appropriate supervised learning algorithm can do the work for us.

1.2.3 Supervised Learning: A Decision Tree Example

We presented the data in Table 1.1 to C4.5 (Quinlan, 1993), a supervised learning program that generalizes a set of input instances by building a decision tree. A *decision tree* is a simple structure where nonterminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes. Decision trees have several advantages in that they are easy for us to understand, can be transformed into rules, and have been shown to work well experimentally. A supervised algorithm for creating a decision tree will be detailed in Chapter 3.

Figure 1.1 shows the decision tree created from the data in Table 1.1. The decision tree generalizes the table data. Specifically,

- If a patient has swollen glands, the disease diagnosis is strep throat.
- If a patient does not have swollen glands and has a fever, the diagnosis is a cold.
- If a patient does not have swollen glands and does not have a fever, the diagnosis is an allergy.

The decision tree tells us that we can accurately diagnose a patient in this data set by concerning ourselves only with whether the patient has swollen glands and a fever. The attributes sore throat, congestion, and headache do not play a role in determining a diagnosis.

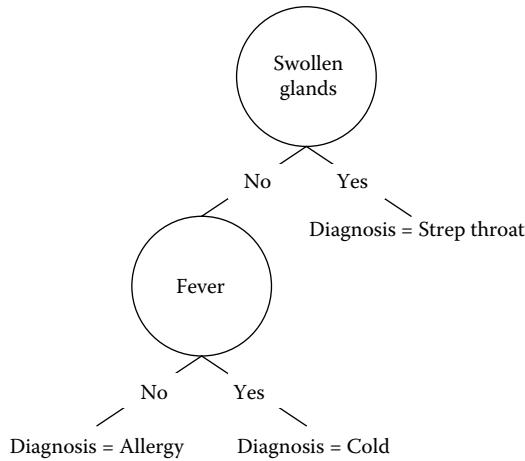


FIGURE 1.1 A decision tree for the data in Table 1.1.

TABLE 1.2 Data Instances with an Unknown Classification

Patient ID	Sore Throat	Fever	Swollen Glands	Congestion	Headache	Diagnosis
11	No	No	Yes	Yes	Yes	?
12	Yes	Yes	No	No	Yes	?
13	No	No	No	No	Yes	?

As we can see, the decision tree has generalized the data and provided us with a summary of those attributes and attribute relationships important for an accurate diagnosis.

Let's use the decision tree to classify the first two instances shown in Table 1.2.

- Since the patient with ID = 11 has a value of *yes* for *swollen glands*, we follow the right link from the root node of the decision tree. The right link leads to a terminal node, indicating that the patient has strep throat.
- The patient with ID = 12 has a value of *no* for *swollen glands*. We follow the left link and check the value of the attribute *fever*. Since *fever* equals *yes*, we diagnose the patient with a cold.

We can translate any decision tree into a set of *production rules*. Production rules are rules of the following form:

IF *antecedent conditions*

THEN *consequent conditions*

The antecedent conditions detail values or value ranges for one or more input attributes. The consequent conditions specify the values or value ranges for the output attributes. The technique for mapping a decision tree to a set of production rules is simple. A rule is

created by starting at the root node and following one path of the tree to a leaf node. The antecedent of a rule is given by the attribute-value combinations seen along the path. The consequent of the corresponding rule is the value at the leaf node. Here are the three production rules for the decision tree shown in Figure 1.1:

1. IF *Swollen Glands* = Yes

THEN *Diagnosis* = *Strep Throat*

2. IF *Swollen Glands* = No and *Fever* = Yes

THEN *Diagnosis* = *Cold*

3. IF *Swollen Glands* = No and *Fever* = No

THEN *Diagnosis* = *Allergy*

Let's use the production rules to classify the table instance with patient ID = 13. Because *swollen glands* equals *no*, we pass over the first rule. Likewise, because *fever* equals *no*, the second rule does not apply. Finally, both antecedent conditions for the third rule are satisfied. Therefore, we are able to apply the third rule and diagnose the patient as having an allergy.

The instances used to create the decision tree model are known as *training data*. At this point, the training instances are the only instances known to be correctly classified by the model. However, our model is useful to the extent that it can correctly classify new instances whose classification is not known. To determine how well the model is able to be of general use, we test the accuracy of the model using a *test set*. The instances of the test set have a known classification. Therefore, we can compare the test set instance classifications determined by the model with the correct classification values. Test set classification correctness gives us some indication about the future performance of the model.

1.2.4 Unsupervised Clustering

Unlike supervised learning, *unsupervised clustering* builds models from data without pre-defined classes. Data instances are grouped together based on a similarity scheme defined by the clustering system. With the help of one or several evaluation techniques, it is up to us to decide the meaning of the formed clusters.

To further distinguish between supervised learning and unsupervised clustering, consider the hypothetical data in Table 1.3. The table provides a sampling of information about five customers maintaining a brokerage account with Acme Investors Incorporated. The attributes *customer ID*, *gender*, *age*, *favorite recreation*, and *income* are self-explanatory. *Account type* indicates whether the account is held by a single person (individual account), two or more persons (joint account), or a person or institution under a safekeeping agreement on behalf of one or more individuals (custodial account). *Transaction method* tells us whether the purchase and sale of stock is made online or through a broker. *Trades/month* indicates the average number of stock transactions per month. Finally, if the account is a

TABLE 1.3 Acme Investors Incorporated

Customer ID	Account Type	Margin Account	Transaction Method	Trades/Month	Gender	Age	Favorite Recreation	Annual Income
1005	Joint	No	Online	12.5	F	30–39	Tennis	40–59K
1013	Custodial	No	Broker	0.5	F	50–59	Skiing	80–99K
1245	Joint	No	Online	3.6	M	20–29	Golf	20–39K
2110	Individual	Yes	Broker	22.3	M	30–39	Fishing	40–59K
1001	Individual	Yes	Online	5.0	M	40–49	Golf	60–79K

margin account, the customer is allowed to borrow cash from the investment firm to purchase new securities.

Suppose we wish to use data mining together with the brokerage database to gain insight about possible patterns in the database. In so doing, we might ask the following four questions:

1. Can I develop a general profile of an online investor? If so, what characteristics distinguish online investors from investors that use a broker?
2. Can I determine if a new customer that does not initially open a margin account is likely to do so in the future?
3. Can I build a model able to accurately predict the average number of trades per month for a new investor?
4. What characteristics differentiate female and male investors?

Each question is a candidate for supervised data mining because each question clearly offers an attribute whose values represent a set of predefined output classes. For question 1, the output attribute is *transaction method*. The output attribute for question 2 is *margin account*. The output attribute for question 3 is given as *trades/month*, and the output attribute for question 4 is *gender*. The answers to each of these questions can be used to develop advertising campaigns for new customers as well as marketing strategies for existing customers.

Alternatively, we can ask one or more general questions about the data. Here are two candidate questions for unsupervised clustering:

1. What attribute similarities group customers of Acme Investors together?
2. What differences in attribute values segment the customer database?

Many unsupervised clustering systems require us to provide an initial best estimate about the total number of clusters in the data. Other clustering systems use an algorithm in an attempt to determine a best number of clusters. In either case, a clustering system

will attempt to group instances into clusters of significant interest. Let's assume we have presented the Acme Investors data to an unsupervised clustering model and three clusters were formed. Here is a possible rule from each cluster:

IF Margin Account = Yes and Age = 20–29 and Income = 40–59K

THEN Cluster = 1

Rule precision: 80%

Rule coverage: 25%

IF Account Type = Custodial and Recreation = Skiing and Income = 80–90K

THEN Cluster = 2

Rule precision: 95%

Rule coverage: 15%

IF Account Type = Joint and Trades/Month > 5 and Transaction Method = Online

THEN Cluster = 3

Rule precision: 82%

Rule coverage: 55%

Values for rule precision and coverage are stated after the consequent condition for each rule. These values give important information about rule confidence and rule significance. To illustrate, assume the entire data set consists of 1000 investors. The rule for cluster 1 shows a 25% rule coverage. This tells us that 250 or 25% of 1000 investors satisfy the rule's antecedent condition. That is, 250 investors hold a margin account, are between 20 and 29 years of age, and have an annual income between \$40,000 and \$59,000. The precision score gives the accuracy of the rule relative to the 250 investors satisfying the antecedent condition. Specifically, 80% or 200 of the 250 individuals that satisfy the antecedent condition are found within cluster 1. The remaining 50 investors are in either cluster 2 or 3. Stated another way, the rule does not apply in 20% of the cases where the antecedent conditions have been met.

The rule for cluster 1 is not a surprise, as we expect younger investors with a reasonable income to take a less conservative approach to investing. In addition, the rule for cluster 3 is not likely to be a new discovery. However, the rule for cluster 2 could be unexpected and therefore useful. The Acme Investors Corporation might take advantage of this knowledge by investing some of their advertising money in ski magazines with the ads promoting custodial accounts for children and/or grandchildren.

The simple examples given help illustrate the basic concepts surrounding supervised and unsupervised learning. The more difficult tasks such as defining a suitable problem to solve, preparing the data, choosing a data mining strategy, and evaluating performance are topics addressed in the remaining chapters of the text. The next section offers guidelines to help us determine when data mining is an appropriate problem-solving strategy.

1.3 IS DATA MINING APPROPRIATE FOR MY PROBLEM?

Making decisions about whether to use data mining as a problem-solving strategy for a particular problem is a difficult task. As a starting point, we offer four general questions to consider:

1. Can we clearly define the problem?
2. Do potentially meaningful data exist?
3. Do the data contain hidden knowledge, or are the data factual and useful for reporting purposes only?
4. Will the cost of processing the data be less than the likely increase in profit seen by applying any potential knowledge gained from the data mining project?

The first two questions as well as the fourth cannot be addressed properly in a few sentences. The tools to help us answer these questions for a particular problem are provided throughout the chapters of this text. However, you can gain insight on how to answer the third question with a few simple examples. We begin by differentiating between four types of knowledge.

1.3.1 Data Mining or Data Query?

Four general types of knowledge can be defined to help us determine when data mining should be considered.

- *Shallow knowledge* is factual in nature. Shallow knowledge can be easily stored and manipulated in a database. Database query languages such as SQL are excellent tools for extracting shallow knowledge from data.
- *Multidimensional knowledge* is also factual. However, in this case, data are stored in a multidimensional format. Online analytical processing (OLAP) tools are used on multidimensional data. OLAP and multidimensional knowledge are covered in Chapter 14.
- *Hidden knowledge* represents patterns or regularities in data that cannot be easily found using a database query language such as SQL. However, data mining algorithms can find such patterns with ease.
- *Deep knowledge* is knowledge stored in a database that can only be found if we are given some direction about what we are looking for. Current data mining tools are not able to locate deep knowledge.

Database query languages and OLAP tools are very good at finding and reporting information within a database when we know exactly what we are looking for. Database queries can easily be written to extract the following information:

- A list of all Acme Department Store customers who used a credit card to buy a gas grill
- A list of all employees over the age of 40 who have averaged five or fewer sick days per year

- A list of all patients who have had at least one heart attack and whose blood cholesterol is below 200
- A list of all credit card holders who used their credit card to purchase more than \$300 in groceries during the month of January

The output of these queries could very well provide valuable information to help make future decisions. Data mining takes us one step further in that it provides us with potentially useful information even when we only have a vague idea about what we are looking for. What is even more exciting is that data mining gives us the ability to find answers to questions we never thought about asking!

Here are a few simple examples of what data mining can do:

- Develop a general profile for credit card customers who take advantage of promotions offered with their credit card billing
- Differentiate individuals who are poor credit risks from those who are likely to make their loan payments on time
- Classify faint objects found within sky image data
- Determine when a patient is likely to return to work after major back surgery

Database query may be able to help us with these examples, but our chances of success are slim. With data mining, if the data contain information of potential value, we are likely to succeed in our efforts. The next section offers a specific example to help make the distinction between data mining and data query clear.

1.3.2 Data Mining versus Data Query: An Example

Let's take a look at a specific example to help us determine when data mining is an appropriate problem-solving strategy. Once again, consider the hypothetical data in Table 1.1. As mentioned earlier, these data contain information about individual patients, their symptoms, and a corresponding affliction. As the data set is small, it seems almost trivial to consider examining the individual instances for patterns. If you like, use your imagination to envision a much larger data set with thousands of records and several additional attributes.

As a first step, we can state a general *hypothesis* about what we hope to find in the data set. A hypothesis is an educated guess about what we believe to be true for some or all of the data. For our data set, we might state a general hypothesis as follows:

One or more of the attributes in the data set are relevant in accurately differentiating between individuals who have strep throat, a cold, or an allergic reaction.

Suppose that through personal experience, we have some additional insight indicating that swollen glands are paramount in predicting strep throat. Therefore, we state a second hypothesis as follows:

Swollen glands are necessary and sufficient for strep throat.

We pose two table queries to test this hypothesis:

1. List all patients with *Diagnosis = Strep Throat* and *Swollen Glands = No*.
2. List all patients with *Diagnosis = Cold* or *Diagnosis = Allergy* and *Swollen Glands = Yes*.

The result of the first query is an empty table. Therefore, we conclude that swollen glands are a necessary condition for strep throat. That is, all patients with strep throat also have swollen glands. The result of the second query is also an empty table. This tells us that any patient who does not have strep throat does not have swollen glands. Combining the results of the two queries, we conclude that swollen glands are a necessary and sufficient condition for strep throat. We form the following general rule:

*IF Swollen Glands = Yes
THEN Diagnosis = Strep Throat*

Our next step is to determine a rule or set of rules to differentiate patients with colds from patients with allergies. Once again, we form a hypothesis about what we believe to be true and test the hypothesis by performing one or more table queries. The process we are describing is sometimes referred to as *manual data mining*. We pose queries and maintain control of the search for patterns in data. We will be effective to the extent that we already almost know what we are looking for. If the data set is small, and if we have a wealth of knowledge about the attributes and the attribute relationships in the data, this approach is appropriate. However, under conditions other than these, our chances of success are limited.

1.4 DATA MINING OR KNOWLEDGE ENGINEERING?

There are times when a data mining or data query approach to problem solving is not feasible. An obvious scenario is any situation where quality data are lacking. When data mining is not a viable choice, other options may be available that allow us to build useful decision-making models. One possibility is to locate one or more human beings who are able to find solutions to the problems we are interested in solving.

Individuals who have the ability to solve problems in one or more difficult problem domains are often referred to as *experts* in their field. Examples include medical doctors who are able to quickly diagnose disease, business executives who are able to make timely decisions, and counselors who are good at helping us with our personal problems. Experts

learn their skills by way of education and experience. Experience over a period of several years helps experts develop skills that enable them to solve problems quickly and efficiently.

Computer scientists develop computer programs called *expert systems* that can emulate the problem-solving skills of human experts in specific problem domains. The word *emulate* means to “act like.” To emulate a human expert with a computer program means that the program must solve problems using methods similar to those employed by the expert. As we are far from understanding the workings of the human brain, these programs concentrate on emulating the knowledge of human experts rather than their methods of applying knowledge to specific problem situations. Because human experts often use rules to describe what they know, most expert systems incorporate rules as a main medium for storing knowledge.

Figure 1.2 uses our hypothetical domain for disease diagnosis to compare the problem-solving methodologies of data mining and expert systems. The data mining approach uses

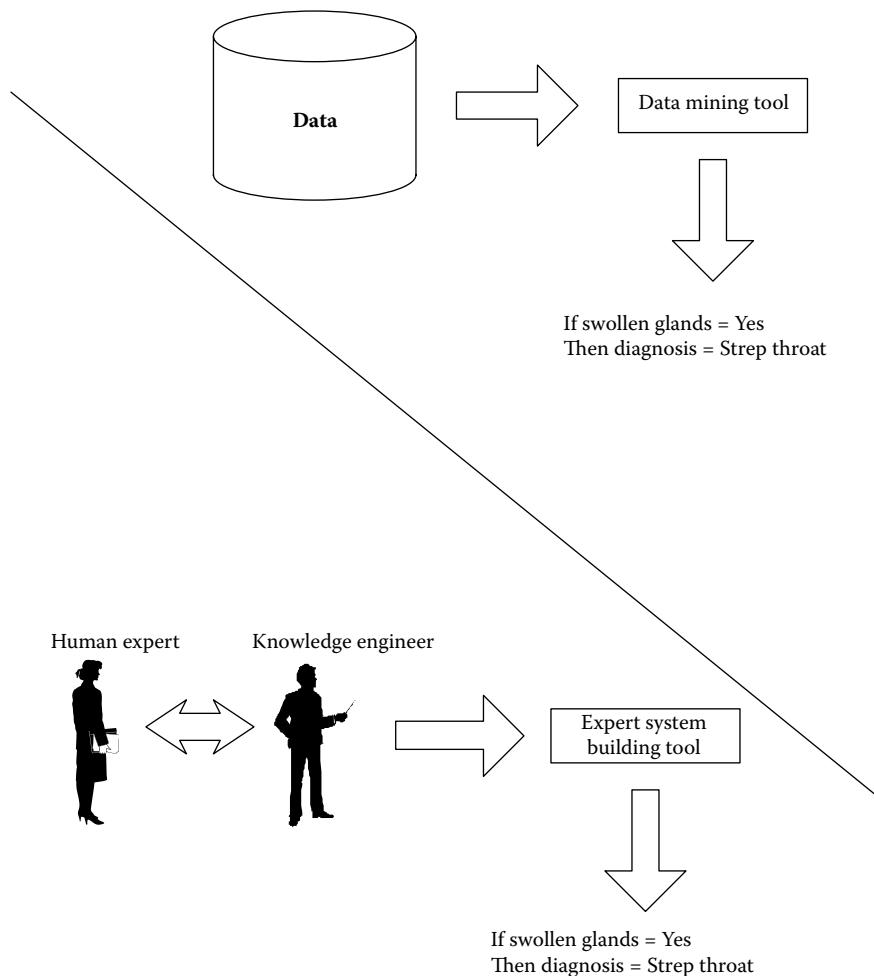


FIGURE 1.2 Data mining versus expert systems.

the data in Table 1.1 together with a data mining tool to create a rule for diagnosing strep throat. In contrast, the expert systems approach employs human beings rather than data. The process involves at least two individuals—an expert and a knowledge engineer. A *knowledge engineer* is a person trained to interact with an expert in order to capture his/her knowledge. Once captured, the knowledge engineer uses one or more automated tools to create a computer model of the new knowledge. For the example shown in Figure 1.2, the human expert is likely a medical doctor. Notice that the model describing the knowledge extracted from the human expert is identical to the model built with the data mining process. In many situations, expert systems and data mining can be collectively applied to solve difficult problems.

1.5 A NEAREST NEIGHBOR APPROACH

Data mining is used to build generalized models to represent unstructured data. For classification problems, we might consider an alternative approach. Specifically,

- Create a classification table containing all data items whose classification is known.
- For each new instance we wish to classify,
 - Compute a score representing the similarity of each table item to the new instance.
 - Give the new instance the same classification as the table item to which it is most similar.
- Add the newly classified instance to the table of known instances.

This approach is called the *nearest neighbor classification* method. As we can see, the approach stores instances rather than a generalized model of the data. There are at least three problems with this approach. First, computation times will be a problem when the classification table contains thousands or millions of records. Second, the approach has no way of differentiating between relevant and irrelevant attributes. Third, we have no way to tell whether any of the chosen attributes are able to differentiate the classes contained in the data.

The first problem is prohibitive when we compare this approach to that of building a generalized classification model such as a decision tree. To illustrate the second problem, we once again turn to the hypothetical data found in Table 1.1. As this table contains instances whose classification is known, the nearest neighbor algorithm can use the table data to classify the new instance defined with the following attribute values:

Patient ID = 14

Sore Throat = Yes

Fever = No

Swollen Glands = No

Congestion = No

Headache = No

When the attributes are numerical, simple Euclidean distance can be used to measure instance similarity. However, as the attributes shown in Table 1.1 are categorical, we will have to define our own measure of similarity. To measure the similarity of each table item with this instance, we can simply count attribute-value matches. Upon doing so, we have one match with patients 1 and 9; two matches with patients 2, 5, and 10; and three matches with patients 3, 6, 7, and 8. Finally, we have four matches with patient 4.

The constructed decision tree shown in Figure 1.1 for these data tells us that the critical attribute differentiating strep throat from allergy and cold is the value of *swollen glands*. However, because the new instance to be classified shows a best match with patient 4, the nearest neighbor approach tells us the patient has a case of strep throat. In addition, if this is indeed an incorrect diagnosis, adding the new instance to the classification table will continue to propagate classification error. To be sure, we are making an assumption that the instances in the training data used to build the decision tree accurately represent the population in general. With this assumption, the correct diagnosis for the new patient is allergy.

A variation of this approach known as a *k-nearest neighbor classifier* classifies a new instance with the most common class of its k nearest neighbors. This variation helps ward off the possibility of a single atypical training instance incorrectly classifying new instances. However, the important issue is that neither approach has a way to determine relevant attributes and will likely produce a high classification error rate when presented with instances containing a wealth of irrelevant attributes. Also, contrary to our intuition, a data set having several independent attributes will show distances between any two instances as being almost equal. This is true even if we are processing hundreds of thousands of records!

The nearest neighbor approach can be successfully applied if attribute preprocessing is able to determine a best set of relevant attributes. Also, computation times can be reduced by comparing instances to be classified with a subset of typical instances taken from each class represented in the data. Even though the nearest neighbor approach does not provide us with a generalization of the data, a general description of each class can be obtained by examining sets of the most typical class instances. Despite its shortcomings, the nearest neighbor remains one of the top 10 data mining algorithms used today (Wu and Kumar, 2009).

1.6 A PROCESS MODEL FOR DATA MINING

In a broad sense, we can define data analytics as a five-step process:

1. Acquire data
2. Preprocess data

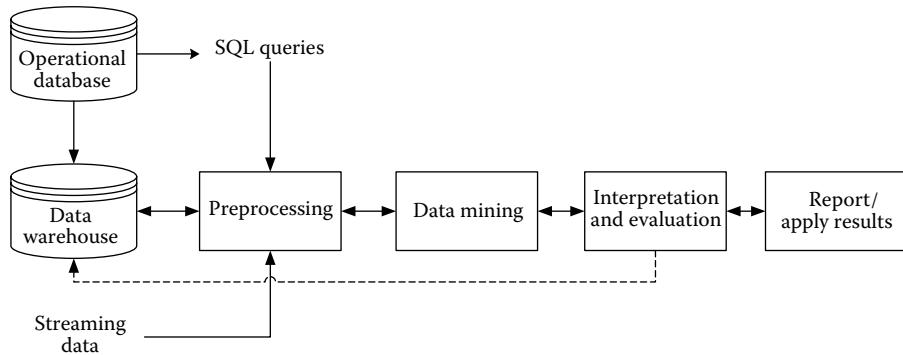


FIGURE 1.3 A process model for data mining.

3. Model data
4. Interpret and evaluate results
5. Report/apply results

Figure 1.3 offers a pictorial diagram of this five-step process model that specifically incorporates data mining. We will use Figure 1.3 to help us describe each step of the process.

1.6.1 Acquiring Data

Data mining requires access to data. The data may be represented as volumes of records in several database files, or the data may contain only a few hundred records in a single file. A common misconception is that in order to build an effective model, a data mining algorithm must be presented with thousands or millions of instances. In fact, most data mining tools work best with a few hundred or a few thousand pertinent records. Therefore, once a problem has been defined, a first step in the data mining process is to extract or assemble a relevant subset of data for processing. Many times, this first step requires a great amount of human time and effort. There are four common data acquisition methods:

1. Access data from a data warehouse.
2. Access data from a relational database.
3. Access data from a flat file or spreadsheet.
4. Access data from servers within a distributed environment.

1.6.1.1 The Data Warehouse

A common scenario for data assembly shows data originating in one or more operational databases. *Operational databases* are transaction based and frequently designed using the relational database model. An operational database fixed on the relational model will contain several normalized tables. The tables have been normalized to reduce redundancy and promote quick access to individual records. For example, a specific customer might have

data appearing in several relational tables where each table views the customer from a different perspective.

Figure 1.3 shows data being transferred from the operational environment to a *data warehouse*. The data warehouse is a historical database designed for decision support rather than transaction processing (Kimball et al., 1998). Thus, only data useful for decision support are extracted from the operational environment and entered into the warehouse database. Data transfer from the operational database to the warehouse is an ongoing process usually accomplished on a daily basis after the close of the regular business day. Before each data item enters the warehouse, the item is time-stamped, transformed as necessary, and checked for errors. The transfer process can be complex, especially when several operational databases are involved. Once entered, the records in the data warehouse become read-only and are not subject to change.

A data warehouse stores all data relating to the same subject (such as a customer) in the same table. This distinguishes the data warehouse from an operational database, which stores information so as to optimize transaction processing. Because the data warehouse is subject oriented rather than transaction oriented, the data will contain redundancies. It is the redundancy stored in a data warehouse that is used by data mining algorithms to develop patterns representing discovered knowledge. Chapter 14 discusses the data warehouse, the relational database model, and OLAP in more detail. As Chapter 14 is self-contained, it can be read any time you wish to learn more about these topics.

1.6.1.2 Relational Databases and Flat Files

Figure 1.3 shows that if a data warehouse does not exist, you can make use of a database query language such as SQL to write one or more queries to create a table suitable for data mining. Whether data are being extracted for mining from the data warehouse or the data extraction is via a query language, you will probably need a utility program to convert extracted data to the format required by the chosen data mining tool. Finally, if a database structure to store the data has not been designed, and the amount of collected data is minimal, the data will likely be stored in a flat file or spreadsheet. Table 1.1 is representative of a flat-file format appropriate for many data mining tools.

1.6.1.3 Distributed Data Access

When data are so complex or voluminous that they must be processed in a distributed environment, they are tagged as *big data*. Examples include data gathered from social media sites, data collected from website visitors, Twitter responses, financial logs, and data streaming from network sensors to name a few. Accessing and modeling data in a distributed environment is extremely difficult. We further address issues with distributed data in Section 1.7.

1.6.2 Data Preprocessing

Data preprocessing is the most-time consuming and least rewarding piece of the analytics process. Data can contain missing values and noise and require one or several

transformations before it is ready to be mined. One basic preprocessing task involves examining the degree of correlation between pairs of numeric input attributes.

A *correlation coefficient* (designated by r for a sample or by ρ , the Greek letter *rho*, for a population) measures the degree of linear association between two numeric attributes. Correlation coefficients take on values ranging between -1 and 1 inclusive. A *positive correlation* means two attributes increase or decrease in unison. Height and weight are two attributes that exhibit a high positive correlation. Two attributes have a *negative correlation* if as one attribute increases, the other decreases. Age and running speed display a high negative correlation. If r is near 0 , the two attributes are not linearly correlated.

We can visualize correlations between pairs of attributes with a *scatterplot* diagram. Figure 1.4 is a scatterplot that shows attributes A and B as having a perfect positive correlation. Figure 1.5 displays two attributes having a perfect negative correlation.

If two candidate input attributes are highly correlated in a positive or a negative direction, only one of the attributes should be selected for data mining. Keeping both attributes is like giving an added weight of importance to a single attribute. Furthermore, caution must be exercised in making deductions from correlational evidence. Although a marked correlation between two variables implies that they are related, the relationship may or may not be of a causal nature.

You will have ample opportunity to learn about several other data preprocessing techniques as they are detailed throughout the text.

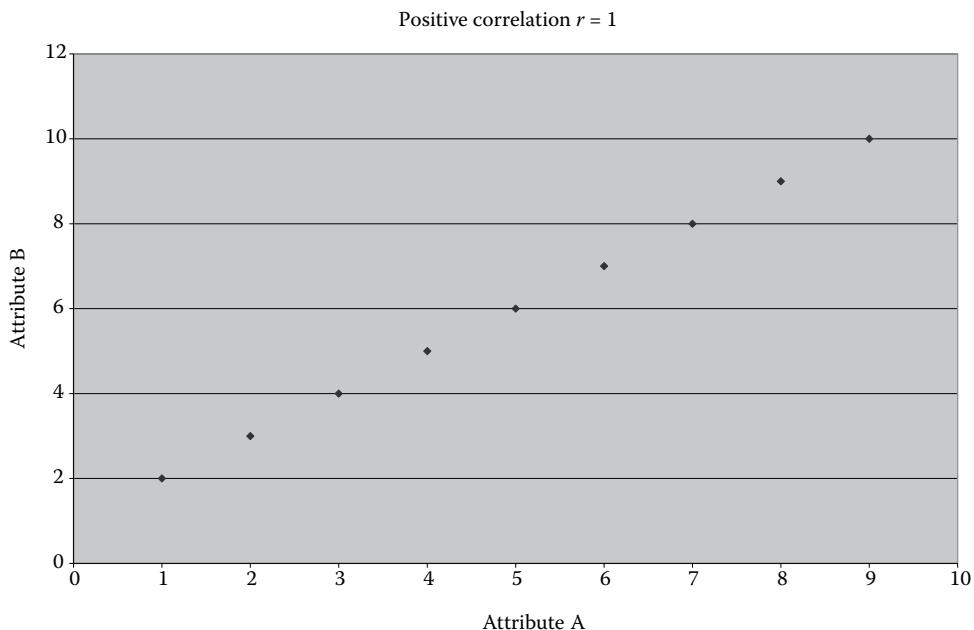


FIGURE 1.4 A perfect positive correlation ($r = 1$).

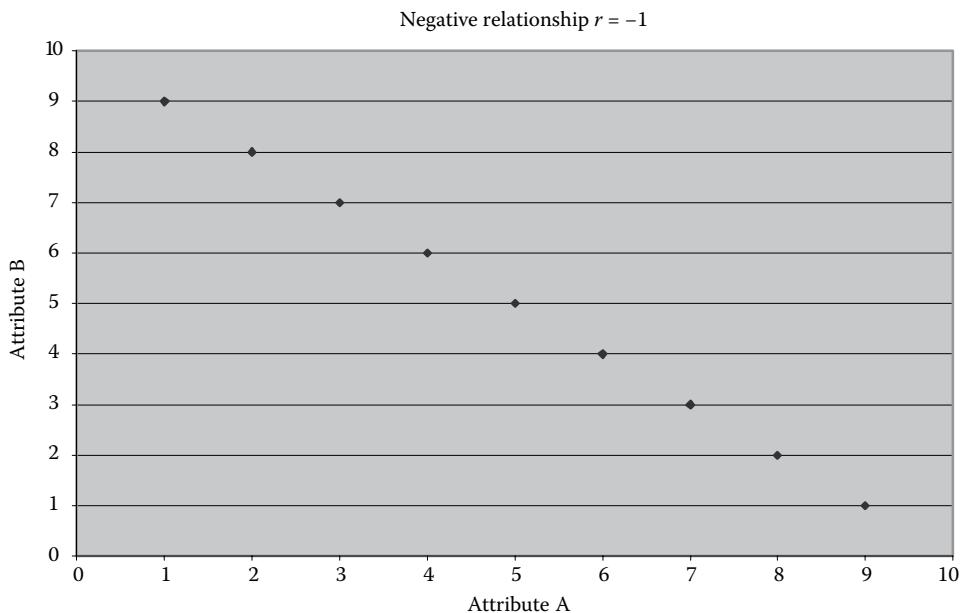


FIGURE 1.5 A perfect negative correlation ($r = -1$).

1.6.3 Mining the Data

Figure 1.3 shows that the next phase of the process is to mine the data. However, prior to giving the data to a data mining tool, we have several choices to make.

1. Should learning be supervised or unsupervised?
2. Which instances in the assembled data will be used for building the model, and which instances will test the model?
3. Which attributes will be selected from the list of available attributes?
4. Data mining tools require the user to specify one or more learning parameters. What parameter settings should be used to build a model to best represent the data?

In Chapter 6, we will provide you with several tools to help you answer each of these questions.

1.6.4 Interpreting the Results

Result interpretation requires us to examine the output of our data mining tool to determine if what has been discovered is both useful and interesting. Figure 1.3 shows that if the results are less than optimal, we can repeat the data mining step using new attributes and/or instances. Alternatively, we may decide to return to the data warehouse and repeat the data extraction process. Chapter 7 examines several techniques to help us make decisions about whether a specific model is useful.

1.6.5 Result Application

Our ultimate goal is to apply what has been discovered to new situations. Suppose through the process of a data mining market analysis, we find that product X is almost always purchased with product Y. A classic example of this is the discovery that an unusually high percentage of people who purchase baby diapers on Thursday also purchase beer. An initial surprise reaction to this finding makes sense when we realize that couples with a young baby at home are not likely to go out on Friday or Saturday night but instead prefer to enjoy the weekend by relaxing at home. A market analyst can take advantage of this finding by making beer an obvious display item for customers buying diapers.

1.7 DATA MINING, BIG DATA, AND CLOUD COMPUTING

Big data and cloud computing are the primary reasons for the evolution of modern decision science. The term *big data* is heard so much today that it seems almost commonplace. But what exactly does it mean?

Big data might be thought of as data of such volume that to process it requires more than available local resources. However, a more complete definition sees big data in four dimensions: volume, variety, velocity, and veracity. These are known as the four V's of big data (<http://www-01.ibm.com/software/data/bigdata/>).

Volume implies that the amount of data that needs to be analyzed grows so quickly that it cannot be stored and processed on a single server.

Variety and *veracity* (bias, noise, or abnormality in the data) are seen with combinations of structured data and unstructured data from various online sources such as Twitter, Facebook, and e-mail servers. Finally, *velocity* is an issue when streaming data enter and exit an infrastructure too quickly for proper analysis or response. This definition makes it clear that solutions for processing big data require several machines working together. To meet this challenge, we look to distributed computing and the cloud.

1.7.1 Hadoop

Hadoop (<http://hadoop.apache.org/>) is the most recognized open-source distributed computing environment for handling big data. The Hadoop framework is made up of two major components, the Hadoop Distributed File System (HDFS) for storing data and MapReduce for data processing (Mone, 2013). HDFS divides and redistributes data to the servers representing the cluster nodes, while MapReduce handles the distribution of tasks throughout the nodes and combines results for a final solution. Although native Hadoop requires complex programming skills, several application packages that run on top of Hadoop have been developed (Zikopoulos et al., 2012).

1.7.2 Cloud Computing

In simple terms, *cloud computing* is a method supported by servers to deliver computing resources over the Internet on a pay-as-you-go basis (Kaur and Mann, 2014). Stated another way, cloud computing offers software as a service (SaaS). The cloud is seen as a best choice for managing big data even though it is not a necessary component of a distributed

computing environment. The virtualization seen with cloud computing has made processing big data a reality.

It is important to note that data mining may or may not be part of a data analytics process. Our focus is on the algorithms applied to data both big and small but not on the implementation details and workings of an environment specifically designed for the analysis of big data. If you would like to experiment within a big data environment, Weka's 3.7 knowledge-flow interface supports a distributed Hadoop package. Also, RapidMiner interfaces with Hadoop and has its own big data package (Radoop) that does not require programming skills.

1.8 DATA MINING ETHICS

Data mining offers sophisticated tools to deduce sensitive patterns from data. Because of this, the use of data containing information about people requires special precautions. Let's look at two examples.

Human capital management (HCM) is an approach that perceives employees as assets with measureable value. The employees of a company that invests in HCM receive consistent communication about their performance expectations. Managers objectively rate employees, thereby holding them accountable for specific goals. Employees that meet or exceed expectations are rewarded, and those consistently falling short are terminated. With this type of system, it is easy to keep objective records of current employees and to maintain previous employee records in a warehouse facility.

Data mining can be used in a positive way as one of several tools to help associate current employees with tasks or positions for which they are best suited. Data mining can also be misused by building a classification model that differentiates former employees relative to their termination. Possibilities include that a former employee quit, was fired, was laid off, retired, or is deceased. The employer then applies the model built from former employee data to current employees and fires those the model predicts are likely to quit. Furthermore, the employer may use the model to lay off or fire employees predicted to retire soon. This use of data mining is certainly less than ethical.

As a second example, anonymized health records are public information as are people's names. However, being able to associate people with their individual health records results in creating private information. This process of being able to deduce unauthorized or private information from publicly available data is known as the *inference problem* (Grossman et al., 2002). Here are three ways to approach this issue.

- Given a database and a data mining tool, apply the tool to determine if sensitive information can be deduced.
- Use an inference controller to detect the motives of the user.
- Give only samples of the data to the user, thereby preventing the user from building a data mining model.

Each approach is appropriate in certain situations but not others. When privacy of the individual is at stake, exercising wisdom and great caution is the best approach.

1.9 INTRINSIC VALUE AND CUSTOMER CHURN

A customer's intrinsic value is the customer's expected value based on the historical value of similar customers (Manganaris, 2000). A customer whose intrinsic expectations are not seen is likely to *churn*. Churning is the process whereby a customer discontinues use of a service or subscription. This is a frequently seen with cellular phone subscriptions, Internet services, lawn care, and credit card usage to name a few. It is well known that it is more expensive to win back a churned customer than to identify those likely to churn and offer an incentive for them to remain a loyal customer.

Data mining has been used to build models for predicting intrinsic value. Once a customer's intrinsic value is determined, an appropriate marketing strategy can be applied. Let's assume value is measured in terms of average dollars spent per month by the customer. This being the case, the idea is to build a model able to predict what an individual customer is likely to spend in any given month. The amount spent by the customer is then compared to the customer's intrinsic value to determine if the customer is spending more or less than individuals with similar characteristics.

To focus on a specific example, suppose a credit card company is concerned about a decline in new customer retention rates as well as negative changes in new customer card usage habits. In an attempt to overcome their concern, the company has hired a data mining consulting firm. The data mining specialists use data about established credit card holders to build a historical model for predicting customer spending and card usage habits. Possible attributes of importance include customer age, income range, demographic location, average monthly balance, and number of credit purchases per month to name a few.

Once built, the model is applied to predict intrinsic value for the population of new customers. Figure 1.6 offers a diagram showing intrinsic value on the *vertical axis* and actual

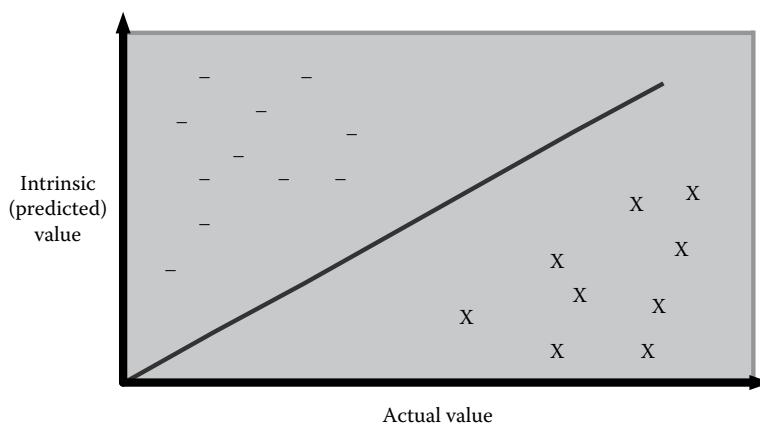


FIGURE 1.6 Intrinsic versus actual customer value.

customer value on the *horizontal* axis. We can use this figure to help us understand three possibilities for each new customer.

1. *A new customer's intrinsic value is in line with his/her current value.* These are the customers falling on or close to the diagonal line shown in Figure 1.6. This group of individuals will likely benefit most from a mildly aggressive marketing strategy aimed at increasing card usage.
2. *A new customer's intrinsic value is greater than his/her actual value.* This group of customers is shown in Figure 1.6 as the region above the diagonal. These individuals are likely to eventually quit using their credit card unless an aggressive marketing strategy is deployed.
3. *A new customer's intrinsic value is less than his/her actual value.* This is seen in Figure 1.6 as the region below the diagonal. Customers in the lowest-right portion of this region show the largest discrepancy between intrinsic and actual value. A nonaggressive marketing strategy offering intermittent rewards for continued card use may be appropriate for these individuals.

This example illustrates how data mining can be applied to profile strategies for increasing customer retention rates and future value. As with all data mining problems, successful application of intrinsic modeling lies in our ability to acquire quality data as well as in our willingness to devote time and effort to the model building process.

1.10 CHAPTER SUMMARY

Data mining was once little more than an experimental component of companies that had sufficient resources to apply to such experiments. Today, data mining is used by thousands of companies and organizations to monitor user experiences, calculate return on investment for advertising campaigns, predict flooding, detect fraud, and even help design vehicles. The applications for data mining seem almost endless (see Exercise 4).

A model created by a data mining algorithm is a conceptual generalization of the data. Common generalizations take the form of a tree, a network, one or more equations, or a set of rules. Data mining differs from data query in that data mining gives us the ability to find answers to questions we never thought about asking.

Data mining is a multistep process that requires accessing and preparing data for a data mining algorithm, mining the data, analyzing results, and taking appropriate action. The most common type of data mining is supervised. With supervised learning, instances whose classification is known are used by the data mining tool to build a general model representing the data. The created model is then employed to determine the classification of new, previously unclassified instances. With unsupervised clustering, predefined concepts do not exist. Instead, data instances are grouped together based on a similarity scheme defined by the clustering model.

In the next chapters, you will learn more about the steps of the data mining process. You will learn about different data mining algorithms, as well as several techniques to help you determine when to apply data mining to your problems. A common theme we hope to convey throughout this book is that data mining is about model building and not about magic. Human nature requires that we generalize and categorize the world around us. For this reason, model building is a natural process that can be fun and very rewarding!

Finally, you can find a wealth of additional information about data mining applications, companies, jobs, public domain and commercial software, as well as seminars and short courses by visiting the website <http://www.kdnuggets.com>.

1.11 KEY TERMS

- *Attribute-value format.* A table format where the first row of the table contains an attribute name. Each row of the table after the first contains a data instance whose attribute values are given in the columns of the table.
- *Churning.* The process whereby a customer terminates a service or subscription with one company in order to obtain the service with a competing company.
- *Classical view.* The view that all concepts have definite defining properties.
- *Cloud computing.* A method supported by nodes of remote servers to deliver computing resources over the Internet.
- *Concept.* A set of objects, symbols, or events grouped together because they share certain characteristics.
- *Correlation coefficient.* A correlation coefficient measures the degree of linear association between two numeric attributes. Correlation coefficients may take on values between -1 and 1 inclusive. A *positive correlation* means two attributes increase or decrease in unison. Two attributes show a *negative correlation* if as one attribute increases, the other decreases.
- *Data mining.* The process of finding interesting structure in data.
- *Data science/data analytics.* The process of extracting meaningful knowledge from data.
- *Data warehouse.* A historical database designed for decision support rather than transaction processing.
- *Decision tree.* A tree structure where nonterminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes.
- *Deep knowledge.* Knowledge stored in a database that can only be found if we are given some direction about what we are looking for.

- *Emulate.* To act like.
- *Exemplar view.* The view that people store and recall likely concept exemplars that are used to classify unknown instances.
- *Expert.* A person skilled at solving difficult problems in a limited domain.
- *Expert system.* A computer program that emulates the behavior of a human expert.
- *Fact.* A simple statement of truth.
- *Hidden knowledge.* Patterns or regularities in data that cannot be easily found using database query. However, data mining algorithms can find such patterns with ease.
- *Hypothesis.* An educated guess about what we believe will be the outcome of an experiment. We can also state a null hypothesis, which is a hypothesis stated in a form to indicate no significant outcome. Experimentation is performed to either accept or reject the null hypothesis.
- *Induction-based learning.* The process of forming a general concept definition by observing specific examples of the concept to be learned versus deductive learning, where general rules are applied to specific situations.
- *Inference problem.* The ability to deduce private information from publicly available data.
- *Input attribute.* An attribute used by a data mining algorithm to help create a model of the data. An input attribute is sometimes referred to as an independent variable.
- *Instance.* An example or nonexample of a concept.
- *K-nearest neighbor classifier.* A variation of nearest neighbor classification where a new instance is classified with the most common class of its k nearest neighbors.
- *Knowledge discovery in databases (KDD).* The application of the scientific method to data mining.
- *Knowledge engineer.* A person trained to interact with an expert in order to capture his/her knowledge.
- *Manual data mining.* The process of posing database queries in an attempt to find hidden patterns in data.
- *Multidimensional knowledge.* Factual knowledge stored in a multidimensional format such as a data cube versus data stored in a two-dimensional structure such as a relational table.
- *Nearest neighbor classification.* An unknown instance is classified by searching the training data for the instance closest in distance to the unknown instance. A variation of this approach classifies a new instance with the most common class of its k nearest neighbors.

- *Output attribute.* For supervised learning, the attribute whose output is to be predicted.
- *Principle.* General laws or truths based on other simpler truths.
- *Probabilistic view.* The view that people store and recall concepts as generalizations created by observation.
- *Procedure.* A step-by-step course of action that is designed to achieve a specific goal.
- *Production rule.* A rule of the form: IF antecedent conditions THEN consequent conditions.
- *Rule coverage.* Given rule R , rule coverage is the percent of all instances that satisfy R 's preconditions.
- *Rule precision.* For rule R , the percent of instances covered by R 's antecedent condition(s) that are also covered by R 's consequent. Rule precision and rule accuracy are often assumed to be interchangeable terms.
- *Scatterplot diagram.* A two-dimensional graph plotting the relationship between two numeric attributes.
- *Shallow knowledge.* Factual knowledge stored in a database.
- *Supervised learning.* A classification model built using data instances of known origin. Once built, the model is able to determine the classification of instances of unknown origin.
- *Test set.* Data instances used to test supervised learning models.
- *Training data.* Data instances used to create supervised learning models.
- *Unsupervised clustering.* Classification models built using an evaluation function that measures the goodness of placing instances into the same class. Data instances are grouped together based on a similarity scheme defined by the clustering model.

EXERCISES

Review Questions

1. Differentiate between the following terms:
 - a. Data science and data mining
 - b. Training data and test data
 - c. Input attribute and output attribute
 - d. Shallow knowledge and hidden knowledge
 - e. Exemplar view and probabilistic view
 - f. Probabilistic view and classical view

- g. Supervised learning and unsupervised clustering
 - h. Intrinsic value and actual value
2. For each of the following problem scenarios, decide if a solution would best be addressed with supervised learning, unsupervised clustering, or database query. As appropriate, state any initial hypotheses you would like to test. If you decide that supervised learning or unsupervised clustering is the best answer, list two or more input attributes you believe to be relevant for solving the problem.
- a. What characteristics differentiate people who have had back surgery and have returned to work from those who have had back surgery and have not returned to their jobs?
 - b. A major automotive manufacturer initiates a tire recall for one of their top-selling vehicles. The automotive company blames the tires for the unusually high accident rate seen with their top seller. The company producing the tires claims the high accident rate only occurs when their tires are on the vehicle in question. Who is to blame?
 - c. An adjustable-bed company wants to develop a model to help determine proper settings for new customers purchasing one of their beds.
 - d. You desire to build a model for a fantasy football team to help determine which running back to play during for a specific week.
 - e. When customers visit my website, what products are they most likely to buy together?
 - f. What percent of my employees miss 1 or more days of work per month?
 - g. What relationships can I find between an individual's height, weight, age, and favorite spectator sport?
3. List five or more attributes likely to be relevant in determining if an individual income tax filing is fraudulent.
4. Visit the website: https://en.wikipedia.org/wiki/Examples_of_data_mining and summarize five applications where data mining has been used.
5. Medical doctors are experts at disease diagnosis and surgery. How do medical doctors use induction to help develop their skills?
6. Go to the website <http://www.kdnuggets.com>.
- a. Summarize one or two articles about how data mining has been applied to solve real problems.
 - b. Follow the *Datasets* link and scroll the UCI KDD Database Repository for interesting data sets. Describe the attributes of two data sets you find interesting.

7. Search the Web to find a reference that gives the top 10 data mining algorithms. Provide a list of the names of these algorithms as well as the reference.
8. Develop a concept definition for *a good student*.
 - a. What attributes would you use in your definition?
 - b. Give a definition of a good student from a classical point of view.
 - c. Define a good student from a probabilistic point of view.
 - d. State the defnition from an exemplar point of view.
9. What happens when you try to build a decision tree for the data in Table 1.1 without employing the attributes *swollen glands* and *fever*?

Data Mining

A Closer Look

CHAPTER OBJECTIVES

- *Determine an appropriate data mining strategy for a specific problem*
- *Understand the structure of the models created by several data mining techniques as well as the form of their output*
- *Know how a confusion matrix is used to help evaluate supervised models having categorical output*
- *Understand basic techniques for evaluating supervised learner models with numeric output*
- *Know how measuring lift can be used to compare the performance of several competing supervised models*
- *Understand basic techniques for evaluating an unsupervised clustering*

Although the field of data mining is in a continual state of change, a few basic strategies have remained constant. In Section 2.1, we define five fundamental data mining strategies and give examples of problems appropriate for each strategy. Whereas a data mining strategy outlines an approach for problem solution, a data mining technique applies a strategy. With the help of a hypothetical database containing customer information about credit card promotions, we introduce several common data mining techniques. We leave detailed explanations about the algorithms used by these techniques to later chapters. Here we focus on the models they create and the form of their output. Section 2.2 is dedicated to supervised learning techniques. In Section 2.3, we present a brief overview of association rules. In Section 2.4, we discuss unsupervised clustering. As you saw in Chapter 1, evaluation is a fundamental step in the data mining process. Section 2.5 provides a few basic tools to help you better understand the evaluation process.

2.1 DATA MINING STRATEGIES

As you learned in Chapter 1, *data mining strategies* can be broadly classified as either supervised or unsupervised. Supervised learning builds models by using input attributes to predict output attribute values. Many supervised data mining algorithms only permit a single output attribute. Other supervised learning tools allow us to specify one or several output attributes. Output attributes are also known as *dependent variables* as their outcome depends on the values of one or more input attributes. Input attributes are referred to as *independent variables*. When learning is unsupervised, an output attribute does not exist. Therefore, all attributes used for model building are independent variables.

Supervised learning strategies can be further labeled according to whether output attributes are discrete or categorical, as well as by whether models are designed to determine a current condition or predict a future outcome. In this section, we examine three supervised learning strategies, take a closer look at unsupervised clustering, and introduce a strategy for discovering associations among retail items sold in catalogs and stores. Figure 2.1 shows the five data mining strategies we will discuss.

2.1.1 Classification

Classification is probably the best understood of all data mining strategies. Classification tasks have three common characteristics:

- Learning is supervised.
- The dependent variable is categorical.
- The emphasis is on building models able to assign new instances to one of a set of well-defined classes.

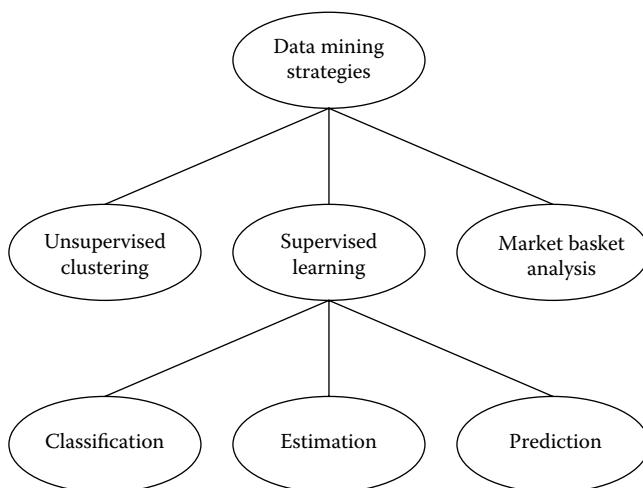


FIGURE 2.1 A hierarchy of data mining strategies.

Some example classification tasks include the following:

- Determine those characteristics that differentiate individuals who have suffered a heart attack from those who have not.
- Develop a profile of a “successful” person.
- Determine if a credit card purchase is fraudulent.
- Classify a car loan applicant as a good or a poor credit risk.
- Determine if an income tax submission has been falsified.

Notice that each example deals with current rather than future behavior. For example, we want the car loan application model to determine whether an applicant is a good credit risk at this time rather than in some future time period. Prediction models are designed to answer questions about future behavior. Prediction models are discussed later in this section.

2.1.2 Estimation

Like classification, the purpose of an *estimation* model is to determine a value for an unknown output attribute. However, unlike classification, the output attribute for an estimation problem is numeric rather than categorical. Here are four examples of estimation tasks:

- Estimate the number of minutes before a thunderstorm will reach a given location.
- Estimate the return on investment for an advertising campaign.
- Estimate the likelihood that a credit card has been stolen.
- Estimate the length of a gamma-ray burst.

Most supervised data mining techniques are able to solve classification or estimation problems, but not both. If our data mining tool supports one strategy but not the other, we can usually adapt a problem for solution by either strategy. To illustrate, suppose the output attribute for the original training data in the aforementioned example of the stolen credit card is a numeric value between 0 and 1, with 1 being a most likely case for a stolen card. We can make discrete categories for the output attribute values by replacing scores ranging between 0.0 and 0.3 with the value *unlikely*, scores between 0.3 and 0.7 with *likely*, and scores greater than 0.7 with *highly likely*. In this case, the transformation between numeric values and discrete categories is straightforward. Cases such as attempting to make monetary amounts discrete present more of a challenge.

THE CARDIOLOGY PATIENT DATA SET

The cardiology patient data set is part of both the RapidMiner and Weka data set packages. The original cardiology patient data was gathered by Dr. Robert Detrano at the VA Medical Center in Long Beach, California. The data set consists of 303 instances. Of these, 138 hold information about patients with heart disease. The original data set contains 13 numeric attributes and a fourteenth attribute indicating whether the patient has a heart condition. The data set was later modified by Dr. John Gennari. He changed seven of the numerical attributes to categorical equivalents for the purpose of testing data mining tools able to classify data sets with mixed data types. The files are named *CardiologyNumerical* and *CardiologyMixed*, respectively. This data set is interesting because it represents real patient data and has been used extensively for testing various data mining techniques. We can use these data together with one or more data mining techniques to help us develop profiles for differentiating individuals with heart disease from those without known heart conditions.

2.1.3 Prediction

It is not easy to differentiate prediction from classification or estimation. However, unlike a classification or estimation model, the purpose of a predictive model is to determine future outcome rather than current behavior. The output attribute(s) of a predictive model can be categorical or numeric. Here are several examples of tasks appropriate for predictive data mining:

- Predict the total number of touchdowns an NFL running back will score during the 2017 NFL season
- Determine whether a credit card customer is likely to take advantage of a special offer made available with his/her credit card billing
- Predict next week's closing price for the Dow Jones Industrial Average
- Forecast which cell phone subscribers are likely to change providers during the next three months

Most supervised data mining techniques appropriate for classification or estimation problems can also build predictive models. Actually, it is the nature of the data that determines whether a model is suitable for classification, estimation, or prediction. To show this, let's consider the *Cardiology Patient Dataset* described in the highlighted box above. The data set contains 303 instances. Of these, 165 hold information about patients who are free of heart disease. The remaining 138 instances contain data about patients who have a known heart condition.

The attributes and possible attribute values associated with this data set are shown in Table 2.1. Two forms of the data set exist. One data set consists of all numeric attributes. The second data set has categorical conversions for seven of the original numeric attributes. The table column labeled *mixed values* shows the value *numeric* for attributes that

TABLE 2.1 Cardiology Patient Data

Attribute Name	Mixed Values	Numeric Values	Comments
Age	Numeric	Numeric	Age in years
Gender	Male, Female	1, 0	Patient gender
Chest pain type	Angina, abnormal angina, NoTang, asymptomatic	1–4	NoTang = nonanginal pain
Blood pressure	Numeric	Numeric	Resting blood pressure upon hospital admission
Cholesterol	Numeric	Numeric	Serum cholesterol
Fasting blood sugar <120	True, false	1, 0	Is fasting blood sugar less than 120?
Resting electrocardiogram (ECG)	Normal, abnormal, Hyp	0, 1, 2	Hyp = left ventricular hypertrophy
Maximum heart rate	Numeric	Numeric	Maximum heart rate achieved
Induced angina?	True, false	1, 0	Does the patient experience angina as a result of exercise?
Old peak	Numeric	Numeric	ST depression induced by exercise relative to rest
Slope	Up, flat, down	1–3	Slope of the peak exercise ST segment
Number of colored vessels	0, 1, 2, 3	0, 1, 2, 3	Number of major vessels colored by fluoroscopy
Thalassemia	Normal, fix, rev	3, 6, 7	Normal, fixed defect, reversible defect
Concept class	Healthy, sick	1, 0	Angiographic disease status

were not converted to a categorical equivalent. For example, the values for attribute *age* are identical for both data sets. However, the attribute *fasting blood sugar <120* has values *true* or *false* in the converted data set and values *0* or *1* in the original data.

Table 2.2 lists four instances from the mixed form of the data set. Two of the instances represent typical examples from each respective class. The remaining two instances are atypical class members. Some differences between a typical healthy and a typical sick patient are easily anticipated, for example, *resting ECG* and *induced angina*. Surprisingly, we do not see expected differences in cholesterol and blood pressure readings between the typical and atypical healthy individuals.

Here are two rules generated from this data set by RapidMiner's *Subgroup Discovery* operator. *Class* is specified as the output attribute:

- IF *Maximum Heart Rate* ≥ 158.333

THEN *Class* = *Healthy*

Rule precision: 75.60%

Rule coverage: 40.60%

TABLE 2.2 Typical and Atypical Instances from the Cardiology Domain

Attribute Name	Typical Healthy Class	Atypical Healthy Class	Typical Sick Class	Atypical Sick Class
Age	52	63	60	62
Gender	Male	Male	Male	Female
Chest pain type	NoTang	Angina	Asymptomatic	Asymptomatic
Blood pressure	138	145	125	160
Cholesterol	223	233	258	164
Fasting blood sugar <120	False	True	False	False
Resting ECG	Normal	Hyp	Hyp	Hyp
Maximum heart rate	169	150	141	145
Induced angina?	False	False	True	False
Old peak	0	2.3	2.8	6.2
Slope	Up	Down	Flat	Down
Number of colored vessels	0	0	1	3
Thalassemia	Normal	Fix	Rev	Rev

- IF $Thal = Rev$

THEN $Class = Sick$

Rule precision: 76.30%

Rule coverage: 38.90%

Let's consider the first rule. Rule coverage gives us the percent of data instances that satisfy the rule's preconditions. Rule coverage reveals that of the 303 individuals represented in the data set, 40.6% or 123 patients have a maximum heart rate greater than 158.333. Rule precision tells us that of the 123 individuals with $maximum\ heart\ rate \geq 158.333$, 75.6% or 93 are healthy patients. That is, if a patient in the data set has a maximum heart rate greater than 158.333, we will be correct more than 75 times out of 100 in identifying the patient as healthy. When we combine this knowledge with the maximum heart rate values shown in Table 2.2, we are able to conclude that healthy patients are likely to have higher maximum heart rate values.

Is this first rule appropriate for classification or prediction? If the rule is predictive, we can use the rule to warn healthy folks with the following statement:

Warning 1: Have your maximum heart rate checked on a regular basis. If your maximum heart rate is low, you may be at risk of having a heart attack!

If the rule is appropriate for classification but not prediction, the scenario reads as follows:

Warning 2: If you have a heart attack, expect your maximum heart rate to decrease.

In any case, we cannot imply the following stronger statement:

Warning 3: A low maximum heart rate will cause you to have a heart attack!

That is, with data mining, we can state relationships between attributes, but we cannot say whether the relationships imply causality. Therefore, entertaining an exercise program to increase maximum heart rate may or may not be a good idea.

The question still remains as to whether either of the first two warnings is correct. This question is not easily answered. A data mining specialist can develop models to generate rules such as those just given. Beyond this, the specialist must have access to additional information—in this case, a medical expert—before determining how to use discovered knowledge.

2.1.4 Unsupervised Clustering

With unsupervised clustering, we are without a dependent variable to guide the learning process. Rather, the learning program builds a knowledge structure by using some measure of cluster quality to group instances into two or more classes. A primary goal of an unsupervised clustering strategy is to discover concept structures in data. Common uses of unsupervised clustering include the following:

- Detect fraud in stock trading activity, insurance claims, or financial transactions
- Determine if meaningful relationships in the form of concepts can be found in data representing gamma-ray burst flashes outside of our solar system
- Determine if publicly available data about banks that have failed is useful for building a supervised model to predict bank failure
- Determine a best set of input attributes for building a supervised learner model to identify cell phone customers likely to churn

You saw an obvious use of unsupervised clustering in Chapter 1 when we showed how clustering was applied to the Acme Investors database to find interesting relationships in the form of concept classes in the data. However, it is not unusual to use unsupervised clustering as an evaluation tool for supervised learning.

To illustrate this idea, let's suppose we have built a supervised learner model using the heart patient data with output attribute *class*. To evaluate the supervised model, we present the training instances to an unsupervised clustering system. The attribute *class* is flagged as unused. Next, we examine the output of the unsupervised model to determine if the instances from each concept class (*healthy* and *sick*) naturally cluster together. If the instances from the individual classes do not cluster together, we may conclude that the attributes are unable to distinguish healthy patients from those with a heart condition. This being the case, the supervised model is likely to perform poorly. One solution is to choose

a best set of attributes for a supervised learner model by repeatedly applying unsupervised clustering with alternative attribute choices. In this way, those attributes best able to differentiate the classes known to be present in the data can be determined. Unfortunately, even with a small number of attribute choices, the application of this technique can be computationally unmanageable.

Unsupervised clustering can also help detect any atypical instances present in the data, that is, instances that do not group naturally with the other instances. Atypical instances are referred to as *outliers*. Outliers can be of great importance and should be identified whenever possible. With data mining, the outliers might be just those instances we are trying to identify. For example, an application that checks credit card purchases would likely identify an outlier as a positive instance of credit card fraud.

2.1.5 Market Basket Analysis

The purpose of *market basket analysis* is to find interesting relationships among retail products. The results of a market basket analysis help retailers design promotions, arrange shelf or catalog items, and develop cross-marketing strategies. Association rule algorithms, described later in this chapter, are often used to apply a market basket analysis to a set of data. Association rules are discussed in detail in Chapter 3.

THE CREDIT CARD PROMOTION DATABASE

Credit card companies often include promotional offerings with their monthly credit card billings. The offers provide the credit card customer with an opportunity to purchase items such as luggage, magazines, or jewelry. Credit card companies sponsoring new promotions frequently send bills to those individuals that have no current card balance hoping that some of these individuals will take advantage of one or more of the promotional offerings. From the perspective of predictive data mining, with the right data, we may be able to find relationships that provide insight about the characteristics of those individuals likely to take advantage of future promotions. In doing so, we can divide the pool of zero-balance cardholders into two classes, where one class will be those persons likely to take advantage of a new credit card promotion. These individuals should be sent a zero-balance billing containing the promotional information. The second class will consist of persons not likely to make a promotional purchase. These individuals should not be sent a zero-balance monthly statement. The end result is savings in the form of decreased postage, paper, and processing costs for the credit card company.

The credit card promotion database shown in Table 2.3 has fictitious data about 15 individuals holding credit cards with the Acme Credit Card Company. The data contain information obtained about customers through their initial credit card application as well as data about whether these individuals have accepted various past promotional offerings sponsored by the credit card company. The data set is titled *CreditCardPromotion* and is given in both Weka and Microsoft (MS) Excel format. Although the data set is small, it serves well for purposes of illustration. We employ this data set for descriptive purposes throughout the text.

2.2 SUPERVISED DATA MINING TECHNIQUES

Now that we have described five common data mining strategies, we turn our attention to the data mining techniques used to apply these strategies to a set of data. A specific *data mining technique* is defined by an algorithm and an associated knowledge structure such as a tree or a set of rules. In Chapter 1, we introduced decision trees as the most studied of all supervised data mining techniques. You will learn how decision trees are constructed in Chapter 3. Furthermore, Chapters 4 and 5 show you how to use Weka and RapidMiner to build and apply decision tree models. Here we present several additional supervised data mining methods. Our goal is to help you develop a basic understanding of the similarities and differences between the various data mining techniques.

2.2.1 The Credit Card Promotion Database

We will use the fictitious data displayed in Table 2.3 and summarized in the box titled *The Credit Card Promotion Database* to help explain the data mining strategies presented here. The table shows data extracted from a database containing information collected on individuals who hold credit cards issued by the Acme Credit Card Company. The first row of Table 2.3 contains the attribute names for each column of data. The first column is for instance identification. The second column gives the salary range for an individual credit card holder. Values in columns 3–5 tell us which cardholders have taken advantage of specified promotions sent with their monthly credit card bill. Column 6 tells us whether an individual has credit card insurance. Column 7 gives the gender of the cardholder, and column 8 offers the cardholder's age. The first cardholder shown in the table has a yearly salary between \$40,000 and \$50,000, is a 45-year-old male, has purchased one or several magazines advertised with one of his credit card bills, did not take advantage of any other credit card promotions, and does not have credit card insurance. Several attributes likely to

TABLE 2.3 Credit Card Promotion Database

ID	Income Range (\$)	Magazine Promotion	Watch Promotion	Life Insurance Promotion	Credit Card Insurance	Gender	Age
1	40–50K	Yes	No	No	No	Male	45
2	30–40K	Yes	Yes	Yes	No	Female	40
3	40–50K	No	No	No	No	Male	42
4	30–40K	Yes	Yes	Yes	Yes	Male	43
5	50–60K	Yes	No	Yes	No	Female	38
6	20–30K	No	No	No	No	Female	55
7	30–40K	Yes	No	Yes	Yes	Male	35
8	20–30K	No	Yes	No	No	Male	27
9	30–40K	Yes	No	No	No	Male	43
10	30–40K	Yes	Yes	Yes	No	Female	41
11	40–50K	No	Yes	Yes	No	Female	43
12	20–30K	No	Yes	Yes	No	Male	29
13	50–60K	Yes	Yes	Yes	No	Female	39
14	40–50K	No	Yes	No	No	Male	55
15	20–30K	No	No	Yes	Yes	Female	19

be relevant for data mining purposes are not included in the table. Some of these attributes are promotion dates, dollar amounts for purchases, average monthly credit card balance, and marital status. Let's turn our attention to the data mining techniques to see what they can find in the credit card promotion database.

2.2.2 Rule-Based Techniques

In Chapter 1, you saw that any decision tree can be translated into a set of production rules. However, there are several other rule generation techniques. One class of rule generators uses what is referred to as a *covering* approach. Instances are covered by a rule if they satisfy the rule's preconditions. For each class, the goal is to create a set of rules that maximizes the total number of covered within-class instances while at the same time minimizing the number of covered nonclass instances.

Another class of rule generators focuses on finding interesting subgroups of instances within the data. RapidMiner's *Subgroup Discovery* operator falls into this category. Earlier, we showed you two rules generated by this operator for the heart patient data set. Here we take a closer look at the subgroup discovery method and concentrate on the covering rule approach in Chapter 3.

Let's assume the Acme Credit Card Company has authorized a new life insurance promotion similar to the previous life insurance promotion specified in Table 2.3. The promotion material will be sent as part of the credit card billing for all cardholders with a nonzero balance. We will use data mining to help us send billings to a select group of individuals who do not have a current credit card balance but are likely to take advantage of the promotion.

Our problem calls for a supervised approach using *life insurance promotion* as the output attribute. Our goal is to develop a profile for individuals likely to take advantage of a life insurance promotion advertised along with their next credit card statement. Here is a possible hypothesis:

A combination of one or more of the data set attributes differentiates between Acme Credit Card Company cardholders who have taken advantage of a life insurance promotion and those cardholders who have chosen not to participate in the promotional offer.

The hypothesis is stated in terms of current rather than predicted behavior. However, the nature of the created rules will tell us whether we can use the rules for classification or prediction.

For our experiment, we presented RapidMiner's *Subgroup Discovery* operator with the data in Table 2.3. We obtained several rules of interest. Here are four such rules:

1. IF *Gender = Female*

THEN *Life Insurance Promotion = Yes*

Rule precision: 85.7%

Rule coverage: 46.7%

2. IF *Gender = Male* and *Income Range = 40–50K*

THEN *Life Insurance Promotion = No*

Rule precision: 100.00%

Rule coverage: 20.00%

3. IF *Credit Card Insurance = Yes*

THEN *Life Insurance Promotion = Yes*

Rule precision: 100.00%

Rule coverage: 20%

4. IF *Credit Card Insurance = No and Watch Promotion = No*

THEN *Life Insurance Promotion = No*

Rule precision: 80.00%

Rule coverage: 33.33%

The first rule tells us that it would be advantageous to send a credit card bill containing the promotion to all female cardholders. The second rule indicates that males who make between \$40,000 and \$50,000 a year are not good candidates for the promotion. The 100.00% precision value tells us that our sample does not contain a single male within the \$40,000–\$50,000 income range who took advantage of the previous life insurance promotion.

The first and second rules are particularly helpful, as neither rule contains an antecedent condition involving a previous promotion. The rule preconditions are based purely on information obtained at the time of initial application. As credit card insurance is always initially offered upon a card approval, the third rule is also useful. However, the fourth rule will not be applicable to new cardholders who have not had a chance to take advantage of a previous promotion. For new cardholders, we should consider the first three rules as predictive and the fourth rule as effective for classification but not predictive purposes. Furthermore, it is important to point out that the precision values shown with each rule are based solely on the training data. The true test of whether these rules are useful comes when they are applied to new data.

Before leaving our discussion about rule-based techniques, two added points are in order. First, we have yet to address the issue of how rules are generated for numeric data. The basic technique is to discretize numeric data into a specified number of equal-sized bins where each bin represents one categorical value for the given attribute. Several rule generators have one or several built-in numeric discretization routines. The subgroup operator discussed here requires discretization to take place during preprocessing.

Secondly, each of the aforementioned rules displays a value for precision and coverage. Where rules are concerned, rule precision and rule accuracy are often considered as

interchangeable terms. That is, both terms are used to represent the ratio of the total number of correctly classified instances to the total number of instances covered by the rule antecedent. However, RapidMiner's subgroup operator makes a notable distinction between rule accuracy and rule precision. Specifically, whereas rule precision is only concerned with rule performance relative to the instances covered by the rule, rule accuracy is measured using all instances within the data set.

To illustrate this, consider a data set having N instances. Suppose class C_1 has rule R covering P of its T_1 instances ($P \leq T_1$). We also have class C_2 with $T_2 = (N - T_1)$ instances. Suppose R's antecedent condition covers Q of the instances within C_2 . The computation for precision is then given as $(P/(P + Q))$. That is, the precision of rule R is the total number of C_1 instances covered by R divided by the total number of data set instances covered by R's antecedent condition. However, in contrast, rule accuracy is measured by the value of $(P + T_2 - Q)/N$. That is, accuracy is defined as the sum of the number of C_1 instances covered by R and the number of C_2 instances whose antecedent condition is not covered by R, divided by the total number of instances. The rationale is that a rule is accurate relative to an instance of the competing class if the instance does not satisfy the rule's preconditions.

As an example, consider rule 1 stating that female instances accepted the life insurance promotion. Upon examining Table 2.3, we see that six of the seven females in the data set accepted the life insurance promotion. Therefore, rule precision is $6/7$, which approximates to 85.7%. To compute rule accuracy, we add the total number of instances from the competing class (male instances) who did not accept the life insurance promotion to the numerator and divide by 15. As instance numbers 1, 3, 8, 9, and 14 represent males not accepting the life insurance promotion, our computation becomes $(6 + 5)/15$, giving us a rule accuracy of 73.3%. We will return to this issue in Chapter 5 when we discuss rule generation with RapidMiner.

2.2.3 Neural Networks

A *neural network* is a set of interconnected nodes designed to imitate the functioning of the human brain. As the human brain contains billions of neurons and a typical neural network has fewer than 100 nodes, the comparison is somewhat superficial. However, neural networks have been successfully applied to problems across several disciplines and for this reason are quite popular in the data mining community.

Neural networks come in many shapes and forms and can be constructed for supervised learning as well as unsupervised clustering. In all cases, the values input into a neural network must be numeric. The feed-forward network is a popular supervised learner model. Figure 2.2 shows a fully connected feed-forward neural network consisting of three layers. With a feed-forward network, the input attribute values for an individual instance enter at the input layer and pass directly through the output layer of the network structure. The output layer may contain one or several nodes. The output layer of the network shown in Figure 2.2 contains two nodes. Therefore, the output of the neural network will be an ordered pair of values.

The network displayed in Figure 2.2 is fully connected, as all the nodes at one layer are connected to all nodes at the next layer. In addition, each network node connection has an

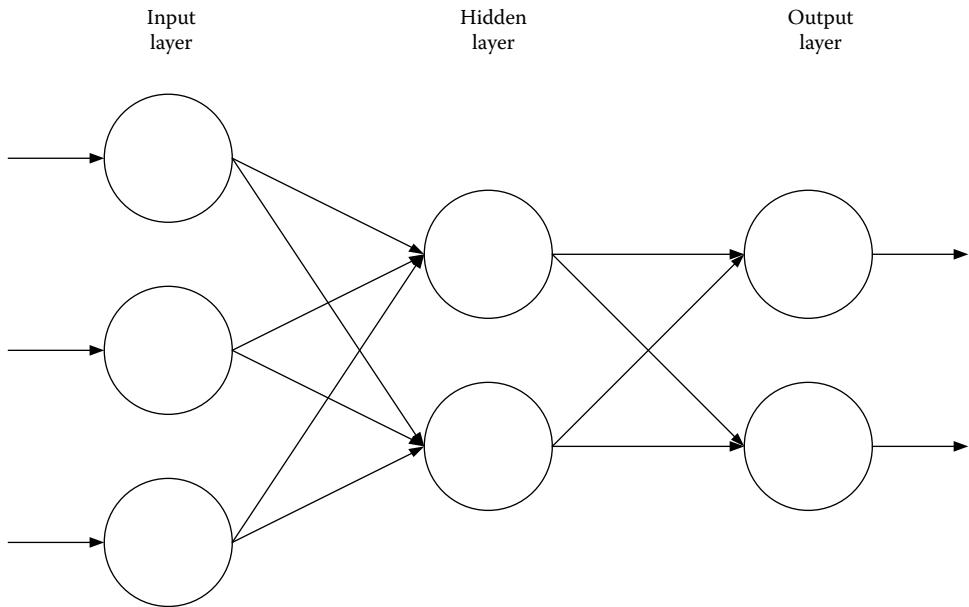


FIGURE 2.2 A fully connected multilayer neural network.

associated weight (not shown in the diagram). Notice that nodes within the same layer of the network architecture are not connected to one another.

Neural networks operate in two phases. The first phase is called the learning phase. During network learning, the input values associated with each instance enter the network at the input layer. One input-layer node exists for each input attribute contained in the data. The actual output value for each instance is computed and compared with the desired network output. Any error between the desired and computed output is propagated back through the network by changing connection-weight values. Training terminates after a certain number of iterations or when the network converges to a predetermined minimum error rate. During the second phase of operation, the network weights are fixed, and the network is used to classify new instances.

We applied a supervised network model to the credit card promotion data to test the aforementioned hypothesis. Once again, *life insurance promotion* was designated as the output attribute. Because we wanted to construct a predictive model, the input attributes were limited to *income range*, *credit card insurance*, *gender*, and *age*. Therefore, the network architecture contained four input nodes and one output node. For our experiment, we chose five hidden-layer nodes. Because neural networks cannot accept categorical data, we transformed categorical attribute values by replacing *yes* and *no* with 1 and 0 respectively, *male* and *female* with 1 and 0, and income range values with the lower end of each range score.

Computed and actual values for the output attribute *life insurance promotion* are shown in Table 2.4. Notice that in most cases, a computed output value is within .03 of the actual value. To use the trained network to classify a new unknown instance, the attribute values for the unknown instance are passed through the network, and an output score is obtained.

TABLE 2.4 Neural Network Training: Actual and Computed Output

Instance Number	Life Insurance Promotion	Computed Output
1	0	0.024
2	1	0.998
3	0	0.023
4	1	0.986
5	1	0.999
6	0	0.050
7	1	0.999
8	0	0.262
9	0	0.060
10	1	0.997
11	1	0.999
12	1	0.776
13	1	0.999
14	0	0.023
15	1	0.999

If the computed output value is closer to 0, we predict the instance to be an unlikely candidate for the life insurance promotion. A value closer to 1 shows the unknown instance as a good candidate for accepting the life insurance promotion.

A major shortcoming of the neural network approach is a lack of explanation about what has been learned. Converting categorical data to numerical values can also be a challenge. Chapter 8 details two common neural network learning techniques. In Chapters 9 and 10, you will learn how to use Weka's and RapidMiner's neural network software packages.

2.2.4 Statistical Regression

Statistical regression is a supervised learning technique that generalizes a set of numeric data by creating a mathematical equation relating one or more input attributes to a single numeric output attribute. A *linear regression* model is a type of statistical regression characterized by an output attribute whose value is determined by a linear sum of weighted input attribute values. Here is a linear regression equation for the data in Table 2.3:

$$\text{Life insurance promotion} = 0.5909 \times (\text{credit card insurance}) - 0.5455 \times (\text{gender}) + 0.7727$$

Notice that *life insurance promotion* is the attribute whose value is to be determined by a linear combination of attributes *credit card insurance* and *gender*. As with the neural network model, we transformed all categorical data by replacing *yes* and *no* with 1 and 0, *male* and *female* with 1 and 0, and income range values with the lower end of each range score.

To illustrate the use of the equation, suppose we wish to determine if a female who does not have credit card insurance is a likely candidate for the life insurance promotion. Using the equation, we have

$$\begin{aligned}\text{Life insurance promotion} &= 0.5909(0) - 0.5455(0) + 0.7727 \\ &= 0.7727\end{aligned}$$

Because the value 0.7727 is close to 1.0, we conclude that the individual is likely to take advantage of the promotional offer.

Although regression can be nonlinear, the most popular use of regression is for linear modeling. Linear regression is appropriate provided that the data can be accurately modeled with a straight line function. In Chapter 11, we experiment with Weka's and RapidMiner's linear regression models.

2.3 ASSOCIATION RULES

As the name implies, *association rule* mining techniques are used to discover interesting associations between attributes contained in a database. Unlike traditional production rules, association rules can have one or several output attributes. Also, an output attribute for one rule can be an input attribute for another rule. Association rules are a popular technique for market basket analysis because all possible combinations of potentially interesting product groupings can be explored. For this reason, a limited number of attributes are able to generate hundreds of association rules.

We applied the Apriori association rule algorithm described by Agrawal et al. (1993) to the data in Table 2.3. The algorithm examines baskets of items and generates rules for those baskets containing a minimum number of items. The Apriori algorithm does not process numerical data. Therefore, before application of the algorithm, we transformed the attribute *age* to the following set of discrete categories: *over15*, *over20*, *over30*, *over40*, and *over50*. To illustrate, an individual with *age* = *over40* is between the ages of 40 and 49 inclusive. Once again, we limited the choice of attributes to *income range*, *credit card insurance*, *sex*, and *age*. Here is a list of three association rules generated by the Apriori algorithm for the data in Table 2.3.

1. IF *Gender* = *Female* and *Age* = *over40* and *Credit Card Insurance* = *No*

THEN *Life Insurance Promotion* = *Yes*

2. IF *Gender* = *Male* and *Age* = *over40* and *Credit Card Insurance* = *No*

THEN *Life Insurance Promotion* = *No*

3. IF *Gender* = *Female* and *Age* = *over40*

THEN *Credit Card Insurance* = *No* and *Life Insurance Promotion* = *Yes*

Each of these three rules has an accuracy of 100% and covers exactly 20% of all data instances. For rule 3, the 20% rule coverage tells us that one in every five individuals is a female over the age of 40 who does not have credit card insurance and has life insurance obtained through the life insurance promotional offer. Notice that in rule 3, *credit card insurance* and *life insurance promotion* are both output attributes.

A problem with association rules is that along with potentially interesting rules, we are likely to see several rules of little value. In Chapter 3, we will explore this issue in more detail and describe the Apriori algorithm for generating association rules. Chapter 4 and Chapter 5 offer tutorials on using the association rule generators available with Weka and RapidMiner. The next section continues our discussion by exploring unsupervised clustering techniques.

2.4 CLUSTERING TECHNIQUES

Several unsupervised clustering techniques are commonly used. One technique is to apply some measure of similarity to divide instances into disjoint partitions. The partitions are generalized by computing a group mean for each cluster or by listing a most typical subset of instances from each cluster. In Chapter 3, we will examine a well-known unsupervised algorithm that uses group means to partition data. A second approach is to partition data in a hierarchical fashion where each level of the hierarchy is a generalization of the data at some level of abstraction. You will learn about two hierarchical clustering techniques in Chapter 12.

We applied a hierarchical clustering model to the data in Table 2.3. Our choice for input attributes was again limited to *income range*, *credit card insurance*, *gender*, and *age*. We set the *life insurance promotion* as a label attribute, meaning that although the attribute is not used by the clustering system, it will appear as part of the summary statistics. As learning is unsupervised, our hypothesis needs to change. Here is a possible hypothesis that is consistent with our theme of determining likely candidates for the life insurance promotion:

By applying unsupervised clustering to the instances of the Acme Credit Card Company database, we will find a subset of input attributes that differentiate cardholders who have taken advantage of the life insurance promotion from those cardholders who have not accepted the promotional offer.

As you can see, we are using unsupervised clustering to find a best set of input attributes for differentiating current customers who have taken advantage of the special promotion from those who have not. Once we determine a best set of input attributes, we can use the attributes to develop a supervised model for predicting future outcomes.

To test the hypothesis, we applied unsupervised clustering to the data several times until we found a set of input attributes that resulted in clusters that differentiate the two classes. The results of one such clustering are displayed in Figure 2.3. The figure indicates that three clusters were formed. As you can see, the three individuals represented in cluster 1 did not take advantage of the life insurance promotion. Two of the individuals in cluster 2 took

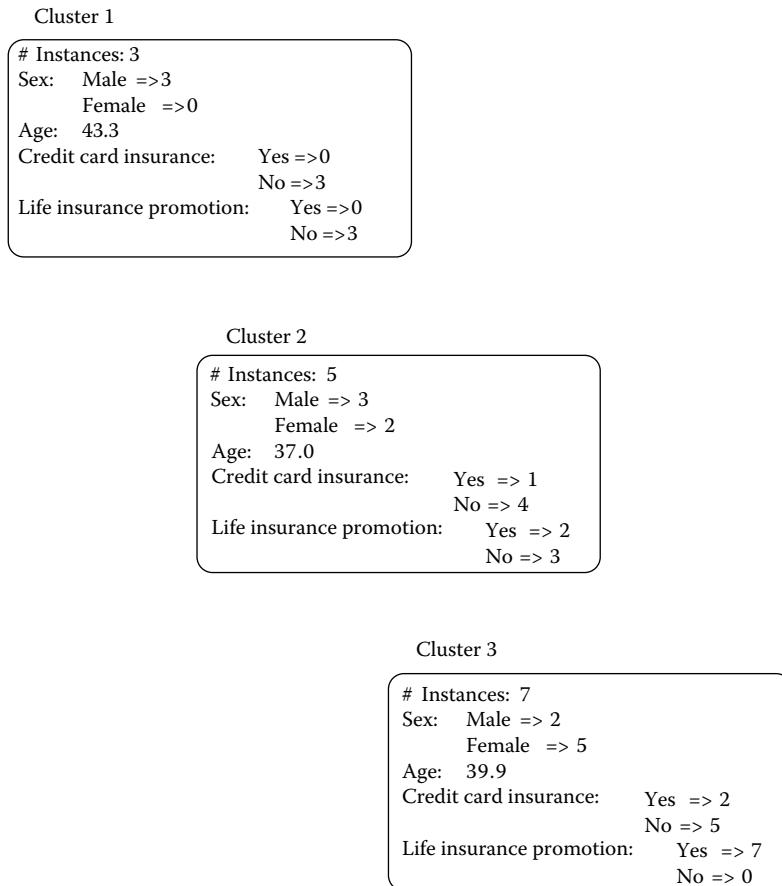


FIGURE 2.3 An unsupervised clustering of the credit card database.

advantage of the promotion, and three did not. Finally, all seven individuals in cluster 3 purchased the life insurance promotion.

It is clear that two of the three clusters differentiate individuals who took advantage of the promotion from those who did not. This result offers positive evidence that the attributes used for the clustering are viable choices for building a predictive supervised learner model. In the next section, we lay the foundation for evaluating the performance of supervised and unsupervised learner models.

2.5 EVALUATING PERFORMANCE

Performance evaluation is probably the most critical of all the steps in the data mining process. In this section, we offer a commonsense approach to evaluating supervised and unsupervised learner models. In later chapters, we will concentrate on more formal evaluation techniques. As a starting point, we pose three general questions:

1. Will the benefits received from a data mining project more than offset the cost of the data mining process?

2. How do we interpret the results of a data mining session?
3. Can we use the results of a data mining process with confidence?

All three questions are difficult to answer. However, the first is more of a challenge because several factors come into play. Here is a minimal list of considerations for the first question:

1. Is there knowledge about previous projects similar to the proposed project? What are the success rates and costs of projects similar to the planned project?
2. What is the current form of the data to be analyzed? Do the data exist, or will they have to be collected? When a wealth of data exist and are not in a form amenable for data mining, the greatest project cost will fall under the category of data preparation. In fact, a larger question may be whether to develop a data warehouse for future data mining projects.
3. Who will be responsible for the data mining project? How many current employees will be involved? Will outside consultants be hired?
4. Is the necessary software currently available? If not, will the software be purchased or developed? If purchased or developed, how will the software be integrated into the current system?

As you can see, any answer to the first question requires knowledge about the business model, the current state of available data, and current resources. Therefore, we will turn our attention to providing evaluation tools for questions 2 and 3. We first consider the evaluation of supervised learner models.

2.5.1 Evaluating Supervised Learner Models

Supervised learner models are designed to classify, estimate, and/or predict future outcome. For some applications, the desire is to build models showing consistently high predictive accuracy. The following three applications focus on classification correctness:

- Develop a model to accept or reject credit card applicants
- Develop a model to accept or reject home mortgage applicants
- Develop a model to decide whether or not to drill for oil

Classification correctness is best calculated by presenting previously unseen data in the form of a test set to the model being evaluated. Test set model accuracy can be summarized in a table known as a *confusion matrix*. To illustrate, let's suppose we have three possible classes: C_1 , C_2 , and C_3 . A generic confusion matrix for the three-class case is shown in Table 2.5.

Values along the main diagonal give the total number of correct classifications for each class. For example, a value of 15 for C_{11} means that 15 class C_1 test set instances were

TABLE 2.5 A Three-Class Confusion Matrix

		Computed Decision		
		C ₁	C ₂	C ₃
C _i	C ₁₁	C ₁₂	C ₁₃	
	C ₂₁	C ₂₂	C ₂₃	
	C ₃₁	C ₃₂	C ₃₃	

correctly classified. Values other than those on the main diagonal represent classification errors. To illustrate, suppose C₁₂ has the value 4. This means that four class C₁ instances were incorrectly classified as belonging to class C₂. The following four observations may be helpful in analyzing the information in a confusion matrix:

1. For any given cell C_{ij}, the subscript *i* represents the actual class, and *j* indicates the computed class.
2. Values along the main diagonal represent correct classifications. For the matrix in Table 2.5, the value C₁₁ represents the total number of class C₁ instances correctly classified by the model. A similar statement can be made for the values C₂₂ and C₃₃.
3. Values in row C_i represent those instances that belong to class C_i. For example, with i = 2, the instances associated with cells C₂₁, C₂₂, and C₂₃ are all actually members of C₂. To find the total number of C₂ instances incorrectly classified as members of another class, we compute the sum of C₂₁ and C₂₃.
4. Values found in column C_i indicate those instances that have been classified as members of C_i. With i = 2, the instances associated with cells C₁₂, C₂₂, and C₃₂ have been classified as members of class C₂. To find the total number of instances incorrectly classified as members of class C₂, we compute the sum of C₁₂ and C₃₂.

We can use the summary data displayed in the confusion matrix to compute model accuracy. To determine the accuracy of a model, we sum the values found on the main diagonal and divide the sum by the total number of test set instances. For example, if we apply a model to a test set of 100 instances and the values along the main diagonal of the resultant confusion matrix sum to 70, the test set accuracy of the model is 0.70 or 70%. As model accuracy is often given as an error rate, we can compute model error rate by subtracting the model accuracy value from 1.0. For our example, the corresponding error rate is 0.30.

If ample test data are not available, we can apply a technique known as *cross-validation*. With this method, all available data are partitioned into *n* fixed-size units. *n* – 1 of the units are used for training, whereas the *n*th unit is the test set. This process is repeated until each of the fixed-size units has been used as test data. Model test set correctness is computed as the average accuracy realized from the *n* training/testing trials. Experimental results have shown a value of 10 for *n* to be maximal in most situations. Several applications

of cross-validation to the data can help ensure an equal distribution of classes within the training and test data sets.

Bootstrapping is an alternative to cross-validation. With bootstrapping, we allow the training set selection process to choose the same training instance several times. This happens by placing each training instance back into the data pool after it has been selected for training. It can be shown mathematically that if a data set containing n instances is sampled n times using the bootstrap technique, the training set will contain approximately two-thirds of the n instances. This leaves one-third of the instances for testing.

2.5.2 Two-Class Error Analysis

The three applications listed at the beginning of this section represent two-class problems. For example, a credit card application is either accepted or rejected. We can use a simple two-class confusion matrix to help us analyze each of these applications.

Consider the confusion matrix displayed in Table 2.6. Cells showing *True accept* and *True reject* represent correctly classified test set instances. For the first and second applications presented in the previous section, the cell with *False accept* denotes accepted applicants that should have been rejected. The cell with *False reject* designates rejected applicants that should have been accepted. A similar analogy can be made for the third application. Let's use the confusion matrices shown in Table 2.7 to examine the first application in more detail.

Assume the confusion matrices shown in Table 2.7 represent the test set error rates of two supervised learner models built for the credit card application problem. The confusion matrices show that each model displays an error rate of 10%. As the error rates are identical, which model is better? To answer the question, we must compare the average cost of credit card payment default to the average potential loss in profit realized by rejecting individuals who are good approval candidates. Given that credit card purchases are unsecured, the cost of accepting credit card customers likely to default is more of a concern. In this case, we should choose model B because the confusion matrices tell us that this model is less likely to erroneously offer a credit card to an individual likely to default. Does the same reasoning apply for the home mortgage application? How about the application where the

TABLE 2.6 A Simple Confusion Matrix

	Computed Accept	Computed Reject
Accept	True accept	False reject
Reject	False accept	True reject

TABLE 2.7 Two Confusion Matrices Each Showing a 10% Error Rate

Model A	Computed Accept	Computed Reject	Model B	Computed Accept	Computed Reject
	600	25		Accept	600
Reject	75	300	Reject	25	300

question is whether to drill for oil? As you can see, although test set error rate is a useful measure for model evaluation, other factors such as costs incurred for false inclusion as well as losses resulting from false omission must be considered.

2.5.3 Evaluating Numeric Output

A confusion matrix is of little use for evaluating supervised learner models offering numeric output. In addition, the concept of classification correctness takes on a new meaning with numeric output models because instances cannot be directly categorized into one of several possible output classes. However, several useful measures of model accuracy have been defined for supervised models having numeric output. The most common numeric accuracy measures are mean absolute error and mean squared error.

The *mean absolute error* (mae) for a set of test data is computed by finding the average absolute difference between computed and desired outcome values. In a similar manner, the *mean squared error* is simply the average squared difference between computed and desired outcome. Finally, the *root mean squared error* (rms) is simply the square root of a mean squared error value. Rms is frequently used as a measure of test set accuracy with feed-forward neural networks. It is obvious that for a best test set accuracy, we wish to obtain the smallest possible value for each measure. You will learn more about the pros and cons of each of these numeric measures of accuracy in Chapter 7.

2.5.4 Comparing Models by Measuring Lift

Marketing applications that focus on response rates from mass mailings are less concerned with test set classification error and more interested in building models able to extract bias samples from large populations. Bias samples show higher response rates than the rates seen within the general population. Supervised learner models designed for extracting bias samples from a general population are often evaluated by a measure that comes directly from marketing known as *lift*. An example illustrates the idea.

Let's consider an expanded version of the credit card promotion database. Suppose the Acme Credit Card Company is about to launch a new promotional offer with next month's credit card statement. The company has determined that for a typical month, approximately 100,000 credit card holders show a zero balance on their credit card. The company has also determined that an average of 1% of all cardholders take advantage of promotional offers included with their card billings. Based on this information, approximately 1000 of the 100,000 zero-balance cardholders are likely to accept the new promotional offer. As zero-balance cardholders do not require a monthly billing statement, the problem is to send a zero-balance billing to exactly those customers who will accept the new promotion.

We can employ the concept of lift to help us choose a best solution. Lift measures the change in percent concentration of a desired class, C_i , taken from a biased sample relative to the concentration of C_i within the entire population. We can formulate lift using conditional probabilities. Specifically,

$$\text{lift} = \frac{P(C_i|\text{sample})}{P(C_i|\text{population})}$$

where $P(C_i|sample)$ is the portion of instances contained in class C_i relative to the biased sample population and $P(C_i|population)$ is the fraction of class C_i instances relative to the entire population. For our problem, C_i is the class of all zero-balance customers who, given the opportunity, will take advantage of the promotional offer.

Figure 2.4 offers a graphical representation of the credit card promotion problem. The graph is sometimes called a *lift chart*. The horizontal axis shows the percent of the total population sampled, and the vertical axis represents the number of likely respondents. The graph displays model performance as a function of sample size. The straight line represents the general population. This line tells us that if we randomly select 20% of the population for the mailing, we can expect a response from 200 of the 1000 likely respondents. Likewise, selecting 100% of the population will give us all respondents. The curved line shows the lift achieved by employing models of varying sample sizes. By examining the graph, you can see that an ideal model will show the greatest lift with the smallest sample size. This is represented in Figure 2.4 as the upper-left portion of the graph. Although Figure 2.4 is useful, the confusion matrix also offers us an explanation about how lift can be incorporated to solve problems.

Table 2.8 shows two confusion matrices to help us understand the credit card promotion problem from the perspective of lift. The confusion matrix showing *no model* tells us that all zero-balance customers are sent a billing statement with the promotional offer. By definition, the lift for this scenario is 1.0 because the sample and the population are identical.

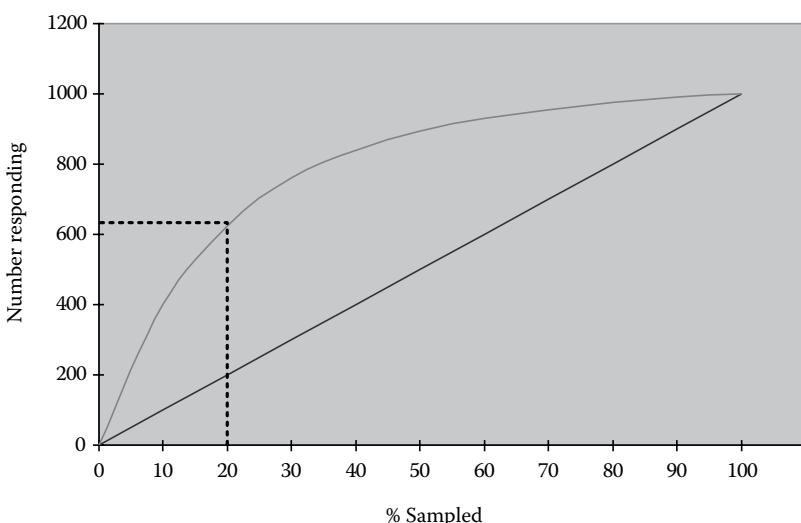


FIGURE 2.4 Targeted versus mass mailing.

TABLE 2.8 Two Confusion Matrices: No Model and an Ideal Model

No Model	Computed Accept	Computed Reject	Ideal Model	Computed Accept	Computed Reject
Accept	1000	0	Accept	1000	0
Reject	99,000	0	Reject	0	99,000

TABLE 2.9 Two Confusion Matrices for Alternative Models with Lift Equal to 2.25

No Model	Computed Accept	Computed Reject	Ideal Model	Computed Accept	Computed Reject
Accept	540	460	Accept	450	550
Reject	23,460	75,540	Reject	19,550	79,450

The lift for the matrix showing *ideal model* is 100 (100%/1%) because the biased sample contains only positive instances.

Consider the confusion matrices for the two models shown in Table 2.9. The lift for model X is computed as

$$\text{lift(model X)} = \frac{540/24,000}{1000/100,000}$$

which evaluates to 2.25. The lift for model Y is computed as

$$\text{lift(model Y)} = \frac{450/20,000}{1000/100,000}$$

which also evaluates to 2.25. As was the case with the previous example, to answer the question about which is a better model, we must have additional information about the relative costs of false-negative and false-positive selections. For our example, model Y is a better choice if the cost savings in mailing fees (4000 fewer mailings) more than offsets the loss in profits incurred from fewer sales (90 fewer sales).

2.5.5 Unsupervised Model Evaluation

Evaluating unsupervised data mining is, in general, a more difficult task than supervised evaluation. This is true because the goals of an unsupervised data mining session are frequently not as clear as the goals for supervised learning. Here we will introduce a general technique that employs supervised learning to evaluate an unsupervised clustering and leave a more detailed discussion of unsupervised evaluation for later chapters.

All unsupervised clustering techniques compute some measure of cluster quality. A common technique is to calculate the summation of squared error differences between the instances of each cluster and their corresponding cluster center. Smaller values for sums of squared error differences indicate clusters of higher quality. A second approach compares the ratio of within-cluster to between-cluster similarity values. For cluster C, within-cluster similarity is measured by computing the similarity of each instance within C to all other instances of C. The similarity scores are averaged to give the within-cluster similarity score for C. Between-cluster similarity for C is determined by computing the similarity of each instance of C to all other instances not in C. These similarity scores are averaged to give the between-cluster similarity score for class C. Larger values for the within-to-between cluster ratio indicates that the instances of C form a meaningful cluster. However,

for a more meaningful evaluation of unsupervised clustering, it is supervised learning that comes to the rescue. The technique is as follows:

1. Perform an unsupervised clustering. Designate each cluster as a class and assign each cluster an arbitrary name. For example, if the clustering technique outputs three clusters, the clusters could be given the class names C_1 , C_2 , and C_3 .
2. Choose a random sample of instances from each of the classes formed as a result of the instance clustering. Each class should be represented in the random sample in the same ratio as it is represented in the entire data set. The percentage of total instances to sample can vary, but a good initial choice is two-thirds of all instances.
3. Build a supervised learner model using the randomly sampled instances as training data. Employ the remaining instances to test the supervised model for classification correctness.

This evaluation method has at least two advantages. First, the unsupervised clustering can be viewed as a structure revealed by a supervised learner model. For example, the results of a clustering created by an unsupervised algorithm can be seen as a decision tree or a rule-based structure. A second advantage of the supervised evaluation is that test set classification correctness scores can provide additional insight into the quality of the formed clusters. We demonstrate how this technique is applied to real data in later chapters.

Finally, a common misconception in the business world is that data mining can be accomplished simply by choosing the right tool, turning it loose on some data, and waiting for answers to problems. This approach is doomed to failure. Machines are still machines. It is the analysis of results provided by the human element that ultimately dictates the success or failure of a data mining project. A formal process model such as the one described in Chapter 6 will help provide more complete answers to the questions posed at the beginning of this section.

2.6 CHAPTER SUMMARY

Data mining strategies include classification, estimation, prediction, unsupervised clustering, and market basket analysis. Classification and estimation strategies are similar in that each strategy is employed to build models able to generalize current outcome. However, the output of a classification strategy is categorical, whereas the output of an estimation strategy is numeric. A predictive strategy differs from a classification or estimation strategy in that it is used to design models for predicting future outcome rather than current behavior. Unsupervised clustering strategies are employed to discover hidden concept structures in data as well as to locate atypical data instances. The purpose of market basket analysis is to find interesting relationships among entities such as retail products. Here, discovered relationships can be used to design promotions, arrange shelf or catalog items, or develop cross-marketing strategies.

A data mining *technique* applies a data mining *strategy* to a set of data. A data mining technique is defined by an algorithm and a knowledge structure. Common features that distinguish the various techniques are whether learning is supervised or unsupervised and whether their output is categorical or numeric. Familiar supervised data mining methods include decision trees, rule generators, neural networks, and statistical methods. Association rules are a favorite technique for marketing applications. Clustering techniques divide the data set into sub-groups of instances that are similar to each other and dissimilar to instances in other sub-groups. Clustering methods are frequently used to help determine a best set of input attributes for building supervised learner models.

Performance evaluation is probably the most critical of all the steps in the data mining process. Supervised model evaluation is often performed using a training/test set scenario. Supervised models with numeric output can be evaluated by computing average absolute, average squared error, or rms error differences between computed and desired outcome. Marketing applications that focus on mass mailings are interested in developing models for increasing response rates to promotions. A marketing application measures the goodness of a model by its ability to lift response rate thresholds to levels well above those achieved by naive (mass) mailing strategies. Unsupervised models support some measure of cluster quality that can be used for evaluative purposes. Supervised learning can also be employed to evaluate the quality of the clusters formed by an unsupervised model.

2.7 KEY TERMS

- *Association rule*. A production rule typically used for market basket analysis whose consequent may contain multiple conditions and attribute relationships. An output attribute in one association rule can be an input attribute in another rule.
- *Bootstrapping*. Allowing instances to appear more than once in a training set.
- *Classification*. A supervised learning strategy where the output attribute is categorical. Emphasis is on building models able to assign new instances to one of a set of well-defined classes.
- *Confusion matrix*. A matrix used to summarize the results of a supervised classification. Entries along the main diagonal represent the total number of correct classifications. Entries other than those on the main diagonal represent classification errors.
- *Cross-validation*. Partitioning a data set into n fixed-size units. $n - 1$ units are used for training, and the n th unit is used as a test set. This process is repeated until each of the fixed-size units has been used as test data. Model test set correctness is computed as the average accuracy realized from the n training/testing trials.
- *Data mining strategy*. An outline of an approach for problem solution.
- *Data mining technique*. One or more algorithms together with an associated knowledge structure.

- *Dependent variable.* A variable whose value is determined by one or more independent variables.
- *Estimation.* A supervised learning strategy where the output attribute is numeric. Emphasis is on determining current rather than future outcome.
- *Independent variable.* Input attributes used for building supervised or unsupervised learner models.
- *Lift.* The probability of class C_i given a sample taken from population P divided by the probability of C_i given the entire population P .
- *Lift chart.* A graph that displays the performance of a data mining model as a function of sample size.
- *Linear regression.* A supervised learning technique that generalizes numeric data as a linear equation. The equation defines the value of an output attribute as a linear sum of weighted input attribute values.
- *Market basket analysis.* A data mining strategy that attempts to find interesting relationships among retail products.
- *Mean absolute error.* For a set of training or test set instances, the mean absolute error is the average absolute difference between classifier predicted output and actual output.
- *Mean squared error.* For a set of training or test set instances, the mean squared error is the average of the sum of squared differences between classifier predicted output and actual output.
- *Neural network.* A set of interconnected nodes designed to imitate the functioning of the brain.
- *Outliers.* Atypical data instances.
- *Prediction.* A supervised learning strategy designed to determine future outcome.
- *Root mean squared error.* The square root of the mean squared error.
- *Statistical regression.* A supervised learning technique that generalizes numerical data as a mathematical equation. The equation defines the value of an output attribute as a sum of weighted input attribute values.

EXERCISES

Review Questions

1. Differentiate between the following terms:
 - a. Data mining technique and data mining strategy
 - b. Dependent variable and independent variable

2. Can a data mining strategy be applied with more than one data mining technique? Can a data mining technique be used for more than one strategy? Explain your answers.
3. State whether each scenario is a classification, estimation, or prediction problem.
 - a. Determine a freshman's likely first-year grade point average from the student's combined Scholastic Aptitude Test (SAT) score, high school class standing, and the total number of high school science and mathematics credits.
 - b. Develop a model to determine if an individual is a good candidate for a home mortgage loan.
 - c. Create a model able to determine if a publicly traded company is likely to split its stock in the near future.
 - d. Develop a profile of an individual who has received three or more traffic violations in the past year.
 - e. Construct a model to characterize a person who frequently visits an online auction site and makes an average of at least one online purchase per month.
4. For each task listed in question 3,
 - a. Choose a best data mining technique. Explain why the technique is a good choice.
 - b. Choose one technique that would be a poor choice. Explain why the technique is a poor choice.
 - c. Develop a list of candidate attributes.
5. Several data mining techniques were presented in this chapter. If an explanation of what has been learned is of major importance, which data mining techniques would you consider? Which of the presented techniques do not explain what they discover?
6. Suppose you have used data mining to develop two alternative models designed to accept or reject home mortgage applications. Both models show an 85% test set classification correctness. The majority of errors made by model A are *False accepts*, whereas the majority of errors made by model B are *False rejects*. Which model should you choose? Justify your answer.
7. Suppose you have used data mining to develop two alternative models designed to decide whether or not to drill for oil. Both models show an 85% test set classification correctness. The majority of errors made by model A are *False accepts*, whereas the majority of errors made by model B are *False rejects*. Which model should you choose? Justify your answer.
8. Explain how unsupervised clustering can be used to evaluate the likely success of a supervised learner model.
9. Explain how supervised learning can be used to help evaluate the results of an unsupervised clustering.

Data Mining Questions

1. Draw a sketch of the feed-forward neural network applied to the credit card promotion database in Section 2.2.3.
2. Do you own a credit card? If so, log your card usage for the next month. Place information about each purchase in an Excel spreadsheet. Keep track of the date of purchase, the purchase amount, the city and state where the purchase was made, and a general purchase category (gasoline, groceries, clothing, etc.). In addition, keep track of any other information you believe to be important that would also be available to your credit card company. In Chapter 9, you will use a neural network to build a profile of your credit card purchasing habits. Once built, the model can be applied to new purchases to determine the likelihood that the purchases have been made by you or by someone else.

Computational Questions

1. Consider the following three-class confusion matrix. The matrix shows the classification results of a supervised model that uses previous voting records to determine the political party affiliation (Republican, Democrat, or independent) of members of the United States Senate.

		Computed Decision		
		Rep	Dem	Ind
	Rep	42	2	1
	Dem	5	40	3
	Ind	0	3	4

- a. What percent of the instances were correctly classified?
 - b. According to the confusion matrix, how many Democrats are in the Senate? How many Republicans? How many Independents?
 - c. How many Republicans were classified as belonging to the Democratic Party?
 - d. How many Independents were classified as Republicans?
2. Suppose we have two classes each with 100 instances. The instances in one class contain information about individuals who currently have credit card insurance. The instances in the second class include information about individuals who have at least one credit card but are without credit card insurance. Use the following rule to answer the following questions:

IF *Life Insurance* = Yes and *Income* > \$50K

THEN *Credit Card Insurance* = Yes

Rule precision = 80%

Rule coverage = 40%

- a. How many individuals represented by the instances in the class of credit card insurance holders have life insurance and make more than \$50,000 per year?
- b. How many instances representing individuals who do not have credit card insurance have life insurance and make more than \$50,000 per year?
3. Consider the following confusion matrices.

Model X	Computed Accept	Computed Reject	Model Y	Computed Accept	Computed Reject
	Accept	54		Accept	55
Reject	2245	7655	Reject	1955	7945

- a. Compute the lift for model X.
- b. Compute the lift for model Y.
4. A certain mailing list consists of P names. Suppose a model has been built to determine a select group of individuals from the list who will receive a special flyer. As a second option, the flyer can be sent to all individuals on the list. Use the notation given in the following confusion matrix to show that the *lift* for choosing the model over sending the flyer to the entire population can be computed with the equation

$$\text{lift} = \frac{C_{11}P}{(C_{11} + C_{12})(C_{11} + C_{21})}$$

Send Flyer?	Computed Send	Computed Don't Send
	Send	C_{11}
Don't Send	C_{21}	C_{22}

5. Use RapidMiner's *Subgroup Discovery* operator's definition of rule accuracy to compute the accuracy of rules 2, 3, and 4 listed in Section 2.2.2 ("Rule-Based Techniques").



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Basic Data Mining Techniques

CHAPTER OBJECTIVES

- *Understand an algorithm for constructing decision trees*
- *Know how the production rules mapped from a decision tree can be simplified*
- *Understand a basic algorithm for generating covering rules*
- *Know an efficient technique for generating association rules*
- *Understand how support and confidence are used to determine the value of an association rule*
- *Understand how the K-means algorithm is used to partition instances containing numeric data into disjoint clusters*
- *Know how genetic algorithms perform supervised learning*
- *Know how to choose a data mining technique for a specific problem*

In Chapters 1 and 2, we focused on the data structures and resultant output produced by several data mining techniques. We now turn our attention to the algorithms used to create these data structures by detailing four fundamental data mining techniques. In Section 3.1, we focus on supervised learning by presenting a standard algorithm for creating decision trees. In Section 3.2, we outline a fundamental covering rule algorithm. In Section 3.3, we demonstrate an efficient technique for generating association rules. The focal point of Section 3.4 is unsupervised clustering and the K-means algorithm. In addition to these basic techniques, Section 3.5 shows you how genetic algorithms can perform supervised learning. A genetic approach can oftentimes be used in conjunction with a more traditional technique to improve model performance. We conclude this chapter with a brief discussion about things to consider when choosing a data mining technique.

3.1 DECISION TREES

Decision trees are a popular structure for supervised learning. Countless articles have been written about successful applications of decision tree models to real-world problems. We introduced the C4.5 decision tree model in Chapter 1. In this section, we take a closer look at the algorithm used by C4.5 for building decision trees. We then apply this algorithm to the credit card promotion database described in Chapter 2.

3.1.1 An Algorithm for Building Decision Trees

Decision trees are constructed using only those attributes best able to differentiate the concepts to be learned. A decision tree is built by initially selecting a subset of instances from a training set. This subset is then used by the algorithm to construct a decision tree. The remaining training set instances test the accuracy of the constructed tree. If the decision tree classifies the instances correctly, the procedure terminates. If an instance is incorrectly classified, the instance is added to the selected subset of training instances, and a new tree is constructed. This process continues until a tree that correctly classifies all nonselected instances is created or the decision tree is built from the entire training set. We offer a simplified version of the algorithm that employs the entire set of training instances to build a decision tree. The steps of the algorithm are as follows:

-
1. Let T be the set of training instances.
 2. Choose an attribute that best differentiates the instances contained in T .
 3. Create a tree node whose value is the chosen attribute. Create child links from this node where each link represents a unique value for the chosen attribute. Use the child link values to further subdivide the instances into subclasses.
 4. For each subclass created in step 3,
 - a. If the instances in the subclass satisfy predefined criteria or if the set of remaining attribute choices for this path of the tree is null, specify the classification for new instances following this decision path.
 - b. If the subclass does not satisfy the predefined criteria and there is at least one attribute to further subdivide the path of the tree, let T be the current set of subclass instances and return to step 2.
-

Before we show you how to apply the algorithm to a set of data, a word about attribute selection (step 2 of the algorithm) is in order. The attribute choices made when building a decision tree determine the size of the constructed tree. A main goal is to minimize the number of tree levels and tree nodes, thereby maximizing data generalization. C4.5 uses a measure taken from *information theory* to help with the attribute selection process. The basic idea is that for any choice point in the tree, C4.5 selects the attribute that splits the data so as to show the largest amount of gain in information. To show this, suppose we have n possible outcomes (classes). The information conveyed by any of these outcomes can be measured in bits as $-\log_2(1/n)$. For example, with $n = 4$, we have $-\log_2(1/4) = 2$. That is, it takes two bits to represent four possible outcomes (00, 01, 10, and 11). Stated another way, two bits uniquely identify four classes. Suppose attribute A is chosen for the next data

split, and the split results in an average of two classes for each new branch of the tree. Because of this, each tree branch will require an average of $-\log_2(1/2) = 1$ bit to represent two possible outcomes. Therefore, choosing attribute A results in an information gain of 1 bit. At each choice point in the tree, C4.5 computes a *gain ratio* based on this idea for all available attributes. The attribute with the largest value for this ratio is selected to split the data.

For the interested reader, an example of how C4.5 uses gain ratio for attribute selection is shown as an optional section titled “C4.5 Attribute Selection.” Here we offer an intuitive approach to the process of attribute selection. Let’s apply the simplified decision tree algorithm to the credit card promotion database defined in Chapter 2!

C4.5 ATTRIBUTE SELECTION (OPTIONAL)

Here we describe the formulas used by C4.5 for attribute selection. At each choice point of the decision tree building process, the attribute showing the largest *gain ratio* is the attribute selected to further subdivide the tree structure. Here is the formula to compute the gain ratio for attribute A.

$$\text{Gain Ratio}(A) = \text{Gain}(A)/\text{Split Info}(A)$$

For a set of I instances, the formula for computing Gain(A) is given as

$$\text{Gain}(A) = \text{Info}(I) - \text{Info}(I, A)$$

where Info(I) is the information contained in the currently examined set of instances and Info(I,A) is the information after partitioning the instances in I according to the possible outcomes for attribute A.

The formulas for computing Info(I), Info(I,A), and Split Info(A) are straightforward.

For n possible classes, the formula for computing Info(I) is

$$\text{Info}(I) = - \sum_{i=1}^n \frac{\# \text{ in class } i}{\# \text{ in } I} \log_2 \left(\frac{\# \text{ in class } i}{\# \text{ in } I} \right)$$

After I is partitioned into k outcomes, Info(I,A) is computed as

$$\text{Info}(I, A) = \sum_{j=1}^k \frac{\# \text{ in class } j}{\# \text{ in } I} \text{info(class } j)$$

Finally, Split Info(A) normalizes the gain computation to eliminate a bias for attribute choices with many outcomes. Without using the value for split info, attributes with unique values for each instance will always be selected. Once again, for k possible outcomes,

$$\text{Split Info}(A) = - \sum_{j=1}^k \frac{\# \text{ in class } j}{\# \text{ in } I} \log_2 \left(\frac{\# \text{ in class } j}{\# \text{ in } I} \right)$$

Let's use the data given in Table 3.1 to see how the formulas are applied. For our example, we designate *life insurance promotion* as the output attribute. For illustrative purposes, we apply the formula to the entire set of training data. We wish to develop a predictive model. Therefore, the input attributes are limited to *income range*, *credit card insurance*, *gender*, and *age*. The purpose of the attribute designated as *ID* is to reference items within the table.

Figure 3.1 shows the results of a top-level split on the attribute *income range*. The total yes and no counts for the output attribute *life insurance promotion* are shown at the bottom of each branch. Although this attribute is not a good choice for splitting the table data, it is a good selection for demonstrating the formulas previously described.

We first compute the value for $\text{Info}(I)$. Since we are trying to determine the top-level node of the tree, I contains all 15 table instances. Given that *life insurance promotion* contains nine yes values and six no values, the computation is

$$\text{Info}(I) = -[9/15\log_2 9/15 + 6/15\log_2 6/15] = 0.97095$$

Income range has four possible outcomes; the computation of $\text{Info}(I, \text{income range})$ is

$$\begin{aligned} \text{Info}(I, \text{income range}) &= 4/15\text{Info}(20-30,000) + 5/15\text{Info}(30-40,000) \\ &\quad + 4/15\text{Info}(40-50,000) + 2/15\text{Info}(50-60,000) \\ &= 0.72365 \end{aligned}$$

where

$$\begin{aligned} \text{Info}(20-30,000) &= -[2/4\log_2 2/4 + 2/4\log_2 2/4] \\ \text{Info}(30-40,000) &= -[4/5\log_2 4/5 + 1/5\log_2 1/5] \\ \text{Info}(40-50,000) &= -[3/4\log_2 3/4 + 1/4\log_2 1/4] \\ \text{Info}(50-60,000) &= -[2/2\log_2 2/2] \end{aligned}$$

and

$$\begin{aligned} \text{Split Info(income range)} &= -[4/15\log_2 4/15 + 5/15\log_2 5/15 + 4/15\log_2 4/15 + 2/15\log_2 1/5] \\ &\approx 1.93291 \end{aligned}$$

We now compute the gain as

$$\begin{aligned} \text{Gain(income range)} &= \text{Info}(I) - \text{Info}(I, \text{income range}) \\ &\approx 0.97905 - 0.72193 = 0.25712 \end{aligned}$$

Finally,

$$\begin{aligned} \text{Gain Ratio}(income range) &= \text{Gain}(income range)/\text{Split Info}(income range) \\ &\approx 0.25712 / 1.93291 = 0.13302 \end{aligned}$$

Scores for the categorical attributes *credit card insurance* and *gender* are computed in a similar manner. The question now becomes how do we apply the formulas in order to obtain a score for the numerical attribute *age*? The answer is to discretize the data by sorting numeric values and to compute a gain ratio score for each possible binary split point. For our example, the ages are first sorted as follows:

19	27	29	35	38	39	40	41	42	43	43	43	45	55	55
Y	N	Y	Y	Y	Y	Y	Y	N	Y	Y	N	N	N	N

Next, we compute a score for each possible split point. That is, the score for a binary split between 19 and 27 is computed as is the score for a split between 27 and 29. This process continues until a score for the split between 45 and 55 is obtained. In this way, each split point is treated as a separate attribute with two values. Finally, by making computations for *income range*, *credit card insurance*, *gender*, and *age*, we find that *credit card insurance* has the best gain ratio score of 3.610.

We follow our previous work with this data set and designate *life insurance promotion* as the output attribute. Once again, we wish to develop a predictive model. Therefore, the input attributes are limited to *income range*, *credit card insurance*, *gender*, and *age*. Table 3.1 shows the training data. With the training data selected, we can proceed to step 2 of the algorithm, which tells us to choose an input attribute to best differentiate the instances

TABLE 3.1 The Credit Card Promotion Database

ID	Income Range	Life Insurance Promotion	Credit Card Insurance	Gender	Age
1	40–50K	No	No	Male	45
2	30–40K	Yes	No	Female	40
3	40–50K	No	No	Male	42
4	30–40K	Yes	Yes	Male	43
5	50–60K	Yes	No	Female	38
6	20–30K	No	No	Female	55
7	30–40K	Yes	Yes	Male	35
8	20–30K	No	No	Male	27
9	30–40K	No	No	Male	43
10	30–40K	Yes	No	Female	41
11	40–50K	Yes	No	Female	43
12	20–30K	Yes	No	Male	29
13	50–60K	Yes	No	Female	39
14	40–50K	No	No	Male	55
15	20–30K	Yes	Yes	Female	19

of the training data. Our choices are *income range*, *credit card insurance*, *gender*, and *age*. Let's look at each possibility.

For our first choice, we consider *income range*. Figure 3.1 shows the partial tree created in step 3 of the algorithm with income range selected as the top-level node. The total *yes* and *no* counts for the output attribute (*life insurance promotion*) are shown at the bottom of each branch of the partial tree. To evaluate the choice of income range as the top node of the decision tree, we first make the value of each path of the partial tree the most frequently encountered class. For the case of the *income range* = 20–30K branch, we have two instances from each class. So for this case, we can select either *life insurance promotion* = *no* or *life insurance promotion* = *yes* as the value of the path. To break the tie, we opt for the most frequently occurring class, which is *life insurance promotion* = *yes* ($2 + 4 + 1 + 2 = 9$). For the branch showing *income range* = 30–40K, we choose *life insurance promotion* = *yes* as the value of the path. For *income range* = 40–50K, we choose *life insurance promotion* = *no*, and for *income range* = 50–60K, we select *life insurance promotion* = *yes*.

Upon making these selections, the partial tree correctly classifies 11 ($2 + 4 + 3 + 2$) of the 15 training set instances. The result is a training set classification correctness of over 73%. This simple measure tells us something about the ability of the attribute to group the instances into the defined classes. However, the measure does not take into account the generalization capabilities of the attribute. For example, what happens when the training data contains an attribute, such as an identification number, that is unique to each instance? Obviously, each training instance is correctly classified by its unique identification number. Therefore, the training set classification correctness score for the attribute will be 100%. However, picking such an attribute is a mistake, as the final decision tree will be a one-level structure having a unique path for each training instance—here, a single node with 15 branches!

A simple way to add a generalization factor to the accuracy measure is to divide training set accuracy by the total number of branches added to the tree (i.e., the total number of unique attribute values) as a result of the attribute choice. In this way, attribute selections resulting in fewer additional tree branches will be favored. To apply this method to the

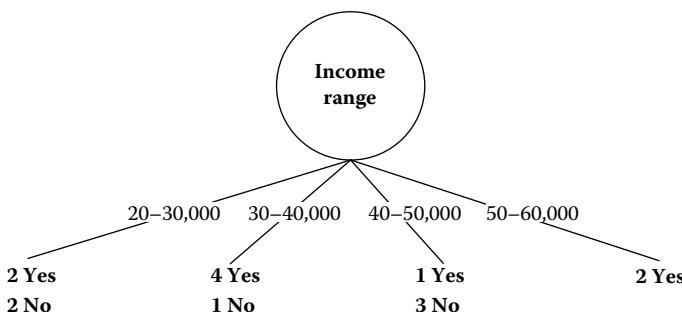


FIGURE 3.1 A partial decision tree with root node = income range.

income range attribute, we divide the accuracy score of 11/15 by 4. This results in a goodness score for attribute income range of approximately 0.183.

Let's consider *credit card insurance* as a candidate for the top-level node of the decision tree. Figure 3.2 displays the partial tree created in step 3 of the algorithm provided that *credit card insurance* is the selected attribute. Using the same reasoning as just discussed, we terminate the tree building process here and calculate the training set classification accuracy of the partial tree. For the branch *credit card insurance* = no, we have six yes and six no responses to the life insurance promotion. Once again, we break the tie by selecting the most frequently occurring class, which is *life insurance promotion* = yes. For the branch showing *credit card insurance* = yes, we choose *life insurance promotion* = yes. To summarize, following either path of the tree, we always make the choice *life insurance promotion* = yes. The resulting training set accuracy is 60% (9 of 15 correct choices). Dividing 0.60 by the number of branches added to the tree as a result of the attribute choice provides a goodness score of 0.30, indicating that *credit card insurance* may be a better choice for the top node in the decision tree than *income range*.

We now consider the numeric attribute *age* as a possible choice for the top-level node of the decision tree. A common method for processing numeric data is to sort the values and consider binary splits between each pair of values. For our example the ages are first sorted as follows:

19	27	29	35	38	39	40	41	42	43	43	43	45	55	55
Y	N	Y	Y	Y	Y	Y	Y	N	Y	Y	N	N	N	N

A goodness score is then computed for each possible split point. That is, the score for a binary split between 19 and 27 is computed, a score for a split between 27 and 29 is computed, and so forth until a score for the split between 45 and 55 is obtained. In this way, each split point is treated as a separate attribute with two values. In making this computation for each choice point, our simple heuristic tells us that 43 results in the best split of the data. We associate *life insurance promotion* = yes with *age* ≤ 43 and *life insurance*

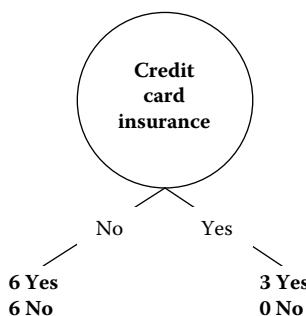


FIGURE 3.2 A partial decision tree with root node = credit card insurance.

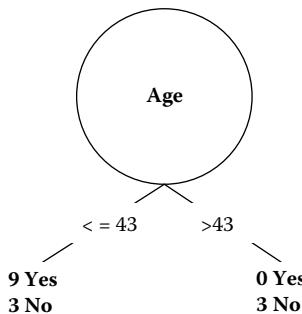


FIGURE 3.3 A partial decision tree with root node = age.

promotion = no with *age > 43*, since the majority of instances above this breakpoint are *yes* and the majority below are *no*. The training set accuracy is 80% (12 of 15 correct), and the goodness score for this attribute is 0.40.

Finally, we consider attribute *gender* as a candidate top-level node. Choosing *gender* results in a goodness score of approximately 0.367. (We leave the computational details for this selection as an exercise.) Comparing the four results, we see that attribute *age* offers the best score among the possible attribute selections. Therefore, we make *age* the attribute of choice and execute step 3 of the decision tree algorithm. The partial tree with *age* as the top-level node is shown in Figure 3.3.

Step 4a of the algorithm requires us to examine each branch of the partial tree to determine if we are to continue the tree building process. The algorithm states two possibilities for terminating a path of the tree. First, the branch becomes a terminal path. The path is then assigned the value of the most frequently occurring class. An obvious termination criterion is that all instances following a specific path must be from the same class. Another termination criterion can be that the path has reached a minimum training set classification accuracy. A third possibility for terminating a path of the tree is the lack of an attribute for continuing the tree splitting process. To be sure, if a categorical attribute is selected, its values are able to divide the tree but once. However, a numerical attribute can be used to split the data several times. For our example, the training instances following the branch having *age > 43* all have a value of *no* for life insurance promotion. Therefore, we terminate this path and label the leaf node as *life insurance promotion = no*.

Next we consider the path with *age <= 43*. Since this path shows nine instances having *yes* for the output attribute and three instances having *no* for the output attribute, we are able to continue building the decision tree. Notice that step 4b of the algorithm tells us that the instances following this path are assigned as the new value of *T*. After the assignment for *T*, steps 2, 3, and 4 of the algorithm are repeated. This process continues until all paths meet the termination criteria or until all possibilities for attribute selection have been exhausted.

3.1.2 Decision Trees for the Credit Card Promotion Database

We applied two implementations of C4.5 to the data in Table 3.1. The versions vary slightly in their process for selecting attributes. The decision trees formed by each implementation

are shown in Figures 3.4 and 3.5. The tree in Figure 3.4 contains three nodes and was created by the most recent version of C4.5. It is worth noting that the gain ratio computation chose attribute *age* with a split at $age = 43$ as the top-level node. Following the right branch of the tree, we see that individuals with age greater than 43 did not take advantage of the life insurance promotion. The 3 shown in parentheses indicates that three of the training instances follow this path. The 0 tells us that all three classifications are correct. Notice

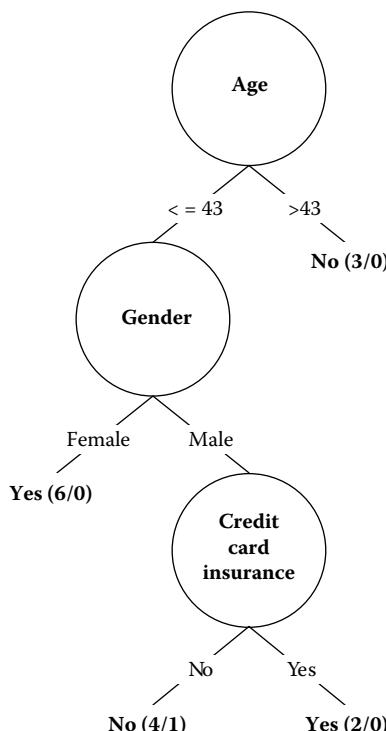


FIGURE 3.4 A three-node decision tree for the credit card database.

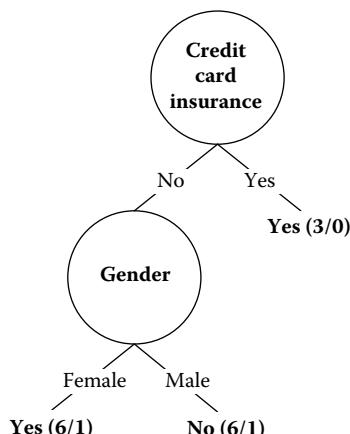


FIGURE 3.5 A two-node decision tree for the credit card database.

that the branch for *credit card insurance = no* shows one incorrect classification. The incorrect classification tells us that one male individual with age less than or equal to 43 did not purchase credit card insurance but said yes to the life insurance promotion. As all the other classifications are correct, the tree is able to accurately classify 14 of the 15 training instances.

The tree in Figure 3.5 has two rather than three nodes containing attribute tests. As with the tree in Figure 3.4, the tree shows attributes *credit card insurance* and *gender*. However, the tree does not have a test for the *age* attribute. As you can see, the decision tree in Figure 3.5 incorrectly classifies two training instances versus one incorrectly classified instance for the tree in Figure 3.4, but has the advantage of requiring only two attributes to classify new instances.

You might be wondering how it is that a decision tree would not correctly classify all training instances. After all, couldn't you just add some more branches to take care of this? Let's use the decision tree in Figure 3.4 to see how this can happen. Table 3.2 displays the four training instances that follow the decision tree path in Figure 3.4 to *credit card insurance = no*. At this point, the algorithm either decides that the predefined criteria stated in step 4a of the algorithm have been satisfied or selects a new attribute to further subdivide the instances. Table 3.2 tells us that the first three instances have *life insurance promotion = no*. The fourth instance has *life insurance promotion = yes*. However, the fourth instance is identical to the second instance with the exception of the value for *life insurance promotion* and the value for *age*. As *life insurance promotion* is the output attribute, the only possible attribute choice is *age*. As noted earlier, once a categorical attribute has been selected as a tree node, it cannot be picked again. However, numerical attributes can be used several times provided that a new split point is chosen each time the attribute is selected. Therefore, the alternatives are to terminate further development of the path or to employ the attribute *age* and create a new node. As you can see, the algorithm chose to terminate the tree building process. Let's examine the alternative.

Table 3.2 shows an age of 29 for the single instance with *life insurance promotion = yes*. The only possibility for a split on age is to make one path with *age <= 29* and the second path having *age > 29*. The first and third instances of Table 3.2 follow *age > 29*. As both instances show *yes* for life insurance promotion, the path is terminal. The path for *age <= 29* requires another split on the age attribute (*age <= 27* and *age > 27* are possibilities) for the tree to successfully classify all training instances. The end result is a decision tree with five nodes. However, instead of creating two new tree nodes to eliminate the incorrect

TABLE 3.2 Training Data Instances Following the Path in Figure 3.4 to *Credit Card Insurance = No*

Income Range	Life Insurance Promotion	Credit Card Insurance	Gender	Age
40–50K	No	No	Male	42
20–30K	No	No	Male	27
30–40K	No	No	Male	43
20–30K	Yes	No	Male	29

classification, the logic of the algorithm gives preference to a more general tree structure and discounts the error.

3.1.3 Decision Tree Rules

In Chapter 1, you saw how a decision tree can be mapped to a set of production rules by writing one rule for each path of the tree. As rules tend to be more appealing than trees, several variations of the basic tree to rule mapping have been studied. Most variations focus on simplifying and/or eliminating existing rules. To illustrate the rule simplification process, consider the decision tree in Figure 3.4. A rule created by following one path of the tree is shown here:

IF Age ≤ 43 and Gender = Male and Credit Card Insurance = No
THEN Life Insurance Promotion = No

The antecedent conditions for this rule cover 4 of the 15 training set instances with a 75% accuracy. Let's simplify the rule by eliminating the antecedent condition for *age*. The simplified rule takes the following form:

IF Gender = Male and Credit Card Insurance = No
THEN Life Insurance Promotion = No

By examining Table 3.1, we see that the antecedent of the simplified rule covers six instances. As the rule consequent covers five of the six instances, the accuracy of the simplified rule is approximately 83.3%. Therefore, the simplified rule is more general and more accurate than the original rule! At first thought, it seems hard to believe that removing a conditional test can actually improve the accuracy of a rule. However, closer examination shows why eliminating the test gives a better result. To see this, notice that removing the *age* attribute from the rule is equivalent to deleting the attribute from the tree in Figure 3.4. In doing so, the three instances following the path *age* > 43 must now follow the same path as those instances traversing *age* ≤ 43 . All three instances with *age* > 43 are members of the *life insurance promotion = no* class. Two of these three instances are of male gender with *credit card insurance = no*. Both instances satisfy the preconditions and consequent condition of the simplified rule. Because of this, the preconditions for the new rule are satisfied by six instances, five of which have *no* as the value of *life insurance promotion*.

Most decision tree implementations automate the process of rule creation and simplification. Once rules have been simplified and/or eliminated, the rules are ordered so as to minimize error. Finally, a default rule is chosen. The default rule states the classification of an instance not meeting the preconditions of any listed rule.

3.1.4 Other Methods for Building Decision Trees

We just described the basics of C4.5—Quinlan's most recent noncommercial decision tree building algorithm. However, several other algorithms for building decision trees exist. Iterative Dichotomiser 3 (ID3) (Quinlan, 1986) has been studied extensively and is the precursor to C4.5. Classification And Regression Tree algorithm (CART) (Breiman et al., 1984)

is of particular interest as several commercial products implement variations of the algorithm. In addition, CART was the first system to introduce *regression trees*. Essentially, regression trees take the form of decision trees where the leaf nodes are numerical rather than categorical values.

CART is very similar to C4.5, but there are several differences. One notable difference is that CART always performs binary splits on the data regardless of whether attributes are categorical or numeric. A second difference is that CART invokes test data to help prune and therefore generalize a created binary tree, whereas C4.5 uses only training data to create a final tree structure. Chi-square automatic interaction detection (CHAID) (Kass, 1980) is a second decision tree building algorithm of interest found in commercial statistical packages such as Statistical Analysis System (SAS) and Statistical Package for the Social Sciences (SPSS). CHAID differs from C4.5 and CART in that it is limited to working with categorical attributes. CHAID has a statistical flavor as it uses the X^2 statistical test of significance to determine candidate attributes for building the decision tree.

3.1.5 General Considerations

Decision trees have several advantages. Here is a list of a few of the many advantages decision trees have to offer.

- Decision trees are easy to understand and map nicely to a set of production rules.
- Decision trees have been successfully applied to real problems.
- Decision trees make no prior assumptions about the nature of the data.
- Decision trees are able to build models with data sets containing numerical as well as categorical data.

As with all data mining algorithms, there are several issues surrounding decision tree usage. Specifically,

- Output attributes must be categorical, and multiple output attributes are not allowed.
- Decision tree algorithms are *unstable* in that slight variations in the training data can result in different attribute selections at each choice point within the tree. The effect can be significant as attribute choices affect all descendent subtrees.
- Trees created from numeric data sets can be quite complex as attribute splits for numeric data are typically binary.

3.2 A BASIC COVERING RULE ALGORITHM

In Chapter 2, we briefly introduced a class of rule generators that use a *covering* approach. Recall that an instance is said to be covered by a rule if it satisfies the rule's preconditions. For each concept class, the goal is to create a set of rules that maximize the total number of covered within-class instances while at the same time minimizing the number of covered nonclass instances.

Here we outline a straightforward covering technique based on the PRISM algorithm (Cendrowska, 1987). The method starts with a single rule whose antecedent condition is empty and whose consequent is given as the class representing the instances to be covered. At this point, the domain of all data set instances is covered by the rule. That is, when the rule is executed, it classifies every instance as a member of the class specified in the rule consequent. In order to restrict rule coverage to those instances within the class, input attribute values are considered one by one as possible additions to the rule antecedent. For each iteration, the attribute-value combination chosen is the one that best represents those instances within the class. A rule is added to the list of covering rules once the preconditions of the rule are satisfied solely by members of the class in question or the attribute list has been exhausted. After the rule has been added, the instances covered by the rule are removed from the pool of uncovered instances. The process continues until all instances from the class in question are covered by one or more rules. The method is then repeated for all remaining classes. Several variations of this general technique exist, but the covering concept is the same.

To better illustrate the workings of this approach, we use the credit card promotion database displayed in Table 3.1. For our experiment, we will assume the Acme Credit Card Company has authorized a new life insurance promotion. We wish to determine which zero-balance credit card holders are likely to show interest in the promotion. The next billing statement will then be sent to those individuals who must submit a payment and to all zero-balance cardholders determined by our model as likely candidates for the promotional offering. Once again, we want to develop a predictive model. Therefore, the input attributes are limited to *income range*, *credit card insurance*, *gender*, and *age*. Learning is supervised with *life insurance promotion* as the output attribute.

To help us with our explanation, we use the Attribute and Cluster Analyzer (ACZ), a basic analytics tool with several capabilities. Here, our discussion is limited to its use as a visualization tool to help with attribute selection and outlier detection. ACZ is a work in progress implemented as an executable Java Archive (JAR) file. If you wish to experiment with ACZ, you can freely download it at our website. ACZ is of interest as it provides two conditional probabilities key to the covering rule approach for rule generation.

Figure 3.6 shows the overall (domain) statistics for the credit card promotions database as they appear in ACZ. We see attribute names, values, frequencies, and *predictability* scores. A predictability score is a conditional probability uniquely associated with each categorical attribute-value combination. Here is the definition of attribute-value predictability.

The predictability of attribute A_i having value V_{ij} given class C_k is denoted as $P(A_i = V_{ij}|C_k)$ and represents the conditional probability that $A_i = V_{ij}$ knowing that the attribute-value combination is in class C_k .

As you will see, although the definition of predictability seems formidable, the idea is not difficult.

The definition indicates that predictability is a probability with a conditional component relative to a specific class. It is important to note that the predictability values shown

Figure 3.6.xlsx - Excel

Domain Statistics for Categorical Attributes				Domain Statistics-Numerical Attributes					
	Name	Value	Freq	Predictability	Name	No	Yes	Domair	Att. Sig
CreditCardIns		No	12	0.8	Age (mean)	44.5	36.33	39.6	0.86
		Yes	3	0.2	(sd)	10.4	7.83	9.51	
Gender		Female	7	0.47					
		Male	8	0.53					
IncomeRange		20-30000	4	0.27					
		30-40000	5	0.33					
		40-50000	4	0.27					
		50-60000	2	0.13					
LifeInsPromo		No	6	0.4					
		Yes	9	0.6					

FIGURE 3.6 Domain statistics for the credit card promotion database.

in Figure 3.6 are for the entire domain of instances rather than a specific class. Therefore, the conditional component is simply that the attribute-value combination is seen in at least one instance within the database.

To illustrate predictability, consider Figure 3.6 with values *yes* and *no* for categorical attribute *credit card insurance*. We see a predictability value of 0.20 associated with *credit card insurance = yes*. This tells us that 20% (3/15) of all individuals within the data set have credit card insurance. The 0.80 predictability score associated with *credit card insurance = no* tells us that 80% (12/15) of all individuals do not have credit card insurance. As *yes* and *no* are the only values for this attribute, the sum of their predictability scores is 1.0. Similar statements can be made about all other categorical attributes. Predictability scores for numeric data are not defined; instead, with numeric attributes such as *age*, Figure 3.6 displays mean and standard deviation values for the overall domain as well as for the individual classes.

Domain statistics offer some information about the nature of the data under analysis, but of greater interest are the individual class statistics. Figure 3.7 displays summary statistics for the class of individuals who took advantage of the life insurance promotion. Here we see a column for attribute-value predictability together with a new column for attribute-value *predictiveness*. Here is the definition of attribute-value predictiveness:

Given class C_k , and attribute A_i with value V_{ij} , the predictiveness score for $A_i = V_{ij}$ denoted as $P(C_k|A_i = V_{ij})$ represents the conditional probability that an instance is in class C_k given that attribute A_i has value V_{ij} . If $P(C_k|A_i = V_{ij})$ is 1, any instance containing this attribute-value pair resides in C_k .

The screenshot shows an Excel spreadsheet titled "Figure 3.7.xlsx - Excel". The data is organized into several sections:

- Section 1: CLASS SPECIFIC RESULTS**

 - Class**: Yes
 - Number of Class Instances**: 9

- Section 2: Predictability and Predictiveness**

	Name	Value	Frequency	Predictability	Predictiveness
CreditCardIns	No	6	0.67	0.5	
	Yes	3	0.33	1	
Gender	Female	6	0.67	0.86	
	Male	3	0.33	0.38	
IncomeRange	20-30000	2	0.22	0.5	
	30-40000	4	0.44	0.8	
	40-50000	1	0.11	0.25	
	50-60000	2	0.22	1	

FIGURE 3.7 Class statistics for *life insurance promotion = yes*.

Unlike predictability, predictiveness is a between-class measure. To see this, once again consider Figure 3.7 and *credit card insurance = yes*. The predictability score of 0.33 tells us that only 33% of all individuals who said *yes* to the life insurance promotion have credit card insurance. Knowing this, we need look no further to conclude that 67% of the life insurance promotion acceptors did not purchase credit card insurance. Now take a look at the predictiveness score of 1.0 for *credit card insurance = yes*. This tells us that all three individuals who purchased credit card insurance reside in the current class. This is easily verified by examining Figure 3.8, which gives statistics for the class *life insurance promotion = no*. Here we see the predictiveness score for *credit card insurance = yes* as 0.00. As a second example, Figure 3.7 shows a value of 0.86 as the predictiveness score for *gender = female*. This tells us that 86% of all females found within the data reside in this class. This also tells us that 14% of the individuals in the class *life insurance promotion = no* are of female gender.

Knowing the definitions of attribute-value predictability and predictiveness, we finally have what we need to better understand the basics of covering rules. Let's celebrate by generating a few covering rules for the class *life insurance promotion = yes*!

We start the rule generation process by choosing an attribute-value combination that best represents the class *life insurance promotion = yes*. If predictability scores are used as a starting point, we have an immediate problem. Specifically, suppose the *life insurance promotion = yes* class shows *yes* for the value of *credit card insurance* with a predictability score of 1.0. This tells us that every instance within this class has the value *yes* for credit card insurance. However, this says nothing about the competing class. That is, every instance within *life insurance promotion = no* could also show credit card insurance as *yes*! If this is the case, the credit card insurance attribute is of no predictive value! Instead, we must first examine

The screenshot shows an Excel spreadsheet titled "Figure 3.8.xlsx - Excel". The data is organized into several sections:

- Section 1: CLASS SPECIFIC RESULTS**
- Section 2: Class** (No)
- Section 3: Number of Class Instances 6**
- Section 4: Predictability and Predictiveness** (Rows 5-7):

Name	Value	Frequency	Predictability	Predictiveness
CreditCardIns	No	6	1	0.5
	Yes	0	0	0
- Section 5: Gender** (Rows 9-10):

Gender	Female	1	0.17	0.14
	Male	5	0.83	0.63
- Section 6: Income Range** (Rows 12-15):

IncomeRange	20-30000	2	0.33	0.5
	30-40000	1	0.17	0.2
	40-50000	3	0.5	0.75
	50-60000	0	0	0

FIGURE 3.8 Class statistics for *life insurance promotion = no*.

predictiveness scores in order to make between-class comparisons. Once we find those attribute-value combinations best able to distinguish between the competing classes, we then use predictability scores to help choose the combinations that cover the greatest number of class instances. With this important information in hand, let's return to our example.

Figure 3.7 shows two attribute-value combinations with a predictiveness score of 1.0. Our first choice is *credit card insurance = yes* as it uniquely covers the most instances (4, 7, and 15). We can immediately generate the following rule and remove the covered instances from the database:

- IF Credit Card Insurance = Yes
THEN Life Insurance Promotion = Yes
Rule precision: 100%
Rule coverage: 20%

Our next choice is the attribute-value combination *income range = 50–60K* as it uniquely covers instances 5 and 13, thereby generating the following rule:

- IF Income Range = 50–60K
THEN Life Insurance Promotion = Yes
Rule precision: 100%
Rule coverage: 13.33%

We now remove instances 5 and 13 from the database. We have covered five of the nine class instances but must still attempt to cover instances 2, 10, 11, and 12.

To get a better picture of the current situation, we invoked ACZ a second time. The result is displayed in Figure 3.9.

Our first choice for the third rule is $gender = female$ with a predictiveness and predictability score of 0.75. A rule with this antecedent condition will correctly cover instances 2, 10, and 11 but erroneously covers instance 6. Therefore, we must look for another attribute in order to add a second condition. The obvious choice is $income range = 30-40K$ as the combination uniquely covers instances 2 and 10. We remove instances 2 and 10 after generating our third rule:

- IF Gender = Female and Income Range = 30–40K

THEN Life Insurance Promotion = Yes

Rule precision: 100%

Rule coverage: 13.33%

Next, we remove instance 11 as it is uniquely covered with the following rule:

- IF Gender = Female and Income Range = 40–50K

THEN Life Insurance Promotion = Yes

Rule precision: 100%

Rule coverage: 6.67%

The screenshot shows an Excel spreadsheet titled "Figure 3.9.xlsx - Excel". The table has the following data:

CLASS SPECIFIC RESULTS				
Class	Yes			
Number of Class Instances 4				
Name	Value	Frequency	Predictability	Predictiveness
CreditCardIns	No	4	1	0.4
Gender	Female	3	0.75	0.75
	Male	1	0.25	0.17
IncomeRange	20-30000	1	0.25	0.33
	30-40000	2	0.5	0.67
	40-50000	1	0.25	0.25

FIGURE 3.9 Statistics for *life insurance promotion = yes* after removing five instances.

We have covered eight of the nine class instances. Instance 12 presents a problem as other than attributes *age* and *life insurance promotion*, instance 12 is identical to instance 8. As we did not discretize age, we are unable to use this attribute. Furthermore, a discretization of age is likely to place both instances in the same bin. That said, we were able to build a set of rules that covered eight of the nine class instances using the basic covering technique outlined previously.

In practice, the method given here is but the first step in constructing a set of useful rules. This is the case as rules based solely on a single data set, although accurate for the instances within the data set, may not do well when applied to the larger population. Techniques for generating useful covering rules are often based on the concept of *incremental reduced error pruning*. The general idea is to divide the instances using a 2:1 ratio into two sets. The larger (grow) set is used during the growing phase of the algorithm for growing perfect rules. The smaller (prune) set is applied during the pruning phase for removing one or more antecedent conditions from perfect rules.

During the growing phase, the *grow* set is used for creating a perfect rule R for a given class C . Next, a heuristic measure of R 's worth such as classification correctness is computed by applying R to the *prune* set. After this, rule antecedent conditions are removed from R one by one each time, creating a new rule $R-$, which contains one less precondition than the previous rule. Upon each removal, a new worth score for $R-$ is computed. This process continues as long as the worth of each new rule is greater than the worth of the original perfect rule. Upon termination of this process, the rule with the highest worth is added to the list of rules for C , and the instances covered by this best rule are removed from the instance set. This procedure repeats for C as long as at least one instance remains in both grow and prune. This entire process repeats itself for all remaining classes within the data. An enhanced version of this basic technique that includes an optimization phase is known as the RIPPER (*repeated incremental pruning to produce error reduction*) algorithm (Cohen, 1995) a variation of which is implemented in both Weka and RapidMiner.

3.3 GENERATING ASSOCIATION RULES

Affinity analysis is the general process of determining which things go together. A typical application is market basket analysis, where the desire is to determine those items likely to be purchased by a customer during a shopping experience. The output of the market basket analysis is a set of associations about customer purchase behavior. The associations are given in the form of a special set of rules known as association rules. The association rules are used to help determine appropriate product marketing strategies. In this section, we describe an efficient procedure for generating association rules.

3.3.1 Confidence and Support

Association rules are unlike traditional classification rules in that an attribute appearing as a precondition in one rule may appear in the consequent of a second rule. In addition, traditional classification rules usually limit the consequent of a rule to a single attribute. Association rule generators allow the consequent of a rule to contain one or several

attribute values. To show this, suppose we wish to determine if there are any interesting relationships to be found in customer purchasing trends among the following four grocery store products:

- Milk
- Cheese
- Bread
- Eggs

Possible associations include the following:

1. If customers purchase milk, they also purchase bread.
2. If customers purchase bread, they also purchase milk.
3. If customers purchase milk and eggs, they also purchase cheese and bread.
4. If customers purchase milk, cheese, and eggs, they also purchase bread.

The first association tells us that a customer who purchases milk is also likely to purchase bread. The obvious question is, “How likely will the event of a milk purchase lead to a bread purchase?” To answer this, each association rule has an associated confidence. For this rule, confidence is the conditional probability of a bread purchase given a milk purchase. Therefore, if a total of 10,000 customer transactions involve the purchase of milk, and 5000 of those same transactions also contain a bread purchase, the confidence of a bread purchase given a milk purchase is $5000/10,000 = 50\%$.

Now consider the second rule. Does this rule give us the same information as the first rule? The answer is no! With the first rule, the transaction domain consisted of all customers who had made a milk purchase. For this rule, the domain is the set of all customer transactions that show the purchase of a bread item. As an example, suppose we have a total of 20,000 customer transactions involving a bread purchase and of these, 5000 also involve a milk purchase. This gives us a confidence value for a milk purchase given a bread purchase of 25% versus 50% for the first rule.

Although the third and fourth rules are more complex, the idea is the same. The confidence for the third rule tells us the likelihood of a purchase of both cheese and bread given a purchase of milk and eggs. The confidence for the fourth rule tells us the likelihood of a bread purchase given the purchase of milk, cheese, and eggs.

One important piece of information that a rule confidence value does not offer is the percent of all transactions containing the attribute values found in an association rule. This statistic is known as the *support* for a rule. Support is simply the minimum percentage of instances (transactions) in the database that contain all items listed in a specific association rule. In the next section, you will see how item sets use support to set limits on the total number of association rules for a given data set.

3.3.2 Mining Association Rules: An Example

When several attributes are present, association rule generation becomes unreasonable because of the large number of possible conditions for the consequent of each rule. Special algorithms have been developed to generate association rules efficiently. One such algorithm is the Apriori algorithm (Agrawal et al., 1993). This algorithm generates what are known as *item sets*. Item sets are attribute-value combinations that meet a specified coverage requirement. Those attribute-value combinations that do not meet the coverage requirement are discarded. This allows the rule generation process to be completed in a reasonable amount of time.

Apriori association rule generation using item sets is a two-step process. The first step is item set generation. The second step uses the generated item sets to create a set of association rules. We illustrate the idea with the subset of the credit card promotion database shown in Table 3.3. The *income range* and *age* attributes have been eliminated.

To begin, we set the minimum attribute-value coverage requirement at four items. The first item set table created contains single-item sets. Single-item sets represent individual attribute-value combinations extracted from the original data set. We first consider the attribute *magazine promotion*. Upon examining the table values for *magazine promotion*, we see that seven instances have a value of *yes* and three instances contain the value *no*. The coverage for *magazine promotion = yes* exceeds the minimum coverage of four and therefore represents a valid item set to be added to the single-item set table. As *magazine promotion = no* with coverage of three does not meet the coverage requirement, *magazine promotion = no* is not added to the single-item set table. Table 3.4 shows all single-item set values from Table 3.3 that meet the minimum coverage requirement.

We now combine single-item sets to create two-item sets with the same coverage restriction. We need only consider attribute-value combinations derived from the single-item set table. Let's start with *magazine promotion = yes* and *watch promotion = no*. Four instances satisfy this combination; therefore, this will be our first entry in the two-item set table (Table 3.5). We then consider *magazine promotion = yes* and *life insurance promotion = yes*. As there are five instance matches, we add this combination to the two-item set table. We

TABLE 3.3 A Subset of the Credit Card Promotion Database

Magazine Promotion	Watch Promotion	Life Insurance Promotion	Credit Card Insurance	Gender
Yes	No	No	No	Male
Yes	Yes	Yes	No	Female
No	No	No	No	Male
Yes	Yes	Yes	Yes	Male
Yes	No	Yes	No	Female
No	No	No	No	Female
Yes	No	Yes	Yes	Male
No	Yes	No	No	Male
Yes	No	No	No	Male
Yes	Yes	Yes	No	Female

TABLE 3.4 Single-Item Sets

Single-Item Sets	Number of Items
<i>Magazine Promotion = Yes</i>	7
<i>Watch Promotion = Yes</i>	4
<i>Watch Promotion = No</i>	6
<i>Life Insurance Promotion = Yes</i>	5
<i>Life Insurance Promotion = No</i>	5
<i>Credit Card Insurance = No</i>	8
<i>Gender = Male</i>	6
<i>Gender = Female</i>	4

TABLE 3.5 Two-Item Sets

Two-Item Sets	Number of Items
<i>Magazine Promotion = Yes and Watch Promotion = No</i>	4
<i>Magazine Promotion = Yes and Life Insurance Promotion = Yes</i>	5
<i>Magazine Promotion = Yes and Credit Card Insurance = No</i>	5
<i>Magazine Promotion = Yes and Gender = Male</i>	4
<i>Watch Promotion = No and Life Insurance Promotion = No</i>	4
<i>Watch Promotion = No and Credit Card Insurance = No</i>	5
<i>Watch Promotion = No and Gender = Male</i>	4
<i>Life Insurance Promotion = No and Credit Card Insurance = No</i>	5
<i>Life Insurance Promotion = No and Gender = Male</i>	4
<i>Credit Card Insurance = No and Gender = Male</i>	4
<i>Credit Card Insurance = No and Gender = Female</i>	4

now try *magazine promotion = yes* and *life insurance promotion = no*. As there are only two matches, this combination is not a valid two-item set entry. Continuing this process results in a total of 11 two-item set table entries.

The next step is to use the attribute-value combinations from the two-item set table to generate three-item sets. Reading from the top of the two-item set table, our first possibility is

Magazine Promotion = Yes and Watch Promotion = No and Life Insurance Promotion = Yes.

As only one instance satisfies the three values, we do not add this combination to the three-item set table. However, two three-item sets do satisfy the coverage criterion:

Watch Promotion = No and Life Insurance Promotion = No and Credit Card Insurance = No
Life Insurance Promotion = No and Credit Card Insurance = No and Gender = Male

As there are no additional member set possibilities, the process proceeds from generating item sets to creating association rules. The first step in rule creation is to specify a minimum rule confidence. Next, association rules are generated from the two- and three-item set tables. Finally, any rule not meeting the minimum confidence value is discarded.

Two possible two-item set rules are as follows:

IF *Magazine Promotion* = Yes
 THEN *Life Insurance Promotion* = Yes (5/7)
 IF *Life Insurance Promotion* = Yes
 THEN *Magazine Promotion* = Yes (5/5)

The fractions at the right indicate the rule accuracy (confidence). For the first rule, there are five instances where magazine promotion and life insurance promotion are both *yes*. There are seven total instances where *magazine promotion* = *yes*. Therefore, in two situations, the rule will be in error when predicting a life insurance promotion value of *yes* when *magazine promotion* = *yes*. If our minimum confidence setting is 80%, this first rule will be eliminated from the final rule set. The second rule states that *magazine promotion* = *yes* any time *life insurance promotion* = *yes*. The rule confidence is 100%. Therefore, the rule becomes part of the final output of the association rule generator. Here are three of several possible three-item set rules:

IF *Watch Promotion* = No and *Life Insurance Promotion* = No
 THEN *Credit Card Insurance* = No (4/4)
 IF *Watch Promotion* = No
 THEN *Life Insurance Promotion* = No and *Credit Card Insurance* = No (4/6)
 IF *Credit Card Insurance* = No
 THEN *Watch Promotion* = No and *Life Insurance Promotion* = No (4/8)

Exercises at the end of the chapter ask you to write additional association rules for this data set. We conclude our discussion on association rules with some general considerations.

3.3.3 General Considerations

Association rules are particularly popular because of their ability to find relationships in large databases without having the restriction of choosing a single dependent variable. However, caution must be exercised in the interpretation of association rules since many discovered relationships turn out to be trivial.

As an example, let's suppose we present a total of 10,000 transactions for a market basket analysis. Also, suppose 70% of all transactions involve the purchase of milk and 50% of all transactions have a bread purchase. From this information, we are likely to see an association rule of the following form:

If customers purchase milk, they also purchase bread.

The confidence for this rule may be well above 40%, but note that most customers purchase both products. The rule does not give us additional marketing information telling us that it would be to our advantage to promote the purchase of bread with milk, since most

customers buy these products together anyway. However, there are two types of relationships found within association rules that are of interest:

- We are interested in association rules that show a lift in product sales for a particular product where the lift in sales is the result of its association with one or more other products. In this case, we can use this information to help promote the product with increased sales as a result of the association.
- We are also interested in association rules that show a lower-than-expected confidence for a particular association. In this case, a possible conclusion is that the products listed in the association rule compete for the same market.

As a final point, huge volumes of data are often stored for market basket analysis. Therefore, it is important to minimize the work required by an association rule generator. A good scenario is to specify an initially high value for the item set coverage criterion. If more rules are desired, the coverage criterion can be lowered and the entire process repeated.

3.4 THE K-MEANS ALGORITHM

The *K*-means algorithm (Lloyd, 1982) is a simple yet effective statistical clustering technique for numeric data. To help you better understand unsupervised clustering, let's see how the *K*-means algorithm partitions a set of data into disjoint clusters.

Here is the algorithm:

-
1. Choose a value for K , the total number of clusters to be determined.
 2. Choose K instances (data points) within the data set at random. These are the initial cluster centers.
 3. Use simple Euclidean distance to assign the remaining instances to their closest cluster center.
 4. Use the instances in each cluster to calculate a new mean for each cluster.
 5. If the new mean values are identical to the mean values of the previous iteration, the process terminates. Otherwise, use the new means as cluster centers and repeat steps 3–5.
-

The first step of the algorithm requires an initial decision about how many clusters we believe to be present in the data. Next, the algorithm randomly selects K data points as initial cluster centers. Each instance is then placed in the cluster to which it is most similar. Similarity can be defined in many ways; however, the similarity measure most often used is simple Euclidean distance.

Once all instances have been placed in their appropriate cluster, the cluster centers are updated by computing the mean of each new cluster. The process of instance classification and cluster center computation continues until an iteration of the algorithm shows no change in the cluster centers, that is, until no instances change cluster assignments in step 3.

3.4.1 An Example Using K-means

To clarify the process, let's work through a partial example containing two numerical attributes. Although most real data sets contain several attributes, the methodology remains the same regardless of the number of attributes. For our example, we will use the six instances shown in Table 3.6. For simplicity, we name the two attributes x and y , respectively, and map the instances onto an x - y coordinate system. The mapping is shown in Figure 3.10.

As the first step, we must choose a value for K . Let's assume we suspect two distinct clusters. Therefore, we set the value for K at 2. The algorithm chooses two points at random to represent initial cluster centers. Suppose the algorithm selects instance 1 as one cluster center and instance 3 as the second cluster center. The next step is to classify the remaining instances.

Recall the formula for computing the Euclidean distance between point A with coordinates (x_1, y_1) and point B with coordinates (x_2, y_2) :

$$\text{Distance}(A - B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Computations for the first iteration of the algorithm with $C_1 = (1.0, 1.5)$ and $C_2 = (2.0, 1.5)$ are as follows where $C_i - j$ is the Euclidean distance from point C_i to the point represented by instance j in Table 3.6.

$$\text{Distance } (C_1 - 1) = 0.00 \quad \text{Distance } (C_2 - 1) = 1.00$$

$$\text{Distance } (C_1 - 2) = 3.00 \quad \text{Distance } (C_2 - 2) \cong 3.16$$

TABLE 3.6 K-means Input Values

Instance	x	y
1	1.0	1.5
2	1.0	4.5
3	2.0	1.5
4	2.0	3.5
5	3.0	2.5
6	5.0	6.0

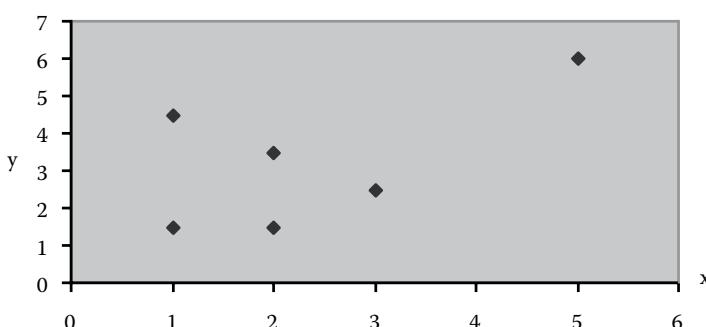


FIGURE 3.10 A coordinate mapping of the data in Table 3.6.

Distance ($C_1 - 3$) = 1.00 Distance ($C_2 - 3$) = 0.00

Distance ($C_1 - 4$) \cong 2.24 Distance ($C_2 - 4$) = 2.00

Distance ($C_1 - 5$) \cong 2.24 Distance ($C_2 - 5$) \cong 1.41

Distance ($C_1 - 6$) \cong 6.02 Distance ($C_2 - 6$) \cong 5.41

After the first iteration of the algorithm, we have the following clustering:

C_1 contains instances 1 and 2.

C_2 contains instances 3, 4, 5, and 6.

The next step is to recompute each cluster center.

For cluster C_1 ,

$$x = (1.0 + 1.0)/2 = 1.0$$

$$y = (1.5 + 4.5)/2 = 3.0$$

For cluster C_2 ,

$$x = (2.0 + 2.0 + 3.0 + 5.0)/4.0 = 3.0$$

$$y = (1.5 + 3.5 + 2.5 + 6.0)/4.0 = 3.375$$

Thus, the new cluster centers are $C_1 = (1.0, 3.0)$ and $C_2 = (3.0, 3.375)$. As the cluster centers have changed, the algorithm must perform a second iteration.

Computations for the second iteration are

Distance ($C_1 - 1$) = 1.50 Distance ($C_2 - 1$) \cong 2.74

Distance ($C_1 - 2$) = 1.50 Distance ($C_2 - 2$) \cong 2.29

Distance ($C_1 - 3$) \cong 1.80 Distance ($C_2 - 3$) = 2.125

Distance ($C_1 - 4$) \cong 1.12 Distance ($C_2 - 4$) \cong 1.01

Distance ($C_1 - 5$) \cong 2.06 Distance ($C_2 - 5$) = 0.875

Distance ($C_1 - 6$) = 5.00 Distance ($C_2 - 6$) \cong 3.30

The second iteration results in a modified clustering:

C_1 contains instances 1, 2, and 3.

C_2 contains instances 4, 5, and 6.

Next, we compute the new centers for each cluster.

For cluster C_1 ,

$$x = (1.0 + 1.0 + 2.0)/3.0 \cong 1.33$$

$$y = (1.5 + 4.5 + 1.5)/3.0 = 2.50$$

For cluster C_2 ,

$$x = (2.0 + 3.0 + 5.0)/3.0 \cong 3.33$$

$$y = (3.5 + 2.5 + 6.0)/3.0 = 4.00$$

Once again, this iteration shows a change in the cluster centers. Therefore, the process continues to a third iteration with $C_1 = (1.33, 2.50)$ and $C_2 = (3.33, 4.00)$. We leave the computations for the third iteration as an exercise.

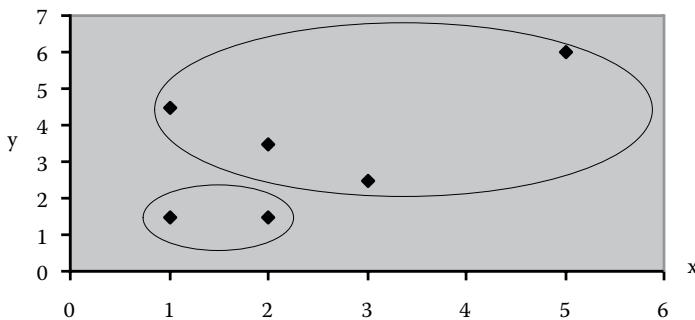
These computations have little meaning other than to demonstrate the workings of the algorithm. In fact, we may see a different final cluster configuration for each alternative choice of the initial cluster centers. Unfortunately, this is a general problem seen with the K -means algorithm. That is, although the algorithm is guaranteed to cluster the instances into a stable state, the stabilization is not guaranteed to be optimal.

An optimal clustering for the K -means algorithm is frequently defined as a clustering that shows a minimum summation of squared error differences between the instances and their corresponding cluster center. Finding a globally optimal clustering for a given value of K is nearly impossible as we must repeat the algorithm with alternative choices for the initial cluster centers. For even a few hundred data instances, it is not practical to run the K -means algorithm more than a few times. Instead, the usual practice is to choose a terminating criterion, such as a maximum acceptable squared error value, and execute the K -means algorithm until we achieve a result that satisfies the termination condition.

Table 3.7 shows three clusterings resulting from repeated application of the K -means algorithm to the data in Table 3.6. Figure 3.11 displays the most frequently occurring clustering. This clustering is shown as outcome 2 in Table 3.7. Notice that the best clustering, as determined by a minimum squared error value, is outcome 3, where the single instance with coordinates (5,6) forms its own cluster and the remaining instances shape the second cluster.

TABLE 3.7 Several Applications of the K -means Algorithm ($K = 2$)

Outcome	Cluster Centers	Cluster Points	Squared Error
1	(2.67,4.67)	2, 4, 6	14.50
	(2.00,1.83)	1, 3, 5	
2	(1.5,1.5)	1, 3	15.94
	(2.75,4.125)	2, 4, 5, 6	
3	(1.8,2.7)	1, 2, 3, 4, 5	9.60
	(5,6)	6	

FIGURE 3.11 A K -means clustering of the data in Table 3.6 ($K = 2$).

3.4.2 General Considerations

The K -means method is easy to understand and implement. However, there are several issues to consider. Specifically,

- The algorithm only works with real-valued data. If we have a categorical attribute in our data set, we must either discard the attribute or convert the attribute values to numerical equivalents. A common approach is to create one numeric attribute for each categorical attribute value. For example, consider the attribute *color* with values *red*, *blue*, and *green*. The *color* attribute is transformed into three numeric attributes: *color_red*, *color_blue*, and *color_green*. If the color attribute for instance I is blue, the conversion shows I with *color_red* = 0, *color_blue* = 1, and *color_green* = 0. That is, all numeric attributes are assigned a value of 0 excepting the numeric attribute representing the original color value. We address additional issues surrounding data conversion in Chapter 5.
- We are required to select a value for the number of clusters to be formed. This is an obvious problem if we make a poor choice. One way to cope with this issue is to run the algorithm several times with alternative values for K . In this way, we are more likely to get a “feel” for how many clusters may be present in the data.
- The K -means algorithm works best when the clusters that exist in the data are of approximately equal size. This being the case, if an optimal solution is represented by clusters of unequal size, the K -means algorithm is not likely to find a best solution.

- There is no way to tell which attributes are significant in determining the formed clusters. For this reason, several irrelevant attributes can cause less-than-optimal results.
- A lack of explanation about the nature of the formed clusters leaves us responsible for much of the interpretation about what has been found. However, as you will see in later chapters, we can use supervised data mining tools to help us gain insight into the nature of the clusters formed by unsupervised clustering algorithms.

Despite these limitations, the *K*-means algorithm continues to be a favorite statistical technique. In Chapter 12, you will see a slightly different approach to numerical clustering that borrows several ideas from evolutionary science. Look closely and you will see similarities between the *K*-means algorithm and the genetic learning approach to unsupervised clustering.

3.5 GENETIC LEARNING

Genetic algorithms apply an evolutionary approach to inductive learning. The genetic algorithm learning model was initially developed by John Holland in 1986 and is based on the Darwinian principle of natural selection. Genetic programming has been successfully applied to problems that are difficult to solve using conventional techniques. Common areas of application include scheduling problems, such as the traveling salesperson problem, network routing problems for circuit-switched networks, and problems in the area of financial marketing.

Genetic algorithms can be developed for supervised and unsupervised data mining. Here we present a basic genetic learning algorithm.

-
1. Initialize a population P of n elements, often referred to as chromosomes, as a potential solution.
 2. Until a specified termination condition is satisfied,
 - a. Use a fitness function to evaluate each element of the current solution. If an element passes the fitness criteria, it remains in P ; otherwise, it is discarded from the population.
 - b. The population now contains m elements ($m <= n$). Use genetic operators to create $(n - m)$ new elements. Add the new elements to the population to return the population size to n .
-

For data mining, we think of elements as instances defined by attributes and values. A common technique for representing genetic knowledge is to transform elements into binary strings. To illustrate, consider the attribute *income range* given in Table 3.1. The values for *income range* are 20–30K, 30–40K, 40–50K, and 50–60K. We can represent *income range* as a string of 2 bits by assigning 00 to 20–30K, 01 to 30–40K, 10 to 40–50K, and 11 to 50–60K. For a numeric attribute such as *age*, we have several options. A frequent approach is to map numeric values to discrete interval ranges and apply the same strategy as that used for categorical data.

The most widespread genetic operators are crossover and mutation. *Crossover* forms new elements for the population by combining parts of two elements currently in the population. The elements most often used for crossover are those destined to be eliminated from the population. *Mutation*, a second genetic operator, is sparingly applied to elements chosen for elimination. Mutation can be applied by randomly flipping bits (or attribute values) within a single element. Alternatively, the choice of whether to flip individual bits can be probability based. *Selection* is a third genetic operator that is sometimes used. With selection, the elements deleted from the population are replaced by copies of elements that pass the fitness test with high scores. In this way, the overall fitness of the population is guaranteed to increase. As selection supports specialization rather than generalization, it should be used with caution.

3.5.1 Genetic Algorithms and Supervised Learning

Figure 3.12 shows a plausible model for supervised genetic learning. Countless variations of the generalized model exist. However, the model suffices to help us explain the genetic learning process. Let's associate this model with the genetic algorithm defined earlier to obtain a better understanding of supervised genetic learning.

Step 1 of the algorithm initializes a population P of elements. The P referred to in the algorithm is shown in Figure 3.12 as the box labeled *population elements*. Figure 3.12 shows that the process of supervised genetic learning iteratively modifies the elements of the population. The algorithm tells us that this process continues until a termination condition is satisfied. The termination condition might be that all elements of the population meet some minimum criteria. As an alternative, the condition can be a fixed number of iterations of the learning process.

Step 2a of the algorithm applies a fitness function to evaluate each element currently in the population. Figure 3.12 shows us that the fitness function uses a set of training data to help with the evaluation. With each iteration of the algorithm, elements not satisfying the fitness criteria are eliminated from the population. The final result of a supervised genetic learning session is a set of population elements that best represent the training data.

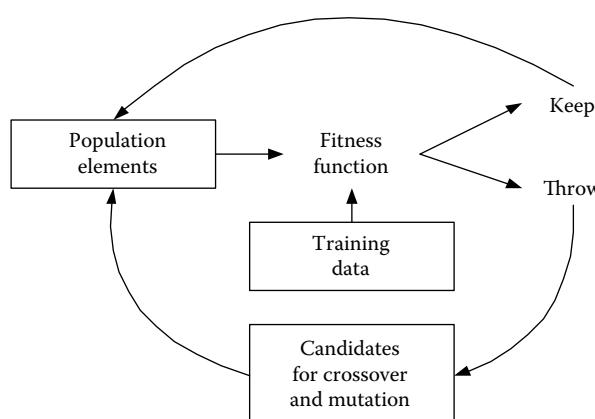


FIGURE 3.12 Supervised genetic learning.

TABLE 3.8 An Initial Population for Supervised Genetic Learning

Population Element	Income Range	Life Insurance Promotion	Credit Card Insurance	Gender	Age
1	20–30K	No	Yes	Male	30–39
2	30–40K	Yes	No	Female	50–59
3	?	No	No	Male	40–49
4	30–40K	Yes	Yes	Male	40–49

Step 2b of the algorithm adds new elements to the population to replace any elements eliminated in step 2a. Figure 3.12 indicates that the new elements are formed from previously deleted elements by applying crossover and mutation. Although the selection operator is not specifically shown in Figure 3.12, it can be an option. If selection is used, some elements with high fitness scores will be reproduced. In any case, a usual scenario holds the number of elements at a constant value. For example, if the population consists of 10 elements and 4 of these elements fail the fitness test, 4 new elements will be added to the population using some combination of crossover, mutation, and selection.

Now that we have a better understanding of what genetic learning is all about, let's look at a specific example that uses a subset of the credit card promotion database. Once again, our goal is to create a model able to differentiate individuals who have accepted the life insurance promotion from those who have not. We must first generate an initial population for our model. As part of this selection process, it is our job to decide how many elements will be part of the population. For simplicity, let's suppose an initial population is represented by the four elements shown in Table 3.8. We require that after each iteration of the algorithm, exactly two elements from each class (*life insurance promotion = yes* and *life insurance promotion = no*) remain in the population.

Notice we have changed attribute values for *age* to discrete categories. This is to remove additional complexity from our explanation. Also, note the question mark as the value for *income range* in the third row of Table 3.8. A question mark is a “don't care” condition, meaning the attribute is not important to the learning process. This is one way to promote attribute significance within the population.

As stated in the algorithm, genetic training involves repeated modification of the population by applying a fitness function to each element. To implement the fitness function, we compare each population element to the six training instances shown in Table 3.9. For a single population element E , we define our fitness function as follows:

-
1. Let N be the number of matches of the input attribute values of E with training instances from its own class.
 2. Let M be the number of input attribute-value matches to all training instances from the competing class.
 3. Add 1 to M .
 4. Divide N by M .
-

TABLE 3.9 Training Data for Genetic Learning

Training Instance	Income Range	Life Insurance Promotion	Credit Card Insurance	Gender	Age
1	30–40K	Yes	Yes	Male	30–39
2	30–40K	Yes	No	Female	40–49
3	50–60K	Yes	No	Female	30–39
4	20–30K	No	No	Female	50–59
5	20–30K	No	No	Male	20–29
6	30–40K	No	No	Male	40–49

That is, the fitness score for any population element E is simply the ratio of the number of matches of the attribute values of E to its own class to the number of matches to all training instances from the competing class. One is added to the denominator of each ratio to avoid a possible division by 0. If the fitness value of a particular element does not show a ratio score above a predefined value, it is eliminated from the population and becomes a candidate for crossover or mutation. If all currently selected elements meet or exceed the fitness score, elements are randomly chosen for removal. Alternatively, elements with the lowest fitness scores can be eliminated.

Now that our fitness function is defined, let's compute the fitness score for element 1. Element 1 is a member of the class *life insurance promotion = no*. Therefore, N is computed as the number of matches element 1 has with the members in Table 3.9 having *life insurance promotion = no*. That is, for element 1, we compute N using the following information.

- *Income Range = 20–30K* matches with training instances 4 and 5.
- There are no matches for *Credit Card Insurance = Yes*.
- *Gender = Male* matches with training instances 5 and 6.
- There are no matches for *Age = 30–39*.

Therefore, the value of N computes to 4. Next, we compute M by comparing element 1 with training instances 1, 2, and 3. The computation is as follows:

- There are no matches for *Income Range = 20–30K*.
- *Credit Card Insurance = Yes* matches with training instance 1.
- *Gender = Male* matches with training instance 1.
- *Age = 30–39* matches with training instances 1 and 3.

As there are four matches with the training instances where *life insurance promotion = yes*, the value of M for instance 1 is 4. Next, we add 1 to M , giving us a final fitness score for element 1 of 4/5, or 0.80. The following is the list of fitness function values for the entire

initial population. Any comparison with a question mark is considered a nonmatch. $F(i)$ denotes the fitness value for population element i .

- $F(1) = 4/5 = 0.80$
- $F(2) = 6/7 = 0.86$
- $F(3) = 6/5 = 1.20$
- $F(4) = 5/5 = 1.00$

We now eliminate a subset of the population and use genetic operators to create new elements. To show the process, we remove elements 1 and 2, because each shows the lowest fitness score for its respective class, and then apply genetic operations to these elements. First, we use the crossover operator. The complete crossover operation is shown in Figure 3.13. Many crossover possibilities exist. Let's make the point of crossover after the *life insurance promotion* attribute. Therefore, the attribute values for *income range* and *life insurance promotion* from element 1 are combined with the values for *credit card insurance*, *gender*, and *age* for element 2. A new second element is created in a similar fashion by combining the first part of element 2 with the last part of element 1. The resultant second-generation population is displayed in Table 3.10. Upon computing the fitness scores for these new elements, we find that both fitness scores improve with $F(1) = 7/5$ and $F(2) = 6/4$.

As a second option, suppose the algorithm chooses to mutate the second element rather than perform the second crossover. A mutation will result in a modification of one or more attribute values of the original second element. For example, the 30–40K value for *income range* can be modified by randomly selecting any one of the alternative values for income

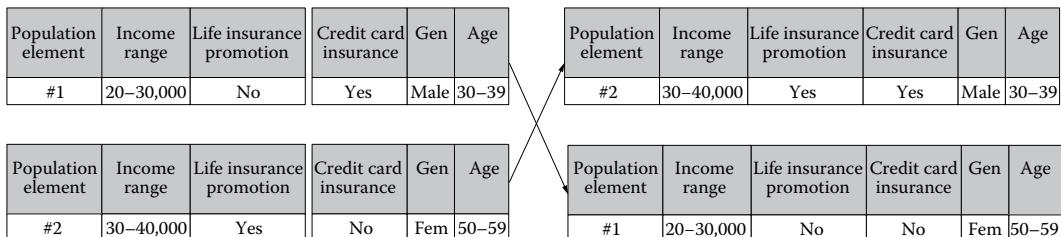


FIGURE 3.13 A crossover operation.

TABLE 3.10 A Second-Generation Population

Population Element	Income Range	Life Insurance Promotion	Credit Card Insurance	Gender	Age
1	20-30K	No	No	Female	50-59
2	30-40K	Yes	Yes	Male	30-39
3	?	No	No	Male	40-49
4	30-40K	Yes	Yes	Male	40-49

range, or the value can be replaced with a question mark. Clearly, a mutation is necessary if an income range value of 50–60K is to be seen in the element population since this particular value does not appear in Figure 3.10. Countless variations of the stated processes for crossover and mutation are limited only by our imagination. The selection operator is not appropriate for this simple example, because the number of elements for each class is limited to two.

Crossover and mutation continue until a termination condition is satisfied. Once the algorithm terminates, we have a model for classifying new instances or predicting future outcome as appropriate. To use the model, we can compare a new unknown instance with the elements of the final population. A simple technique is to give the unknown instance the same classification as the population element to which it is most similar. A second possibility is for the model to keep track of the m elements from P most similar to the instance presented for classification. The algorithm then randomly chooses one of the m elements and gives the unknown instance the classification of the randomly selected element.

3.5.2 General Considerations

Genetic algorithms continue to gain popularity, and several commercial data mining products now contain a genetic learning component. Here is a list of considerations when using a problem-solving approach based on genetic learning:

- Genetic algorithms are designed to find globally optimized solutions. However, there is no guarantee that any given solution is not the result of a local rather than a global optimization.
- The fitness function determines the computational complexity of a genetic algorithm. A fitness function involving several calculations can be computationally expensive.
- Genetic algorithms explain their results to the extent that the fitness function is understandable.
- Transforming the data to a form suitable for a genetic algorithm can be a challenge.

In addition to supervised learning, Chapter 12 demonstrates how genetic learning can be used for unsupervised clustering. Furthermore, genetic techniques can be used in conjunction with other learning techniques. For example, genetic algorithms can be applied to train feed-forward neural networks. Each chromosome or element in a population represents one possibility for the weights contained in the network. As genetic operators are then applied to the various elements, the connection weights of the network change. As a second example, in Chapter 5, you will see how genetic learning can be applied to help select a best set of input attributes for supervised learning.

3.6 CHOOSING A DATA MINING TECHNIQUE

It is apparent that when it comes to solving a particular problem, we have several techniques to choose from. The question now becomes, “How do we know which data mining

technique to use?" Let's examine this issue in more detail. A formal statement of the problem reads as follows:

Given a set of data containing attributes and values to be mined together with information about the nature of the data and the problem to be solved, determine an appropriate data mining technique.

The following questions may be useful in determining which techniques to apply:

- Is learning supervised or unsupervised?
- Do we require a clear explanation about the relationships present in the data?
- Is there one set of input attributes and one set of output attributes, or can attributes interact with one another in several ways?
- Is the data categorical, numeric, or a combination of both?
- If learning is supervised, is there one output attribute, or are there several output attributes? Are the output attribute(s) categorical or numeric?

For a particular problem, these questions have obvious answers. For example, we know that neural networks and regression models are black-box structures. Therefore, these techniques are poor choices if an explanation about what has been learned is required. Also, association rules are usually a best choice when attributes are allowed to play multiple roles in the data mining process.

If the answers to these questions still leave us with several choices, we can try turning to the research community for direction. Several studies offer generalizations about the behavior of different data mining techniques (Quinlan, 1994; Shavlik et al., 1990). Unfortunately, many of these studies give conflicting results. Still, answers to several additional questions may provide some guidance. Specifically,

1. Do we know the distribution of the data?

Data sets containing more than a few hundred instances can be a problem for data mining techniques that require the data to conform to certain standards. For example, many statistical techniques assume the data to be normally distributed.

2. Do we know which attributes best define the data to be modeled?

Decision trees and certain statistical approaches can determine those attributes most predictive of class membership. Neural network, nearest neighbor, and various clustering approaches assume attributes to be of equal importance. This is a problem when several attributes not predictive of class membership are present in the data.

3. Do the data contain several missing values?

Most data mining researchers agree that, if applicable, neural networks tend to outperform other models when a wealth of noisy data are present.

4. Is time an issue?

Algorithms for building decision trees and production rules typically execute much faster than neural network or genetic learning approaches.

5. Which technique is most likely to give a best classification accuracy?

The last question is of particular interest, as a best accuracy is always a desireable feature of any model. As previous research has shown different levels of performance among data mining techniques, it seems reasonable to assume that a maximum classification accuracy is obtainable using an approach where several models built with the same training data vote on the classification of new unknown instances. An instance is then given the classification chosen by the majority of the competing models.

Unfortunately, our experience with a variety of data mining tools and several data sets, both large and small, has consistently shown a high degree of overlap among individual instance misclassifications for competing classifier models. Because of this, multiple-model approaches making use of several data mining techniques are not likely to improve the results seen with an individual model. However, multiple-model approaches that apply the same data mining technique for building each model have met with some success (Maclin and Opitz, 1997). We discuss multiple-model approaches of this type in Chapter 13. Also, in Appendix A, we show you how a rule-based expert system approach using some of the ideas presented here can be developed to help determine a best data mining technique.

3.7 CHAPTER SUMMARY

Decision trees are probably the most popular structure for supervised data mining. A common algorithm for building a decision tree selects a subset of instances from the training data to construct an initial tree. The remaining training instances are then used to test the accuracy of the tree. If any instance is incorrectly classified, the instance is added to the current set of training data, and the process is repeated. A main goal is to minimize the number of tree levels and tree nodes, thereby maximizing data generalization. Decision trees have been successfully applied to real problems, are easy to understand, and map nicely to a set of production rules. Covering rule generators attempt to create a set of rules that maximize the total number of covered within-class instances while at the same time minimizing the number of covered nonclass instances. Association rules are able to find relationships in large databases. Association rules are unlike traditional production rules in that an attribute that is a precondition in one rule may appear as a consequent in another rule. Also, association rule generators allow the consequent of a rule to contain one or several attribute values. As association rules are more complex, special techniques have been developed to generate association rules efficiently. Rule confidence and support

help determine which discovered associations are likely to be interesting from a marketing perspective. However, caution must be exercised in the interpretation of association rules because many discovered relationships turn out to be trivial.

The *K*-means algorithm is a statistical unsupervised clustering technique. All input attributes to the algorithm must be numeric, and the user is required to make a decision about how many clusters are to be discovered. The algorithm begins by randomly choosing one data point to represent each cluster. Each data instance is then placed in the cluster to which it is most similar. New cluster centers are computed, and the process continues until the cluster centers do not change. The *K*-means algorithm is easy to implement and understand. However, the algorithm is not guaranteed to converge to a globally optimal solution, lacks the ability to explain what has been found, and is unable to tell which attributes are significant in determining the formed clusters. Despite these limitations, the *K*-means algorithm is among the most widely used clustering technique.

Genetic algorithms apply the theory of evolution to inductive learning. Genetic learning can be supervised or unsupervised and is typically used for problems that cannot be solved with traditional techniques. A standard genetic approach to learning applies a fitness function to a set of data elements to determine which elements survive from one generation to the next. Those elements not surviving are used to create new instances to replace deleted elements. In addition to being used for supervised learning and unsupervised clustering, genetic techniques can be employed in conjunction with other learning techniques.

Here and in Chapter 2 we have previewed several data mining techniques. Later chapters introduce several additional techniques for you to study. Several factors, including the nature of the data, the role of individual attributes, and whether learning is supervised or unsupervised, help us determine the data mining technique most appropriate for a particular problem.

3.8 KEY TERMS

- *Affinity analysis*. The process of determining which things are typically grouped together.
- *Confidence*. Given a rule of the form “if A then B,” confidence is defined as the conditional probability that B is true when A is known to be true.
- *Crossover*. A genetic learning operation that creates new population elements by combining parts of two or more elements from the current population.
- *Genetic algorithm*. A data mining technique based on the theory of evolution.
- *Item set*. A set of several combinations of attribute-value pairs that cover a prespecified minimum number of data instances.
- *Mutation*. A genetic learning operation that creates a new population element by randomly modifying a portion of an existing element.

- *Regression tree.* A decision tree where the leaf nodes of the tree are numerical rather than categorical values.
- *Selection.* A genetic learning operation that adds copies of current population elements with high fitness scores to the next generation of the population.
- *Support.* The minimum percentage of instances in the database that contain all items listed in a given association rule.
- *Unstable algorithm.* A data mining algorithm that is highly sensitive to small changes in the training data. As a result, the models built using the algorithm show significant changes in structure when slight changes are made to the training data.

EXERCISES

Review Questions

1. A decision tree built using the algorithm described in Section 3.1 allows us to choose a categorical attribute for data division only once. However, the same numeric attribute may appear several times in a decision tree. Explain how this is true.
2. Use an example to explain in simple terms the difference between attribute-value predictiveness and attribute-value predictability.
3. How do association rules differ from traditional production rules? How are they the same?
4. What is the *usual* definition of an optimal clustering for the K -means algorithm?
5. What do you suppose happens when selection is the only operator allowed for supervised genetic learning?
6. One of the problems with the K -means algorithm is its inability to explain what has been found. Describe how you could apply a decision tree algorithm to a set of clusters formed by the K -means algorithm to better determine the meaning of the formed clusters.

Computational Questions

1. Use the training data in Table 3.1 to construct a confusion matrix for the decision tree shown in Figure 3.5.
2. Answer the following.
 - a. Write the production rules for the decision tree shown in Figure 3.4.
 - b. Can you simplify any of your rules without compromising rule accuracy?
 - c. Repeat (a) and (b) for the decision tree shown in Figure 3.5.
3. Using the attribute *age*, draw an expanded version of the decision tree in Figure 3.4 that correctly classifies all training data instances.

4. In Section 3.1.1 (“An Algorithm for Building Decision Trees”), we defined a simple measure for decision tree attribute selection. Use our defined measure to compute a goodness score for using the attribute *gender* as the top-level node of a decision tree to be built from the data in Table 3.1.
5. Use the measure described in Section 3.1.1 (“An Algorithm for Building Decision Trees”) to compute a goodness score for choosing *credit card insurance* as the node to follow the branch *income range = 20–30K* in Figure 3.5.
6. Use the data in Table 3.3 to give confidence and support values for the following association rule.

IF Gender = Male and Magazine Promotion = Yes

THEN Life Insurance Promotion = Yes

7. Use the information in Table 3.5 to list three two-item set rules. Use the data in Table 3.3 to compute confidence and support values for each of your rules.
8. List three rules for the following three-item set. Use the data in Table 3.3 to specify the confidence and support for each rule.

Watch Promotion = No and Life Insurance Promotion = No and Credit Card Insurance = No

9. Perform the third iteration of the *K*-means algorithm for the example given in Section 3.4.1 (“An Example Using *K*-means”). What are the new cluster centers?
10. Use crossover and mutation as appropriate to compute third-generation elements for the population shown in Table 3.10. Compute fitness scores for the new population.
11. Use the procedure outlined in Section 3.2 together with Table 3.1 and Figure 3.8 to create a best set of covering rules for the class *life insurance promotion = no*.
12. Concept class C_1 shows the following information for the categorical attribute *color*. Use this information and the information in the table to answer the following questions:

Name	Value	Frequency	Predictability	Predictiveness
Color	Red	30	0.4	
	Black	20		1.0

- a. What percent of the instances in class C have a value of *black* for the *color* attribute?
- b. Suppose that exactly one other concept class, C_2 , exists. In addition, assume all domain instances have a color value of either *red* or *black*. Given the information in the table, can you determine the predictability score of *color = red* for class C_2 ? If your answer is yes, what is the predictability value?

- c. Using the same assumption as in part b, can you determine the predictiveness score for $color = red$ for class C_2 ? If your answer is yes, what is the predictiveness score?
 - d. Once again, use the assumption stated in part b. How many instances reside in class C_2 ?
13. Develop a table of instances where the output attribute is one of the data mining techniques discussed in this chapter and the input attributes are several of the characteristics described in Section 3.6. For example, one input attribute to consider is *type of learning* with value *supervised* or *unsupervised*. Another is *explanation required* with value *yes* or *no*. Once you have your table defined, use the decision tree algorithm described in Section 3.1 to build a decision tree able to determine which data mining technique to use.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

II

Tools for Knowledge Discovery



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Weka—An Environment for Knowledge Discovery

CHAPTER OBJECTIVES

- *Understand Weka's Explorer interface*
- *Know how to preprocess data with Weka*
- *Know how to use Weka for building supervised learner models*
- *Know how to perform unsupervised clustering with Weka*
- *Know how to use Weka's cost/benefit analysis tool*
- *Know how to use Weka to create association rules*

In this chapter, we introduce Weka, a multipurpose tool for data mining and analysis. Weka, shorthand for *Waikato Environment for Knowledge Analysis*, was developed at the University of Waikato in New Zealand. Weka is written in the Java programming language and is publicly available under the terms of the GNU General Public License.

Weka contains a wealth of data mining tools including classifiers, clustering algorithms, association rule algorithms, and functions. Weka has nice graphical features with pre-processing and visualization capabilities. Some of the more popular Weka tools include decision tree and production rule algorithms, association rule mining, Bayes classifier, regression analysis, and several clustering algorithms. Useful information including a link to obtain help through the Weka user community forum is available at <http://www.cs.waikato.ac.nz/ml/weka/>.

Weka offers four graphical user interfaces (GUI's). Here we concentrate on Weka's *Explorer* interface as it provides the fastest way to get you started mining your own data. The Explorer allows you to apply all of the algorithms available with Weka. The main disadvantage is that the Explorer interface requires all data to be in the main memory, thereby

limiting data set size. However, this limitation does not inhibit our ability to learn how to apply various data mining algorithms to solve interesting problems.

Weka's *Experimenter* interface gives us the capability of automating the process of testing various classifiers and preprocessing methods as well as collecting statistics with various parameter settings. It is worth noting that any process automated with the Experimenter can be performed interactively within the Explorer environment.

Weka's *Knowledge Flow* interface offers the capability of mining large-sized data. The Knowledge Flow interface represents data sources and learner algorithms as boxes that can be configured and connected. We look closely at this type of interface in Chapter 5 when we examine RapidMiner. Finally, the *Weka Workbench* provides one location for accessing all three interfaces.

Section 4.1 provides the basic information we need to get started with Weka. In Section 4.2, we focus on building decision trees with Weka's implementation of C4.5. Section 4.3 uses Weka's PART rule generator to show how to save and thereby use data mining models multiple times. In Section 4.4, the emphasis is on preprocessing. Section 4.5 is devoted to generating association rules. Section 4.6 highlights Weka's cost/benefit analysis tool that helps when misclassification costs vary. This section is optional as the material is a bit challenging. Covering this material at a later time is a viable option. The topic of Section 4.7 is unsupervised clustering with Weka's implementation of the *K*-means algorithm.

4.1 GETTING STARTED WITH WEKA

The latest version of Weka can be downloaded for free by clicking the download link at the website <http://www.cs.waikato.ac.nz/ml/weka/>.

We suggest that you download and install the latest stable version of the software appropriate for your specific machine—currently, for a Windows x64 machine with Java installed, the download file is `weka-3-8-0-x64.exe`; for OS X, the download file is `weka-3-8-0-oracle-jvm.dmg`. Further, we encourage you to download and open the MS PowerPoint file `Screens_04.ppt` as it contains screenshots clearly showing what you will see on your display as you work through the tutorials.

Once installation is complete, the WEKA GUI Chooser shown in Figure 4.1 together with a message box notifying you about Weka's package manager will appear. The package manager is used to install algorithms not part of Weka's initial installation package. For now, close the message box and click on the Explorer option to see the Explorer GUI shown in Figure 4.2. The arrow labeled 0 tells us that we are in preprocess mode. This mode allows us to load and preprocess data to be mined. Let's load some data and create a simple decision tree.

To get started, we need a data set formatted for the Weka environment. Several sample data sets come with your Weka installation. These data sets are structured in Weka's standard Attribute-Relation File Format (ARFF—described later) and reside in the *data* subfolder within the install folder. Locate and open the install folder—currently `Weka-3-8`—on your machine. Figure 4.3 gives a partial list of and subfolders contained within the install folder. Notice that Weka's reference manual titled *WekaManual.pdf* is included in this folder.

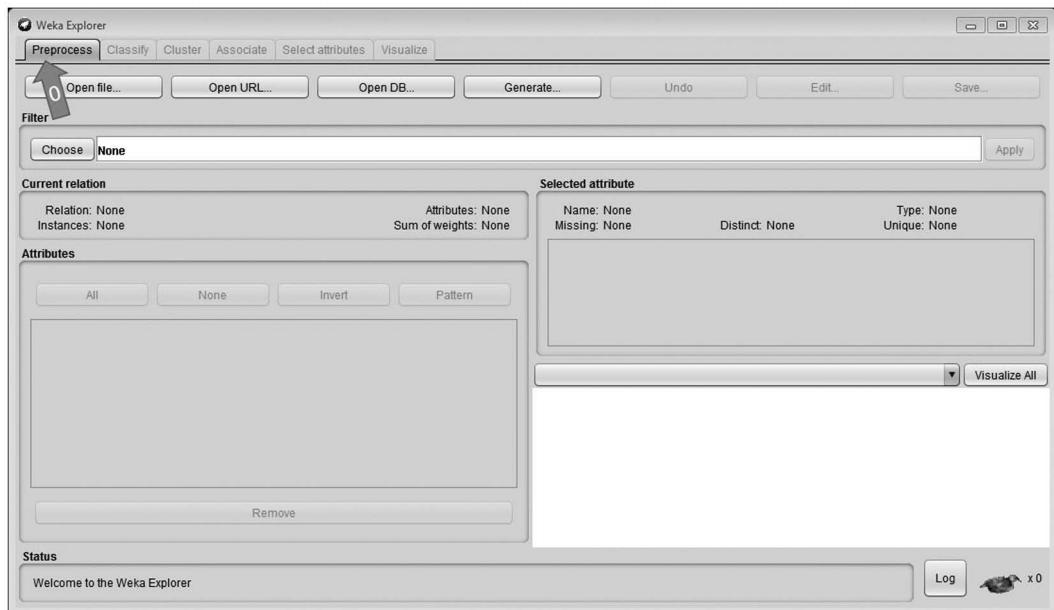
FIGURE 4.1 Weka GUI *Chooser*.

FIGURE 4.2 Explorer four graphical user interfaces (GUI's).

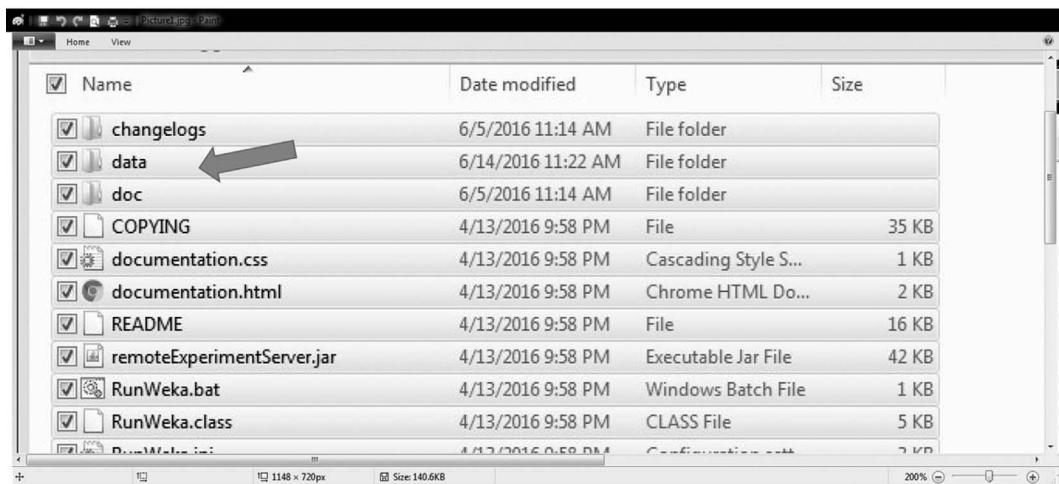


FIGURE 4.3 Weka install folder.

Next, open the *data* subfolder to observe the list displayed in Figure 4.4 of sample data sets. Let's examine the file *contact-lenses.arff*.

Use any standard text or word processor to open the file. Scroll the file until it appears similar to Figure 4.5. All lines beginning with a percent symbol contain documentation about the file. The documentation tells us about the origin and nature of the file.

The *@relation* symbol tells us that the attribute definitions and values follow. Each attribute begins with *@attribute* and is followed by a list enclosed in braces of all possible values for the given attribute. You will notice that all data are categorical. If an attribute is numeric, the attribute is defined simply by its name followed by either the word *numeric* or *real*. The data are designed for supervised learning as there are three input attributes

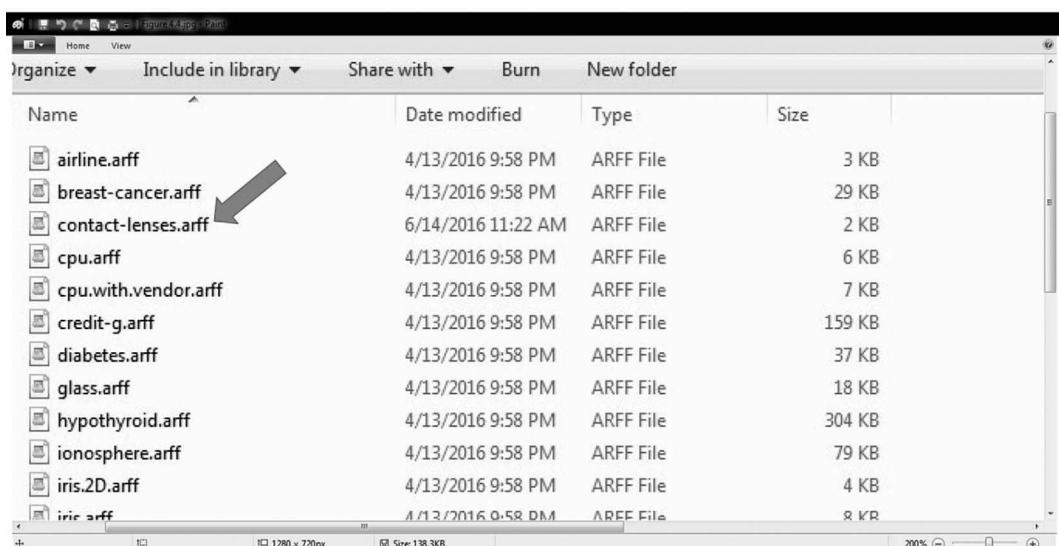
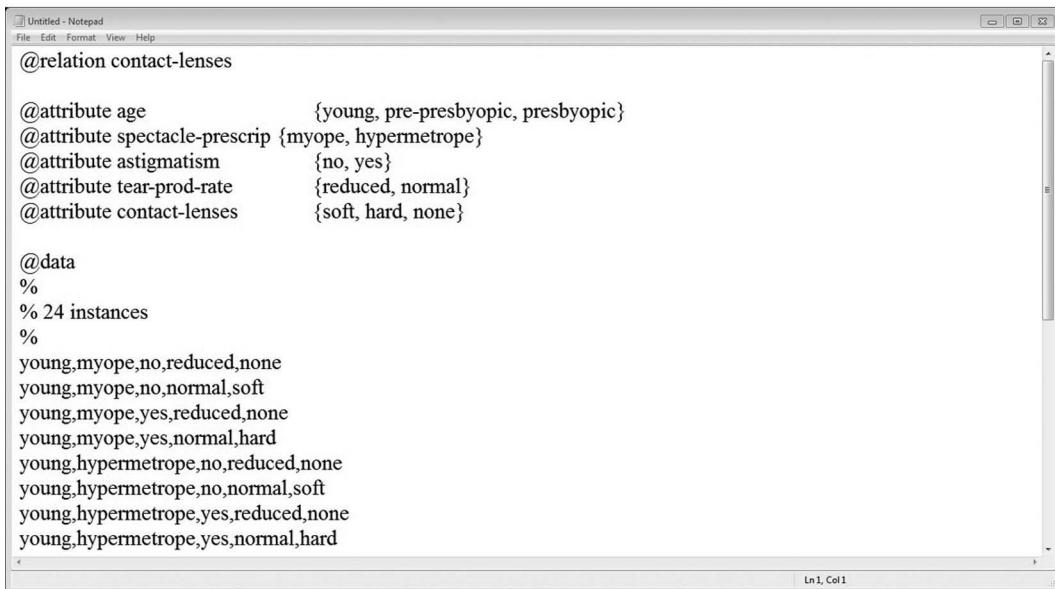


FIGURE 4.4 Sample data sets.



```

@relation contact-lenses

@attribute age {young, pre-presbyopic, presbyopic}
@attribute spectacle-prescrip {myope, hypermetrope}
@attribute astigmatism {no, yes}
@attribute tear-prod-rate {reduced, normal}
@attribute contact-lenses {soft, hard, none}

@data
%
% 24 instances
%
young,myope,no,reduced,none
young,myope,no,normal,soft
young,myope,yes,reduced,none
young,myope,yes,normal,hard
young,hypermetrope,no,reduced,none
young,hypermetrope,no,normal,soft
young,hypermetrope,yes,reduced,none
young,hypermetrope,yes,normal,hard

```

FIGURE 4.5 Instances of the contact-lenses file.

and an outcome attribute—*contact-lenses*—with possible values *hard*, *soft*, and *none*. The ARFF file format gives no indication as to whether an attribute is to be used for input or for output. This is a job handled through the Weka interface. The *@data* symbol terminates the attribute definitions and tells us that the data instances follow. The attribute values in each line of data are separated by a comma. A question mark is used to specify an unknown attribute value. By scanning the lines of the file, we see that this data set is without unknown values. It's time to load this data set into Weka and get started with our data mining session!

4.2 BUILDING DECISION TREES

Our goal is to build a decision tree that will tell us if an individual who wears glasses is able to wear hard or soft contact lenses. First, we need to explore a few basic preprocessing options. While in preprocess mode, click *open file* and load *contact-lenses.arff* into the Explorer. Figure 4.6 shows the result of this action. The contact lenses file contains 24 instances, where each instance represents an individual who wears either soft contact lenses, hard contact lenses, or no contact lenses (regular glasses).

The file attributes are listed on the left as *age*, *spectacle-prescrip*, *astigmatism*, *tear-prod-rate*, and *contact-lenses*. Note that the *age* attribute is highlighted on this list. On the right of the screen, values for the attribute *age* can be seen. *Type:Nominal* indicates that the *age* attribute is not numeric. Eight instances have a value for age of *young*, eight show age as *pre-presbyopic*, and eight give age as *presbyopic*. Because age is highlighted on the list of attributes, the visualization chart displays the distribution of the instances relative to the age attribute within each of the three classes. Figure 4.6 also contains three labeled arrows pointing to locations of particular interest. Specifically,

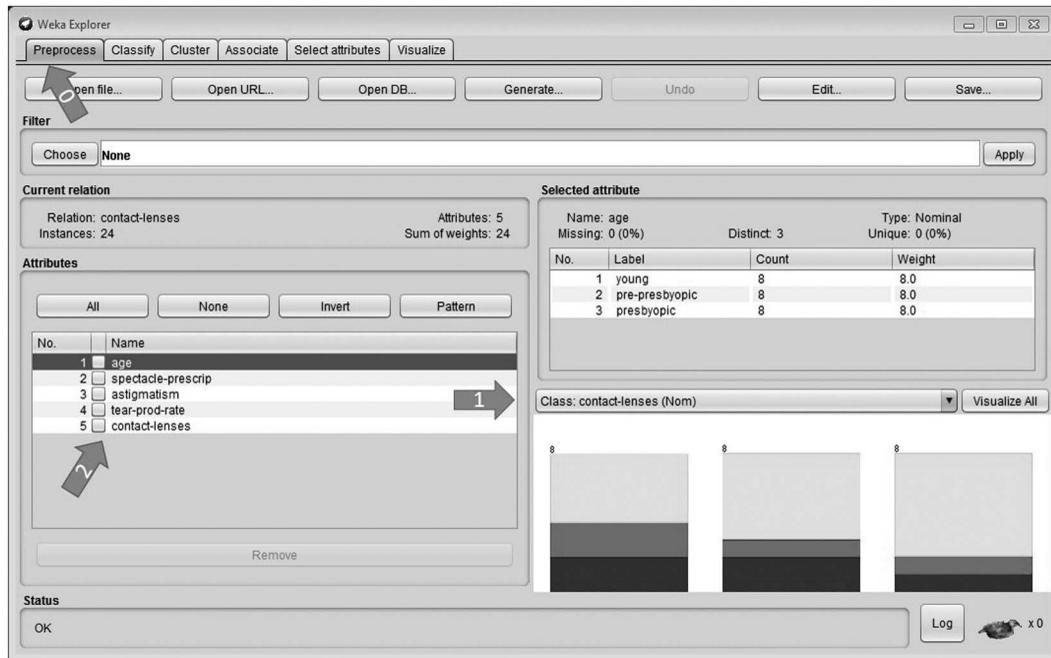


FIGURE 4.6 Loading the contact-lenses data set.

- The arrow labeled 0 points to *Preprocess*. Preprocess mode is our current position within the Explorer.
- The arrow labeled 1 points to the Class or outcome designator for supervised learning. This value can be changed with a click on the Class field. Above this arrow, we see that the selected attribute is *age* with three distinct values. Each bar in the visualize area highlights the values of *age* relative to the three class attribute values.
- Click on the *contact-lenses* attribute pointed to by the arrow labeled 2 to change the attribute information displayed on the right from *age* to *contact-lenses*. The result of this action is shown in Figure 4.7.

Figure 4.7 has five additional pointers to locations of interest. Here is a brief description of each location.

- The arrow labeled 3 points to the *Edit* button. A click on this button presents the data set in spreadsheet format. Click on a specific attribute for several options including sorting, deleting, replacing missing values, and finding mean values.
- Click on *Visualize All* (arrow 4) to see bar charts for all attributes as they relate to the output attribute.
- The arrow in Figure 4.7 labeled 6 points to *Remove*. To eliminate any attribute from the data set, check the box next to the attribute name and click *Remove*.

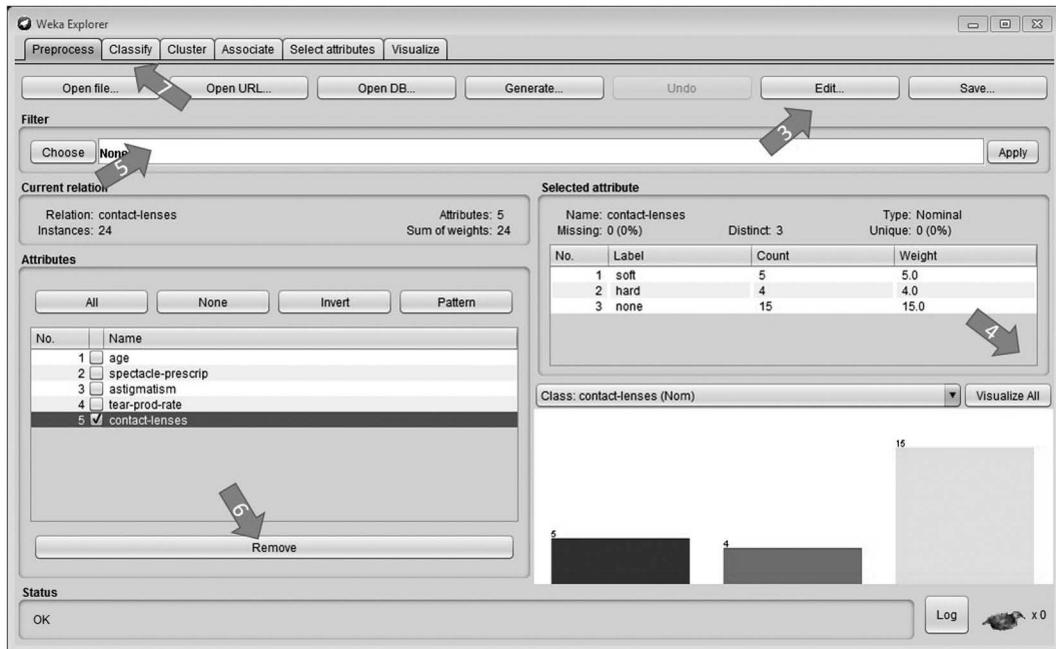


FIGURE 4.7 Navigating the *Explorer* interface.

- The arrow labeled 7 points to *Classify*. We click here when we are finished with data preprocessing and are ready to choose a model for supervised learning.
- The arrow labeled 5 points to the *Choose* command line. In preprocess mode, the command line is used for selecting and applying various filters. Figure 4.8 offers a partial list of supervised and unsupervised attribute filters. Supervised filters filter data relative to a specific output attribute, whereas unsupervised filters are more general. Unsupervised filters can be applied when learning is supervised or unsupervised. For example, the unsupervised *InterquartileRange* filter looks for outliers while skipping the output attribute. To display the entire list of filters, click *Choose, filters, supervised, attribute, instance, unsupervised, attribute, instance*.

Refer to the Weka reference manual for more information about these and other features of the Explorer interface. We now have enough information to leave preprocessing mode and build our decision tree.

- Mouse to the top left of the interface and click on *Classify* (arrow 7 in Figure 4.7). You will see the classifier name *zeroR* appear in the *Choose* command line.

ZeroR is the default model for supervised learning, but we want to use the *J48* decision tree model. Here's how to locate *J48*:

- Click *Choose* and find the *trees* subfolder. Click on *J48*. Your screen will appear as in Figure 4.9.

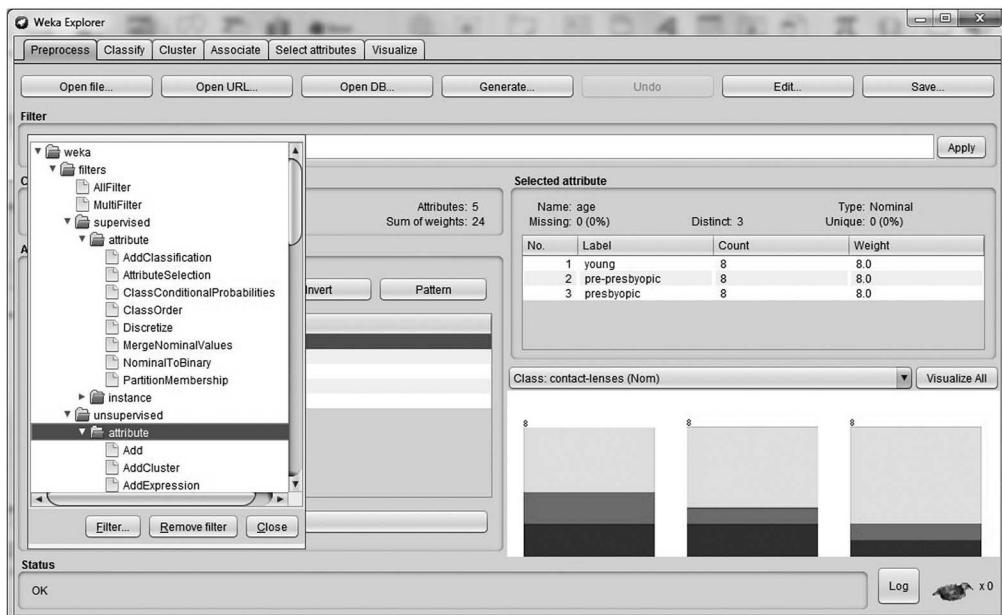


FIGURE 4.8 A partial list of attribute filters.

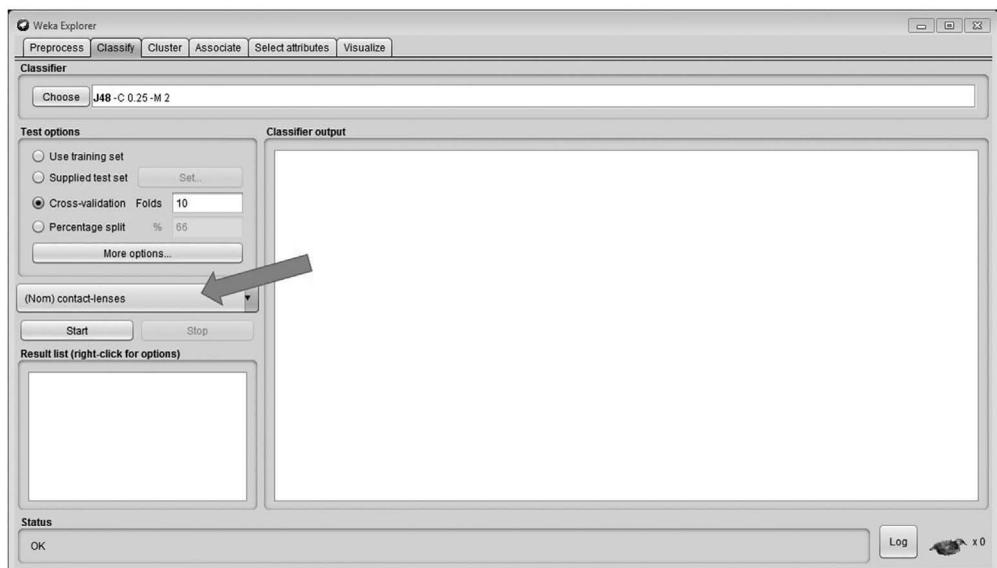


FIGURE 4.9 Command line call for J48.

- Click anywhere on the *Choose* command line to invoke Weka's GenericObjectEditor displayed in Figure 4.10. This editor gives us the opportunity to modify several of J48's learning parameters. End of chapter Exercise 2 looks at how these parameters are used to regulate decision tree size and how the parameter settings affect classification correctness.

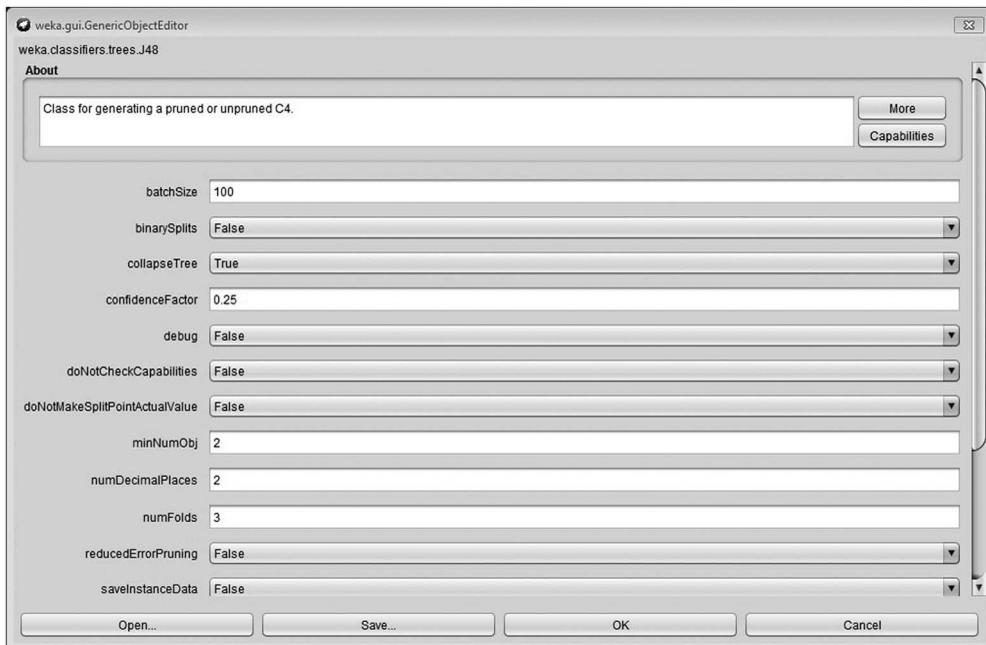


FIGURE 4.10 Parameter setting options for J48.

- Click *More* to obtain a short synopsis of J48 and to learn about the individual parameters. We see that J48 is the Java version of C4.5. Click *Capabilities* to see information about allowable input and output attribute types as well as whether missing data is a problem. Let's leave all parameter values at their default settings.
- In Figure 4.9, directly under the *Choose* command line, we see four model testing options. As there is a lack of ample data for both training and testing, we use the default option, which is the 10-fold cross-validation method discussed in Chapter 2.
- Below *Test options* is the class designation, which must be set as *nom(contact-lenses)*. Click *Start* to begin the mining session.

Figure 4.11 shows the first part of the output. It takes a few data mining sessions to get used to Weka's default textual output format for decision tree structures. The top-level

```
J48 pruned tree
-----
tear-prod-rate = reduced: none (12.0)
tear-prod-rate = normal
|   astigmatism = no: soft (6.0/1.0)
|   astigmatism = yes
|   |   spectacle-prescrip = myope: hard (3.0)
|   |   spectacle-prescrip = hypermetrope: none (3.0/1.0)

Number of Leaves :      4
Size of the tree :     7
```

FIGURE 4.11 Decision tree for the contact-lenses data set.

node of the tree is tear-prod-rate with values *reduced* and *normal*. The first stated line given as *tear-prod-rate = reduced: none (12.0)* represents a terminal node within the decision tree. That is anyone with a value of *reduced* for tear-prod-rate cannot wear contact lenses. The number in parentheses tells us that 12 of the 24 data instances follow this path. If the value of tear-prod-rate is *normal*, the tree structure continues with a test on the attribute astigmatism. The value of *astigmatism = no* gives us a second terminal node. The representation (6.0/1.0) tells us that six data instances follow this path where one of the six instances is an incorrect classification. Any new instance following this path will be classified as someone able to wear soft contacts with an associated prediction confidence of 0.833 (5.0/6.0). Lastly, if the value of astigmatism is *yes*, a final node representing the attribute spectacle-prescrip differentiates those individuals able to wear hard contact lenses from those unable to wear contact lenses. It is important to note that even though cross-validation was used for testing purposes, the tree shown in Figure 4.11 was created using all available data.

We can see a graphical representation of this tree with the following procedure:

- Right-click on *trees.j48* located in the window on the left of the screen with title *Result list (right-click for options)*. Several options are displayed.
- Left-click on *Visualize tree*. Your screen will appear as in Figure 4.12.

This graphical view allows us to better see the most important splits at the top level of the tree.

Next, scroll the display window until it shows the confusion matrix displayed in Figure 4.13. We see that 83.333% of the instances were correctly classified. This value is an

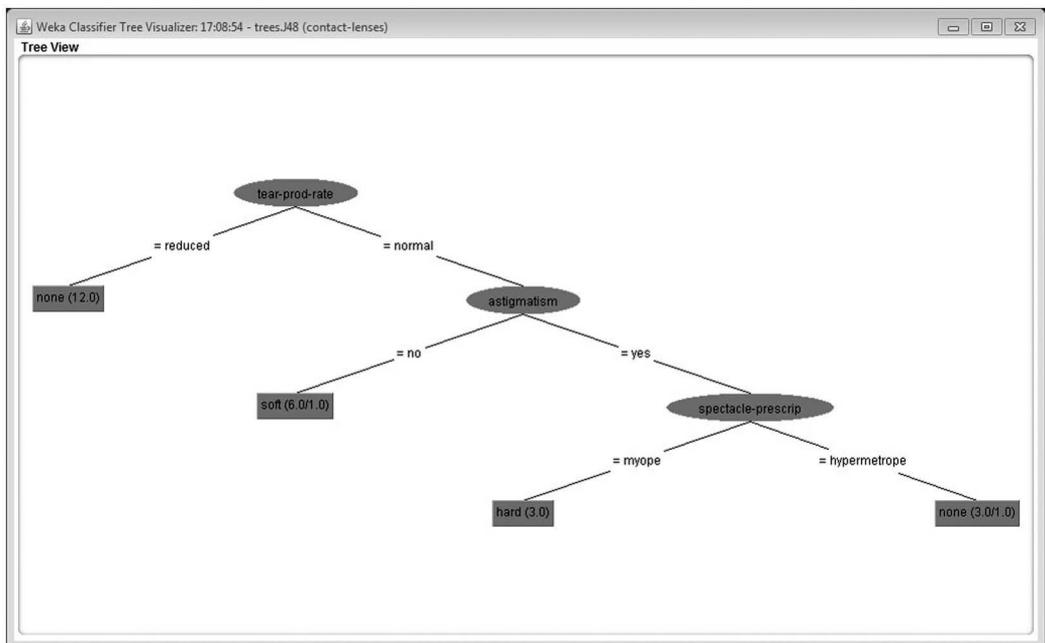


FIGURE 4.12 Weka's tree visualizer.

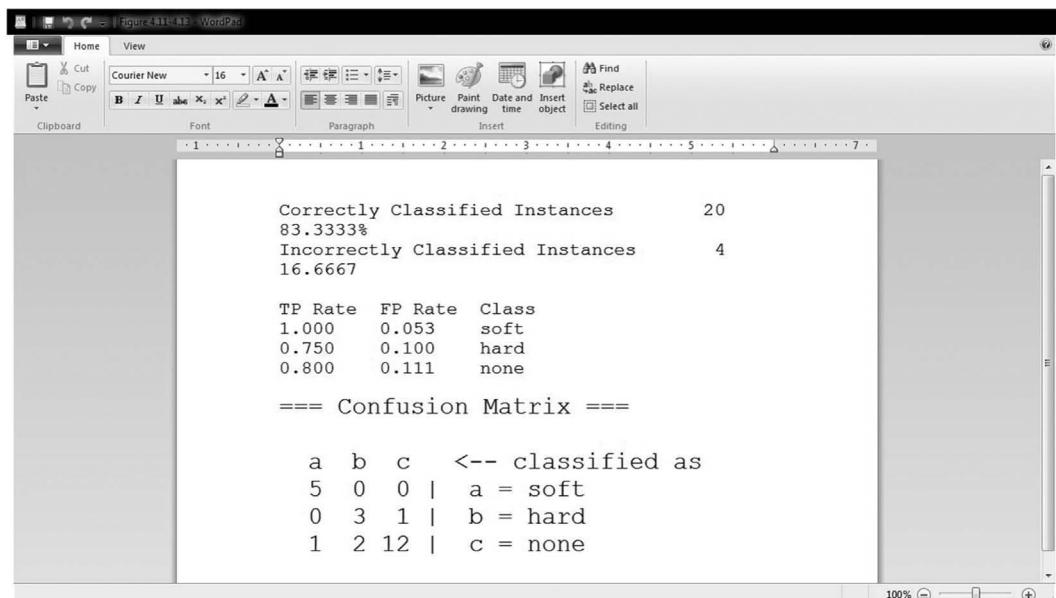


FIGURE 4.13 Decision tree output for the contact-lenses data set.

average correctness score given by the 10-fold cross-validation. The corresponding confusion matrix shows actual class values in the rows and predicted values in the columns. The matrix tells us that one instance representing someone unable to wear contact lenses was classified as able to wear soft lenses. Two instances that should have been classified as *none* were classified as able to wear hard lenses. Finally, one instance able to wear hard contact lenses was incorrectly placed with the class unable to wear contacts.

Here is a procedure we can use to see specific instance classifications.

- On the left of the display screen, click *More options* to see the list of options shown in Figure 4.14.
- Mouse to *Output predictions*, choose *Plain Text*, and click *OK*.
- Click *Start*. When data mining is complete, scroll the output screen until you see the output shown in Figure 4.15.

The screen displays actual and predicted output together with a prediction probability. The *inst#* gives each test set instance an assigned number relative to the cross-validation. As there are 24 instances, the 10-fold cross-validation is carried out using four three-instance test set groups and six two-instance test set groups. That is, four of the cross-validations use 21 instances for training, and six of the cross-validations build their model with 22 training instances. Scroll down until you observe a "+" in the error column. Each "+" designates a classification error.

Before we move to our next example, two additional measures displayed in Figure 4.13 are of interest. *TP Rate* measures positive-instance classification correctness. Consider the

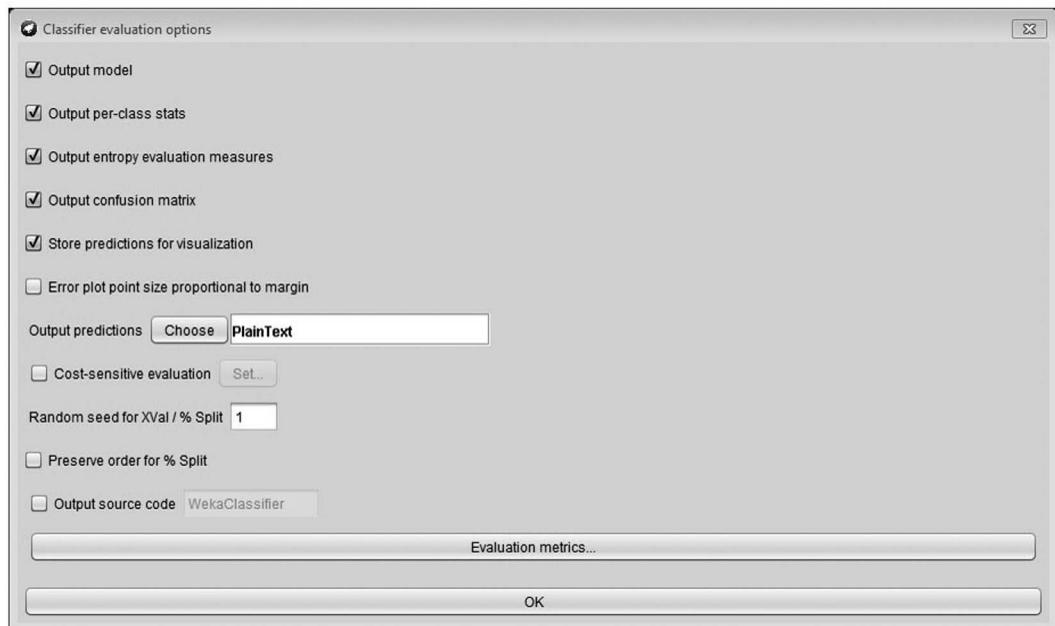


FIGURE 4.14 Classifier output options.

inst#	actual	predicted	error	prediction
1	2:hard	2:hard	1	
2	3:none	3:none	1	
3	1:soft	1:soft	0.8	
1	2:hard	2:hard	0.6	
2	3:none	3:none	1	
3	1:soft	1:soft	0.8	
1	2:hard	3:none	+	1
2	3:none	3:none	1	
3	1:soft	1:soft	0.8	
1	2:hard	2:hard	0.6	
2	3:none	3:none	1	
3	1:soft	1:soft	0.8	
1	3:none	3:none	1	
2	3:none	2:hard	+	0.8
1	3:none	3:none	1	
2	3:none	3:none	1	
1	3:none	3:none	1	

FIGURE 4.15 Actual and predicted output.

true-positive (TP) rate for the class *contact-lenses = none*, which is given as 0.8, computed by dividing 12 by 15. This is the total number of instances correctly classified divided by the total number of instances belonging to the class. Stated another way, TP is computed by dividing the number of instances correctly classified with the given class by the total number of instances actually belonging to the class. The false-positive (FP) rate is computed as the total number of instances incorrectly classified with a specific class divided by the total number of instances that are not part of the class. For the class *contact-lenses = none*, we see this as 1 divided by 9 or 0.111. Values for TP and FP are of special interest when the cost of incorrectly classifying a nonclass instance as a class member is significantly different from the cost of incorrectly classifying a class instance as a member of another class.

4.3 GENERATING PRODUCTION RULES WITH PART

Production rules are a popular technique for finding structure in data. In this section, we show how Weka’s production rule generator, known as PART, is able to uncover interesting rules when presented with data in ARFF format. PART creates a set of production rules with the help of the C4.5 decision tree algorithm. At each iteration, the algorithm creates a partial C4.5 decision tree and then builds a rule following the path of the tree having a best leaf node. Unlike traditional production rules that can be executed in any order, PART’s rules are ordered and must be executed as such. For this reason the production rules generated by PART are often called a *decision list*.

To illustrate PART we utilize two files having customer churn information. The original data set, *customer-churn-data.xls*, has been featured in several RapidMiner tutorials. Here we use these data to create two files. One file holds all instances of known outcome. The second file contains the instances whose outcome is to be predicted. Here is the procedure.

- Load *customer-churn-data-KN.arff* into Weka’s Explorer. If your screen does not show *Churn* as the output attribute—indicated above the visualization screen—be sure to change the output attribute to reflect this.
- Per Figure 4.16, click the *Churn* attribute. The figure shows that the data set contains five attributes and 900 instances. The output attribute is *Churn* with values *loyal* or *churn*, where *loyal* tells us if the individual represented by the instance is still a customer and *churn* indicates that the individual is no longer a customer. We see that 322 individuals are no longer customers.
- Mouse from preprocess to classify mode, choose rules, and click on *PART*.
- Directly above *Start*, change the output attribute to *Nom (Churn)*. Be sure that the test option is 10-fold cross-validation.
- Click *Start* to begin the mining session. Scroll your output screen until it appears as in Figure 4.17. The output shows a classification accuracy near 85%. The output also displays five rules intended to be executed in order. That is, to classify a new instance, we always start by checking if the first rule applies followed by the second, third, and fourth rules. The final rule is a default with no precondition and applies only if

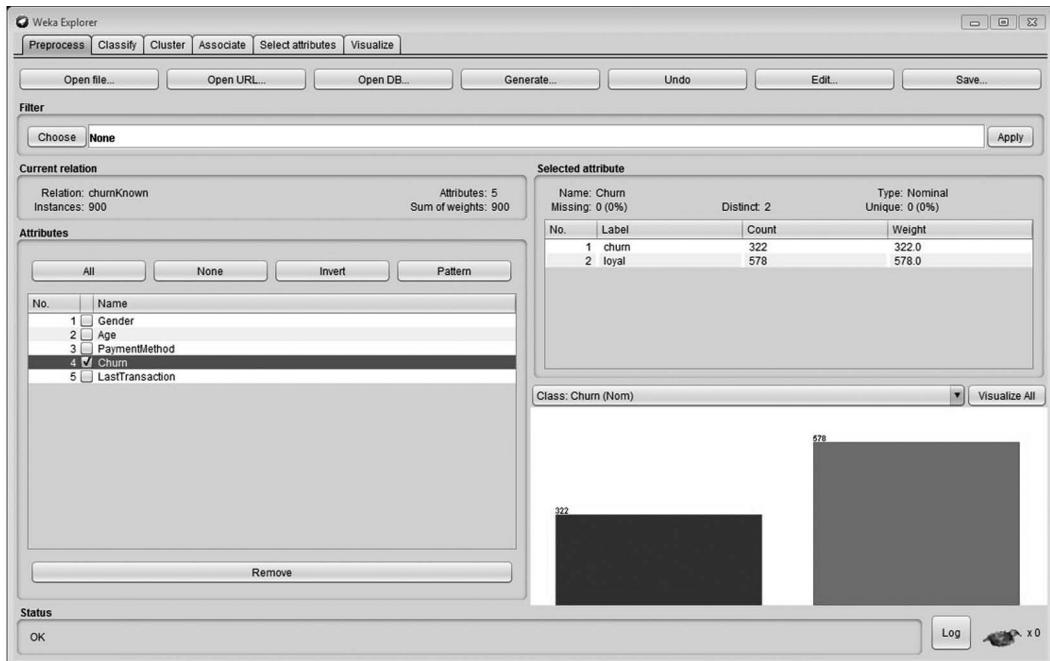


FIGURE 4.16 Customer churn data.

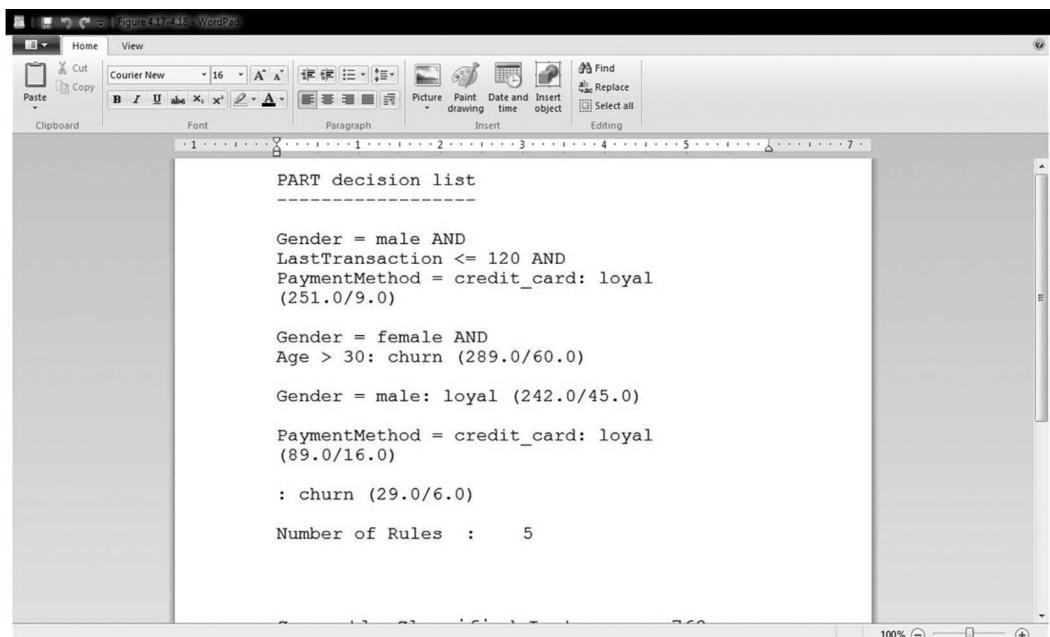


FIGURE 4.17 A decision list for customer churn data.

the first four rules are not applicable. Notice that each of the five rules misclassifies several instances.

- Scroll the output until you see the confusion matrix shown in Figure 4.18.

The confusion matrix tells us that our model incorrectly classified 66 loyal customers as churners. Also, 71 individuals were incorrectly determined to be loyal customers. Given the same classification accuracy, a better result would see fewer individuals incorrectly determined to be loyal customers and more loyal customers classified as churners. However, let's assume that we are satisfied with the current model and plan to use it as a prediction tool for likely churners. Therefore, our next step is to save the model.

- The model is listed in the section titled *Result list (right-click for options)*. The model has the name *rules.PART*. Right-click on the model name.
- Left-click on *Save model* to save the model to a desired destination folder. The model is saved as a Java object file with a .model extension.
- Close Weka Explorer.

We can now use the saved model to predict which current customers are likely churners. To do this, we use the file *customer-churn-data-UNK.arff*. This file contains 96 instances representing current customers with the class attribute showing a question mark rather than the name of the class. Here is the method.

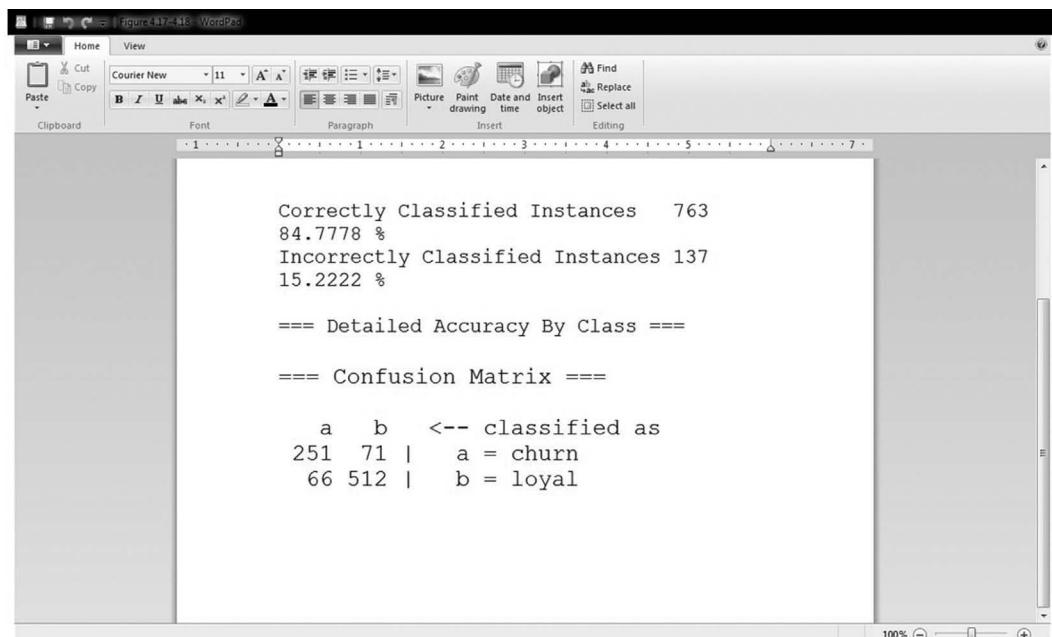


FIGURE 4.18 Customer churn output generated by PART.

- Open Weka's Explorer and open the file *customer-churn-data-UNK.arff*.
- Move from preprocess mode to classify mode.
- Right-click in the *Result list* (*right-click for options*) area, and choose the *Load model* option. Find and load the previously saved model. Be sure to include the .model extension when you load the file. The rules for the model will appear to the right.
- Click *More options*. Mouse to *Output predictions*, Choose *Plain Text*, and click *OK*.
- Under *Test options*, highlight the *Supplied Test set* radio button. Click on *Set* to see Figure 4.19. Click on *open*, and locate *customer-churn-data-UNK.arff*. To load the file, either double-click on the file name or type the entire name including the .arff extension into the space provided. Finally, change the class from *no class* (arrow 2 in Figure 4.19) to *(Nom) Churn*. Click *Close*.
- Mouse back to the *Result-List* area and right-click on the model name. Click *Re-evaluate model on current test set* to obtain the predictions. Scroll your display until it is similar to the output in Figure 4.20.

For each instance, we are given a predicted class together with a prediction probability. The prediction probability gives us a value between zero and one that indicates how confident we can be that the prediction made by the model is correct. To illustrate, the first instance in Figure 4.20 is predicted to churn with probability 0.792. This instance is a 39-year-old female and was therefore predicted to churn based on the second rule in Figure 4.18. Two hundred eighty-nine instances of known outcome were classified by this rule. Sixty of the 289 classifications were incorrect, giving us a classification accuracy of 0.792 (229/289). The question mark in the *actual* column informs us that the classification of each

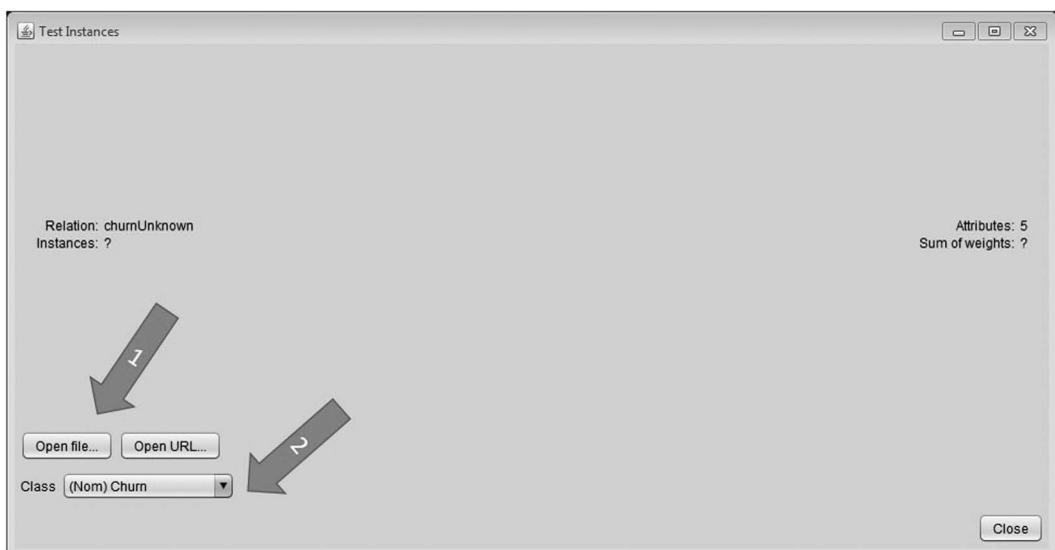


FIGURE 4.19 Loading the customer churn instances of unkown outcome.

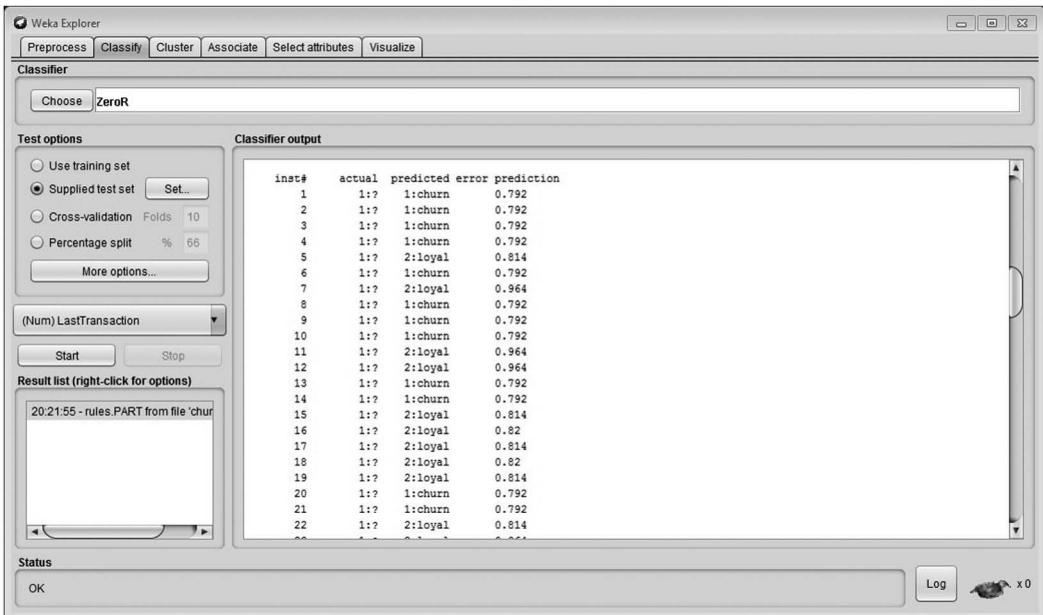


FIGURE 4.20 Predicting customers likely to churn.

instance is unknown and is to be predicted. The resultant output buffer can be saved for further investigation with a right-click on the model name and then *save result buffer*. Once saved, we can sort the predictions to help make decisions about which individuals should receive special attention in order to avoid having them go elsewhere for their purchases.

Figure 4.20 used two files. One file held instances for training and model testing. The second file contained instances where the value of the output attribute was not known. It is worth noting that we can bring about the same result by using a single file that holds the training data as well as the data used for predictive purposes. To do this, we assume that the parameters used to build the model have previously been established. Here is the procedure.

- Append the unknown instances to be classified to the end of the file containing the training data. Make sure that the unclassified instances contain a question mark in the class field.
- Load the file containing both the training data and the unclassified instances into the Explorer.
- Set the desired parameters for building the model. Set the test option to *Use training set*. Be sure that you mouse to *Output predictions* within *More options* and set the *Choose* box to show *Plain Text*. Click *Start*.

The resultant output will contain the summary information obtained from training and will also show the predicted value for each of the unknown instances. That is, all Weka classifiers including PART use only those instances whose classification is known for training.

The remaining instances are listed with their predicted class together with a probability value. The downside to this technique is that the model must be recreated each time before it is applied to the unknown data.

A main advantage of both PART and J48 is that the attribute selection process is built into their model building algorithms. This is often the exception rather than the rule. Fortunately, Weka offers a large selection of preprocessing filters that help determine a best set of attributes prior to the model building process. The next section demonstrates how one such preprocessing filter can be applied in conjunction with a nearest neighbor classifier.

4.4 ATTRIBUTE SELECTION AND NEAREST NEIGHBOR CLASSIFICATION

In Chapter 1, we discussed the nearest neighbor approach to classification. We gave three potential problems with this technique. Specifically,

- Computation time is a problem when the classification table contains thousands or millions of records.
- The approach has no way of differentiating between relevant and irrelevant attributes.
- We have no way to tell whether any of the chosen attributes are able to differentiate the classes contained in the data.

We can partially overcome the second and third problems by eliminating irrelevant attributes. Here we use a supervised attribute filter to choose a best set of attributes for supervised classification. For our example, we apply a supervised attribute filter to the spam data set (*spam.arff*) prior to using Weka's IBk nearest neighbor classifier. IBk uses linear search together with simple Euclidean distance to tag an unknown instance with the class most common to its K nearest neighbors. IBk is listed in the *Choose* drop-down menu under *lazy classifiers*.

The spam data set contains 4601 data instances. Each instance represents an e-mail message—2788 are valid messages, and the remaining are spam. All 57 input attributes are continuous. The output attribute is nominal, with 0 representing a valid e-mail message. Let's build a real-time model using IBk that can distinguish between valid and invalid e-mail messages. We hope to use the model to determine whether a new e-mail message will be delivered to a user's inbox or placed in the junk folder. Desirable model characteristics include both fast processing and high accuracy levels where the majority of errors classify spam e-mail as valid. That is, we would rather have the user delete spam from the inbox than miss valid messages that reside in the junk folder.

Before employing the attribute filter, let's first see what happens when we apply IBk to the original set of attributes.

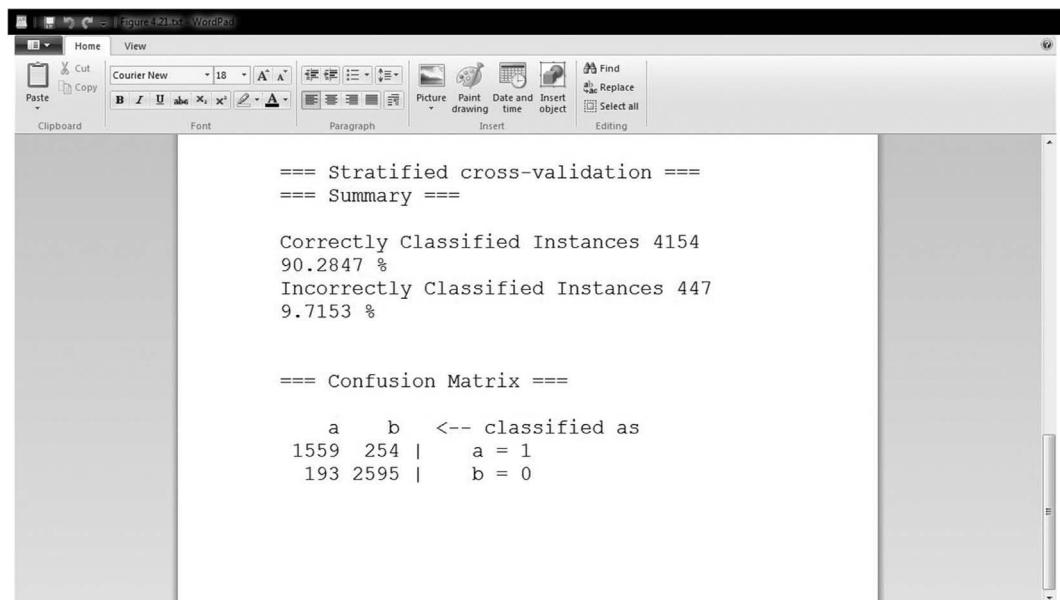
- Load *spam.arff* into Weka's Explorer and proceed to classify mode.
- Use *Choose* to locate and load IBk. IBk is found in the *lazy* subfolder under *classifiers*.

- Click on the *Choose* command line to see the GenericObjectEditor. Set the value of KNN to 5. This instructs the algorithm to classify a new instance based on its five nearest neighbors.
- Choose *Cross-Validation* and click *Start* to see the result displayed in Figure 4.21.

The output shows an overall classification accuracy of more than 90%. We see that 254 spam e-mails are classified as valid and 193 valid e-mails are given a spam label. Using this result as a benchmark, we pose this question: Can we achieve a comparable or higher classification accuracy using only those attributes most predictive of class membership?

To address this question, we return to preprocess mode and apply a supervised *AttributeSelection* filter to the original data set. Once the filter is chosen, we then select a specific evaluator and search technique. For our example, we choose *InfoGainAttributeEval* as the evaluator and *Ranker* as the search technique. *InfoGainAttributeEval* determines the goodness of an attribute by measuring the class information gained as a result of adding it to the list of input attributes. *Ranker* ranks the input attributes from most to least predictive of class membership based upon the chosen evaluation method. Here's the procedure.

- Return to preprocess mode, click *Choose* on the command line, and follow the path *filters* → *supervised* → *attribute* → *Attribute Selection* (Figure 4.22). Click *AttributeSelection* to see the call to the filter in the command line.
- Click on the white space area of the *Choose* line to invoke the GenericObjectEditor. Change the evaluator to *InfoGainAttributeEval* and search to *Ranker* (Figure 4.23).



The screenshot shows a Microsoft WordPad window titled "Figure 4.21.txt - WordPad". The content of the document is as follows:

```

=====
Stratified cross-validation ===
===== Summary ===

Correctly Classified Instances 4154
90.2847 %
Incorrectly Classified Instances 447
9.7153 %

===== Confusion Matrix ===

      a      b    <- classified as
1559  254 |      a = 1
193   2595 |      b = 0

```

FIGURE 4.21 Nearest neighbor output for the spam data set.

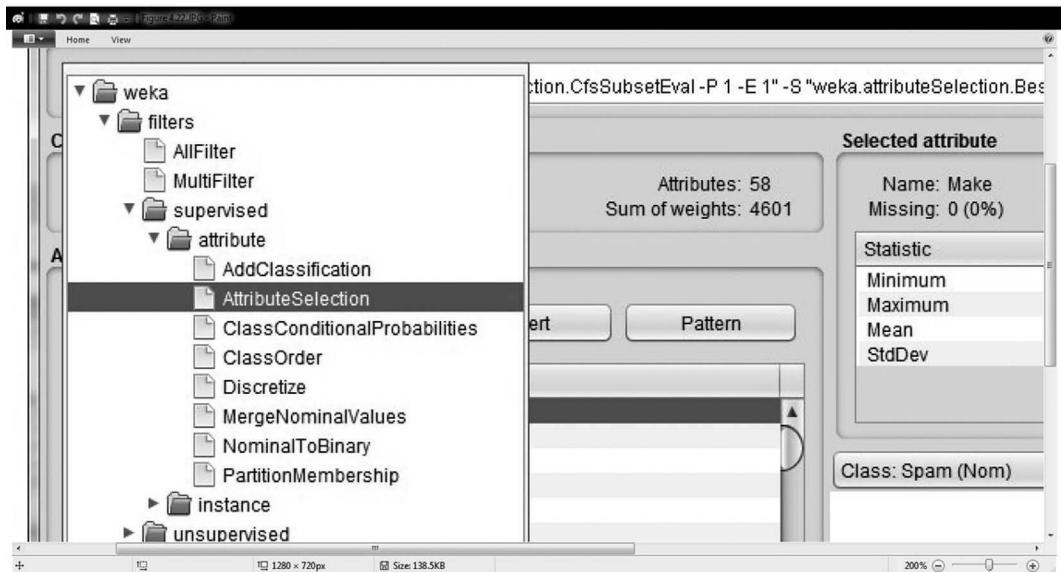


FIGURE 4.22 Weka's attribute selection filter.

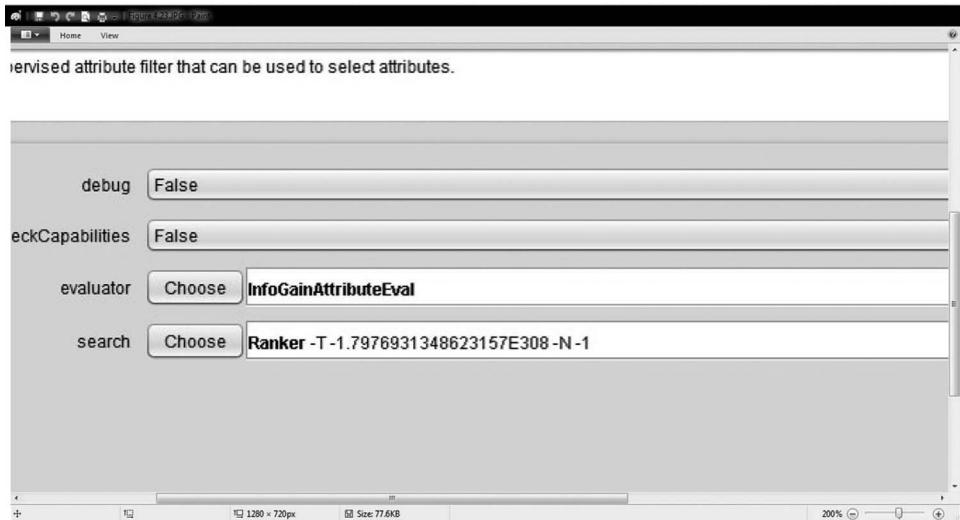


FIGURE 4.23 Options for the attribute selection filter.

- Click in the white space area of the *Choose* line for *Ranker*. The result of this action is shown in Figure 4.24.

Notice that the variable *numToSelect* is set at -1 . This tells us that when the filter is applied, the input attributes will be ranked from most to least relevant, but none of the attributes will be deleted. We can modify this value to determine the exact number of

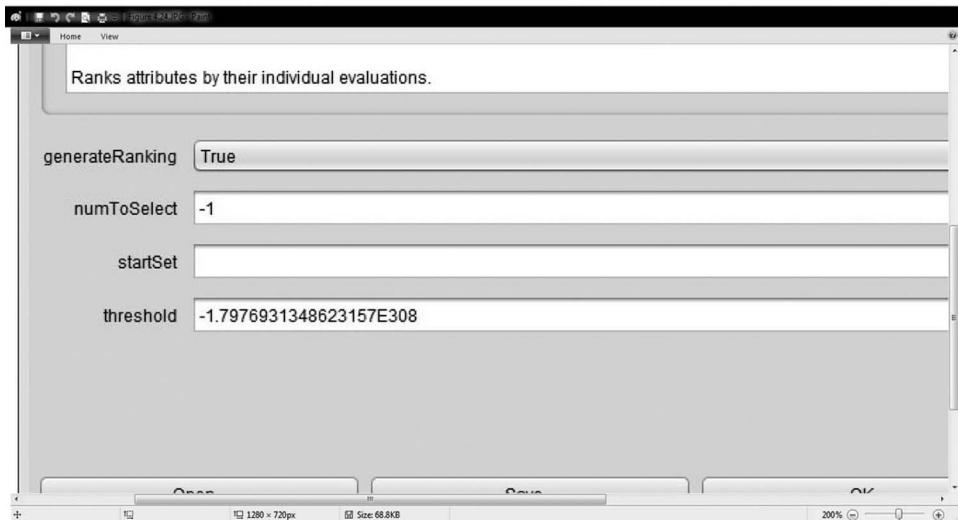


FIGURE 4.24 Parameter settings for ranker.

input attributes to be saved. That is, if *numToSelect* has value 5, the five most predictive input attributes will be saved, and the others will be removed.

Rather than attempting to guess the number of attributes to be removed, we can use ranker to first rank the attributes without removal. After this, we conduct several experiments. In each experiment, we mine the data after removing one or more of the least predictive attributes. This method gives us a best chance of retaining the most predictive attributes. Let's give this a try.

- Click OK to close the GenericObjectEditor.
- To apply the filter, click *apply* located at the rightmost position of the *Choose* command line. The attributes will display as in Figure 4.25.
- Check attributes 48 through 57 and click *Remove*. This removes the 10 least predictive input attributes from the data set.
- Mouse to *Classify*. Make sure that IBk is the chosen classifier with KNN set at 5. Click *Start*.

The result is displayed in Figure 4.26 and shows a classification accuracy close to 90% with 267 spam e-mails classified as valid and 203 valid e-mails classified as spam.

- Return to preprocess mode and remove the next 10 least predictive attributes. Click *Start*. This results in a classification accuracy slightly under 90% with 293 spam e-mails classified as valid and 178 valid e-mails classified as spam.

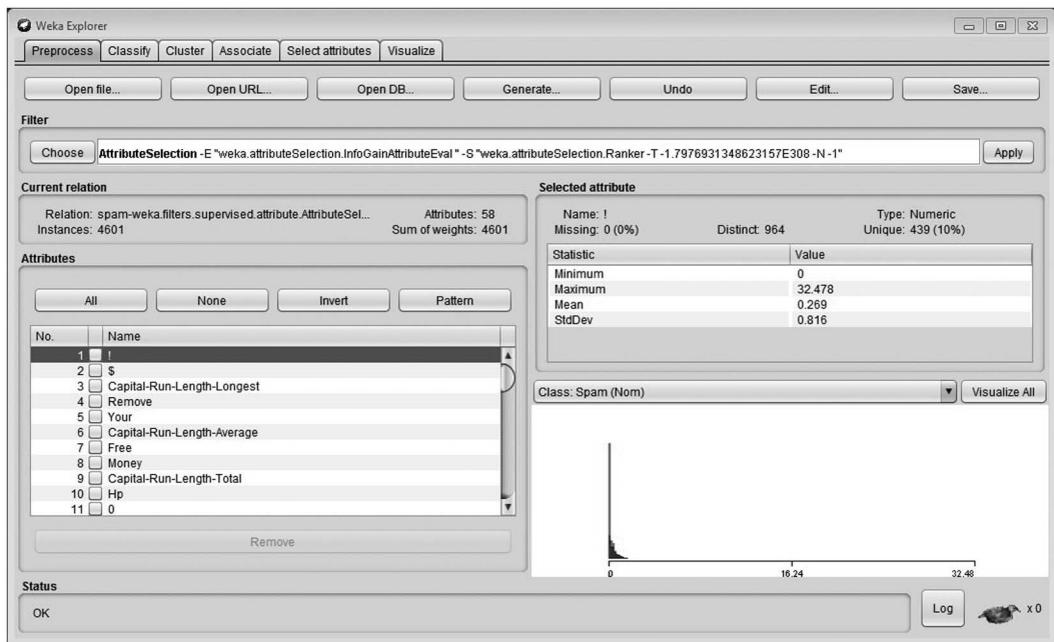


FIGURE 4.25 Most predictive attributes for the spam data set.

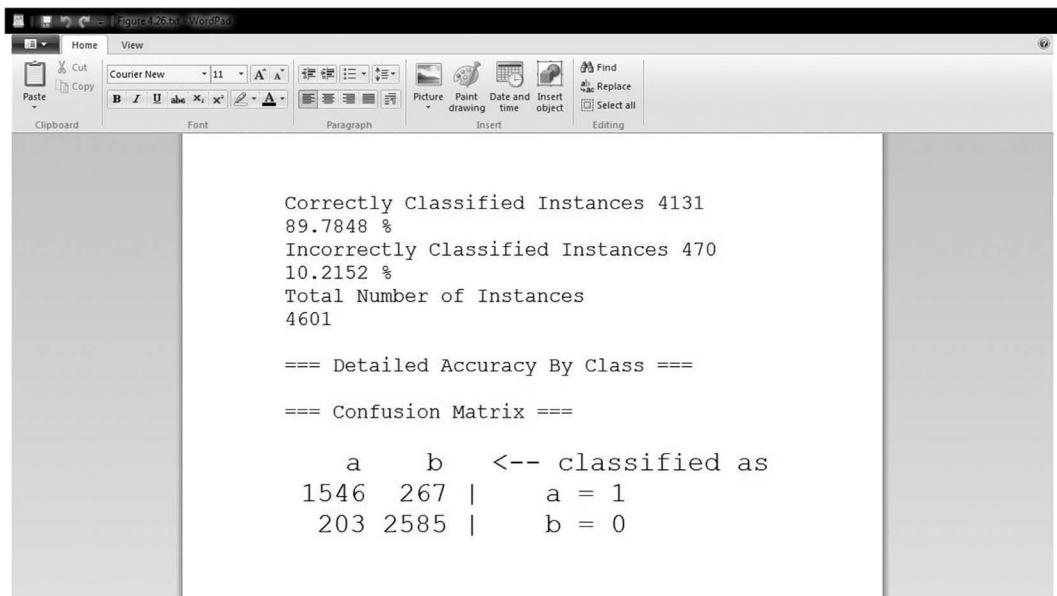


FIGURE 4.26 IBk output after removing the 10 least predictive attributes.

- Once again, return to preprocess mode and delete the 10 least predictive attributes. With 28 input attributes, we see a classification accuracy of 89.5% where 331 spam e-mails are classified as valid and 146 valid e-mails are classified as spam.

Note that our first experiment yielded 193 valid e-mails classified as spam, while our last experiment produced 146 valid e-mails classified as spam with no significant loss in incorrectly classified instances. These results indicate that by removing the least predictive attributes, we are getting closer to our goal of maintaining high accuracy levels, increasing classification speed, and reducing the number of valid e-mails classified as spam. It will take additional experimentation to determine if we have achieved a best result.

Here we have examined a single supervised filter to get you started in understanding the attribute selection process. We encourage you to experiment with several of the supervised and unsupervised filters offered within the Weka toolkit.

4.5 ASSOCIATION RULES

Weka has six association rule generators. Here we highlight the well-known Apriori algorithm presented in Chapter 3. Most association rule generators including Apriori are limited to data sets containing nominal (categorical) data. If a data set contains numeric attributes we have one of two choices. One option is to delete all numeric attributes prior to generating association rules. Our second option is to convert numeric attributes to categorical equivalents. This can be done in preprocess mode using Weka’s *discretize* filter found within the list of both supervised and unsupervised attribute filters. For our example, we return to the contact-lenses data set, where all attributes are categorical.

- Load the *contact-lenses.arff* into Weka’s Explorer.
- Click on *Associate* located at the top of the Explorer screen. A call to the Apriori algorithm will appear in the *Choose* command line.
- Click *Start*. Scroll your display until it appears as in Figure 4.27.

The output shows that 10 association rules were generated from the data. The first five rules give *contact-lenses* as the output attribute. Rules six through ten have either *astigmatism* or *tear-prod-rate* as the output attribute.

Consider the second rule given as:

If *spectacle-prescrip* = *myope* *tear-prod-rate* = *reduced* 6 → *contact-lenses* = *none* 6
conf:(1)

The 6 in the rule precondition (left of → symbol) tells us that 6 data instances show both *myope* and *reduced* as values for the given attributes. The 6 in the rule consequent (right of → symbol) tells us that these same six instances have the value *none* for the attribute *contact-lenses*. This results in a confidence value of 1 for the rule. All 10 rules have a

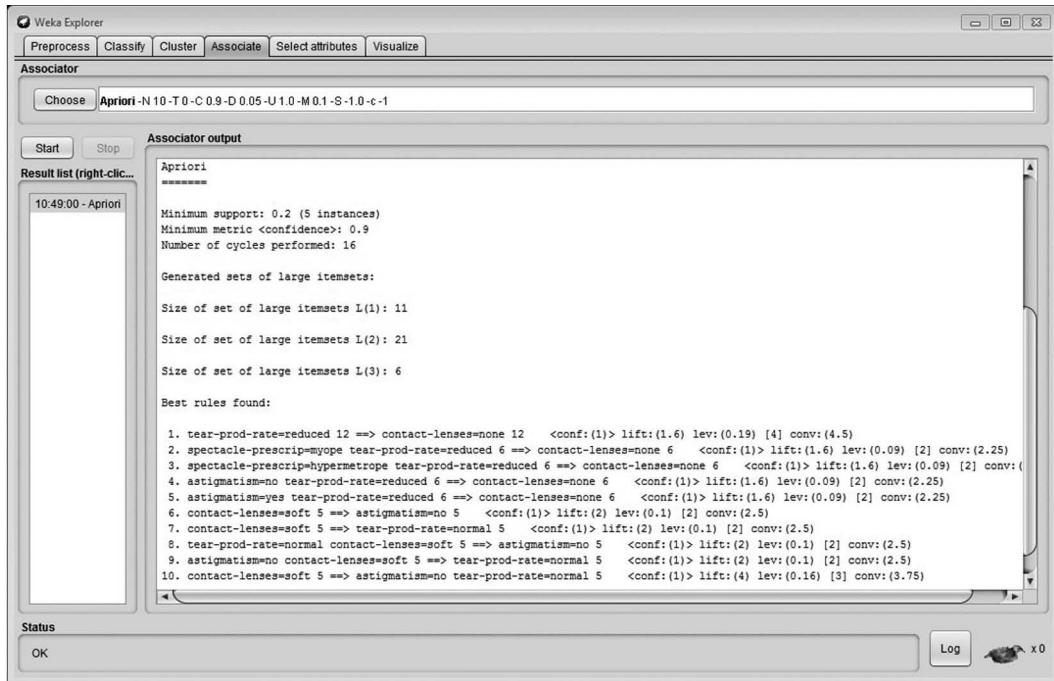


FIGURE 4.27 Association rules for the contact-lenses data set.

confidence score of 1, meaning that each rule is 100% accurate. Rule support is not shown. We are able to manipulate confidence and support as well as other parameters with a click on the *Choose* command line. Doing so invokes the GenericObjectEditor seen in Figure 4.28. We can choose a lower bound for confidence, set lower and upper bounds on support, use a metric other than confidence, and control the maximum number of rules to be displayed. One other parameter of special interest is *delta*. The value of delta determines the iterative decrease in the support factor. Support is reduced by this value until the lower bound for support has been reached or the required number of rules has been generated. LowerBoundMinSupport is currently set at 0.1. If we reset this value at 50% (0.5) and run the application a second time we see but a single rule. Specifically,

$$\text{tear-prod-rate} = \text{reduced} \rightarrow \text{contact-lenses} = \text{none}$$

Let's turn our attention to a more interesting data set designed for market basket analysis. The *supermarket.arff* data set found in Weka's data folder represents actual shopping data collected from a supermarket in New Zealand. Figure 4.29 shows the result of loading this data set into the Explorer. The data set contains 4627 instances and 217 attributes. Each instance gives the purchases of a shopper in a single trip to the supermarket. Some attributes are very general in that they simply indicate a purchase within a given department. Other attributes are more specific as their value is an individual item type such as canned fruit. The last attribute is *total* with a value of *high* or *low*. The value is high if the total bill is more than \$100. However, this attribute is not necessarily of major importance

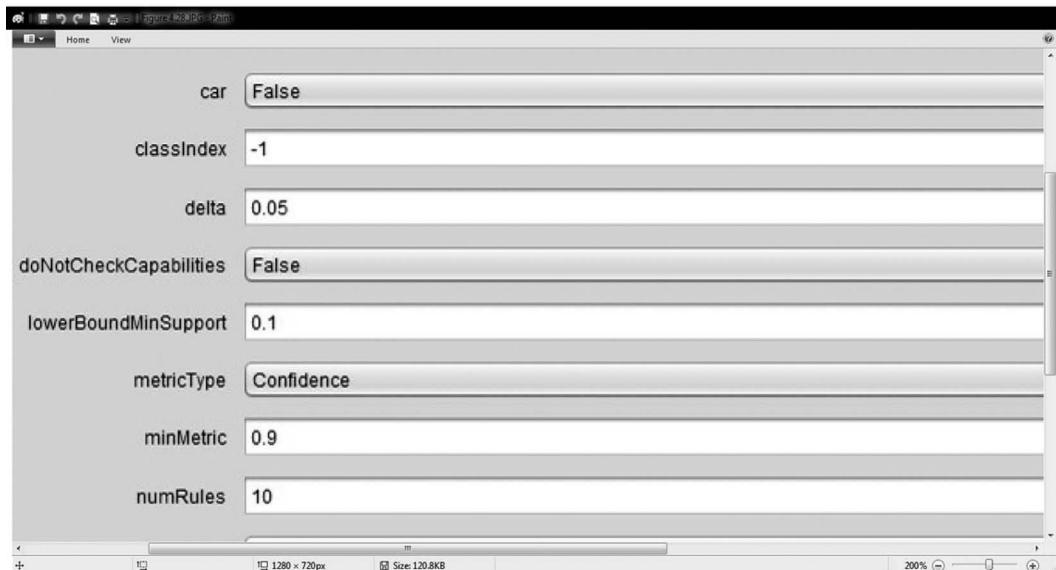


FIGURE 4.28 Parameters for the Apriori algorithm.

as we are looking for association rules featuring combinations of attribute values in order to determine those items likely to be purchased together.

- Load *supermarket.arff* into the Explorer.
- Click *Edit* to see the first few instances displayed in Figure 4.30.

We see that a purchase is represented by a *t*. The first listed customer purchased an item from *department1*, a *baby needs* item, a *bread and cake* item, and several other items seen by moving the slide bar on the bottom of the screen to the right, a product classified as *baking needs*, and a few other items. Customer 2 purchased an item from *department1* as well as the additional items seen by moving the scroll bar to the right. It is obvious that most of the

THE CREDIT CARD SCREENING DATA SET

The Credit Card Screening data set contains information about 690 individuals who applied for a credit card. The data set contains 15 input attributes and 1 output attribute indicating whether an individual credit card application was accepted (+) or rejected (-). All input attribute names and values have been changed to meaningless symbols to protect confidentiality of the data. The original data set was submitted by Ross Quinlan to the machine learning data set repository referenced in Appendix A.

The data set is interesting for several reasons. First, the instances represent real data about credit card applications. Second, the data set offers a nice mix of categorical and numeric attributes. Third, 5% of the data set records contain one or more pieces of missing information. Finally, as the attributes and values are without semantic meaning, we cannot introduce biases about which attributes we believe to be important.

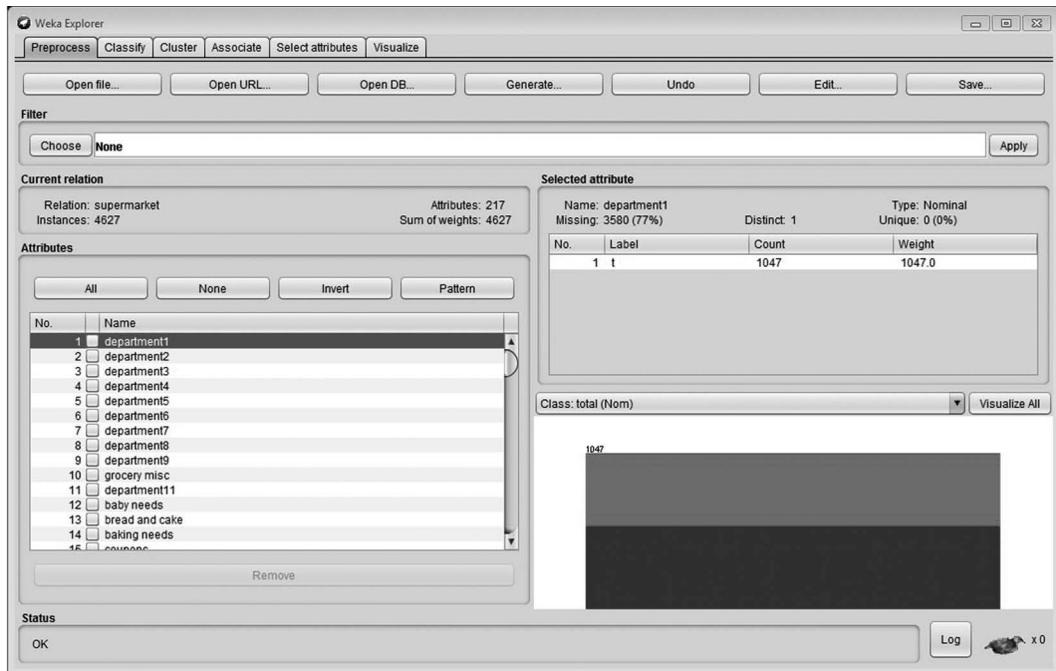


FIGURE 4.29 The supermarket data set.

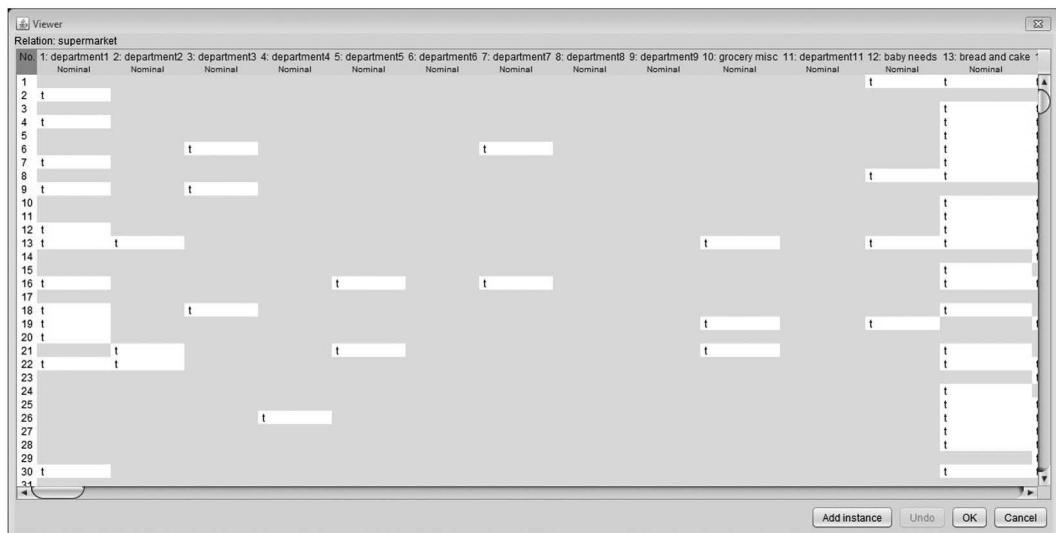


FIGURE 4.30 Instances of the supermarket data set.

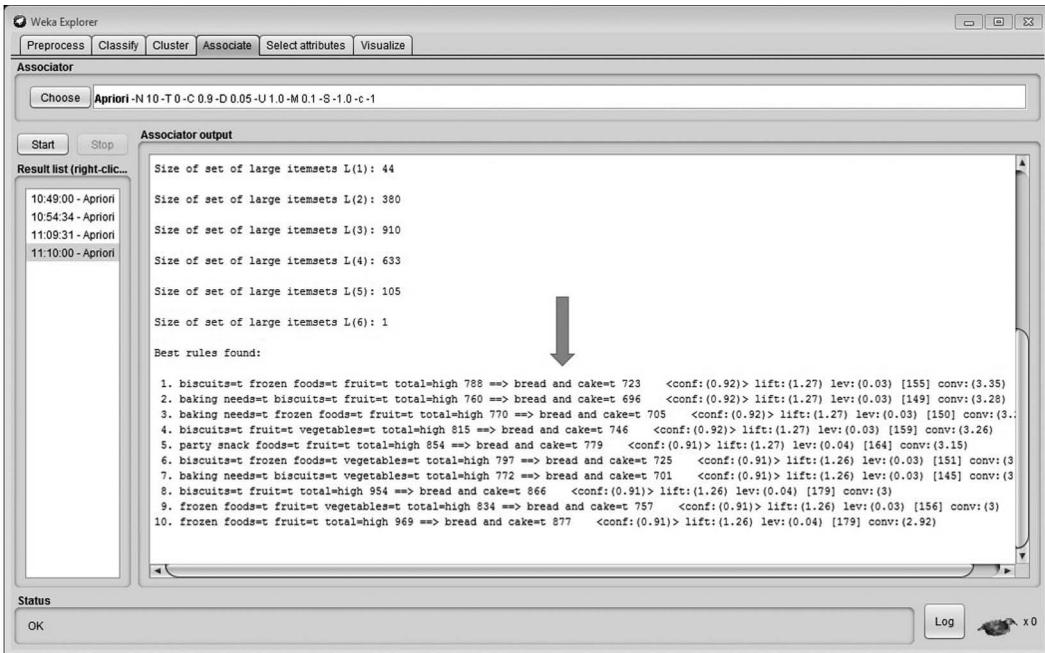


FIGURE 4.31 Ten association rules for the supermarket data set.

data are quite sparse in that most attribute values are empty. With *LowerBoundMinSupport* set to 0.1, a click on *start* results in 10 association rules using the default parameter settings of the Apriori algorithm. Figure 4.31 shows that the consequent for each rule is *bread and cake*. Exercise 6 challenges you to extract interesting association rules from this data set.

4.6 COST/BENEFIT ANALYSIS (OPTIONAL)

Some applications are less concerned with test set classification accuracy and are more interested in building models able to extract bias samples from large populations. Response rates for mass mailings immediately come to mind. It is also frequently the case that we want to apply weights to various types of classification error. For example, it is likely to be advantageous for a credit card company to deny cards to some individuals who would not default if they are, in return, able to accurately deny individuals who will default.

Here we address this issue by illustrating how we use Weka's cost/benefit analyzer to associate cost values with individual classifications. For our example, we use the data set described in the aside box titled "The Credit Card Screening Data Set." The data set contains information about 690 individuals who applied for a credit card. The data include 15 input attributes and 1 output attribute indicating whether an individual credit card application was accepted (+) or rejected (-). Here is our goal:

We are to build a predictive model using the instances of the credit card screening data set able to predict which future credit card applicants to accept and which to reject. The model must allow varying costs to be associated with incorrectly accepting or rejecting credit card applicants.

Here's how the cost/benefit analysis tool together with J48 can help us find a solution!

- Load the CreditScreening.arff data set into Weka.
- Mouse to *Classify*. Directly above *Start*, change the output attribute to *class*. Be sure that the test option is 10-fold cross-validation.
- Use *Choose* to locate J48.
- Click on *More options* and mouse to *Output predictions*. Click on *Choose* and select *Plain Text*. Click *OK*. This will give us the needed prediction confidence scores for our analysis.
- Perform a data mining session using J48's default settings.

The output is given in Figure 4.32. We see a classification accuracy of 84.783% (arrow 2). The confusion matrix (arrow 3) tells us that 50 individuals incorrectly received a credit card and 55 applicants that should have been accepted were not. Arrow 1 points to the individual predictions for the last two instances of the final cross-validation.

- Locate *Result list (right-click for options)* just under *Start* on the left of the display screen.
- Right-click on the line that displays *trees.j48*. Mouse down the menu to locate *Cost/Benefit analysis*. Your screen will show as in Figure 4.33.
- Left-click on the "+" sign. Your screen will appear similar to the display in Figure 4.34.

The display shows the interface for Weka's cost/benefit analysis tool. The cost/benefit tool offers several analytic capabilities.

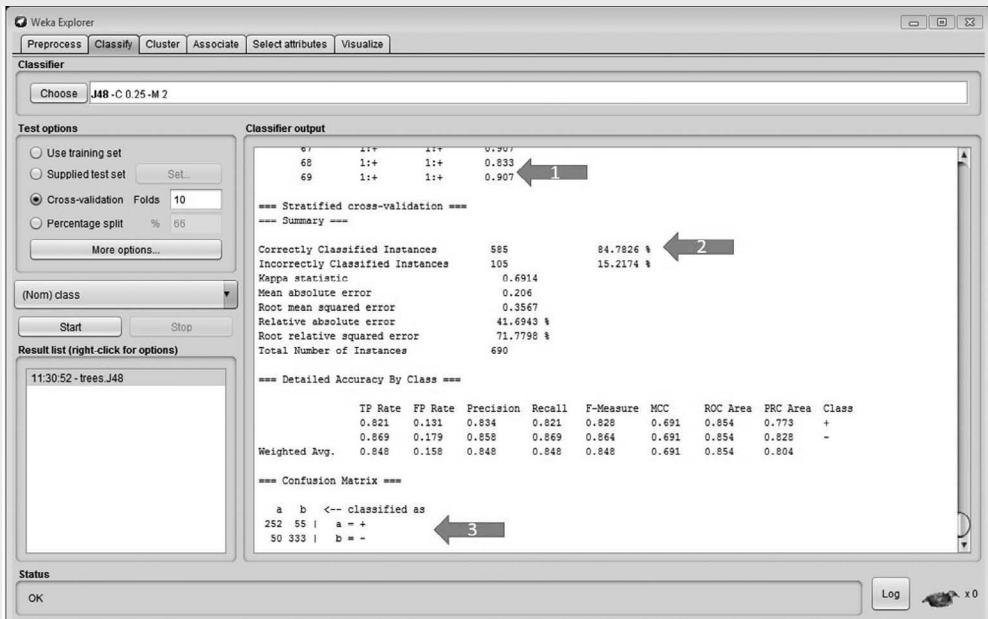


FIGURE 4.32 A J48 classification of the credit card screening data set.

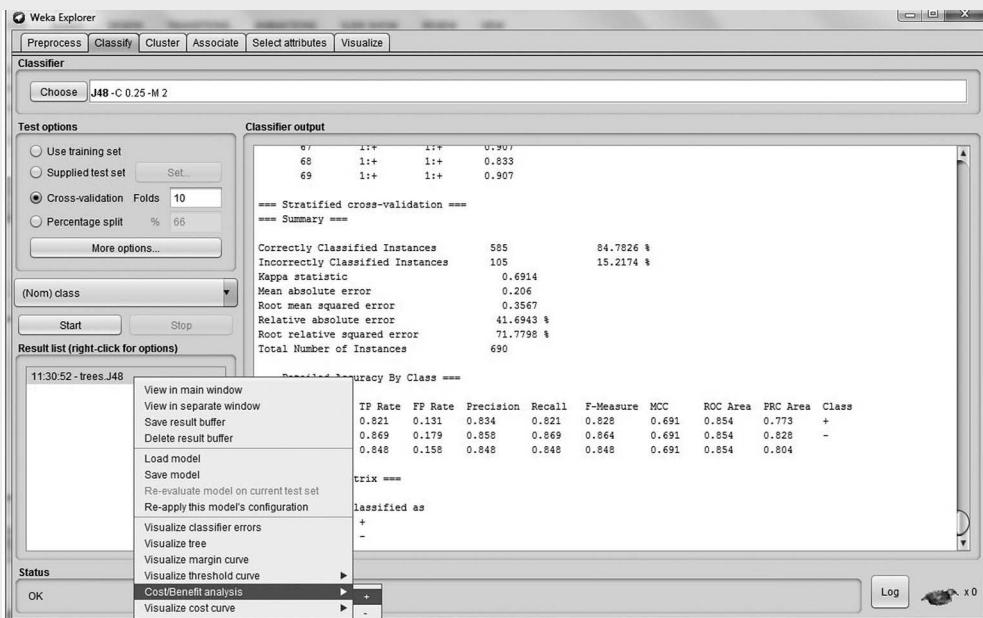


FIGURE 4.33 Invoking a cost/benefit analysis.

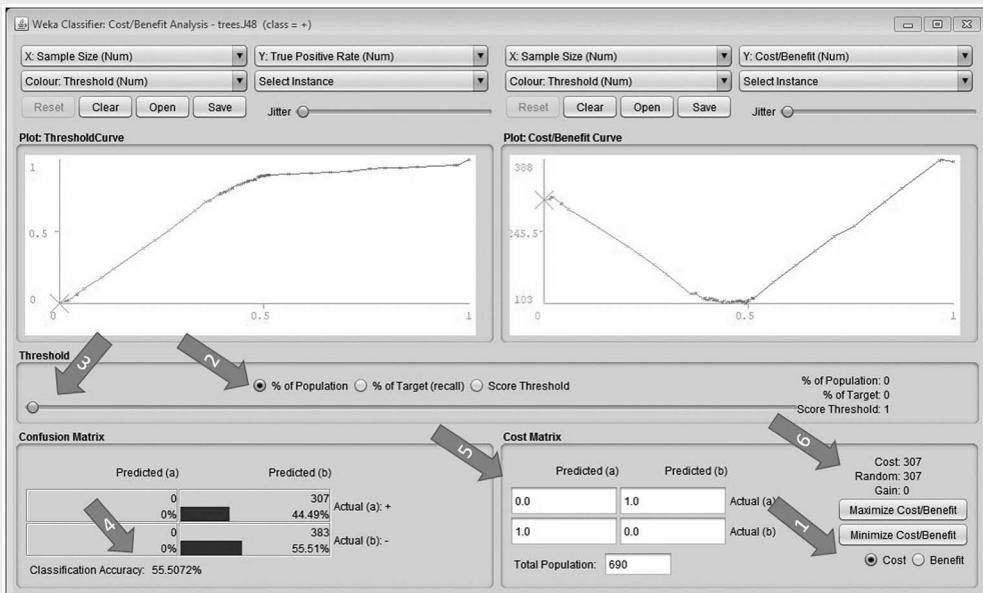


FIGURE 4.34 Cost/benefit output for the credit card screening data set.

- As our problem is concerned with cost, highlight the *Cost* button located in the lower right of your screen (arrow 1) in Figure 4.34.

Arrow 2 in Figure 4.34 points to the *% of Population* button, which should be highlighted. The slide bar for *% of Population* (arrow 3) is positioned at the far left of the screen. The confusion matrix shows that with the slide bar in this position, all applicants are denied a credit card. This positioning does not reflect the model produced by J48. Arrow 4 points to a classification accuracy of 55.51%, which is exactly what we have if all 690 applicants are denied a credit card.

Arrow 5 points to the *cost matrix*. It currently shows that the cost for incorrectly accepting or rejecting an applicant is 1. As all 307 applicants have been rejected, the cost (arrow 6) shows 307. Move the slide bar (arrow 3) to the right until the confusion matrix matches the confusion matrix seen in the data mining session with J48 (Figure 4.32). Your screen will appear as in Figure 4.35.

This is the location where J48 chose 304 instances for the “+” class (252 correct/52 incorrect), which represents 44.058% of the total population [(252 + 52)/690].

The cost shown in Figure 4.35 is 107 units. That is, $252(0) + 55(1) + 52(1) + 331(0) = 107$.

Next, let's assume that the average cost of giving someone a credit card when they shouldn't have one is 5, whereas the average cost of not giving someone a credit card when they should have one is 2.

- Make these changes to the cost matrix (arrow 5 in Figure 4.34) to see the result in Figure 4.36. Notice that our cost has increased to 370 units.
- Given this new cost structure, we can use the cost/benefit tool to minimize our cost. Click on *Minimize Cost/Benefit* located in the lower right of your screen. The result is shown in Figure 4.37.

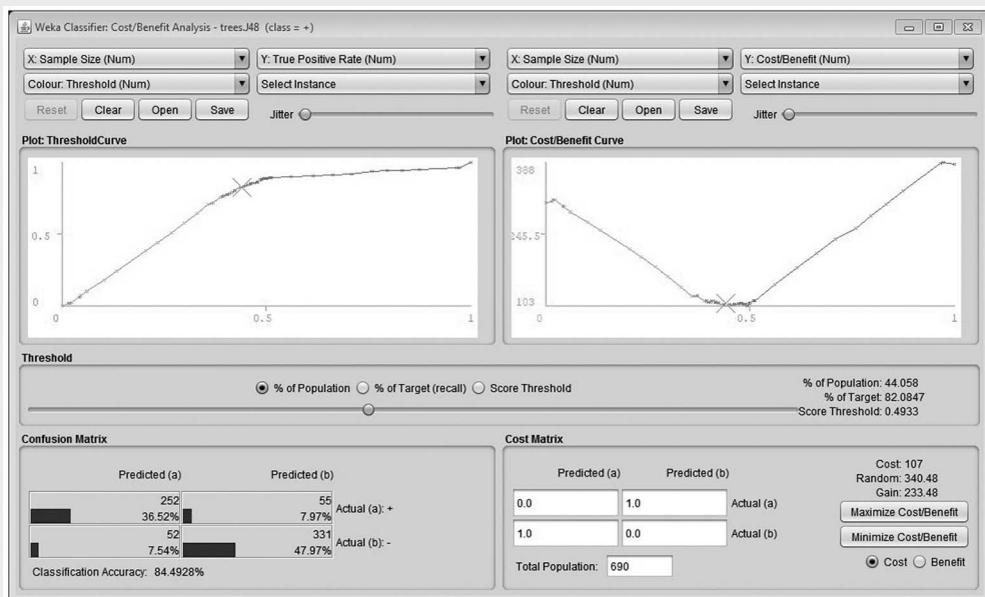


FIGURE 4.35 Cost/benefit analysis set to match J48 classifier output.

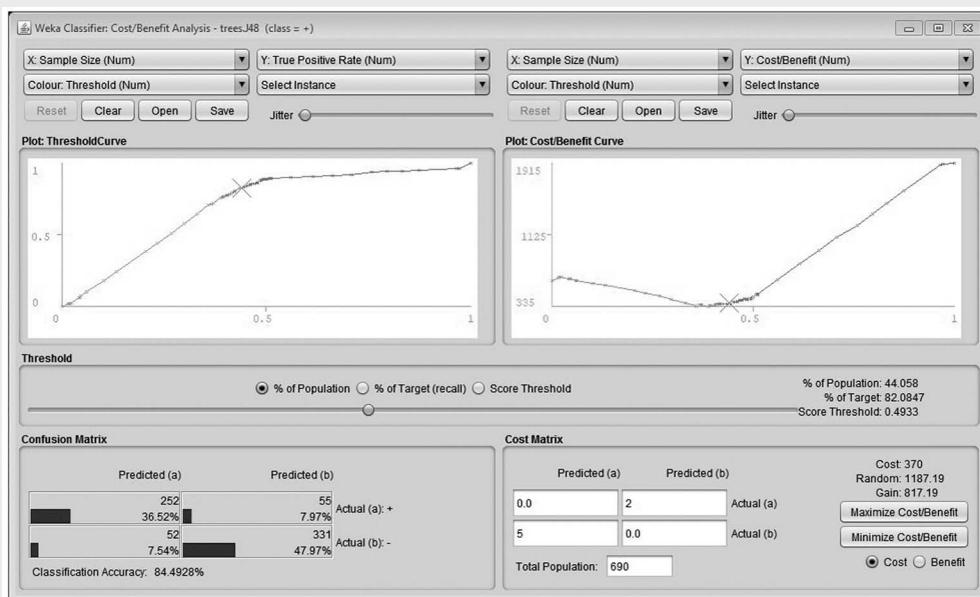


FIGURE 4.36 Invoking a cost/benefit analysis.

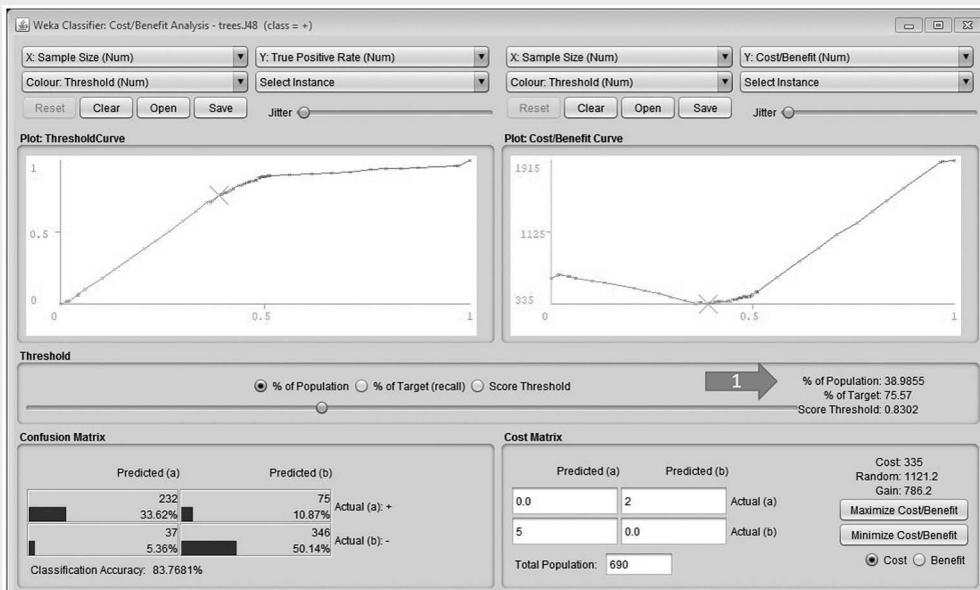


FIGURE 4.37 Minimizing total cost.

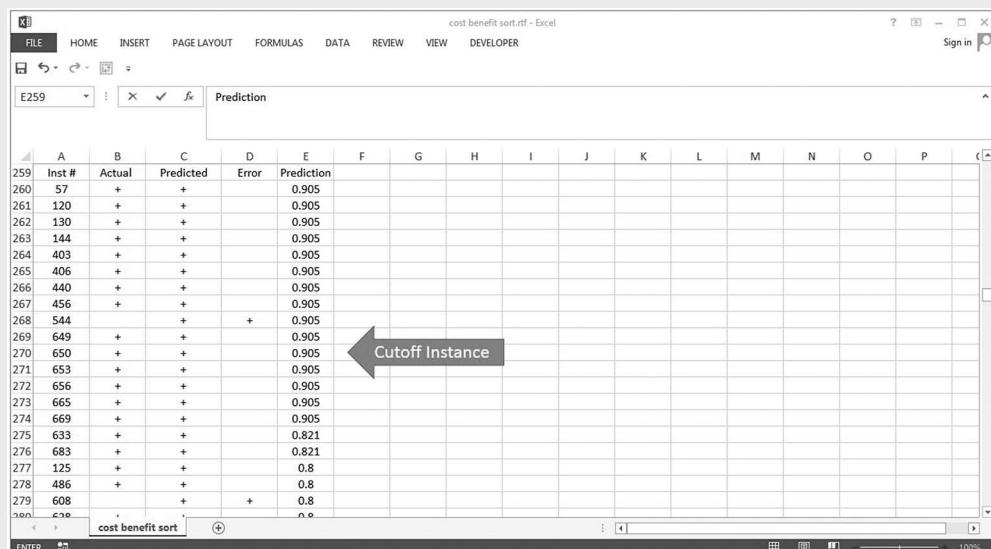
The new cost computation is $232(0) + 75(2) + 37(5) + 346(0) = 335$. The cost has decreased from 370 to 335. This has been accomplished by accepting 20 fewer applicants (232 versus 252), which in turn also gives us fewer incorrect classifications (52 versus 37). Our credit card acceptance drops from the top 44.058% of all applicants to the top 38.986% (arrow 1 in Figure 4.37) of all applicants.

Lastly, in order to apply our model, we must sort the predictions to determine the top 38.986% cutoff value. The process for finding this value is somewhat tedious. Here is an outline of the process:

- Use J48 to create a model using all of the training data. In this way, we will have a top-to-bottom listing of all confidence values.
- Copy the individual instance predictions into an Excel spreadsheet and sort the instances by their probability of belonging to the “+” class.
- Scroll the sorted list until you see the 269th positive prediction (232 correct, 37 incorrect).

Figure 4.38 shows the result of this action. Our model tells us that, using the cost matrix shown in Figure 4.37, any individual with a predicted confidence score of 90.5 or above for the “+” class should receive a credit card.

Here we presented an introduction to Weka’s cost/benefit tool. In Chapter 13, we will use the cost/benefit analysis tool’s graphical capabilities as a method to deal with imbalanced data. As time permits, it is well worth exploring the additional features of this powerful analytical tool.



A screenshot of a Microsoft Excel spreadsheet titled "cost benefit sort.xlsx". The spreadsheet contains a table of data with columns: Inst #, Actual, Predicted, Error, and Prediction. The rows show various instances with Inst # ranging from 57 to 608. An arrow points to row 269, which is highlighted in yellow and labeled "Cutoff Instance". The "Prediction" column for row 269 shows a value of 0.905.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Inst #	Actual	Predicted	Error	Prediction												
57	+	+		0.905												
120	+	+		0.905												
130	+	+		0.905												
144	+	+		0.905												
403	+	+		0.905												
406	+	+		0.905												
440	+	+		0.905												
456	+	+		0.905												
544		+	+	0.905												
649	+	+		0.905												
650	+	+		0.905												
653	+	+		0.905												
656	+	+		0.905												
665	+	+		0.905												
669	+	+		0.905												
633	+	+		0.821												
683	+	+		0.821												
125	+	+		0.8												
486	+	+		0.8												
608		+	+	0.8												

FIGURE 4.38 Cutoff scores for credit card application acceptance.

4.7 UNSUPERVISED CLUSTERING WITH THE K-MEANS ALGORITHM

In Chapter 3, we showed you how the *K*-means algorithm performs unsupervised clustering with numeric data. We can also apply strictly numeric algorithms to categorical or mixed data by first preprocessing categorical data with a nominal-to-binary filter. Weka's *NominalToBinary* filter is specifically designed for this task. However, this preprocessing is not necessary with *K*-means as the nominal-to-binary conversion as well as data normalization are built into the algorithm. Furthermore, if an output attribute is present in the data, we have the option of observing how the output values are distributed within the formed clusters. This offers a general picture of how well the input attributes define the class attribute. Let's see how the *K*-means algorithm works by applying it to the *CreditScreening.arff* data set described in the aside box titled "The Credit Card Screening Data set."

- Load *creditscreening.arff* into the Explorer and click the *Cluster* option at the top of your display. Click *Choose* and load *SimpleKMeans*.
- Highlight the *Classes to clusters evaluation* radio button—arrow in Figure 4.39—for the cluster mode option. Doing so excludes the class attribute from affecting the clustering and allows us to see how the class values are distributed within the formed clusters.
- Click the command line to invoke the *GenericObjectEditor*. Change the value of *displayStdDevs*—arrow in Figure 4.40—to *True*.

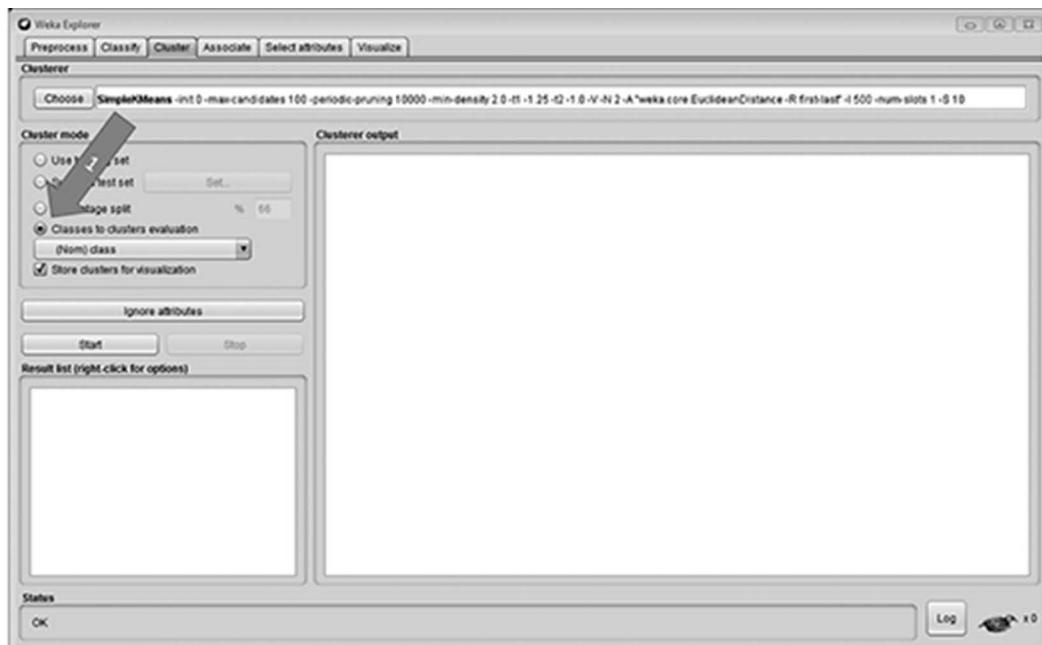


FIGURE 4.39 Classes to clusters evaluation for SimpleKMeans.

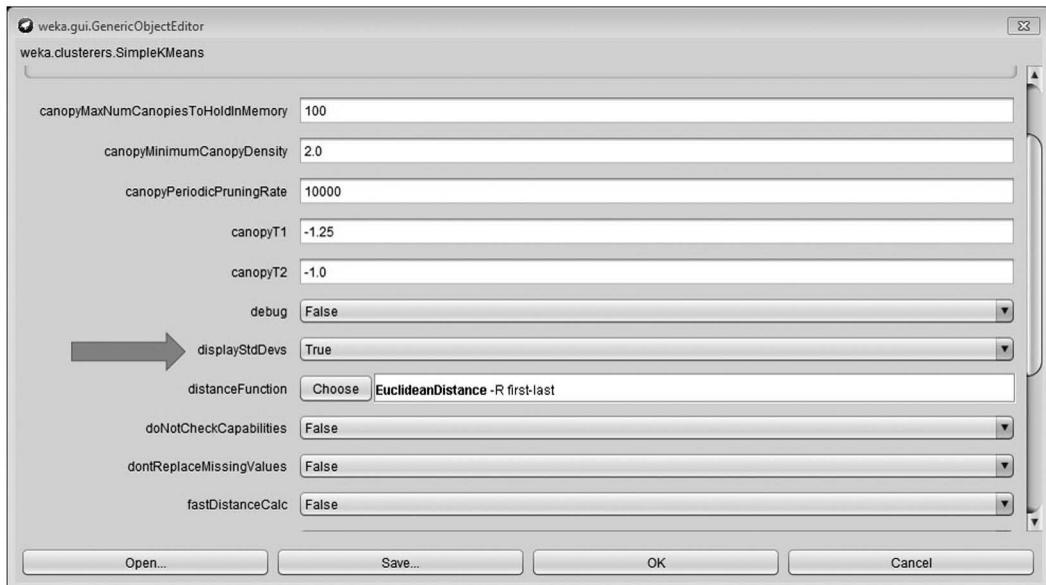


FIGURE 4.40 Include standard deviation values for SimpleKMeans.

The object editor has a default value of 2 for the number of clusters and 10 for the seed. The seed determines the initial instances chosen as cluster centers, which in turn affects the overall clustering. Recall that in Chapter 3, we stated that an optimal clustering for the K -means algorithm is frequently defined as the clustering that shows a minimum summation of squared error differences between the instances and their corresponding cluster center. As alternative seed values will result in different values for this summation, we can experiment with various settings of the seed in an attempt to obtain a best possible clustering.

- Allow the default settings to remain for the number of clusters and the seed. Click OK to close the GenericObject editor.
- Click Start to begin the SimpleKMeans clustering.

Your screen will appear as in Figure 4.41. Cluster 0 contains 48% and cluster 1 holds 52% of the instances. Of greater interest is the relationship between the clusters relative to the class attribute. Cluster 0 contains 247 of the 307 instances representing individuals whose credit card application was accepted. It also contains 82 instances whose application was rejected. Cluster 1 holds 301 of the 383 instances whose application was rejected as well as 60 instances representing accepted applicants. In summary, almost 80% of the instances clustered according to the value of their class attribute. As the class attribute is not part of the clustering, this gives us an initial indication that a supervised learner model built with a subset of the input attributes is likely to perform well.

```

Figure4.41.4.42.4.43.txt - WordPad

Time taken to build model (full training data) :
0.05 seconds

==== Model and evaluation on training set ====

Clustered Instances

0      329 ( 48%)
1      361 ( 52%)

Class attribute: class
Classes to Clusters:

0   1 <-- assigned to cluster
247 60 | +
82 301 | -

Cluster 0 <-- +
Cluster 1 <-- -
Incorrectly clustered instances : 142.0 20.5797%

```

FIGURE 4.41 Classes to clusters output.

- Scroll the output until it displays the information in Figure 4.42.

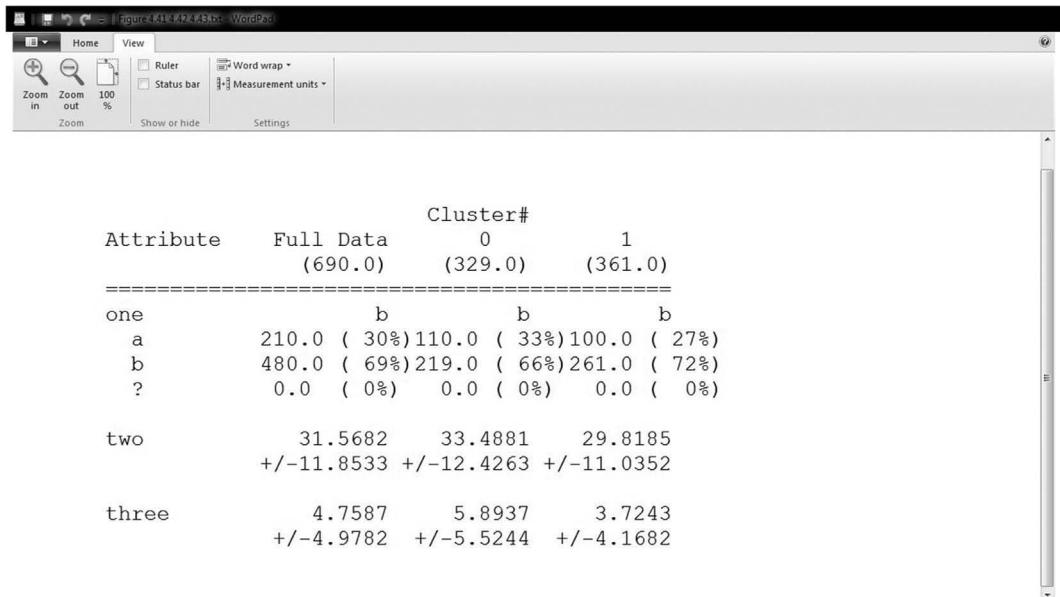
The output shows that the most common value for attribute *one* is *b*. This is the case for the full data set as well as both clusters. This tells us that attribute *one* does not differentiate the clusters or the classes.

- Next, scroll your screen until it appears as in Figure 4.43.

Consider categorical attribute *nine*. Notice that 91% of the instances with value *t* for attribute *nine* are found in cluster 0, whereas 83% of the instances with value *f* for attribute *nine* are in cluster 1. This clearly indicates that attribute *nine* helps to differentiate the clusters as well as the classes. Notice that in addition to attribute *nine*, the clusters clearly show dissimilar values for categorical attributes *ten* and *twelve*.

Mean and standard deviation values for numeric attributes are given for the domain as well as the individual clusters. We can apply a simple heuristic to measure numeric attribute significance using mean and standard deviation scores. Here's one way to compute an attribute significance score for attribute *two*.

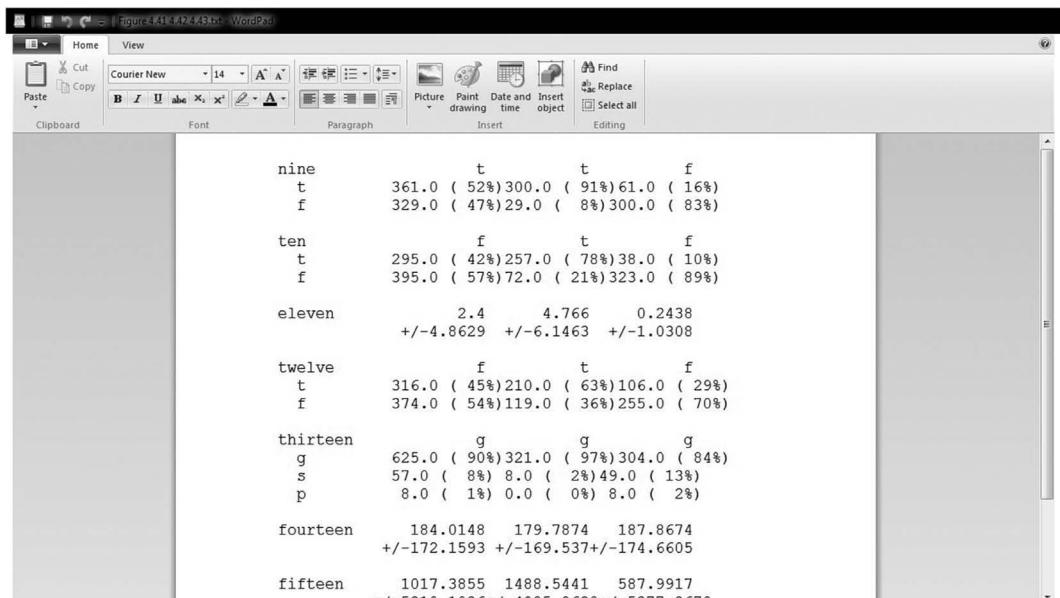
- Subtract the smallest cluster mean for attribute *two* (cluster 1 = 29.819) from the largest attribute *two* cluster mean (cluster 0 = 33.488).
- Divide this result by the domain standard deviation for attribute *two* ($sd = 11.853$).



The screenshot shows a Microsoft WordPad window with the title "Figure 4.41 4.42 4.43.txt - WordPad". The menu bar includes Home, View, and several toolbars for zoom, ruler, status bar, and measurement units. The main content area displays a table of attribute values:

Attribute	Cluster#		
	Full Data	0	1
	(690.0)	(329.0)	(361.0)
<hr/>			
one	b	b	b
a	210.0 (30%)	110.0 (33%)	100.0 (27%)
b	480.0 (69%)	219.0 (66%)	261.0 (72%)
?	0.0 (0%)	0.0 (0%)	0.0 (0%)
two	31.5682 +/-11.8533	33.4881 +/-12.4263	29.8185 +/-11.0352
three	4.7587 +/-4.9782	5.8937 +/-5.5244	3.7243 +/-4.1682

FIGURE 4.42 Partial list of attribute values for the K-means clustering in Figure 4.41.



The screenshot shows a Microsoft WordPad window with the title "Figure 4.41 4.42 4.43.txt - WordPad". The menu bar includes Home, View, and several toolbars for clipboard, font, paragraph, insert, and editing. The main content area displays a table of attribute values:

nine	t	t	f
t	361.0 (52%)	300.0 (91%)	61.0 (16%)
f	329.0 (47%)	29.0 (8%)	300.0 (83%)
<hr/>			
ten	f	t	f
t	295.0 (42%)	257.0 (78%)	38.0 (10%)
f	395.0 (57%)	72.0 (21%)	323.0 (89%)
<hr/>			
eleven	2.4 +/-4.8629	4.766 +/-6.1463	0.2438 +/-1.0308
<hr/>			
twelve	f	t	f
t	316.0 (45%)	210.0 (63%)	106.0 (29%)
f	374.0 (54%)	119.0 (36%)	255.0 (70%)
<hr/>			
thirteen	g	g	g
g	625.0 (90%)	321.0 (97%)	304.0 (84%)
s	57.0 (8%)	8.0 (2%)	49.0 (13%)
p	8.0 (1%)	0.0 (0%)	8.0 (2%)
<hr/>			
fourteen	184.0148 +/-172.1593	179.7874 +/-169.537	187.8674 +/-174.6605
<hr/>			
fifteen	1017.3855 +/-5210	1488.5441 +/-1026+	587.9917 +/-4985
			+/-5377
			2678

FIGURE 4.43 Additional attribute values for the SimpleKMeans clustering in Figure 4.41.

The result gives an attribute significance for attribute *two* of 0.310. Making this same computation for attribute *eleven* computes to a value above 0.90. The division by the standard deviation normalizes mean difference values, thus allowing us to compare the attribute significance scores for all numeric attributes. Numeric attributes with lower significance values (usually less than 0.25) will likely be of little value in differentiating one

cluster from another. If a wealth of irrelevant attributes are present, eliminating attributes of low significance and mining the data a second time can result in a clustering that more clearly differentiates the instances within the data.

Here we have used Weka's SimpleKMeans algorithm to show you how unsupervised clustering can be applied to a data set designed for supervised learning. Many additional uses of unsupervised clustering are highlighted throughout the chapters that follow.

4.8 CHAPTER SUMMARY

Data mining is an iterative process that involves a degree of art as well as science. Because of this, everyone develops their own special techniques for creating better models through data mining. This chapter introduced Weka, a powerful analytics tool containing algorithms for data preprocessing, supervised learning, clustering, association rule mining, and visualization. The data sets found in the Weka data set library represent a good starting point to help you learn about the basic data mining algorithms. Data sets not part of this library but used in this chapter are contained in the file *datasetsWeka.zip* available at our website. Links to additional data sets can be found by clicking on *datasets* listed under the *Further Information* heading at Weka's home page site.

Throughout the remainder of the text, we apply the algorithms presented here to several data sets and introduce additional algorithms for clustering, neural network modeling, and support vector machine applications. In the next chapter, we show you how to build problem-solving models with the help of RapidMiner's knowledge flow interface.

EXERCISES

1. Use PART and 10-fold cross-validation to perform a data mining session with the contact-lenses data set introduced in Section 4.2. Compare the production rules with the decision tree created by J48. Compare the confusion matrices given by applying each model. Summarize your results.
2. This exercise compares decision trees built with Weka's SimpleCart algorithm to J48. Unlike J48, SimpleCart always generates a binary decision tree. A second difference between the algorithms is in how tree pruning is accomplished. Both algorithms attempt tree simplification using *postpruning*, where selected subtrees are replaced with leaf nodes after the decision tree has been built. SimpleCart uses a technique known as minimal cost complexity pruning where tree complexity is a function of the total number of leaf nodes and test set classification error rate. J48's approach uses the training data to compute error rates seen with subtree removal.

As SimpleCart is not part of Weka's initial installation package, it must be installed. Refer to Appendix A for instructions on how to install SimpleCart. Here is the approach for comparing trees created by the two algorithms.

Open Weka's Explorer and load the *CreditCardPromotion.arff* dataset. Mouse to classify and set the output attribute as *LifeInsPromo*. Set the Test option to *Use training set*. For each scenario listed below:

- List the attributes used to build each tree.
 - Give the accuracy of each tree in classifying the training data.
 - Record the number of leaves and the size of each tree.
- a. Scenario 1: Compare the trees created using the default settings for each algorithm.
 - b. Scenario 2: Compare the trees created by setting the J48 parameter *unpruned to true* and the SimpleCart parameter *usePrune* to *false*. This turns off the tree pruning process for both techniques.
 - c. Scenario 3: J48 and SimpleCart both have a parameter denoted as *minNumObj*. The value of this parameter determines the minimum number of instances per leaf node. Compare the trees created when *minNumObj* is set at 5.
 - d. Manipulate one or several of the parameters for one or both algorithms and summarize your findings.
3. Work through the example of Section 4.3 using J48. Show the predictions for the unknown data. Compare the predictions with those seen in Figure 4.19.
4. For this exercise, use J48 and PART to perform a data mining session with the cardiology patient data described in Chapter 2. Open Weka's Explorer and load cardiologymixed.arff. This is the mixed form of the data set containing both categorical and numeric data. Recall that the data contain 303 instances representing patients who have a heart condition (sick) as well as those who do not.

Preprocess Mode Questions:

- a. How many of the instances are classified as healthy?
- b. What percent of the data is female?
- c. What is the most commonly occurring domain value for the attribute *slope*?
- d. What is the mean age within the data set?
- e. How many instances have the value 2 for number of colored vessels?

Classification Questions using J48:

Perform a supervised mining session using 10-fold cross-validation with J48 and *class* as the output attribute. Answer the following based on your results:

- a. What attribute did J48 choose as the top-level decision tree node?
- b. Draw a diagram showing the attributes and values for the first two levels of the J48-created decision tree.

- c. What percent of the instances were correctly classified?
- d. How many healthy class instances were correctly classified?
- e. How many sick class instances were falsely classified as healthy individuals?
- f. Show how true-positive rate (TP rate) and false-positive rate (FP rate) are computed.

Classification Questions using PART:

- a. List one rule for the healthy class that covers at least 50 instances.
 - b. List one rule for the sick class that covers at least 50 instances.
 - c. List one rule that is likely to show an inaccuracy rate of at least 0.05.
 - d. What percent of the instances were correctly classified?
 - e. How many healthy class instances were correctly classified?
 - f. How many sick class instances were falsely classified as healthy individuals?
5. Use a word processor or text editor to open the *soybean* data set located in the *data* subfolder of the Weka install folder. This data set represents one of the more famous data mining successes. This exercise uses this data set to explore the effect that attribute selection has on classification correctness.
- a. Scroll through the file to get a better understanding of the data set. Open Weka's Explorer and load this data set. Classify the data by applying J48 with a 10-fold cross-validation. Record your results.
 - b. Repeat your experiment using SimpleCart. Detail the differences between this result and the output in (a). Specify differences between the decision trees and their classification accuracies.
 - c. Return to preprocess mode. Apply Weka's supervised *AttributeSelection* filter to the data set using *InfoGainAttributeEval* as the evaluator and *Ranker* as the search technique. Make a table showing classification correctness values for J48 and SimpleCart when they are applied to the data set using the best 20, 15, 10, and 5 attributes. Summarize your results.
 - d. Experiment with the GenericObjectEditor *minNumObj* parameter in an attempt to create a smaller, easier to interpret tree. Summarize your results.
6. Perform experiments with the supermarket data set described in Section 4.5 by modifying the parameter settings for the Apriori algorithm in an attempt to find interesting association rules. Summarize the results of your experiments.
7. For this exercise you will experiment with *SimpleKMeans* using the mixed form of the cardiology patient dataset. Recall the K-means algorithm requires numeric data

but automatically makes the nominal to binary numeric conversions for you. Here's what to do:

- a. Load *CardiologyMixed.arff* into the Explorer and mouse to *Cluster*. Choose *SimpleKMeans* and highlight *Classes to clusters evaluation*. Be sure the parameter *numClusters* is set at its default value of 2. Cluster the data. Record the number and percent of incorrectly clustered instances as well as the specific instances used as the initial cluster centers.
 - b. Repeat part (a) but first open the GenericObjectEditor and change *initialization Method* from *Random* to *Farthest First*. *Random* randomly chooses one instance to represent the first cluster and a second instance for the second cluster. *Farthest First* also randomly selects the first instance. However, the second instance is chosen to be as far distant as possible from the initial selection.
 - c. Summarize your results.
8. Load *spam.arff* into the Explorer. Use IBk to mine the data and record the classification correctness.
- a. Use *InfoGainAttributeEval* together with *ranker* to rank the attributes from most to least predictive of class membership. Delete all but the 10 *least* predictive attributes. Apply IBk together with a 10-fold cross-validation. Compare the classification correctness with the value seen when all attributes are applied.
 - b. Repeat the experiment but remove all but the five *least* predictive attributes. Summarize your results. Are the results in line with what you expected?
9. Choose a data set from <http://repository.seasr.org/Datasets/UCI/arff/>. Choose an appropriate data mining algorithm to mine the data. Write a short report detailing the contents of the file as well as the results of your experiment(s).
10. Apply PART together with Weka's cost/benefit analyzer to the spam data set to determine a least cost classification based on the information given below. Here is the procedure.
- a. Load the spam data set. Select PART as the classifier. Select *Use training set* as the *Test option*. Under *More options*, mouse to *Output predictions* then *Choose Plain Text*. Click Start to mine the data.
 - b. Perform the cost/benefit analysis in terms of the spam class (output =1). Set the cost of an incorrect classification of a valid email at 5 and the cost of a spam message classified as valid at 1.
 - c. Apply the cost/benefit analysis using the *Minimize Cost/Benefit* option.
 - d. Save the individual predictions together with their probabilities to an Excel file.

Sort the file to determine the minimum probability value necessary for a message to be classified as spam.

Knowledge Discovery with RapidMiner

CHAPTER OBJECTIVES

- *Understand the basic object and workflow structure of RapidMiner*
- *Know how to use RapidMiner for preprocessing data*
- *Know how to use RapidMiner for building supervised learner models*
- *Know how to perform unsupervised clustering with RapidMiner*
- *Know how to use RapidMiner to create association rules*
- *Know about RapidMiner's template interface*

RapidMiner is a well-known commercial data mining and predictive analytics tool developed by a company with the same name. In this chapter, we introduce RapidMiner Studio is an easy-to-use, open-source, and code-free version of RapidMiner's commercial product. RapidMiner uses a workflow paradigm for building models to solve complex problems. RapidMiner's software platform offers several options for interacting with its interface. We leave exploring these various options to you. Here we concentrate on showing you how to use RapidMiner to design and execute workflows that solve problems with the techniques discussed in this text.

Fundamental to RapidMiner's structure is the operator. Operators are used to preprocess, visualize, transform, create, evaluate, and optimize models. To get you started with RapidMiner Studio, we concentrate on a basic subset of its 1500-plus operators. Once familiar with this subset of operators, you will find it easy to add interesting operators to your list. In Section 5.1, we introduce you to RapidMiner Studio and show you how to write your first process. The focus of Section 5.2 is on decision trees and model evaluation. Section 5.3 is all about generating rules. The topic of Section 5.4 is association rule learning. Section 5.5 concentrates on unsupervised clustering with the *K*-means algorithm. In Section 5.6,

the emphasis is on attribute selection. The end-of-chapter exercises help you better understand how RapidMiner can create generalized models from data.

5.1 GETTING STARTED WITH RAPIDMINER

5.1.1 Installing RapidMiner

Go to website <http://docs.rapidminer.com/studio/> and click on *Installation* for instructions on downloading and installing the latest version of RapidMiner Studio. This website also contains links to the RapidMiner Studio manual, operator reference guide, tutorials, and reference notes. During the installation process, you will have the option to create a free RapidMiner user account. By doing so, you join the RapidMiner community, which offers several benefits, including a community forum and support.

RapidMiner accepts and processes data in several formats. Most of the data files you will be using here are contained in *datasetsRapidMiner.zip*. These files are in Microsoft (MS) Excel format, with the first row containing the names of the attributes represented in the data. The section titled “Data Sets for Data Mining” in Appendix A lists several locations for downloading this file and other supplementary materials. Please download and unzip this file to a folder of your choice as we will be using several of these files for the tutorials just ahead. Further, most of the figures in this chapter are screenshot enhancements designed for readability. We encourage you to download and make use of *Screens_05* (MS PowerPoint or PDF format), which contains screenshots showing what you will see on your screen as you work through the tutorials. Let’s get started!

5.1.2 Navigating the Interface

Use the RapidMiner Studio shortcut on your desktop to open RapidMiner. Your screen will appear similar to either Figure 5.1 or Figure 5.2. A click on *Get Started* in Figure 5.2 gives Figure 5.1, and a click on *New Process* in Figure 5.1 gives Figure 5.2. Figure 5.2 shows that RapidMiner offers a choice of eight templates (plus a blank process template) from which to begin your analytics process. The templates give examples directed toward specific application areas. We will examine one of these templates later in this chapter. Here, our goal is to learn how to use the workflow structure offered by RapidMiner to solve problems of our own choosing. Therefore, let’s start by creating a new blank process.

- Click on *New Process* and then on *Blank*.

Your screen will appear as in Figure 5.3. Ten pointers have been inserted to help us explain RapidMiner’s main interface. Before providing you with an explanation of the interface, we must mention the free extensions available with RapidMiner Studio. In Figure 5.3, right above arrow 8, click on *Extensions*, then *Marketplace (Updates and Extensions)*, and finally on *Top Downloads*. At some point, install the *Text Processing*, *Web mining*, *Weka*, *Finance and Economics*, and *Text Analysis by Aylien* extensions. These extensions will be useful once the basics of RapidMiner are in place.

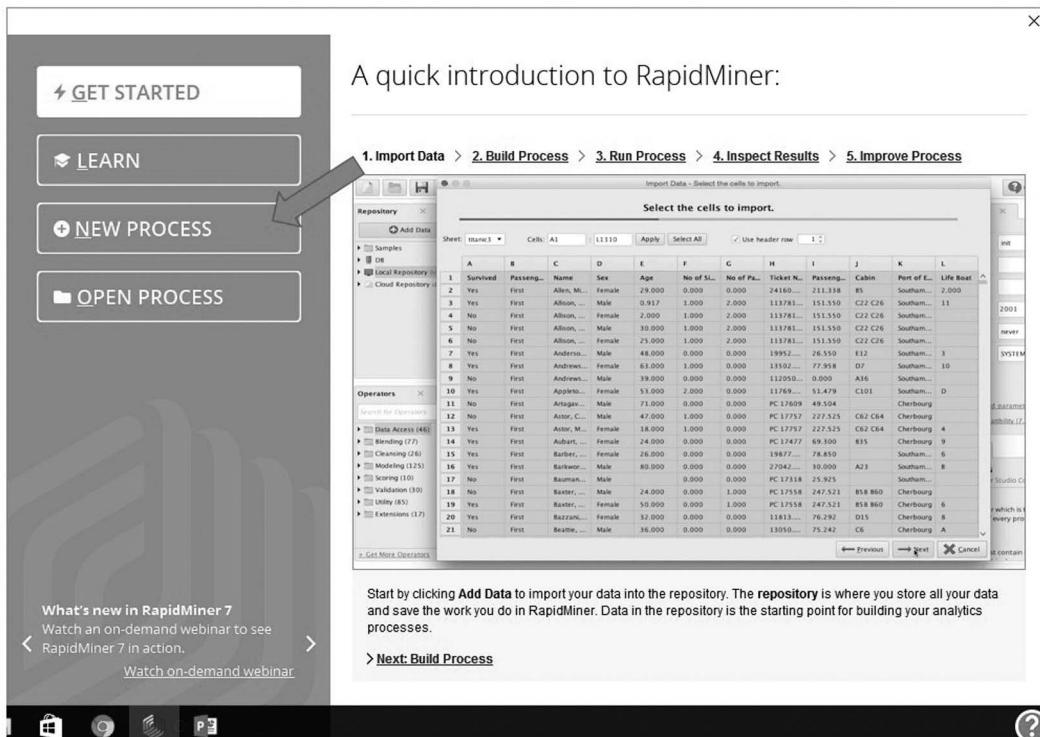


FIGURE 5.1 An introduction to RapidMiner.

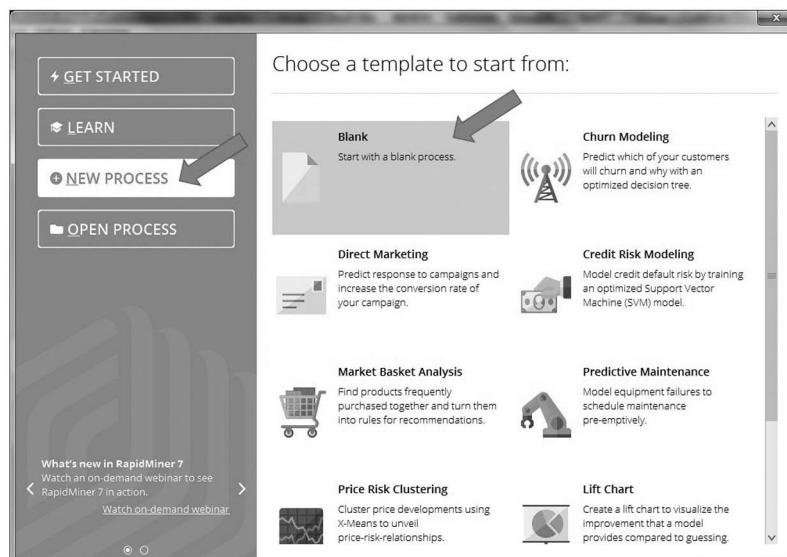


FIGURE 5.2 Creating a new blank process.

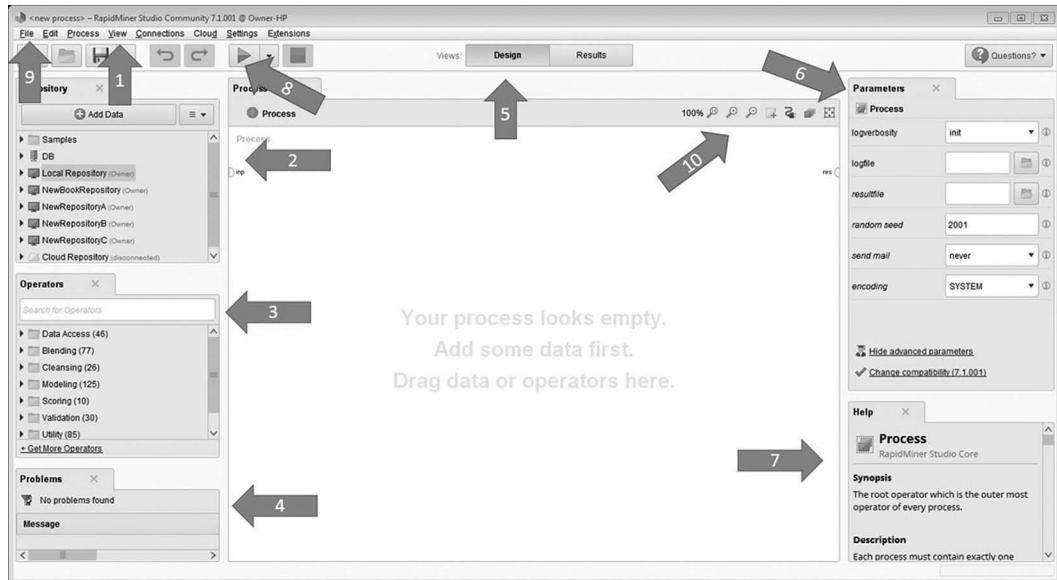


FIGURE 5.3 A new blank process with helpful pointers.

It's time for our premium around-the-horn tutorial! Referring to the pointers in Figure 5.3,

1. One of our main goals is to keep you from getting into too much trouble prior to learning the ins and outs of RapidMiner. The *Restore Default View* option is a great aid in achieving this goal. Any time your screen becomes difficult to interpret, click on *View* and then *Restore Default View*.
2. All input and output take place here. The items in the *Samples* folder are very helpful as they contain a wealth of information for your use in the form of data files, processes, tutorials, and the like. Below the samples folder, you will see a local repository. Double-click on *Local Repository* to see a *data* folder and a *processes* folder. Typically, the data folder will contain your data files, and the processes folder will contain the workflows of operators and their connections you develop for your applications. RapidMiner allows you to create as many local repositories as you desire. Local repositories reside on your computer and can be easily transferred to other computers.
3. The heart and soul of RapidMiner resides in its 1500-plus operators. Arrow 3 directs you to the folders containing these operators. Operators are used to perform all workflow tasks. The easiest way to find an operator is to type what you think might be the operator's name or category into the search box.
4. The fourth arrow directs you to any error messages encountered in your work. By default, the *Problems* interface does not appear. This is the case as major processing errors terminate the execution of a process and a balloon with an explanatory message appears. To have the *Problems* interface show on your screen, click

View, Show Panel, and then Problems. A click on *Restore Default View* removes the *Problems* interface.

5. The fifth arrow is directed to the current view. We can conveniently toggle between the *Design* and *Results* views. As we have yet to execute a process, a click on *Results* gives a blank screen.
6. An operator may accept zero or more parameters. A click on an operator in the main process screen allows us to modify its parameter settings.
7. Anytime we highlight an operator in our process screen, the corresponding operator documentation appears in the process window. We can conveniently read about the details of a given operator here rather than in the user manual. We can expand, contract, or delete this and all windows within the main process screen as we desire. Remember, *Restore Default View* puts everything back in its proper place.
8. The eighth arrow is directed to the arrow used to execute a process. We can terminate process execution at any time with a click on the square to the right of this arrow.
9. The ninth arrow takes us to *File*, where we create, open, and save processes. *Immediately after creating a new blank process and prior to loading any operators into the main process window, it is to our advantage to save the blank process.* Although this isn't necessary, doing so eliminates warning messages about possible limitations on the portability of the input data.
10. The tenth and final pointer directs us to icons for zooming the process window, creating comment blocks, automatically connecting operators, and altering the order of operator execution.

With pointers in hand, you know enough about the workings of the main process window to create your first process. Let's get started!

5.1.3 A First Process Model

For your first process, we return to the credit card promotion data set first defined in Chapter 2. Be sure your main process window is open. The initial task is to create and save a new blank process.

- Click on *File* and then *Save Process as*.
- Double click on *Local Repository* and then click *Processes*. Give your process a name; we gave our process the name *MyFirstProcess*. Your screen will appear similar to Figure 5.4. Click *OK* to create and save your process.
- The next step is to import some data. Per Figure 5.3, click on *Add Data* and then on *My Computer*. Navigate to the folder you created to store your MS Excel data files. Locate and select *CreditCardPromotion.xlsx*. Your screen will appear similar to Figure 5.5.

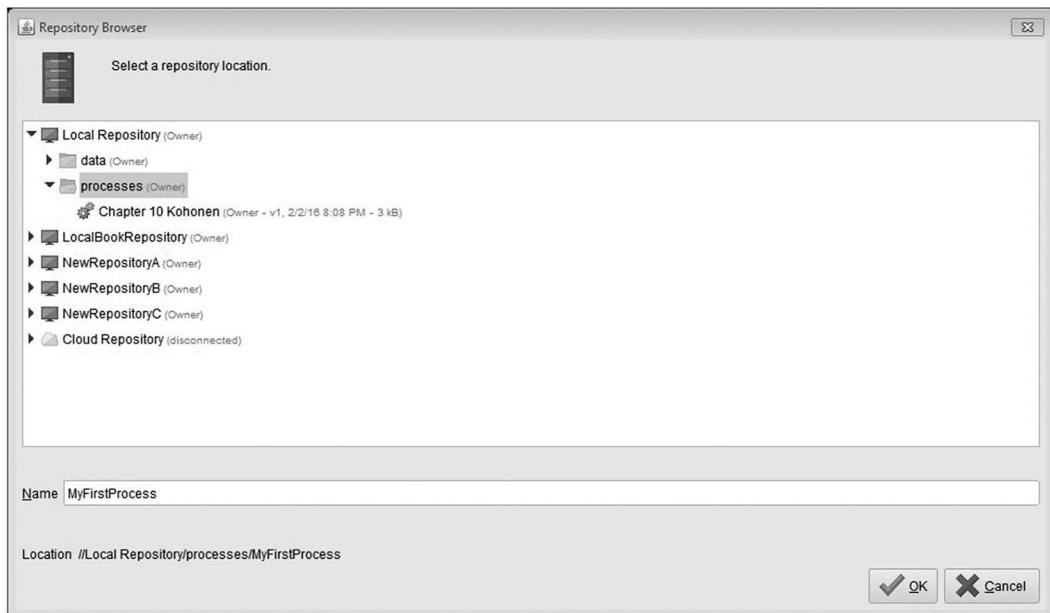


FIGURE 5.4 Creating and saving a process.

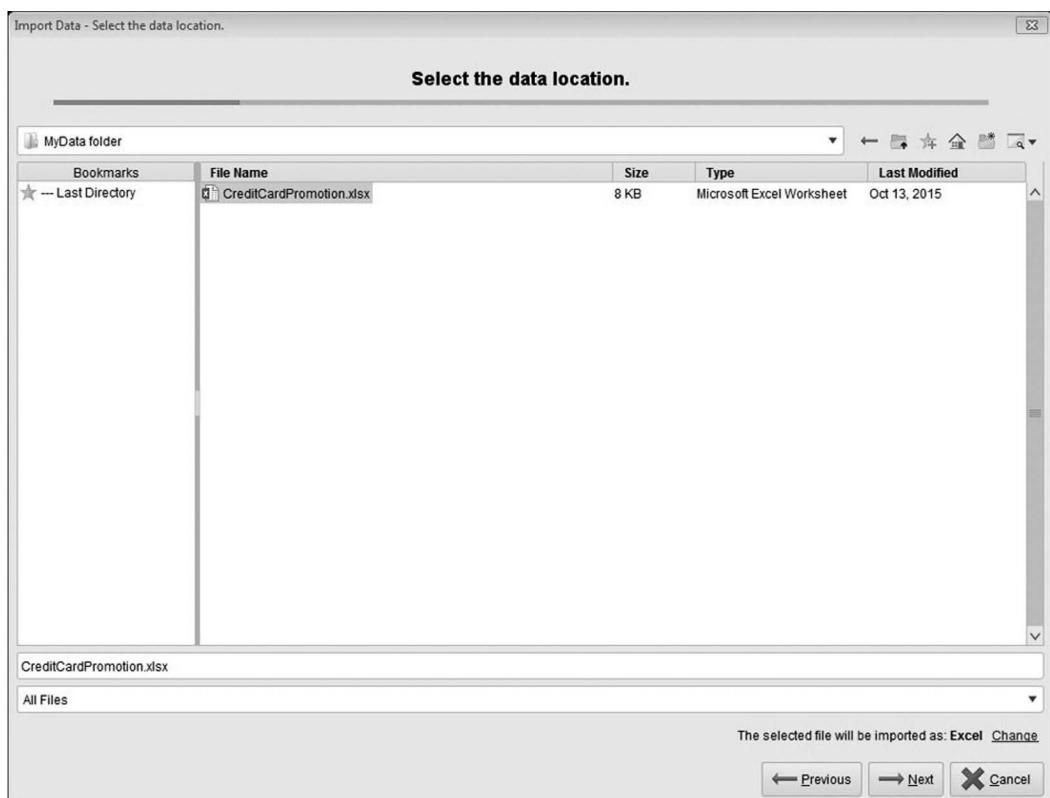


FIGURE 5.5 Importing the credit card promotion database.

- Click *Next*. Your screen will appear as in Figure 5.6. Notice that the *Define header row* is checked and the cell range correctly specifies A through G.
- Click *Next*. Figure 5.7 displays the result seen after clicking on the small downward arrow to the right of *Income Range* and highlighting *Change Type*. The types listed in Figure 5.7 represent the range of allowable data types. *Polynomial* is another way to specify that data are nominal or categorical. *Binomial* is of a polynomial type allowing but two possible attribute values. Notice that all attributes excepting age default to *polynomial*. We can change the data types of individual attributes as needed.
- Next, mouse to *Life Ins Promo*; click the downward arrow and then *Change Role*. By default, the role of each attribute is *regular*. The role of an output attribute must be designated as *label*. This can be done here or in the main process window with the *Set Role* operator. As our output attribute is known to be *Life Ins Promo*, let's change it here. Figure 5.8 shows the process. Make the label change, click *OK*, and then click *Next*.
- Figure 5.9 informs us that we must specify a repository location and name for our file. As we have created a new process, our current location is the process folder of the local repository. We want our data file to be located in the data folder. Use your mouse to highlight the *data folder*, give the file a name—we chose *CreditCardPromotion*—and click *Finish*. This action displays the *file contents* file as seen in Figure 5.10.
- Click on *Design* to return to the main process window displayed in Figure 5.11.

Import Data - Select the cells to import.

Select the cells to import.

Sheet	Sheet1 ▾	Cell range:	A:G	Select All	<input checked="" type="checkbox"/> Define header row:	1	
1	Income Range	Magazine Promo	Watch Promo	Life Ins Promo	Credit Card Ins.	Gender	Age
2	40-50,000	Yes	No	No	No	Male	45.000
3	30-40,000	Yes	Yes	Yes	No	Female	40.000
4	40-50,000	No	No	No	No	Male	42.000
5	30-40,000	Yes	Yes	Yes	Yes	Male	43.000
6	50-60,000	Yes	No	Yes	No	Female	38.000
7	20-30,000	No	No	No	No	Female	55.000
8	30-40,000	Yes	No	Yes	Yes	Male	35.000
9	20-30,000	No	Yes	No	No	Male	27.000
10	30-40,000	Yes	No	No	No	Male	43.000
11	30-40,000	Yes	Yes	Yes	No	Female	41.000
12	40-50,000	No	Yes	Yes	No	Female	43.000
13	20-30,000	No	Yes	Yes	No	Male	29.000
14	50-60,000	Yes	Yes	Yes	No	Female	39.000
15	40-50,000	No	Yes	No	No	Male	55.000
16	20-30,000	No	No	Yes	Yes	Female	19.000

← Previous → Next X Cancel

FIGURE 5.6 Selecting the cells to import.

Format your columns.

Date format: MMM d, yyyy h:mm:ss a z Replace errors with missing values ⓘ

	Income Range	Magazine Pro...	Watch Promo	Life Ins Promo	Credit Card Ins.	Gender	Age
	polynomial	polynomial	polynomial	polynomial	polynomial	polynomial	integer
1	40-50,000	Yes	No	No	No	Male	45
2	30-40,000	Yes	Yes	Yes	No	Female	40
3	40-50,000	No	Yes	No	No	Male	42
4	30-40,000	Yes	Yes	Yes	Yes	Male	43
5	50-60,000	Yes	Yes	Yes	No	Female	38
6	20-30,000	No	Yes	No	No	Female	55
7	30-40,000	Yes	No	Yes	Yes	Male	35
8	20-30,000	No	Yes	No	No	Male	27
9	30-40,000	Yes	No	No	No	Male	43
10	30-40,000	Yes	Yes	Yes	No	Female	41
11	40-50,000	No	Yes	Yes	No	Female	43
12	20-30,000	No	Yes	Yes	No	Male	29
13	50-60,000	Yes	Yes	Yes	No	Female	39
14	40-50,000	No	Yes	No	No	Male	55
15	20-30,000	No	No	Yes	Yes	Female	19

(✓) no problems. ← Previous Next → ✖ Cancel

FIGURE 5.7 A list of allowable data types.

Import Data - Format your columns.

Format your columns.

Date format: MMM d, yyyy h:mm:ss a z Replace errors with missing values ⓘ

	Income Range	Magazine Pro...	Watch Promo	Life Ins Promo	Credit Card Ins.	Gender	Age
	polynomial	polynomial	polynomial	polynomial	polynomial	polynomial	integer
1	40-50,000	Yes	No	No	No	Male	45
2	30-40,000	Yes	Yes	Yes	No	Female	40
3	40-50,000	No	Yes	Yes	Yes	Female	42
4	30-40,000	Yes	Yes	Yes	Yes	Male	43
5	50-60,000	Yes	Yes	Yes	Yes	Female	38
6	20-30,000	No	Yes	Yes	Yes	Male	55
7	30-40,000	Yes	Yes	Yes	Yes	Female	35
8	20-30,000	No	Yes	Yes	Yes	Male	27
9	30-40,000	Yes	Yes	Yes	Yes	Female	43
10	30-40,000	Yes	Yes	Yes	Yes	Male	41
11	40-50,000	No	Yes	Yes	Yes	Female	43
12	20-30,000	No	Yes	Yes	Yes	Male	29
13	50-60,000	Yes	Yes	Yes	Yes	Female	39
14	40-50,000	No	Yes	No	No	Male	55
15	20-30,000	No	No	Yes	Yes	Female	19

Change role

Please enter the new role:

OK Cancel

(✓) no problems. ← Previous Next → ✖ Cancel

FIGURE 5.8 Changing the role of *Life Ins Promo*.

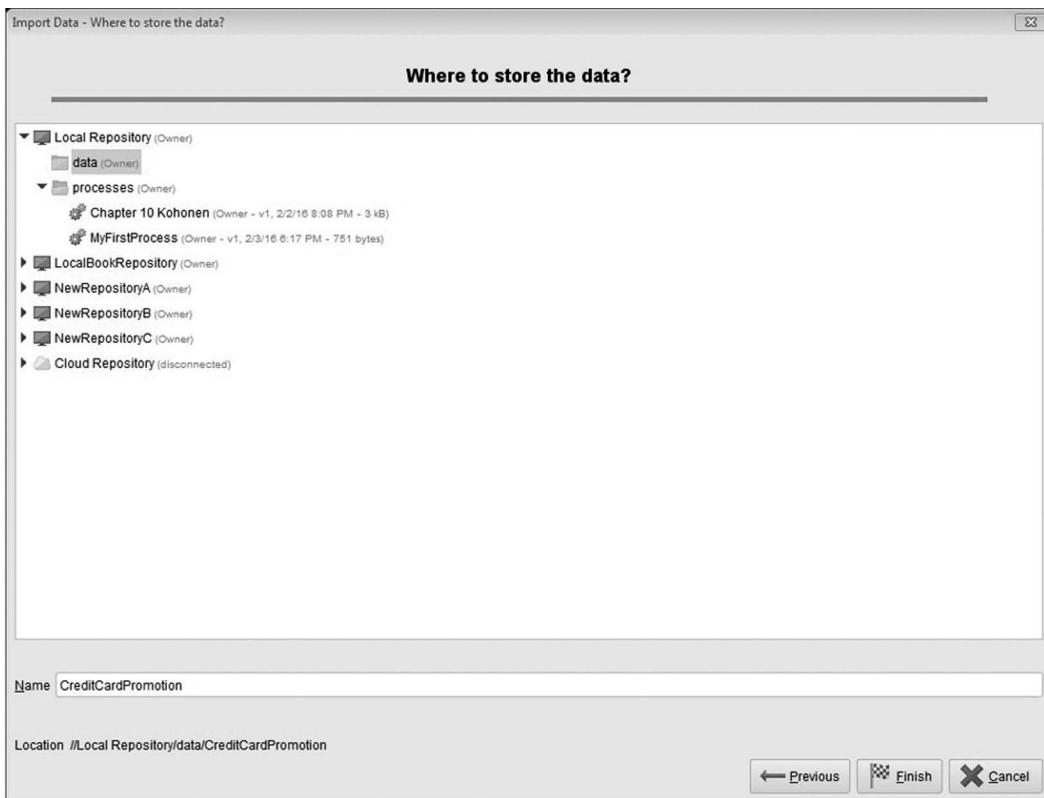


FIGURE 5.9 Storing a file in the data folder.

Row No.	Life Ins Pro...	Income Ran...	Magazine Pr...	Watch Promo	Credit Card L...	Gender	Age
1	No	40-50,000	Yes	No	No	Male	45
2	Yes	30-40,000	Yes	Yes	No	Female	40
3	No	40-50,000	No	No	No	Male	42
4	Yes	30-40,000	Yes	Yes	Yes	Male	43
5	Yes	50-60,000	Yes	No	No	Female	38
6	No	20-30,000	No	No	No	Female	55
7	Yes	30-40,000	Yes	No	Yes	Male	35
8	No	20-30,000	No	Yes	No	Male	27
9	No	30-40,000	Yes	No	No	Male	43
10	Yes	30-40,000	Yes	Yes	No	Female	41
11	Yes	40-50,000	No	Yes	No	Female	43
12	Yes	20-30,000	No	Yes	No	Male	29
13	Yes	50-60,000	Yes	Yes	No	Female	39
14	No	40-50,000	No	Yes	No	Male	55
15	Yes	20-30,000	No	No	Yes	Female	19

FIGURE 5.10 The credit card promotion database.

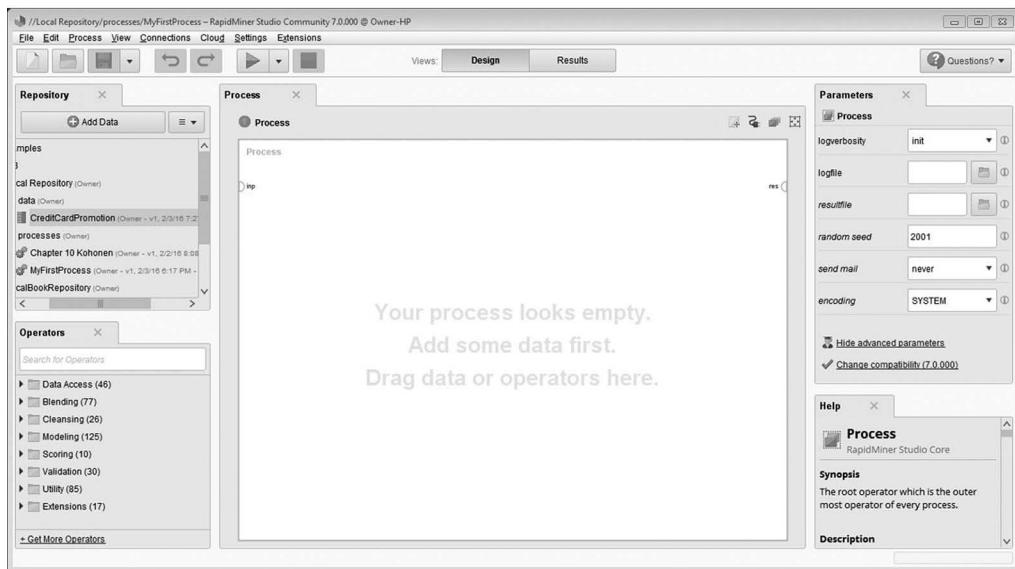


FIGURE 5.11 A successful file import.

The main process window shows two ports pictured as half circles, one for input (*inp*) and one for output (*res*). Notice that the CreditCardPromotion file is listed under the *data* folder in the local repository.

- The *Retrieve* operator is used to bring the file into the main process window. To invoke the *Retrieve* operator, use your mouse to drag the CreditCardPromotion into the main process window.
- Click and hold the *out* port given on the right side of the *Retrieve* operator to make a connection between *out* and *res* as in Figure 5.12. Notice that a second *res* port

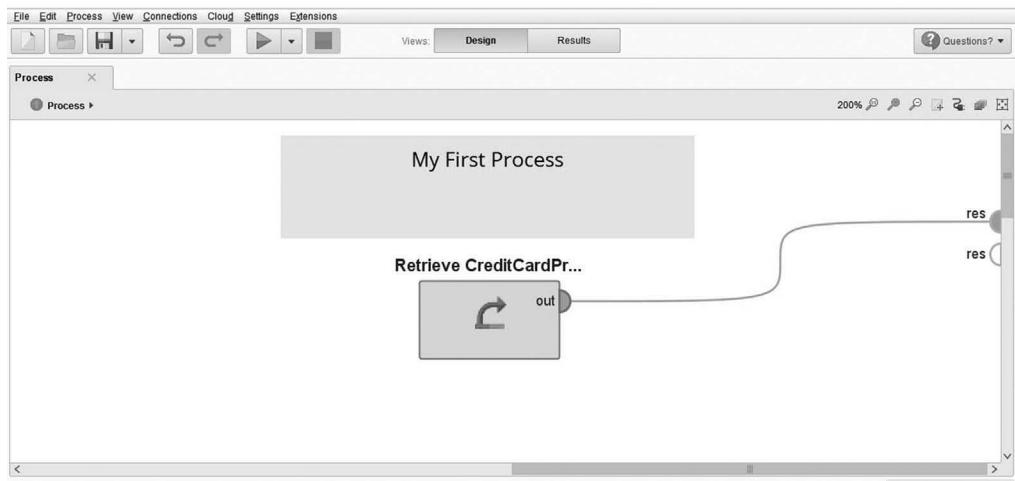


FIGURE 5.12 Connecting the credit card promotion database to an output port.

appears for future use. Click on the arrow (Figure 5.3 arrow 8) to run your process. The result simply displays the contents of the data set as shown in Figure 5.10.

It is to your advantage to become comfortable with the interface by toggling between *Design*, *Results*, *Results History*, as well as all of the data display options presented on the left of your screen. Let's take a look at the data display options.

- On the *Results* screen, click on *Statistics* to see the summary statistics in Figure 5.13.
- Click on *Charts* to see the bar graph for *Income Range* given in Figure 5.14.

Label					
Life Ins Promo	Polynomial	0	Least No (6)	Most Yes (9)	Values Yes (9), No (6)
Income Range	Polynomial	0	Least 50-60,000 (2)	Most 30-40,000 (5)	Values 30-40,000 (5), 20-30,000 (4), ...[2 more]
Magazine Promo	Polynomial	0	Least No (7)	Most Yes (8)	Values Yes (8), No (7)
Watch Promo	Polynomial	0	Least No (7)	Most Yes (8)	Values Yes (8), No (7)
Credit Card Ins.	Polynomial	0	Least Yes (3)	Most No (12)	Values No (12), Yes (3)
Gender	Polynomial	0	Least Female (7)	Most Male (8)	Values Male (8), Female (7)
Age	Integer	0	Min 19	Max 55	Average 39.600

FIGURE 5.13 Summary statistics for the credit card promotion database.

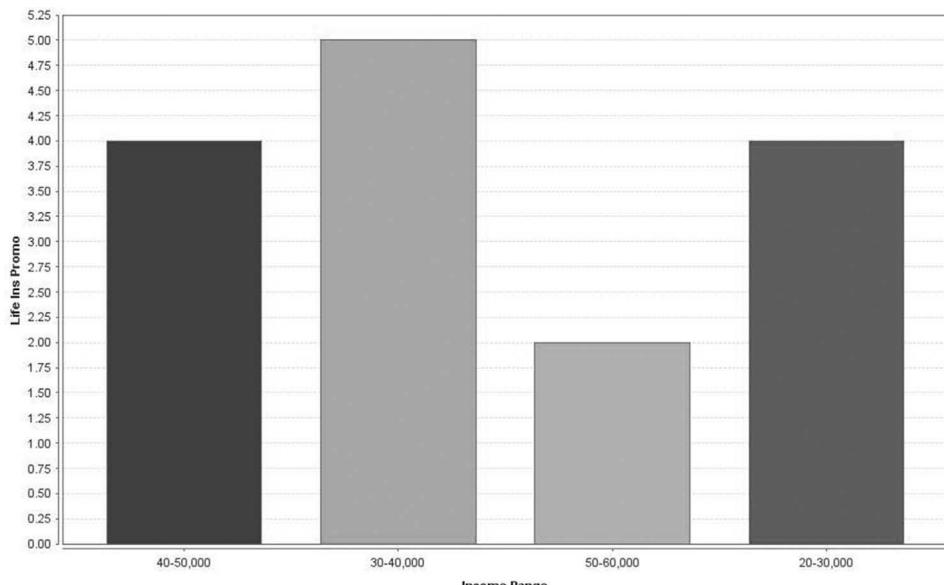


FIGURE 5.14 A bar graph for income range.

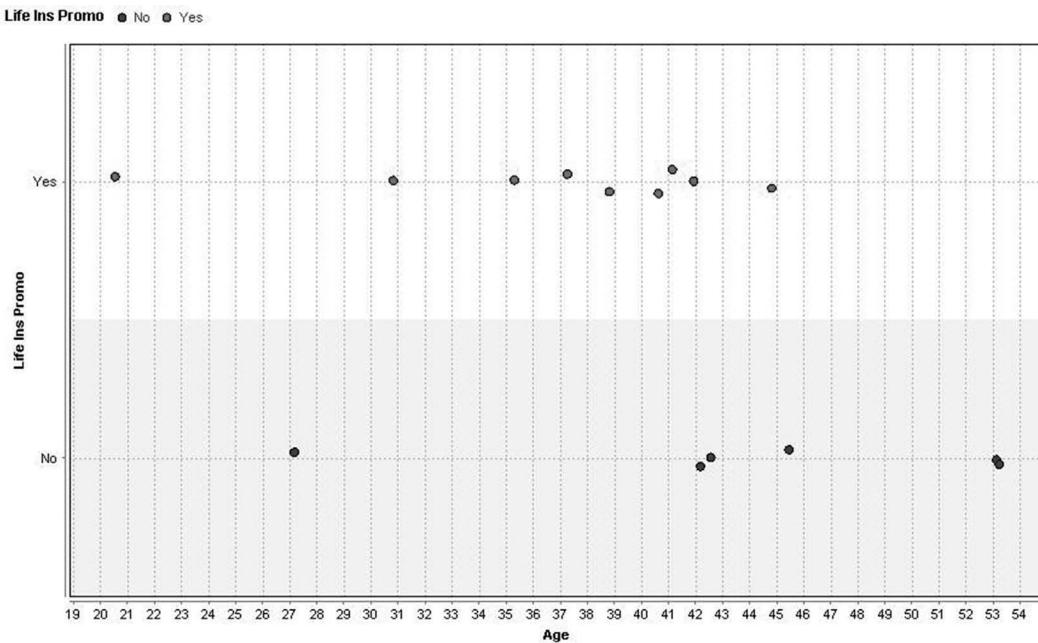


FIGURE 5.15 A scatterplot comparing age and life insurance promotion.

Our options for creating charts are unlimited. Scatterplot charts are especially useful for visualizing possible correlations between real-valued attributes. We can also use scatterplots to examine other relationships.

- Choose the *Scatter* chart style with *x*-axis *Age* and *y*-axis *Life Ins Promo* and look for possible relationships between *Age* and the output attribute.

Figure 5.15 shows the scatterplot diagram. Notice that almost everyone under the age of 44 said yes to the life insurance promotion. This indicates that *Age* is a viable candidate for a decision tree created from these data. Let's find out!

5.1.4 A Decision Tree for the Credit Card Promotion Database

- Click on *Design* to return to the main process screen and enter *Decision* in the operator search box to locate the *Decision Tree* operator.
- Drag the operator into the main process window and make the connections shown in Figure 5.16 as follows:
 - First, right-click on the current link between *out* and *res* and confirm its removal.
 - Connect the output of the *Retrieve* operator to the training data input port (*tra*) of the *Decision Tree* operator.
 - Connect the *mod* (model) port to *res*. When the process executes, the decision tree model will pass through the *mod* port to the output port.

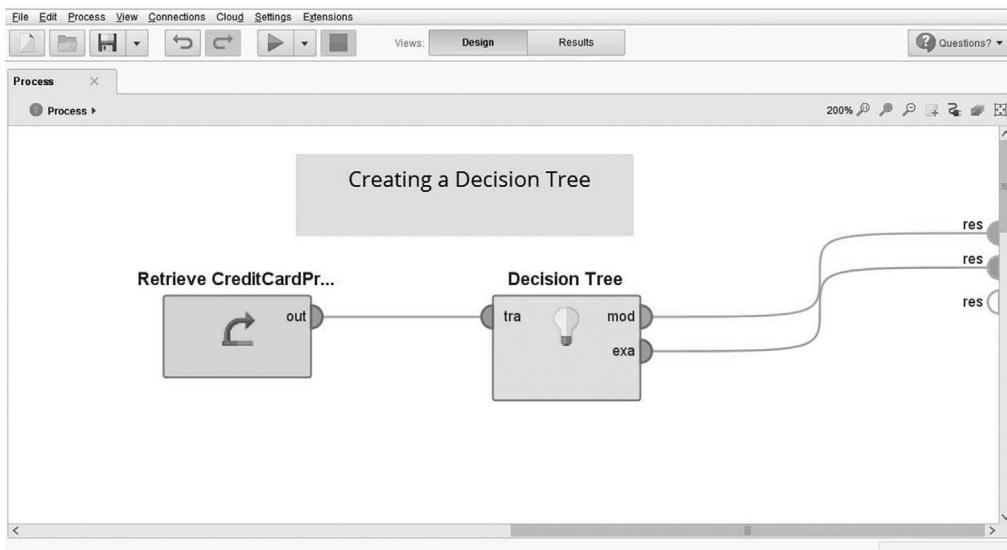


FIGURE 5.16 A decision tree process model.

- Connect the *exa* (example set) to the second output port. When the process executes, the example set will pass through the *Decision Tree* operator unchanged to the second output port.

The parameters window shows that the decision tree defaults to the gain ratio computation described in Chapter 3. With the parameters set at their default values, it's time to execute our process.

- Click the run process *arrow*. The output displays the data in their original form as in Figure 5.10. This output comes by way of the link between *exa* and *res*.
- To see the decision tree (*mod-to-res* link), look at the top of your screen and click on *Tree (Decision Tree)*. This gives the decision tree seen in Figure 5.17. This is the same decision tree we saw in Figure 3.4 of Chapter 3 with *Age* as the top-level decision tree node. Toggle between *Example Set (Retrieve CreditCardPromotion)*, *Result History*, and the *Tree (Decision Tree)* to become more familiar with the options within the *results* view.
- Return to the decision tree. Click on *Description* to see Figure 5.18, which gives the tree in textual format. The branch stating *Credit Card Ins. = No: No {No = 3, Yes = 1}* tells us that four data instances follow this path. Three instances are correctly identified as individuals who did not accept the life insurance promotion. One individual who accepted the life insurance promotion is incorrectly identified as a rejection.

We can remove the link between *exa* and *res* to limit our output to the decision tree. Let's remove the link.

- Click on *Design* then right-click the link between *exa* and *res* and confirm its removal. Run your process again to see the decision tree.

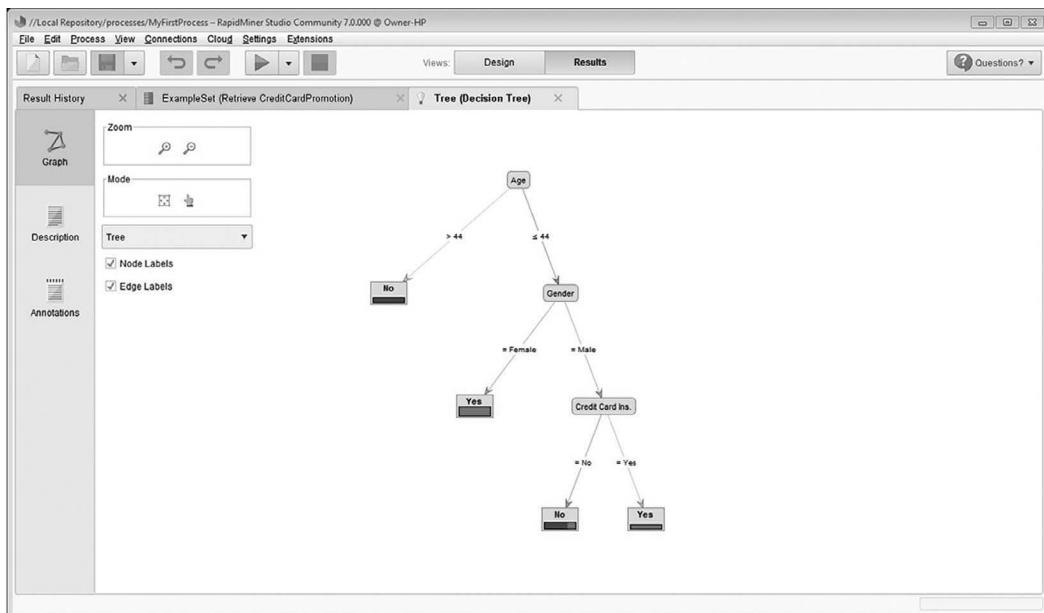


FIGURE 5.17 A decision tree for the credit card promotion database.

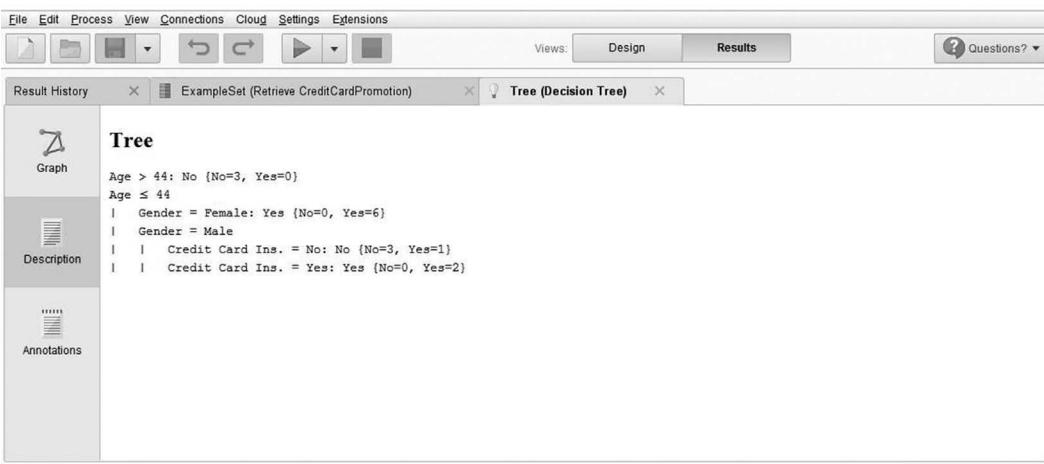


FIGURE 5.18 A decision tree in descriptive form.

5.1.5 Breakpoints

As problems and their solutions become more complex, it is important for us to have the option of observing the results of applying each operator. We can do this with the help of breakpoints. To see this, return to your process. Right-click on the *Decision Tree* operator to see the options given in Figure 5.19. Toward the bottom of the options list, you will find *Breakpoint Before* and *Breakpoint After*. By setting a breakpoint before, the process interrupts prior to operator execution. This allows us to see the information as it enters the operator. The breakpoint after shows us the result of applying the operator to the data.

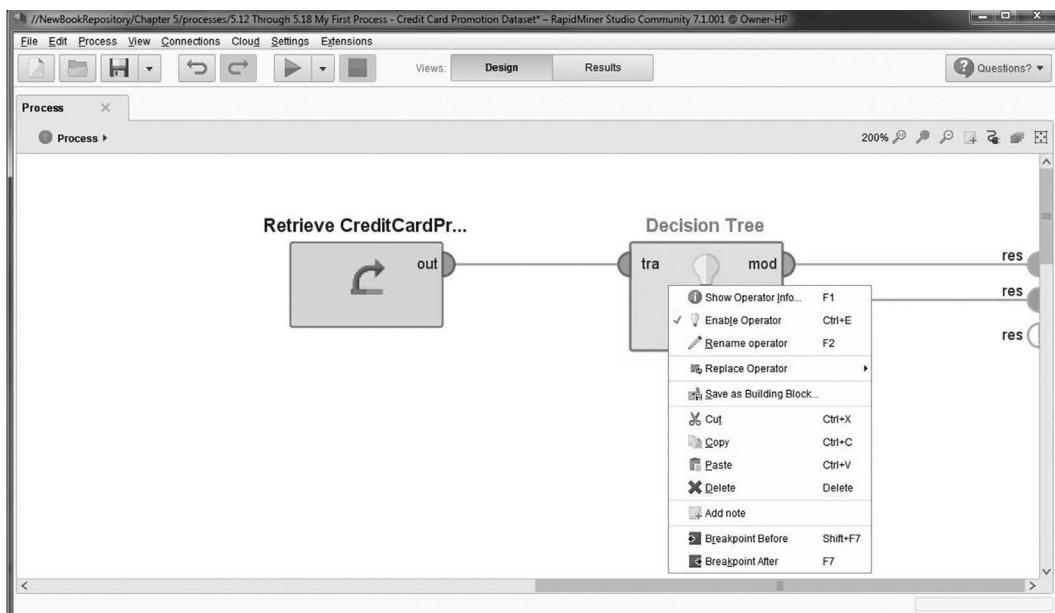


FIGURE 5.19 A list of operator options.

- Set the *Breakpoint Before* and *Breakpoint After* options for the decision tree operator and execute your process. The breakpoint before displays the input data as they enter the decision tree operator.
- Click the run process *arrow* to resume your process. The breakpoint after will execute, and our screen will show a graph of the decision tree.
- As the breakpoint after has interrupted the process, click the run process a third time. The process is now following the *mod-to-res* link. Therefore, prior to process termination, we once again see the graphical form of the decision tree.

As no operators follow the breakpoint after, it simply duplicates the final output. However, in general, the breakpoint operators are a tremendous aid when building models to solve complex problems. In the next section, we use decision trees to help us present two methods for testing the accuracy of supervised learner models.

5.2 BUILDING DECISION TREES

A standard way of measuring the goodness of a model is to test its performance on previously unseen data. A common scenario is to use two-thirds of the data for model creation and the remaining third for testing purposes. When sufficient data are lacking, the cross-validation technique discussed in Chapter 2 is a viable alternative. Here we illustrate both approaches with the help of a customer churn data set—*customer-churn-data.xlsx*—that has been highlighted in several RapidMiner tutorials. The data set contains 996 instances, 96 of which are of unknown outcome.

5.2.1 Scenario 1: Using a Training and Test Set

The main process window for our first scenario is displayed in Figure 5.20. Our goal is to build a decision tree model that accurately classifies instances as loyal customers or churners. If our test results are acceptable, we will use all of the data to construct a final model. This final model will be put into practice to help identify likely churners. Let's build the process given in Figure 5.20. As each new operator is introduced, locate the operator, bring it into the main process window, and make the connections shown in Figure 5.20.

- Create and save a new blank process.
- Add *customer-churn-data.xlsx* to the data folder of your local repository. This causes the contents of the data set to appear in your main process window. The file is now part of your local repository but has yet to be added to your process.
- Click on *Design*, locate *customer-churn-data.xlsx* in your local repository, and drag it into the main process window.

The *Set Role* operator allows us to change the role of any given attribute. We commonly use this operator to change the role of an attribute from regular to label.

- In the *Operators* panel, search for *Set Role*. Drag or double-click on *Set Role* to add it to your workspace. Be sure the *Parameters* panel shows *Set Role* as the operator. In the *Parameters* panel, use the drop-down to find *Churn* within *attribute name* and set its *target role* to *label*. Per Figure 5.20, link the *Retrieve* operator to the *Set Role* operator.

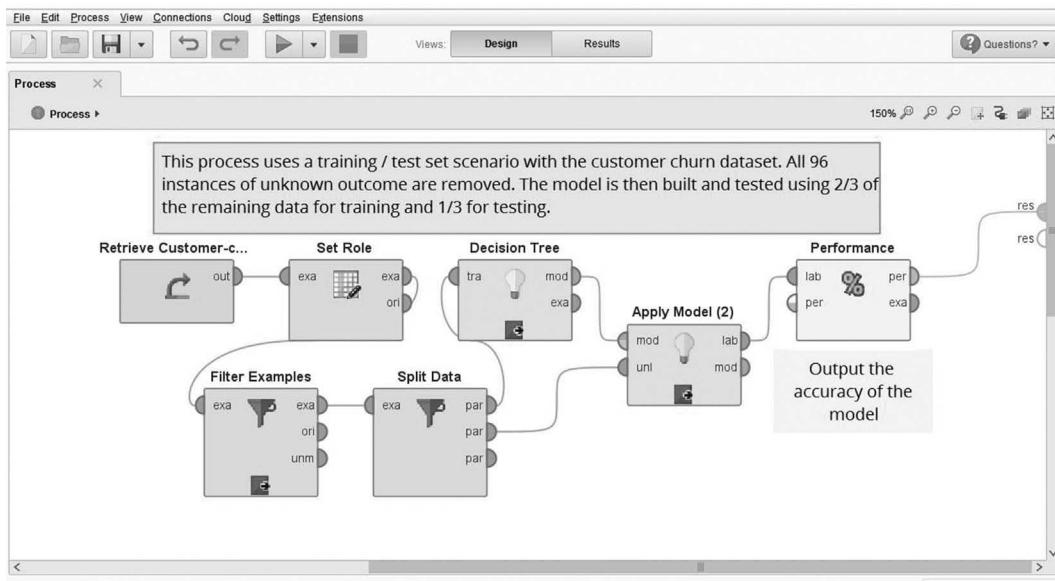


FIGURE 5.20 Customer churn—A training and test set scenario.

The *Filter Examples* operator is used to remove instances from the data set based on one or several criterion. As we are in the model building business, we must eliminate all instances of unknown outcome from the data.

- As you just did for the *Set Role* operator, add a *Filter Examples* operator to the workspace. In the *Parameters* panel, click on *Show Advanced Parameters* and make sure *condition class* is set to *custom filters*. Click *Add Filter* and fill in the filter criteria as specified in Figure 5.21. In this way, all instances of unknown outcome are removed. Link *Set Roles* to *Filter Examples*.
- The *Split Data* operator is applied to subdivide the data using one of several sampling criteria. Insert a *Split Data* operator. In the *partitions* section of the *Parameters* panel, click the *Edit* button, click *Add Entry* to fill in the partition ratios as specified in Figure 5.22, and click *OK*.

At this point, go ahead and add the remaining operators shown in Figure 5.20. Notice that the *Split Data* operator shows one output port labeled *par*, but if you attach this port to the *Apply Model* operator's *unl* (unlabeled) input port, a second *par* port will appear on *Split Data*. These ports represent the two partitions of the data. The first partition having two-thirds of the data is for model building, so the top *par* port should be linked to the *Decision Tree* operator. The second partition has the remaining data, so the second *par* port should be linked to the *unl* input port of the *Apply Model* operator. The *unl* connection tells *Apply Model* that these instances are to be used for model testing.

The *Apply Model* operator shows two input ports. The *mod* port takes as input the created decision tree model. The *unl* port accepts data that are either unlabeled or are to be

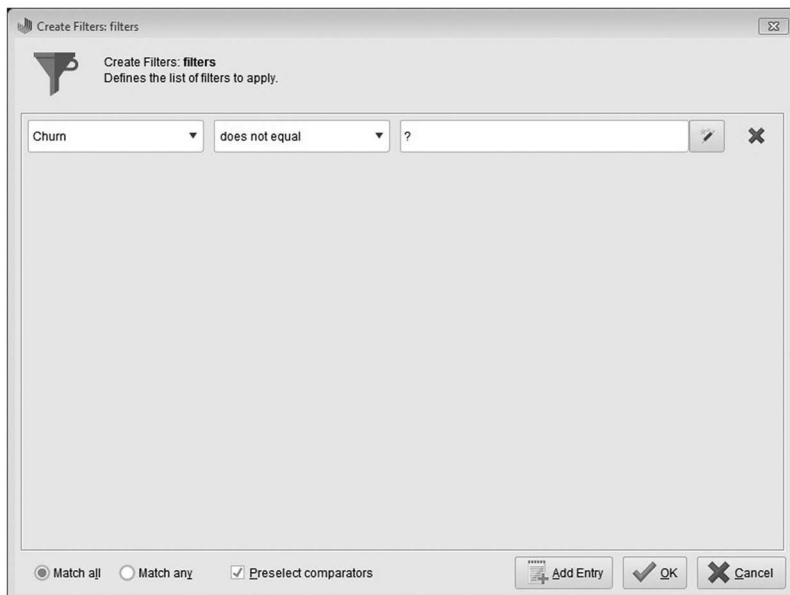


FIGURE 5.21 Removing instances of unknown outcome from the churn data set.

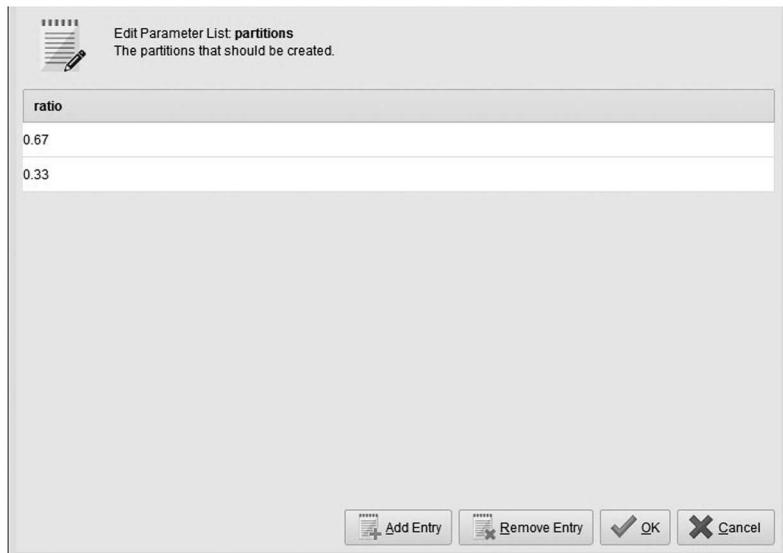


FIGURE 5.22 Partitioning the customer churn data.

treated as unlabeled for testing purposes. The *Apply Model* operator uses the decision tree to determine the class of the unlabeled data. Confidence scores are associated with each classified instance.

- To examine the confidence scores, place a *Breakpoint After* within the *Apply Model* operator.
- The *Performance* operator offers a summary of model accuracy. Highlight the *Performance* operator to see its parameters. Click on the *Performance* box, and in the *Parameters* panel, check to make sure the accuracy default is selected. In this way, we will see how accurate our model is in classifying the test data.
- Finally, place a *Breakpoint After* within the *Retrieve*, *Filter Examples*, and *Decision Tree* operators.
- Run your process to see the output displayed in Figure 5.23. *LastTransaction* gives the number of days since the individual last interacted with the customer website while still a customer.

This output is the data in their original form. Depending on how you loaded the data, the churn attribute may or may not be shown as a label (indicated by the highlighted column in Figure 5.23).

- Resume your process to see Figure 5.24. This is the result of applying the *Filter Examples* operator whereby all instances of unknown outcome have been removed.

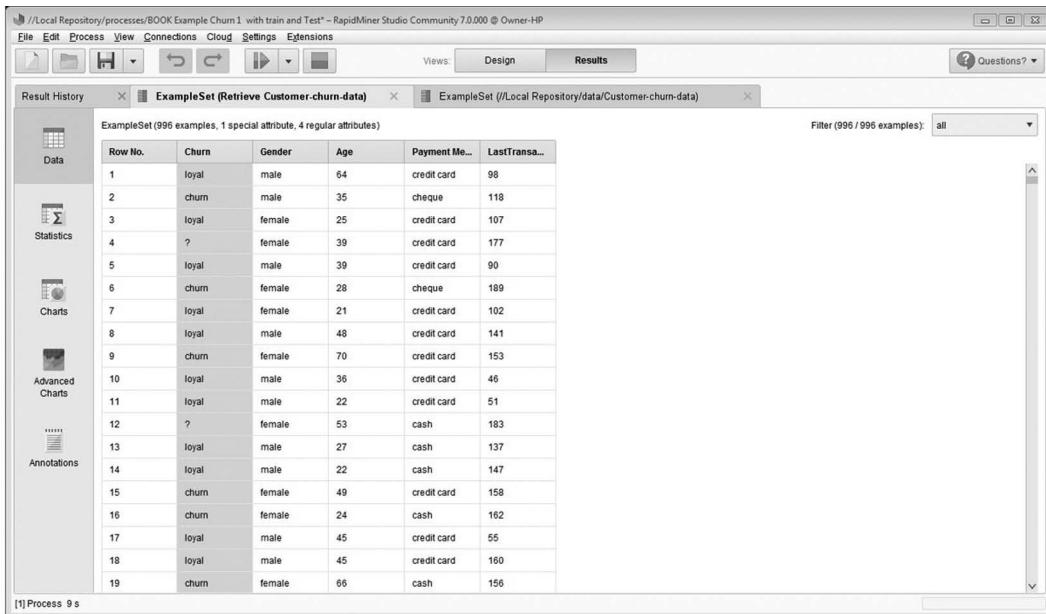
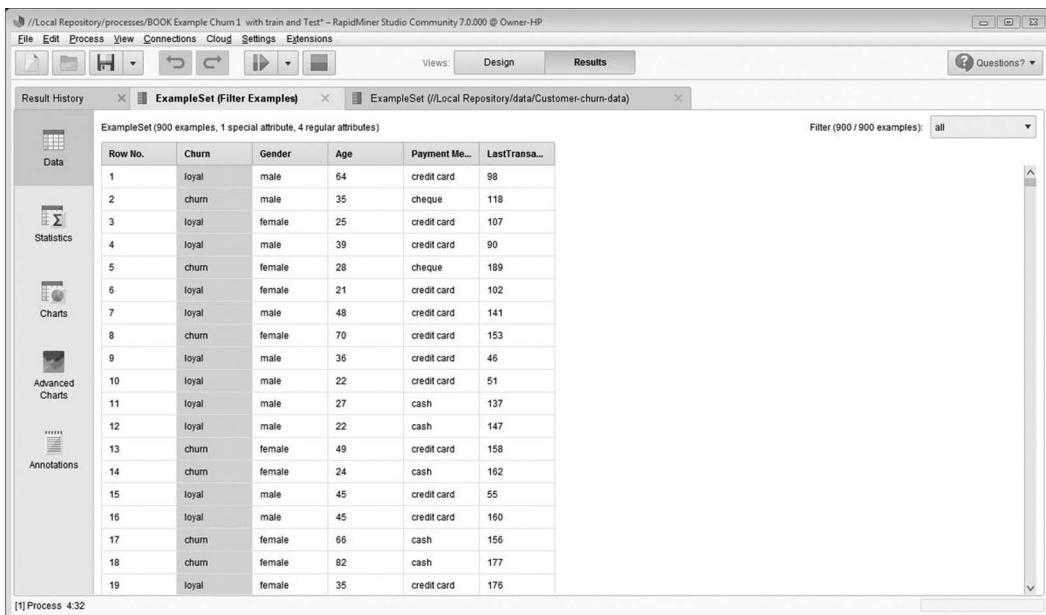


FIGURE 5.23 The customer churn data set.

FIGURE 5.24 *Filter Examples* has removed all instances of unknown outcome.

- Resume your process to observe the decision tree in Figure 5.25. Click on *Description* to see that there are two loyal customers over 89.5 years of age.
- Resuming, the next breakpoint takes us to Figure 5.26. This is the labeled output of the *Apply Model* operator. Scroll your screen to see the classifications of the test set instances together with their associated confidence scores.
- Continue the process to see the *Performance* operator displayed in Figure 5.27.

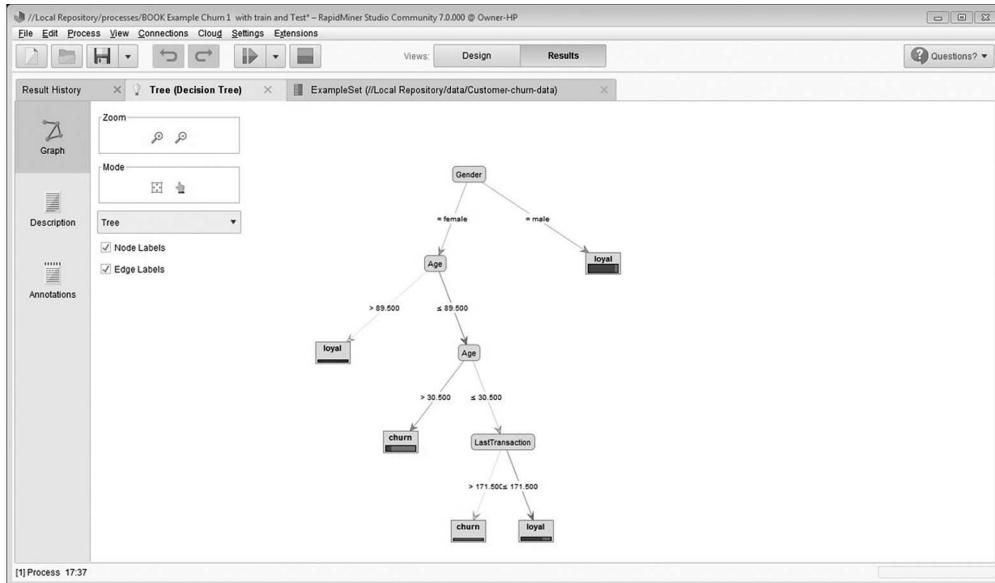


FIGURE 5.25 A decision tree for the customer churn data set.

The screenshot shows the RapidMiner Studio interface with the 'ExampleSet (Split Data)' operator selected. The table displays the results of the model application:

Row No.	Churn	prediction(Churn)	confidence(Churn)	confidence(Loyal)	Gender	Age	Payment Method	LastTransac...
1	loyal	loyal	0.899	0.101	male	64	credit card	98
2	churn	loyal	0.899	0.101	male	35	cheque	118
3	loyal	loyal	0.725	0.275	female	25	credit card	107
4	churn	churn	0	1	female	28	cheque	189
5	churn	loyal	0.725	0.275	female	24	cash	162
6	loyal	loyal	0.899	0.101	male	45	credit card	55
7	churn	churn	0.203	0.797	female	82	cash	177
8	loyal	churn	0.203	0.797	female	35	credit card	176
9	loyal	loyal	0.725	0.275	female	17	credit card	133
10	loyal	churn	0.203	0.797	female	84	cash	195
11	loyal	loyal	0.899	0.101	male	34	cheque	96
12	churn	churn	0.203	0.797	female	49	cash	188
13	loyal	loyal	0.725	0.275	female	22	credit card	72
14	churn	churn	0.203	0.797	female	37	credit card	162
15	churn	loyal	0.899	0.101	male	55	cash	126
16	loyal	churn	0.203	0.797	female	60	credit card	142
17	churn	churn	0.203	0.797	female	47	cheque	212
18	loyal	churn	0.203	0.797	female	33	credit card	30
19	loyal	churn	0.203	0.797	female	35	credit card	153

FIGURE 5.26 Output of the *Apply Model* operator.

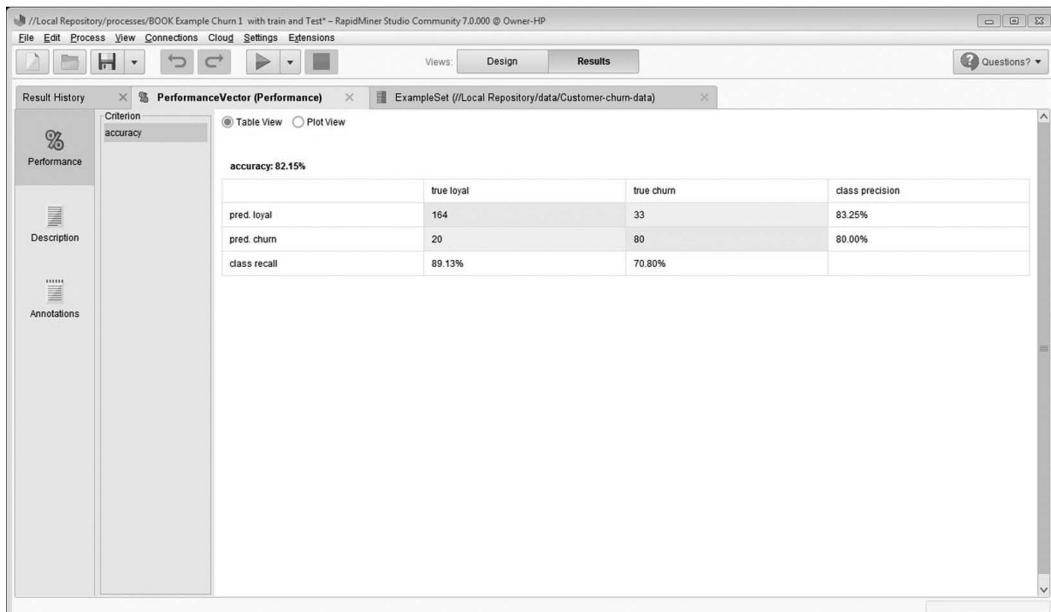


FIGURE 5.27 A performance vector for the customer churn data set.

Figure 5.27 shows that the test set accuracy is 82.15%. The figure also offers output in the form of a confusion matrix with true churn and true loyal values in column rather than row orientation. That is, 20 loyal customers were predicted to be churners, and 33 churners were seen as loyal customers. Precision for the loyal class is computed as 164/197, which is the number correctly classified as loyal divided by the sum of the number correctly classified as loyal and the number incorrectly classified as loyal. Recall is given by 164/184, which represents the number correctly classified as loyal divided by the total number of loyal customers. Similar statements can be made for precision and recall for the class of churners.

It is clear that the 70.8% recall for churners is less than optimal. However, provided that these results meet the minimum standards set for model accuracy, precision, and recall, our next step is to use all instances with known outcome to build a final decision tree model. This model will then be put into practice to help determine current customers likely to churn. A subset of these customers will be offered incentives in an attempt to prevent them from churning. Before we explore model application, the next section revisits this same problem but uses a subprocess to improve process structure and readability.

5.2.2 Scenario 2: Adding a Subprocess

The number of operators required to solve a problem is directly related to problem complexity. As the number of operators grows, it becomes more difficult to keep our thoughts directed toward solving the problem rather than on operator manipulation within the main process window. The solution to this dilemma is the *Subprocess* operator. The *Subprocess* operator can be thought of as a process within a process. Within each subprocess, we place operators that naturally group together to solve the larger problem.

To illustrate the use of the *Subprocess* operator, consider the main process shown in Figure 5.20. Of the seven operators used for this problem, the first four perform preprocessing tasks. As these four operators are part of the same task, let's place them in the same subprocess. Here's how it's done:

- Mouse to *File* and then *Save Process as* to make a copy of the previous process.
- Use the *Operators* search window to locate and drag the *Subprocess* operator into the main process window.
- Double-click the *Subprocess* operator to see a blank *Subprocess* window. We will move the preprocess operators into this window. First we must return to the main process. This is accomplished with a single click on *Process* or the *up* arrow, both of which are located in the upper-left portion of the subprocess screen.
- Cut and paste operators from the main process window to the subprocess window until they appear as in Figures 5.28 and 5.29. Take special note that the two output ports in Figure 5.29 match the two output ports of the *Subprocess* operator in Figure 5.28.
- Execute your process as in the previous example.

Provided that the subprocess operator is correctly configured, the output will be identical to that of the previous example. Although we have not done so here, we can right-click on the *Subprocess* operator to give the operator a new name that more clearly reflects its task.

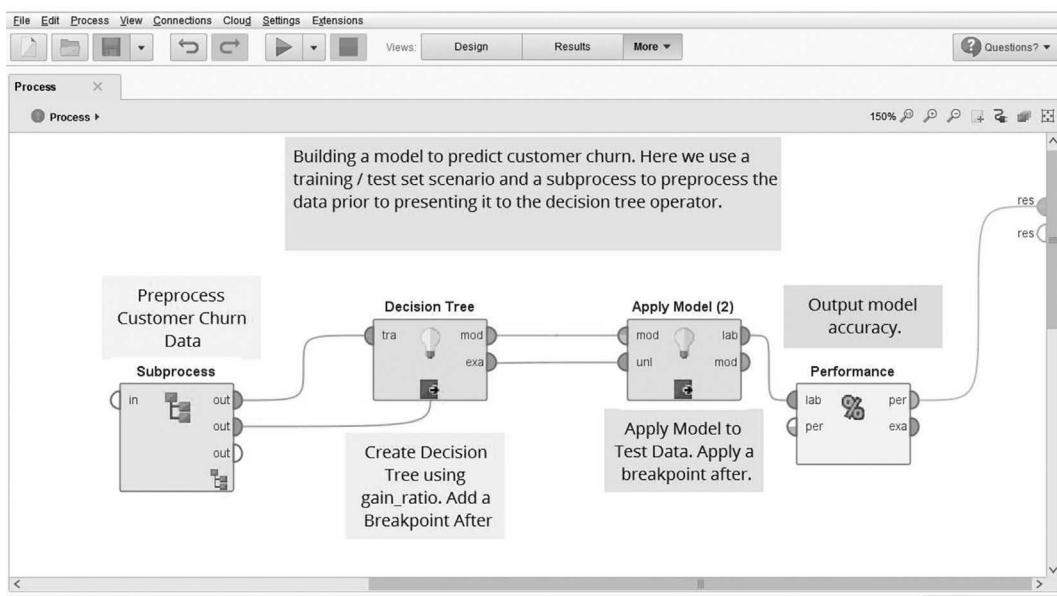


FIGURE 5.28 Adding a subprocess to the main process window.

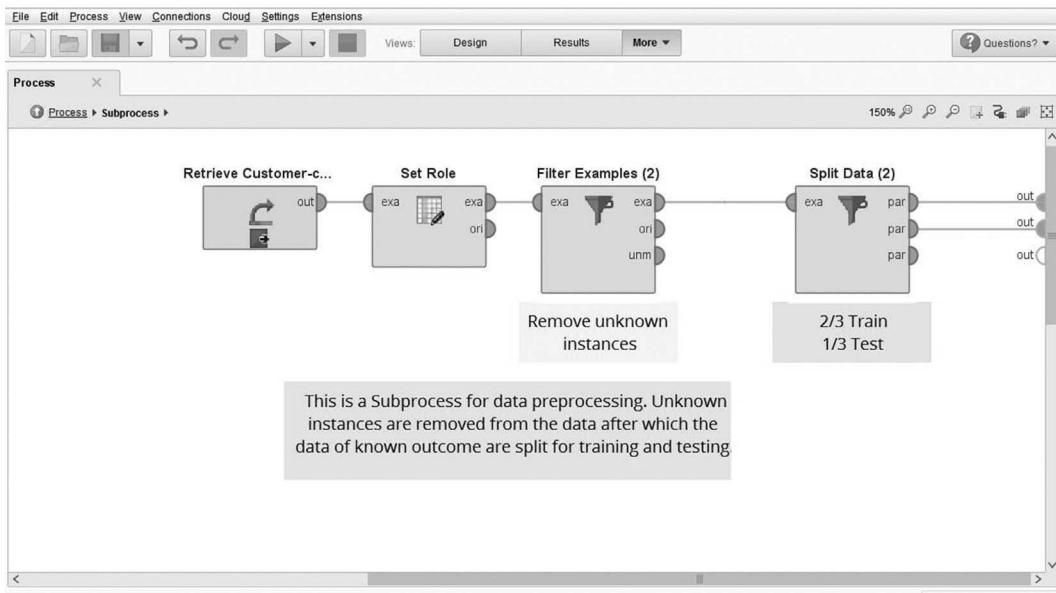


FIGURE 5.29 A subprocess for data preprocessing.

5.2.3 Scenario 3: Creating, Saving, and Applying the Final Model

With the assumption that the results obtained by testing our model are acceptable, it's time to create and save the final model. This model will then be applied to data of unknown outcome.

5.2.3.1 Saving a Model to an Output File

The process for writing the decision tree to an output file is shown in Figure 5.30. You can create this process by modifying the main process given in the model of Figure 5.20.

The process reads the customer churn data, filters out the unknown examples, and creates a decision tree from all data of known outcome. After the decision tree is created, the *Write Model* operator saves the decision tree in binary form to an output file of your choice. It is important to note that the *Apply Model* operator is not necessary. That is, we could remove both the *Apply Model* and *Performance* operators and link the decision tree *mod* output port to the *Write Model* input port. The purpose of adding the performance testing operators is simply to have a record of model performance relative to the training data.

- Create the process for writing to an output file.
- Before you execute the process, click on the *Write* operator and give your output file a name by providing a value for the *Model File* parameter. Set the *output type* parameter to *Binary*.
- After your process has executed, make sure the output file containing the decision tree model has been written to a folder on your computer.

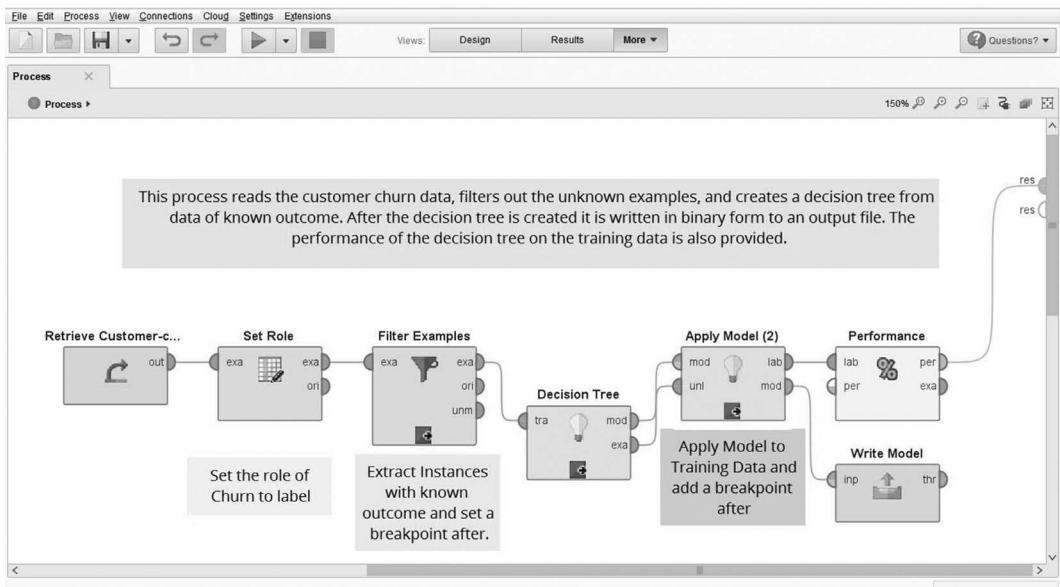


FIGURE 5.30 Creating and saving a decision tree model.

5.2.3.2 Reading and Applying a Model

With the saved decision tree model on hand, it's time to create a process that reads the model and applies it to the churn file data of unknown outcome. Figure 5.31 displays the process model for applying the saved model. The process reads the customer churn data set, removes from the data all examples of *known* outcome, and reads the saved decision tree model. *Apply Model* then uses the saved decision tree to predict outcome. The predicted outcome is written to an Excel file, where we can easily identify those individuals most likely to churn. To implement this process, be sure the *Read Model* parameter setting for *model file* gives the correct path to the saved model. Furthermore, have the *Filter Examples* operator to remove from the data all instances of known outcome. This is accomplished by modifying the test in Figure 5.21 from *does not equal* to *equals*. Finally, identify an output file for the *Write Excel* operator.

Figure 5.32 displays the first few lines of the Excel file after it has been sorted on the column labeled *prediction(Churn)*. Instances with the highest confidence scores for churning represent those individuals to receive special retention incentives.

5.2.4 Scenario 4: Using Cross-Validation

When the amount of data is not sufficient for a training/test set approach, cross-validation is a viable choice. Here we use the same customer churn data but replace the training/test set scenario with a 10-fold cross-validation. Recall that a 10-fold cross-validation splits the data into 10 equal-sized partitions whereby 9 partitions are used for training and the remaining partition is applied for testing. The model building and testing process is repeated 10 times, with each partition playing the role of test set a single time. The average performance of the 10 created models measures model accuracy.

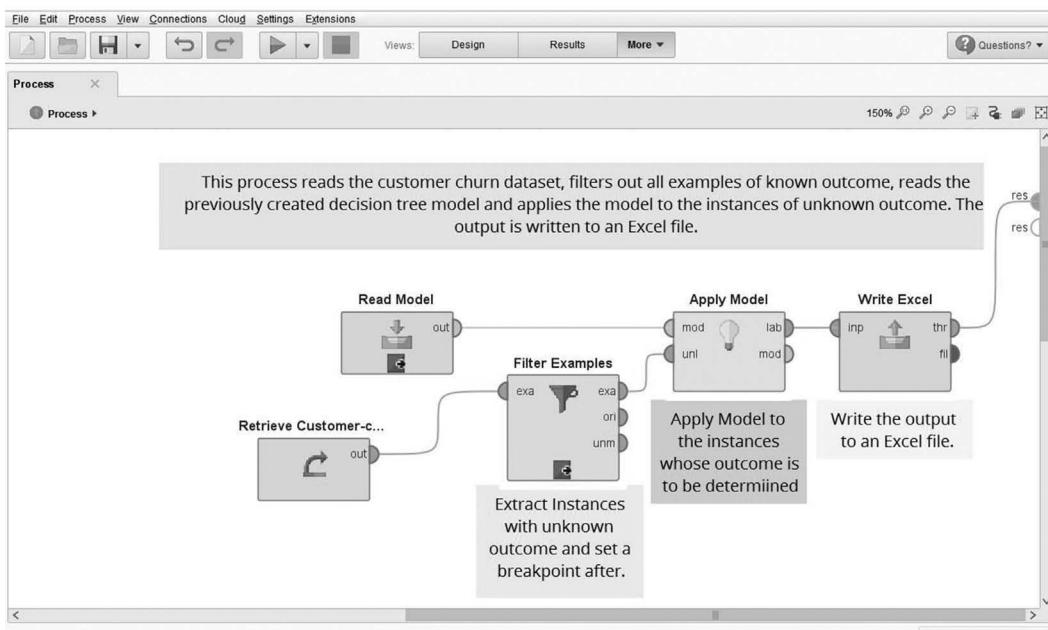


FIGURE 5.31 Reading and applying a saved model.

The screenshot shows a Microsoft Excel spreadsheet titled "My Churn File Output.xlsx - Excel". The data is organized into columns:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Gender	Age	Payment Meth	LastTransactio	Churn	confidence()	confidence(c)	confidence(c)	prediction(Churn)				
2	female	39.0	credit card	177.0		.2		.8	churn				
3	female	53.0	cash	183.0		.2		.8	churn				
4	female	33.0	credit card	194.0		.2		.8	churn				
5	female	71.0	credit card	27.0		.2		.8	churn				
6	female	54.0	cheque	146.0		.2		.8	churn				
7	female	58.0	credit card	176.0		.2		.8	churn				
8	female	45.0	credit card	150.0		.2		.8	churn				
9	female	33.0	credit card	144.0		.2		.8	churn				
10	female	72.0	credit card	158.0		.2		.8	churn				
11	female	66.0	cash	199.0		.2		.8	churn				
12	female	75.0	credit card	207.0		.2		.8	churn				
13	female	51.0	credit card	142.0		.2		.8	churn				
14	female	39.0	credit card	173.0		.2		.8	churn				
15	female	52.0	credit card	161.0		.2		.8	churn				
16	female	38.0	cash	163.0		.2		.8	churn				
17	female	50.0	cash	141.0		.2		.8	churn				
18	female	34.0	credit card	167.0		.2		.8	churn				
19	female	70.0	credit card	162.0		.2		.8	churn				
20	female	47.0	credit card	210.0		.2		.8	churn				
21	female	52.0	cash	194.0		.2		.8	churn				
22	female	71.0	credit card	148.0		.2		.8	churn				
23	female	39.0	credit card	128.0		.2		.8	churn				

At the bottom left, there is a button labeled "RapidMiner Data".

FIGURE 5.32 An Excel file stores model predictions.

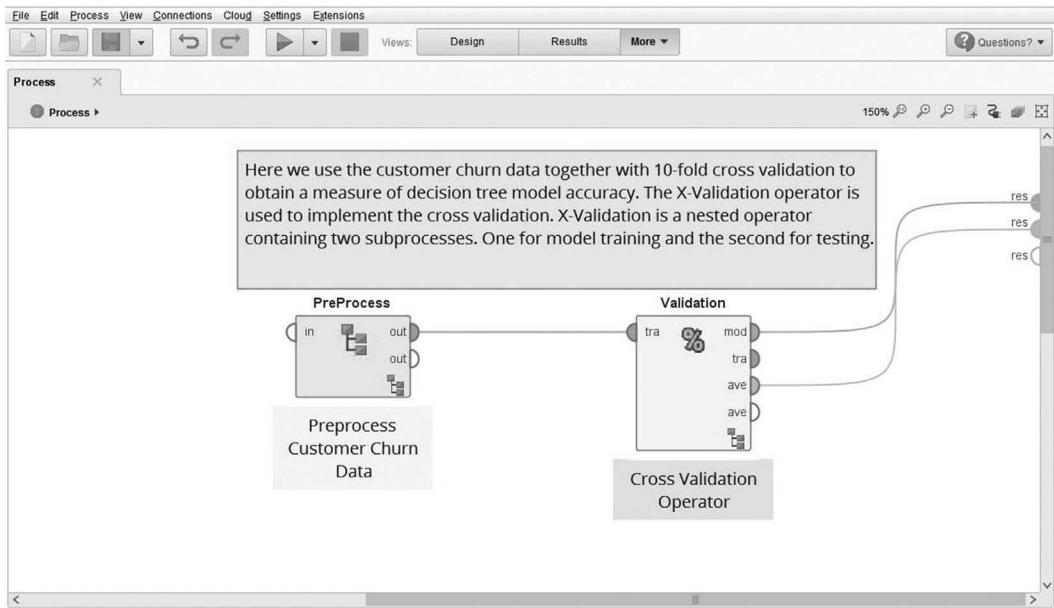


FIGURE 5.33 Testing a model using cross-validation.

Figure 5.33 displays the main process window for the cross-validation. Although Figure 5.33 shows the name of the cross-validation operator as *Validation*, the cross-validation operator is shown in the *Operators* panel as *X-Validation*. Here's how to implement a cross-validation for our problem.

- Create and save a new process.
- Locate and drag the *Subprocess* and *X-Validation* operators into the main process window. We use the subprocess displayed in Figure 5.34 to read and filter the data. As before, we change the subprocess name to *PreProcess*.
- Here's how to build *PreProcess*:
 - Double-click on *PreProcess* and retrieve *customer-churn-data.xlsx*.
 - Use *Set Role* to specify *Churn* as the label attribute.
 - Use *Filter Examples* to remove instances of unknown outcome from the data.
 - Return to the main process window and connect the *out* port of *PreProcess* to the *tra* input port of *Validation*.

Cross-validation is a nested operator with two subprocesses. One subprocess is for model training and the second is for model testing. Figure 5.35 shows the nested subprocesses of the cross-validation operator for this problem. Keep in mind that the cross-validation

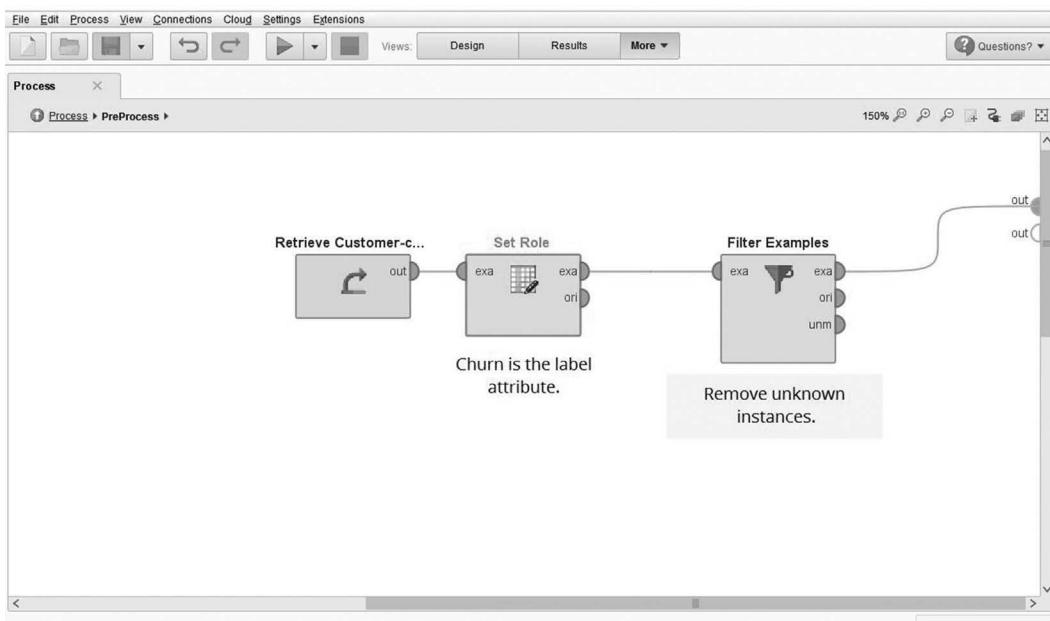


FIGURE 5.34 A subprocess to read and filter customer churn data.

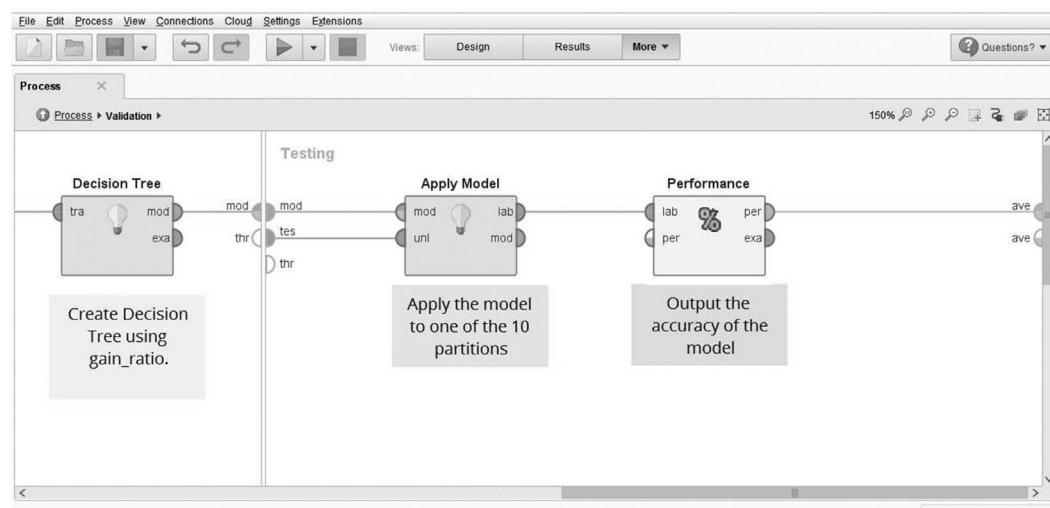


FIGURE 5.35 Nested subprocesses for cross-validation.

operator controls model training and testing. It's easy to get confused into thinking it's the other way around. Let's continue building our model:

- Double-click on *Validation* to reveal two empty subprocesses.
- Drag the *Decision Tree* operator into the training window and the *Apply Model* and *Performance* (classification) operators into the testing window.
- In the training window, connect the two *tra* ports. This connection represents the cross-validation operator generating a partition and giving the training data of the partition to the decision tree model.
- In the testing window, connect the *tes* port to the *uml* input port of the *Apply Model* operator. This is where the cross-validation operator presents the test data for the current partition to the *Apply Model* operator. In addition, connect the *mod* ports so *Apply Model* receives the created decision tree.
- Insert the input and output links for the performance vector and remove all unnecessary breakpoints as each breakpoint will be part of the 10-fold cross-validation.
- Return to the main design window to make the *mod-to-res* and the *ave-to-res* connections. The first connection outputs the model, and the second connection displays the performance vector.
- Run your process.

The output, given in Figure 5.36, shows the model accuracy to be 82.67%. This accuracy represents the overall average correctness of the 10 cross-validations. The standard

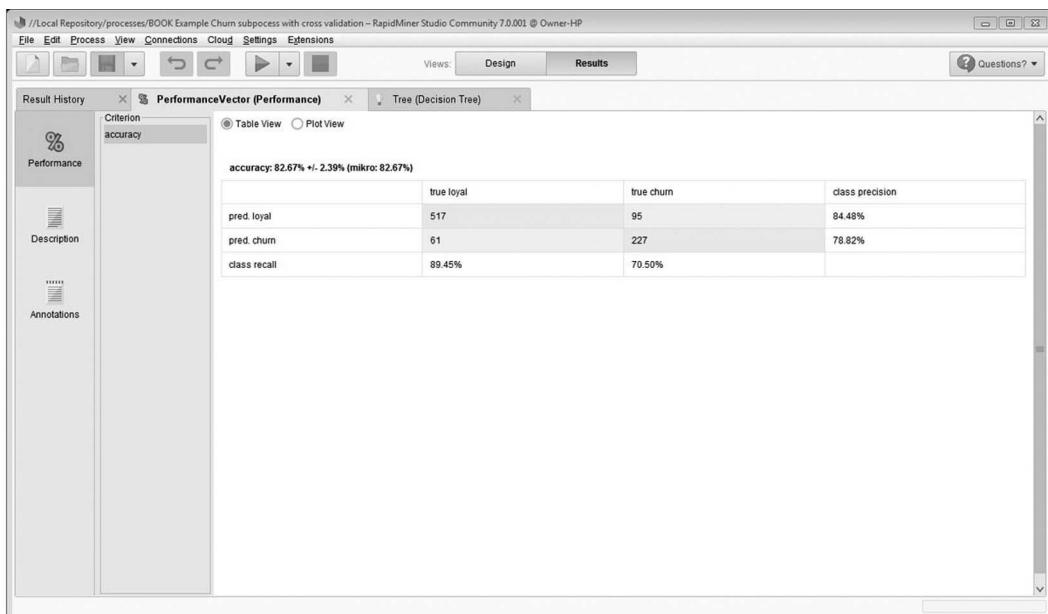


FIGURE 5.36 Performance vector for a decision tree tested using cross-validation.

deviation of the accuracy values seen as a result of the cross-validations is 2.39%. The standard deviation is given as a percent as each number in the computation is also a percent. Given the small sample size, the value is of little meaning other than giving a very rough estimate of the variation seen between the 10 accuracy scores. The mikro score of 82.67% is the model accuracy when applied to all instances. In this case, the mikro and the average model accuracy computed by averaging the accuracy scores from all 10 cross-validations are identical.

5.3 GENERATING RULES

RapidMiner offers several options for generating interesting rules. Here we examine three of RapidMiner's rule operators. The *Rule Induction* operator is based on the repeated incremental pruning to produce error reduction (RIPPER) algorithm and uses a rule growing and pruning procedure similar to the technique discussed in Chapter 3. The *Tree to Rules* operator works by first creating a decision tree and then mapping the tree to a set of production rules. This basic technique was also discussed in Chapter 3. In Chapter 2, we examined rules generated by the *Subgroup Discovery* operator. Let's apply each operator to the customer churn data in an attempt to find similarities and differences in their performance.

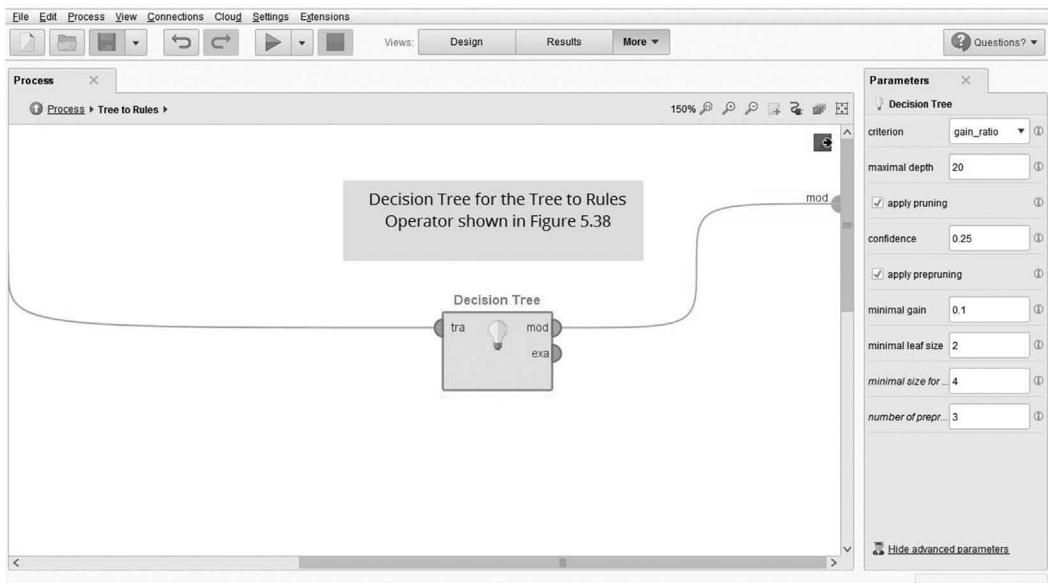
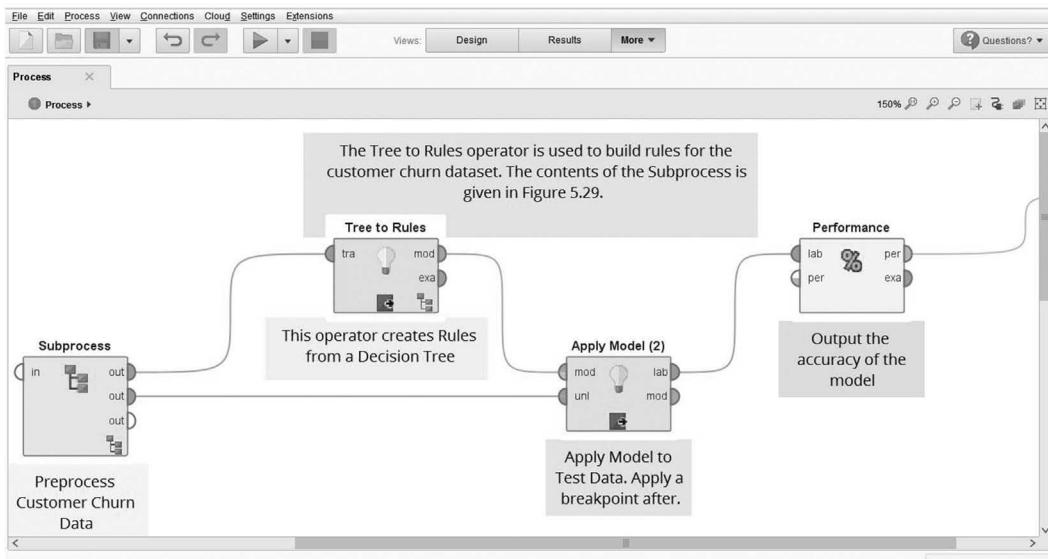
For our experiment, we return to the training/test set scenario given in Figure 5.28. Let's try to make life easy by using this process model as a template and replacing the decision tree operator with each of the three rule generators. Let's hope this works!

5.3.1 Scenario 1: Tree to Rules

The *Tree to Rules* operator is what is known as a meta-operator. This is the case as the *Tree to Rules* operator uses another operator (i.e., a *Decision Tree* operator) as the basis for its rules. Therefore, simply replacing the *Decision Tree* operator in the model shown in Figure 5.28 with the *Tree to Rules* operator generates an error. In addition to the operator replacement, we must also instruct the *Tree to Rules* operator as to which decision tree model should be the basis for the rules it creates. Here's what to do.

- Delete the *Decision Tree* operator and replace it with the *Tree to Rules* operator.
- Make the appropriate port connections and then double-click on the operator. Your screen will show an empty subprocess. Drag the *Decision Tree* operator into the subprocess window and make the connections as they appear in Figure 5.37.
- Return to the main process shown in Figure 5.38 and place a *Breakpoint After* within the *Tree to Rules* operator. This will allow us to see the generated rules.
- Run your process to see the seven rules displayed in Figure 5.39. As expected, the rules mimic the decision tree given in Figure 5.25.
- Continue the execution of your process to observe the performance vector in Figure 5.40.

The performance vector shows a test set accuracy of 83.16%, with 28 churners classified as loyal and 22 loyal customers classified as having churned.

FIGURE 5.37 Subprocess for the *Tree to Rules* operator.FIGURE 5.38 Building a model with the *Tree to Rules* operator.

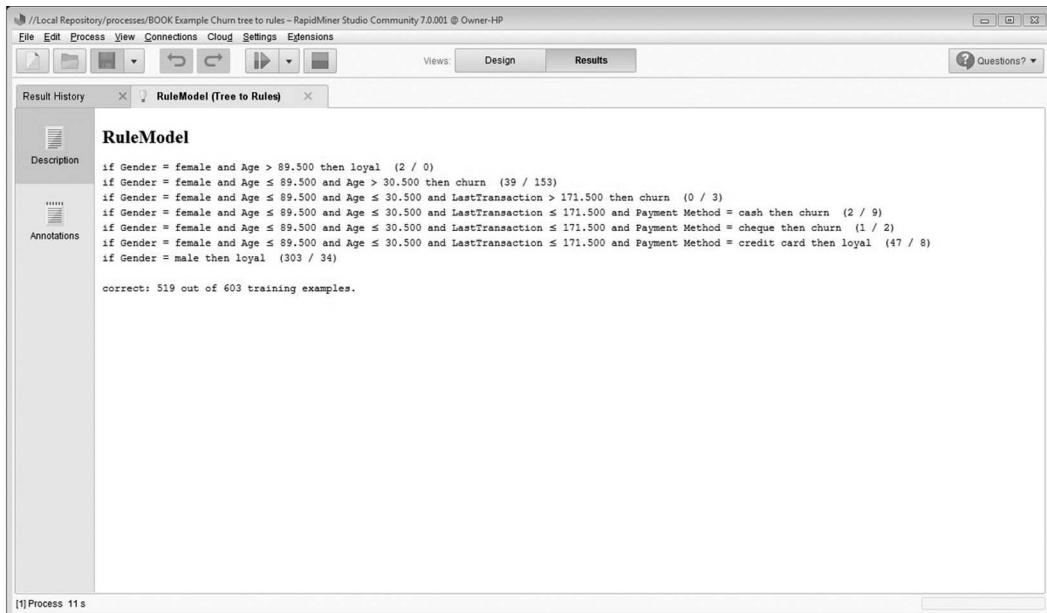
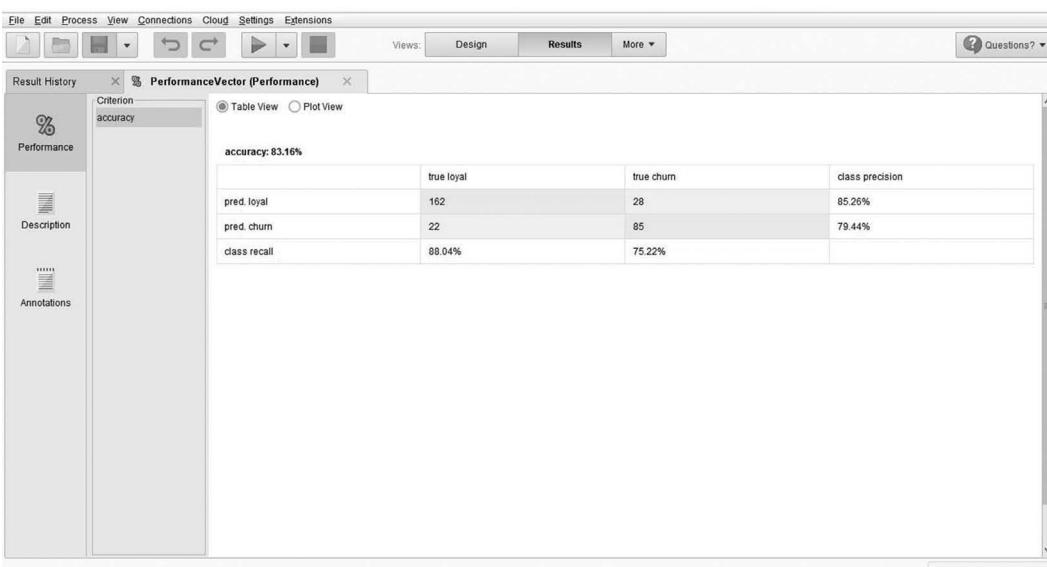
FIGURE 5.39 Rules generated by the *Tree to Rules* operator.

FIGURE 5.40 Performance vector for the customer churn data set.

5.3.2 Scenario 2: Rule Induction

The *Rule Induction* operator is next on the list.

- Replace the *Tree to Rules* operator with the *Rule Induction* operator.
- Add the *Discretize by Binning* operator with a bin size of four to the preprocessing subprocess. Although this operator can handle numeric data, it is best to discretize numeric attributes when generating rules.

Figure 5.41 displays the main process window. The revised subprocess, which now includes the *Discretize by Binning* operator, is given in Figure 5.42.

- Place a *Breakpoint After* within the *Rule Induction* operator and run your process. The resultant set of 15 rules is seen in Figure 5.43.
- Continue your process to display the performance vector shown in Figure 5.44. (Note that the *Apply Model* link from its *mod* port to the *res* port allows you to toggle between the generated rules and the performance vector.)

Here we see a test set accuracy of 82.15% with 26 churners classified as loyal and 27 loyal customers classified as churners. Comparing these results with those of the *Tree to Rules* operator offers little insight into which is the better performer. Both models show similar precision and recall values relative to both classes. However, one major difference is that the rule induction model required twice as many rules to cover the instances.

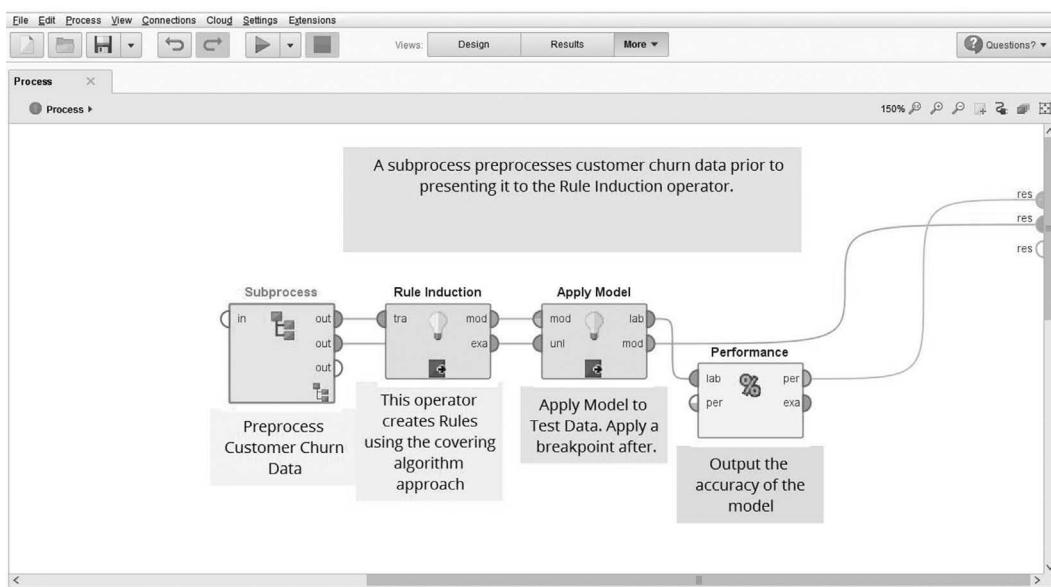


FIGURE 5.41 A process design for rule induction.

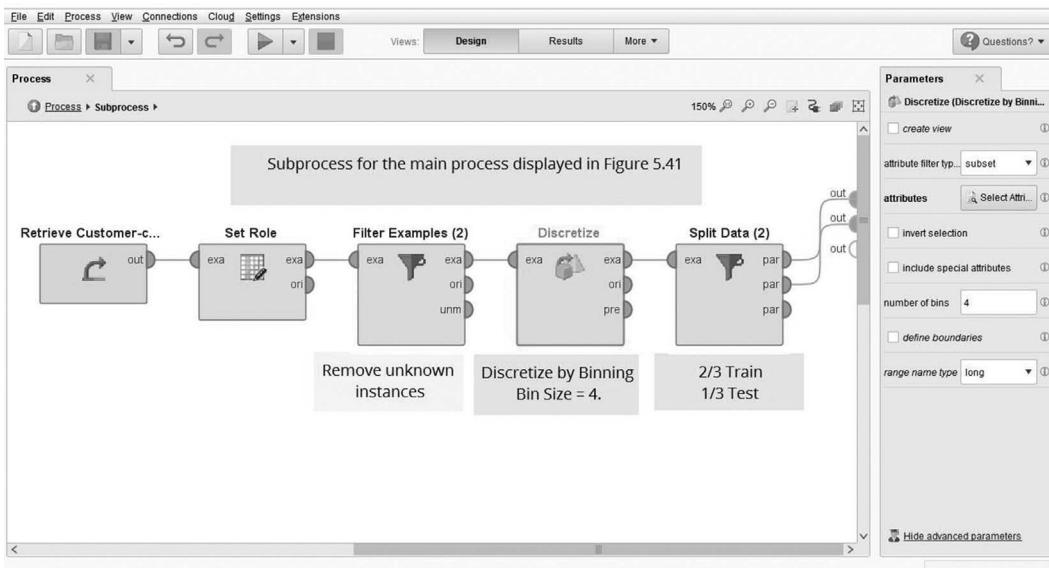
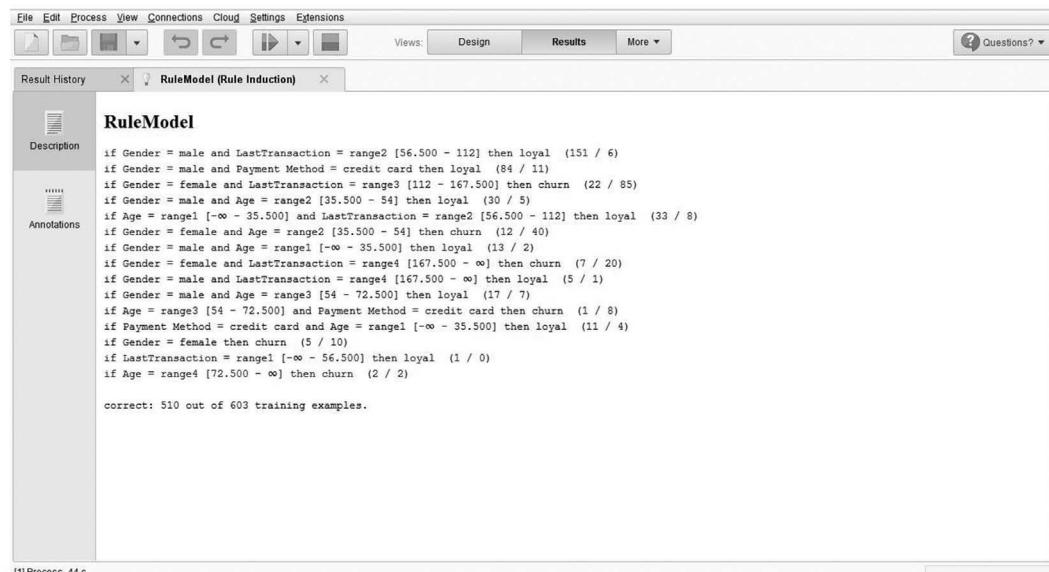
FIGURE 5.42 Adding the *Discretize by Binning* operator.

FIGURE 5.43 Covering rules for customer churn data.

Furthermore, close examination of the individual rules shows that each rule generated by the *Tree to Rules* operator uses the gender attribute, whereas one-third of the rules generated by the *Rule Induction* operator are without the gender attribute. In Chapter 7, you will see a technique that uses statistics to determine whether the difference in the performance of two models is statistically significant.

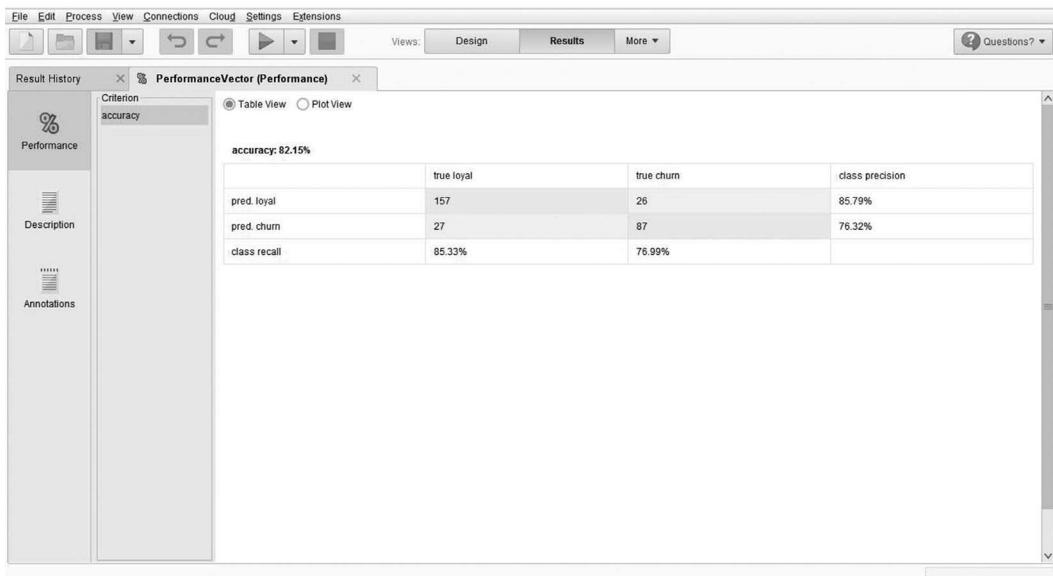


FIGURE 5.44 Performance vector for the covering rules of Figure 5.43.

5.3.3 Scenario 3: Subgroup Discovery

In Chapter 2, we described how the focus of subgroup discovery is on finding rules to describe interesting population subgroups within the data rather than on overall classification accuracy. Our process model needs revision to reflect this change in emphasis. Figures 5.45 and 5.46 display the revised main process and subprocess windows. We have removed the *Apply Model* and *Performance* operators from the main process window and the *Split Data* operator from the subprocess. With these changes, the focus of our data mining session becomes the individual rules and corresponding subgroups within the data. As the *Subgroup Discovery* operator is limited to two-class problems with nominal data types, the *Discretize* operator remains part of our subprocess. Therefore, to follow the example,

- Make the previously described changes and highlight *Subgroup Discovery* to display the operator's parameters.

As you can see, we have several parameter options. Most of the parameter settings can have a dramatic effect on the number and type of rules we see. It is worth your time to scroll through the operator documentation to get a general idea about the workings of each parameter as well as subgroup discovery's procedure for searching the instance space. For our example, we are interested in the *k best rules* parameter, which allows us to specify the number of rules to be generated, and the *rule generation* parameter, which specifies whether we want to see rules from one or both classes.

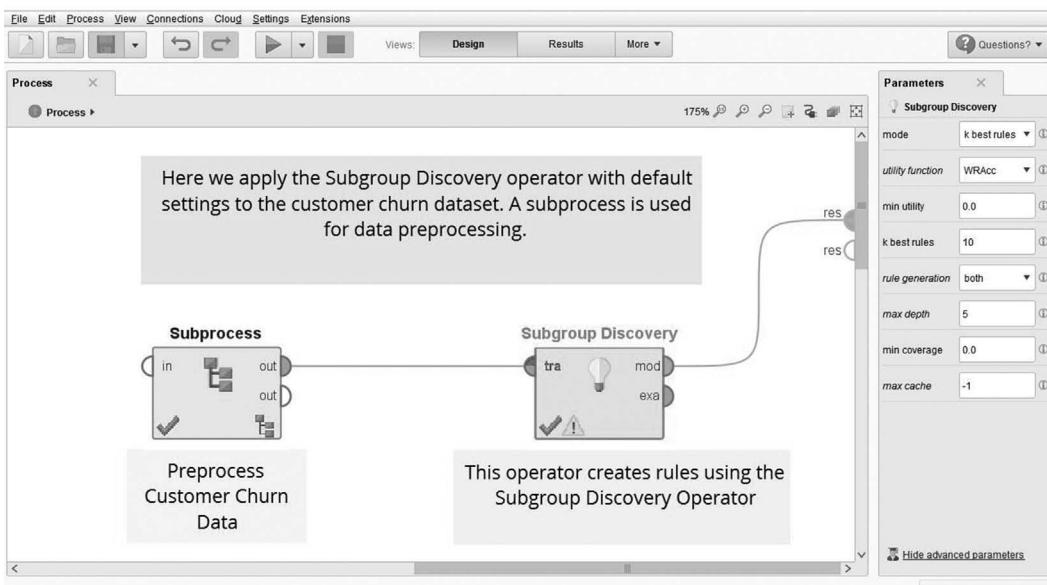


FIGURE 5.45 Process design for subgroup discovery.

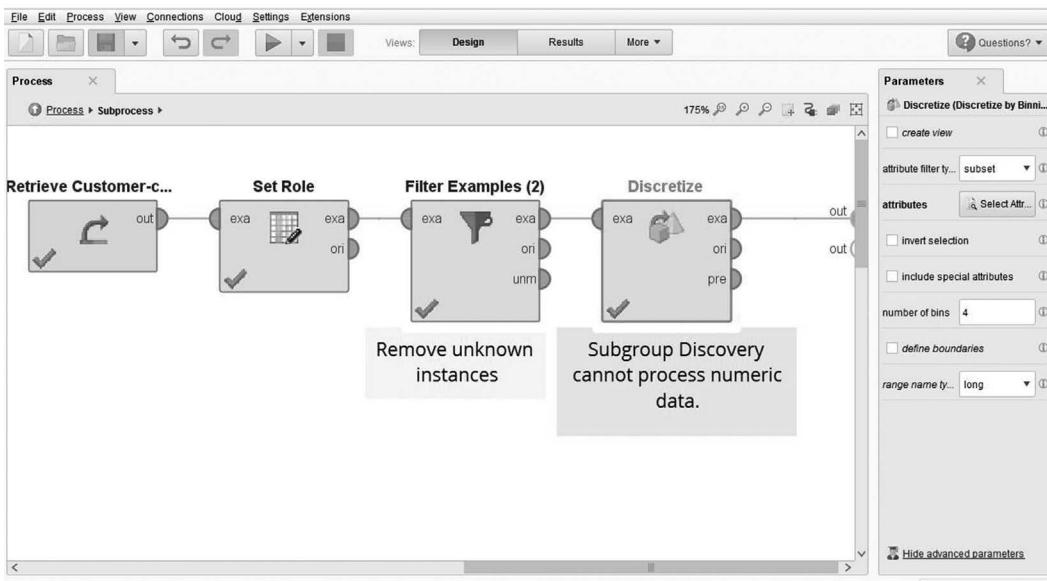


FIGURE 5.46 Subprocess design for subgroup discovery.

- Use the default value of 10 for k best rules and both for rule generation, and run your process.

Figure 5.47 shows a table format of the 10 best rules based on the parameter settings. Consider the first rule. The rule can be stated as follows:

- IF $Gender = Female$

THEN $Conclusion = Churn$

Rule Precision: 65.8%

Rule Coverage: 45.20%

The Pos value of 268 and Neg value of 139 tell us that of the total population of 407 instances covered by the rule antecedent condition, 268 churned, and 139 did not churn. It should be pointed out that the class associated with the first instance defaults as the negative class. As the first instance is a loyal customer, churn is treated as the positive class. Rule precision is determined by the ratio 268:407 and rule coverage by 407:900.

Although accuracy for the set of rules relative to the entire instance population has little meaning, a computation is made for the accuracy of each individual rule relative to the entire population. We see that the first rule shows an accuracy of 78.6%. This tells us that when we apply this rule to the entire instance population, 78.6% of all instances are correctly classified. This value is actually greater than the rule precision value of 65.8%!

The screenshot shows the KNIME interface with the 'Results' tab selected. On the left, there are three vertical tabs: 'Data', 'Description', and 'Annotations'. The main area displays a table titled 'RuleSet (Subgroup Discovery)' with the following data:

Premise	Conclusion	Pos	Neg	Size	Coverage	Precision	Accuracy	Bias	Lift	BI
Gender=female	churn	268	139	407	0.452	0.658	0.786	0.3...	1.840	0...	0...	
Gender=ma...	loyal	54	439	493	0.548	0.890	0.786	0.2...	1.387	0...	0...	
Gender=ma...	loyal	18	297	315	0.350	0.943	0.668	0.3...	1.468	0...	0...	
LastTransaction=range2 [56.500 - 112], Gender=ma...	loyal	11	221	232	0.258	0.953	0.591	0.3...	1.483	0...	0...	
LastTransaction=range3 [112 - 167.500], Gender=fema...	churn	130	38	168	0.187	0.774	0.744	0.4...	2.163	0...	0...	
Age=range2 [35.500 - 54], Gender=fema...	churn	113	27	140	0.156	0.807	0.738	0.4...	2.256	0...	0...	
Gender=fema...	churn	157	108	265	0.294	0.592	0.697	0.2...	1.656	0...	0...	
LastTransaction=range2 [56.500 - 112], Gender=ma...	loyal	5	176	181	0.201	0.972	0.548	0.3...	1.514	0...	0...	
LastTransaction=range2 [56.500 - 112]	loyal	70	285	355	0.394	0.803	0.597	0.1...	1.250	0...	0...	
LastTransaction=range2 [56.500 - 112], Payment Method=credit card	loyal	41	230	271	0.301	0.849	0.568	0.2...	1.322	0...	0...	

FIGURE 5.47 Rules generated by the Subgroup Discovery operator.

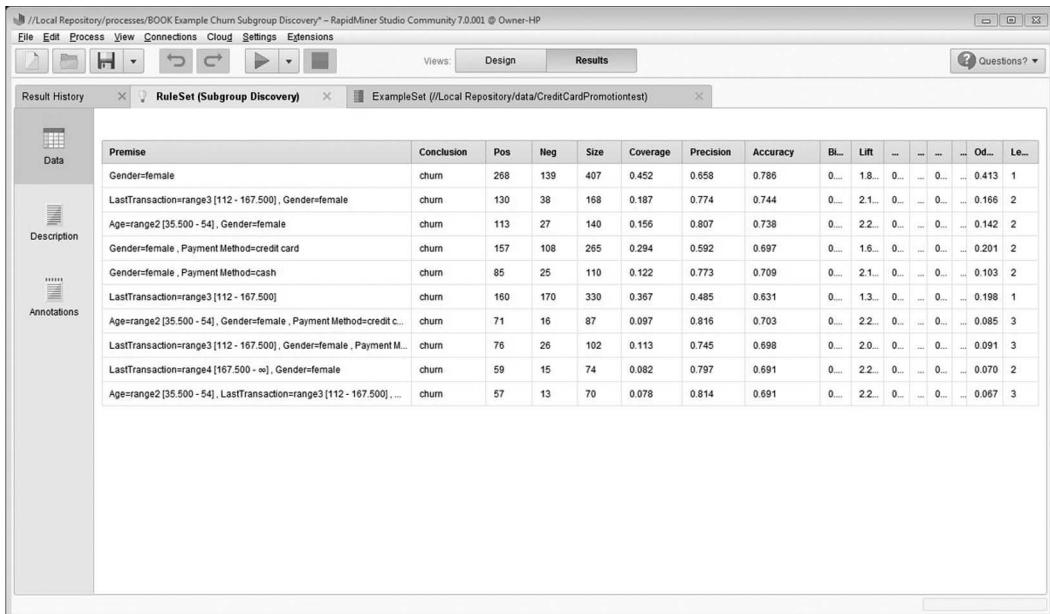


FIGURE 5.48 Ten rules identifying likely churn candidates.

How can this be? Recalling the explanation given in Chapter 2, the rule is considered to correctly classify any instance within the population that does not satisfy its antecedent condition provided that it resides in the competing class. Although this method of measuring rule accuracy is unusual, it does give us a metric as to how well each rule performs as a single representative of the entire population.

Finally, you may have noticed that the rules in Figure 5.47 emphasize customer loyalty. As we are more interested in identifying future churners, a better approach is to limit the rule generation to the class of churners. We can do this by highlighting the subgroup operator and changing rule generation from *both* to *positive*. The 10 rules identifying likely churning are given in Figure 5.48. With these rules, we can turn our attention to those rules with higher precision to best identify customers likely to churn.

5.4 ASSOCIATION RULE LEARNING

Market basket analysis attempts to determine those items likely to be purchased by a customer during a shopping experience. The output of a market basket analysis is a set of associations about customer-purchase behavior. In Chapter 3, we showed you how the Apriori algorithm creates frequent item sets from which association rules are generated.

RapidMiner's *Fp-Growth* operator is an efficient alternative to the Apriori algorithm. This is especially true for larger data sets as *Fp-Growth* requires but two scans of the data. Here we provide two association rule examples. The first example is a tutorial illustrating the basic features of the *Fp-Growth* operator. The second example investigates RapidMiner's template for market basket analysis.

5.4.1 Association Rules for the Credit Card Promotion Database

For our first example, we apply the *Fp-Growth* operator to a modified form of the credit card promotion database to generate item sets. These item sets are then given to the *Create-Association-Rules* operator for association rule generation. Here is the process:

- Create and save a new empty process and add *CreditCardPromoAssoc.xlsx* to your local repository.

The data set is a modified form of the credit card promotion database where the gender column has been changed to two columns, one for female and the second for male, with true/false entries as appropriate. All yes/no values have also been changed to true/false values.

- Build the main and subprocess structures shown in Figures 5.49 and 5.50. The discretize operator is *Discretize by Frequency* with attribute filter type *single*, attribute *age*, and number of bins *three*. You can read about this operator in the documentation.
- Highlight the *Fp-Growth* operator to display its parameters. Set the parameters as shown in Figure 5.49.

Several parameters require explanation. The *positive value* parameter indicates which binomial value represents the positive case. If *find min number of itemsets* is checked, the value placed in the box *min number of itemsets* overrides the value of

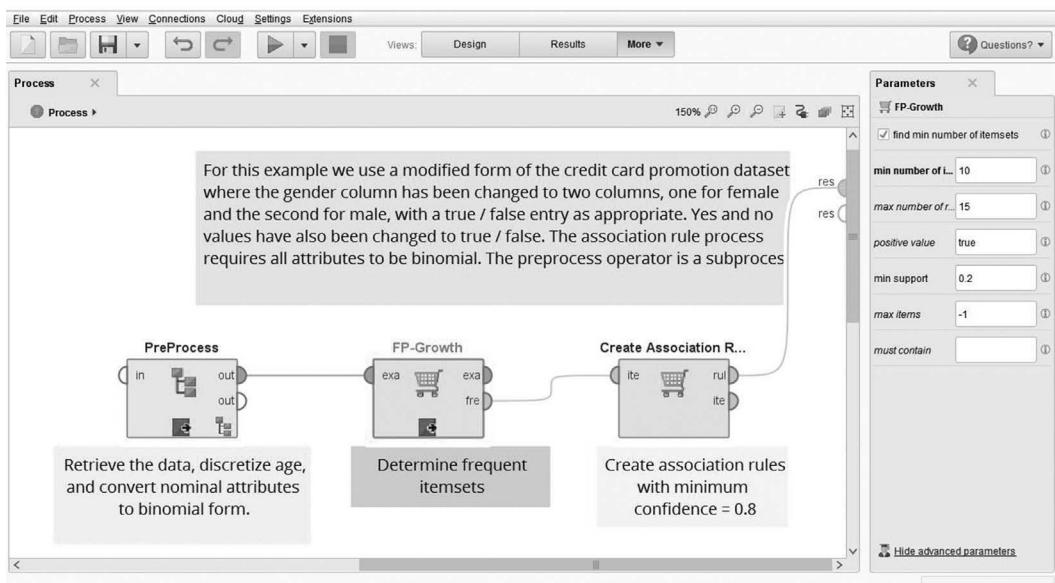


FIGURE 5.49 Generating association rules for the credit card promotion database.

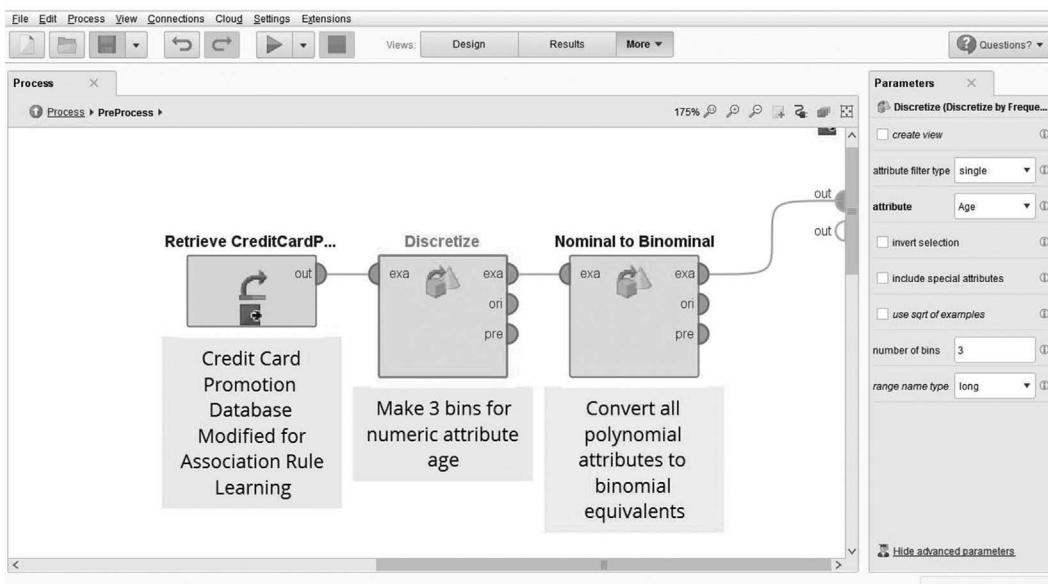


FIGURE 5.50 Preparing data for association rule generation.

min support. That is, *min support* will be ignored if its value causes fewer than the minimum number of item sets specified to be generated.

- You may wish to make use of the *Breakpoint After* to see the result of each operator. Run your process.

Figure 5.51 presents a partial list of the association rules where *Life Ins Promo* is part of the rule conclusion. Scroll to the right to see confidence and support values. It is obvious that there is a wealth of support rule generation options. Of particular importance is the *Min. Criterion* option in the lower-left part of your display. As you move the bar to the right, the minimum criterion for rule confidence increases. Click on *Description* to see the rules displayed as in Figure 5.52.

5.4.2 The Market Basket Analysis Template

Association rules are a powerful tool for analyzing purchasing trends. The primary issue with association rules is editing and transforming the data into a format suitable for analysis. This preprocessing of the data is often referred to as *edit, transform, and load (ETL)*. To illustrate the ETL process, let's follow the logic of RapidMiner's *Market Basket Analysis* template example.

- Go to *File, New Process*, and then click on *Market Basket Analysis*. Your screen will appear as in Figure 5.53.
- Place a *Breakpoint After* within the first four operators and run the process.

The first breakpoint shows the data in their original form. As the objective of the market basket analysis is to determine which products are purchased together,

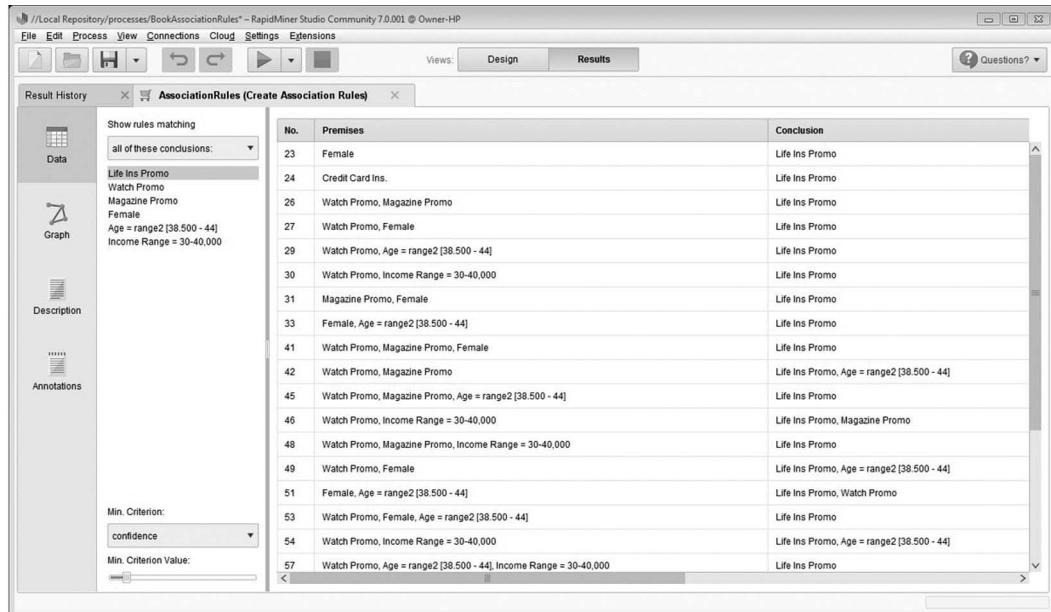


FIGURE 5.51 Interface for listing association rules.

```
//Local Repository/processes/BookAssociationRules* - RapidMiner Studio Community 7.0.01 @ Owner-HP
File Edit Process View Connections Cloud Settings Extensions
Views: Design Results Questions?

Result History AssociationRules (Create Association Rules)

AssociationRules
[Income Range = 30-40,000] --> [Life Ins Promo] (confidence: 0.800)
[Age = range1 [-> - 38.500]] --> [Life Ins Promo] (confidence: 0.800)
[Income Range = 30-40,000] --> [Age = range2 [38.500 - 44]] (confidence: 0.800)
[Life Ins Promo, Age = range2 [38.500 - 44]] --> [Magazine Promo] (confidence: 0.800)
[Magazine Promo, Age = range2 [38.500 - 44]] --> [Life Ins Promo] (confidence: 0.800)
[Income Range = 30-40,000] --> [Life Ins Promo, Magazine Promo] (confidence: 0.800)
[Magazine Promo, Income Range = 30-40,000] --> [Life Ins Promo] (confidence: 0.800)
[Life Ins Promo, Age = range2 [38.500 - 44]] --> [Female] (confidence: 0.800)
[Watch Promo, Age = range2 [38.500 - 44]] --> [Magazine Promo] (confidence: 0.800)
[Magazine Promo, Age = range2 [38.500 - 44]] --> [Watch Promo] (confidence: 0.800)
[Watch Promo, Age = range2 [38.500 - 44]] --> [Female] (confidence: 0.800)
[Magazine Promo, Age = range2 [38.500 - 44]] --> [Income Range = 30-40,000] (confidence: 0.800)
[Income Range = 30-40,000] --> [Magazine Promo, Age = range2 [38.500 - 44]] (confidence: 0.800)
[Magazine Promo, Income Range = 30-40,000] --> [Age = range2 [38.500 - 44]] (confidence: 0.800)
[Life Ins Promo, Age = range2 [38.500 - 44]] --> [Watch Promo, Magazine Promo] (confidence: 0.800)
[Watch Promo, Age = range2 [38.500 - 44]] --> [Life Ins Promo, Magazine Promo] (confidence: 0.800)
[Life Ins Promo, Watch Promo, Age = range2 [38.500 - 44]] --> [Magazine Promo] (confidence: 0.800)
[Magazine Promo, Age = range2 [38.500 - 44]] --> [Life Ins Promo, Watch Promo] (confidence: 0.800)
[Life Ins Promo, Age = range2 [38.500 - 44]] --> [Watch Promo, Female] (confidence: 0.800)
[Watch Promo, Age = range2 [38.500 - 44]] --> [Life Ins Promo, Female] (confidence: 0.800)
[Life Ins Promo, Watch Promo, Age = range2 [38.500 - 44]] --> [Female] (confidence: 0.800)
[Life Ins Promo, Watch Promo] --> [Age = range2 [38.500 - 44]] (confidence: 0.833)
[Female] --> [Life Ins Promo] (confidence: 0.857)
[Credit Card Ins.] --> [Life Ins Promo] (confidence: 1.000)
[Income Range = 30-40,000] --> [Magazine Promo] (confidence: 1.000)
[Watch Promo, Magazine Promo] --> [Life Ins Promo] (confidence: 1.000)
[Watch Promo, Female] --> [Life Ins Promo] (confidence: 1.000)
[Life Ins Promo, Age = range2 [38.500 - 44]] --> [Watch Promo] (confidence: 1.000)
```

FIGURE 5.52 Association rules for the credit card promotion database.

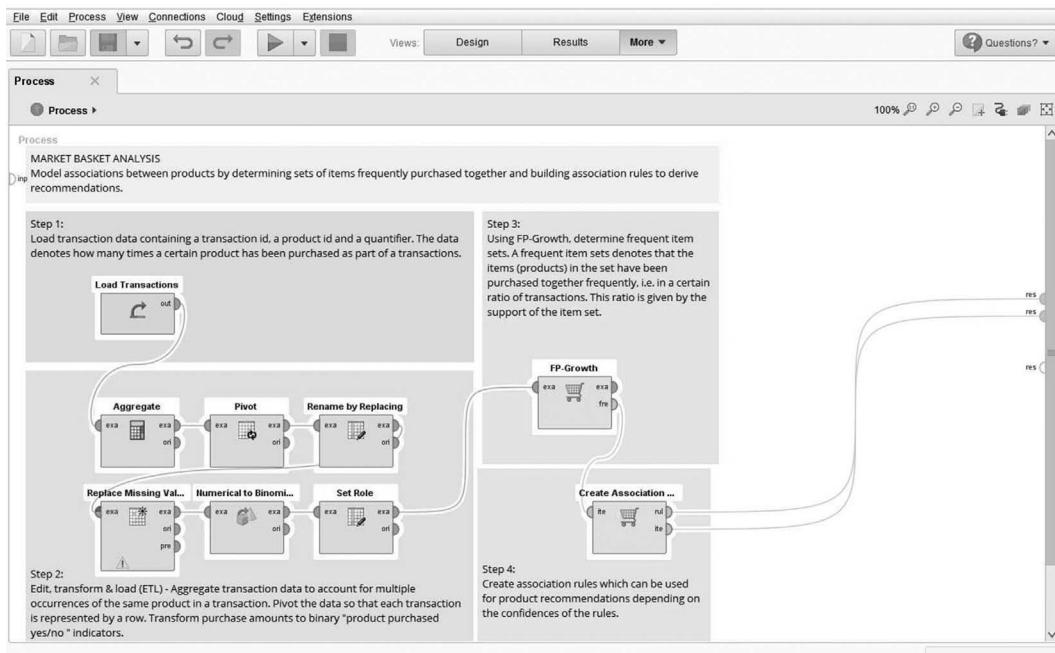


FIGURE 5.53 *Market basket analysis* template.

the product IDs currently listed in a single column must be transformed such that each product has its own column. For example, invoice 131510 shows two purchases of product 20 on line 7 and one purchase of product 20 on line 8.

- Resume the process to see the output of the second breakpoint.

The second breakpoint presents the data after execution of the *Aggregate* operator. The purpose of the *Aggregate* operator is to combine multiple occurrences of the same product on one invoice. The *Aggregate* operator combines the purchases of product 20 so as to show invoice 131510 on row 109 with product 20 purchased three times.

- Once again, resume the process.

The third breakpoint is the result of the *Pivot* operator. The output of the *Pivot* operator is given in Figure 5.54. Notice that each row shows a unique invoice and each column represents a single product. Moving to the next breakpoint, the *Rename by Replacing* operator removes the *sum(Orders)* prefix from each product name. Moving to the next breakpoint, as *Fp-Growth* cannot process missing data, the *Replace Missing Values* operator replaces all missing values with 0. Moving to the last breakpoint, Figure 5.55 shows a subset of the resultant association rules generated by this process. This example makes it clear that solving nontrivial problems requires a comprehensive understanding of the capabilities of your analytical tool.

Result History ExampleSet (Pivot)

ExampleSet (493 examples, 0 special attributes, 24 regular attributes)

Filter (493 / 493 examples): all

Row No.	Invoice	sum(Orders)_Product_11	sum(Orders)_Product_12	sum(Orders)_Product_13	sum(Orders)_Product_14	sum(Orders)_Product_15	sum(Orders)_Product_16	sum(Orders)_Product_17	sum(Orders)_Product_18	sum(Orders)_Product_19	sum(Orders)_Product_20	sum(Orders)_Product_21	sum(Orders)_Product_22	sum(Orders)_Product_23	sum(Orders)_Product_24
3	1306800	?	1	2	?	?	?	?	?	?	?	?	?	?	?
4	1306824	?	?	?	1	1	?	?	?	?	?	?	?	?	?
5	1306825	?	?	?	?	?	?	1	?	?	?	?	?	?	?
6	1306835	?	?	?	?	?	?	?	1	?	?	?	?	?	?
7	1306845	?	?	?	?	?	?	?	?	?	?	2	?	?	?
8	1306872	?	?	?	?	?	?	?	?	?	?	?	?	?	1
9	1306873	?	?	?	?	?	?	?	?	?	?	?	?	?	?
10	1306874	?	1	?	?	?	?	3	?	?	?	?	?	?	?
11	1306875	?	?	?	?	?	?	?	?	?	?	?	?	?	?
12	1306896	?	?	?	?	?	?	?	?	?	?	?	?	?	?
13	1306915	?	?	?	?	?	?	?	?	?	?	?	?	?	?
14	1306916	?	?	?	?	?	?	?	?	1	?	?	?	?	?
15	1306917	?	?	1	1	?	?	?	?	?	?	?	?	?	?
16	1306918	?	1	?	?	?	?	?	?	?	?	?	?	?	?
17	1306939	?	?	?	?	?	?	?	?	?	?	?	?	?	?
18	1306940	?	?	?	?	1	?	?	?	?	?	?	?	?	?
19	1306953	?	?	?	1	?	?	?	?	?	?	?	?	?	?
20	1306954	?	1	1	?	?	?	?	?	?	?	?	?	?	1

[1] Process 2/48

FIGURE 5.54 The pivot operator rotates the example set.

Result History FrequentItemSets (FP-Growth) AssociationRules (Create Association Rules)

Association Rules

```

[Product 18] --> [Product 12] (confidence: 0.103)
[Product 12] --> [product 1] (confidence: 0.104)
[Product 26] --> [Product 20] (confidence: 0.114)
[Product 20] --> [Product 19] (confidence: 0.118)
[Product 19] --> [Product 11] (confidence: 0.125)
[Product 19] --> [Product 12] (confidence: 0.125)
[product 1] --> [Product 16] (confidence: 0.125)
[Product 19] --> [Product 11, Product 20] (confidence: 0.125)
[Product 16] --> [Product 12] (confidence: 0.130)
[Product 16] --> [product 1] (confidence: 0.130)
[Product 20] --> [Product 18] (confidence: 0.137)
[Product 20] --> [Product 27] (confidence: 0.137)
[Product 21] --> [Product 12] (confidence: 0.138)
[Product 29] --> [Product 12] (confidence: 0.167)
[Product 11, Product 20] --> [Product 12] (confidence: 0.176)
[Product 11, Product 20] --> [Product 19] (confidence: 0.176)
[Product 18] --> [Product 20] (confidence: 0.179)
[Product 12] --> [Product 20] (confidence: 0.194)
[Product 27] --> [Product 12] (confidence: 0.214)
[Product 27] --> [Product 12, Product 20] (confidence: 0.214)
[Product 29] --> [Product 20] (confidence: 0.222)
[Product 12, Product 20] --> [Product 11] (confidence: 0.231)
[Product 12, Product 20] --> [Product 27] (confidence: 0.231)
[Product 11] --> [Product 20] (confidence: 0.250)
[Product 19] --> [Product 20] (confidence: 0.250)
[Product 20] --> [Product 12] (confidence: 0.255)
[product 1] --> [Product 12] (confidence: 0.292)
[Product 20] --> [Product 11] (confidence: 0.292)

```

FIGURE 5.55 Association rules for the *Market Basket Analysis* template.

5.5 UNSUPERVISED CLUSTERING WITH K-MEANS

THE GAMMA-RAY BURST DATA SET

Gamma-ray bursts are brief gamma-ray flashes with origins outside of our solar system. More than 1000 such events have been recorded. The gamma-ray burst data in this data set are from the BATSE 4B catalog. The bursts in the BATSE 4B catalog were observed by the Burst and Transient Source Experiment (BATSE) aboard the National Aeronautics and Space Administration's (NASA's) Compton Gamma-Ray Observatory between April 1991 and March 1993. Although many attributes have been measured for these bursts, the data set is limited to seven attributes. Attribute *burst* gives the assigned burst number. All other attributes have been preprocessed by applying a logarithmic normalization. Normalized attributes *T90* and *50* measure burst duration (burst length), *P256* and *fluence* measure burst brightness, and *HR321* and *HR32* measure burst hardness.

This data set is interesting as it allows astronomers to develop and test various hypotheses about the nature of gamma-ray bursts. In doing so, astronomers have an opportunity to learn more about the structure of the universe. Also, the raw gamma-ray burst data had to be pre-processed and transformed several times before a set of significant attributes were developed. The data set clearly demonstrates the importance of data preprocessing and data transformation. The data are strictly real-valued and formatted for Weka and RapidMiner under the name *Grb4u*. If you would like more information about the BATSE project, visit the website at <http://www.batse.msfc.nasa.gov/batse/grb/catalog/>.

In Chapter 3, we illustrated the *K-means* unsupervised clustering algorithm. RapidMiner's implementation of *K-means* is very versatile and can be used with nominal, mixed, and numeric data. Here we use RapidMiner's *K-means* operator to experiment with the data described in the box titled "The Gamma-Ray Burst Data Set." The data are interesting as gamma-ray bursts are the most powerful singular events in our universe and there is considerable disagreement as to the number of clusters present in the data set (Mukherjee et al., 1998). Let's see what we can find!

Figure 5.56 shows the main process window for our experiment. Our purpose is to have *K-means* cluster the data into three clusters, after which the cluster attribute is used as the label attribute for supervised learning with a decision tree. In this way, we have a graphical depiction of the clustering that would not be otherwise available to us. Here's how to set things up.

- Create and save a new blank process.
- Add *Grb4u.xlsx* to the data folder of your local repository.
- The *burst* attribute should be removed as it is unique. Use the *Select Attributes* operator with attribute filter type *subset* to select all the attributes other than the *burst* attribute. This is most easily accomplished as follows:
 - Drag the *Select Attributes* operator into your workspace. Make sure the filter type is *subset*. Click on the *Select Attributes* parameter. Move *burst* into the selected attribute area to the right.
 - Click *apply* and then check the parameter box titled *invert selection*. The *burst* attribute will be removed upon operator execution.

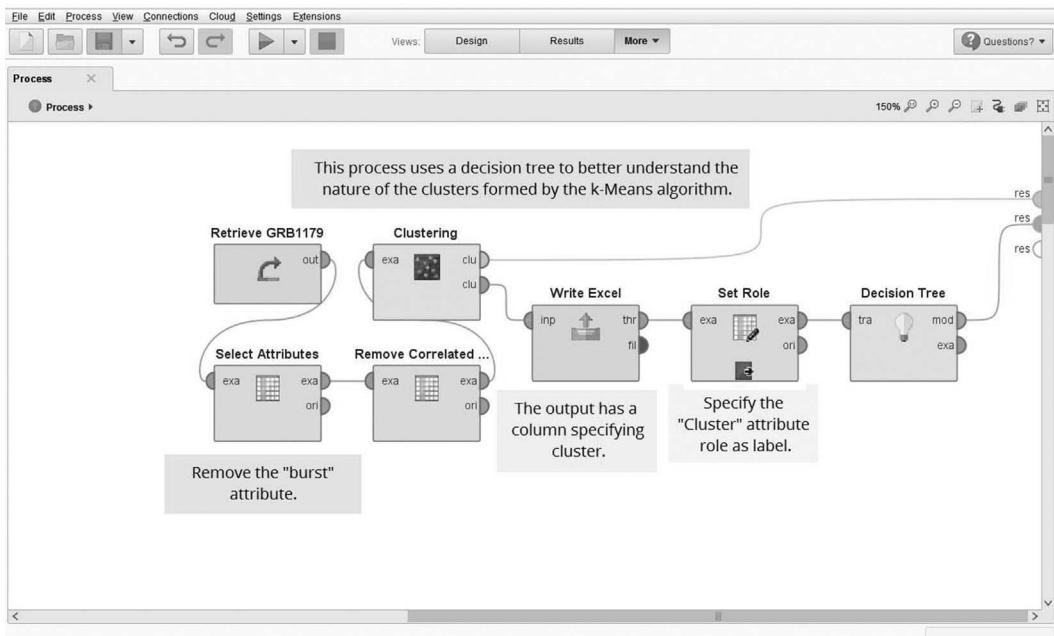


FIGURE 5.56 Process design for clustering gamma-ray burst data.

- One of each pair of attributes for burst length brightness and hardness should be removed as each pair is highly correlated. Use the *Remove Correlated Attributes* operator with the setting of 0.5 for attribute removal.
- Add the *K-means* clustering operator with $k = 3$. Be sure to check the add cluster attribute box shown in the Figure 5.56. This will allow us to use the cluster attribute for supervised learning.
- Use the *Write Excel* operator to write the clustering to an output file. Be sure to specify a name for the output file.
- Use the *Set Role* operator and set the role of the cluster attribute as *label*. *Cluster* will be used as the output attribute for the created decision tree. Place a *Breakpoint After* within this operator.
- Add the *Decision Tree* operator with its maximum depth parameter set at 4.
- Run your process to see the result of the *Breakpoint After* within the *Set Role* operator, as shown in Figure 5.57. Notice that *cluster* has been added as the label attribute.

Prior to creating the decision tree, let's take a look at a scatterplot of the clustering.

- Click on *charts*, and fill in the chart options as in Figure 5.58. By making the color column *cluster*, we get a clear graphical picture of the clustering relative to burst length (t50) and hardness (hr32). The scatterplot makes it clear that the harder bursts are shorter in length. We also see one very soft short burst that represents an outlier. This is likely a false reading, but we can't be certain.

File Edit Process View Connections Cloud Settings Extensions

Views: Design Results More ▾

Result History ExampleSet (Set Role)

Data Statistics Charts Advanced Charts Annotations

ExampleSet (1179 examples, 2 special attributes, 3 regular attributes)

Filter (1,179 / 1,179 examples): all ▾

Row No.	id	cluster	p256	hr32	t50
1	1	cluster_2	0.006	0.514	0.730
2	2	cluster_2	0.119	0.469	0.850
3	3	cluster_0	0.022	0.431	0.933
4	4	cluster_2	-0.027	0.450	0.867
5	5	cluster_0	0.043	0.484	1.093
6	6	cluster_0	-0.000	0.556	0.937
7	7	cluster_2	0.114	0.411	0.878
8	8	cluster_0	0.035	0.492	1.064
9	9	cluster_2	0.116	0.380	0.878
10	10	cluster_2	0.035	0.515	0.681
11	11	cluster_2	-0.025	0.426	0.741
12	12	cluster_2	0.035	0.396	0.736
13	13	cluster_2	-0.032	0.516	0.698
14	14	cluster_2	-0.068	0.444	0.704
15	15	cluster_2	0.180	0.525	0.802
16	16	cluster_2	0.007	0.530	0.728
17	17	cluster_2	-0.036	0.542	0.827
18	18	cluster_0	0.060	0.407	0.985
19	19	cluster_0	-0.047	0.440	0.952

[1] Process 2:25

FIGURE 5.57 A partial clustering of gamma-ray burst data.

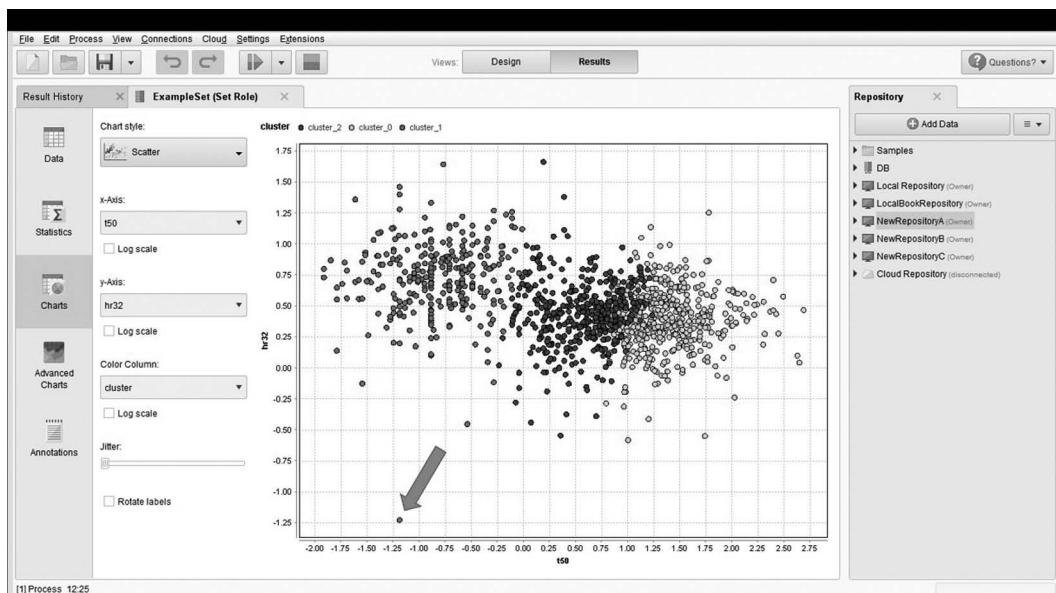


FIGURE 5.58 Three clusters of gamma-ray burst data.

- Resume the process to observe the decision tree, whose graphical and descriptive forms are given in Figures 5.59 and 5.60. The figures tell us that burst length is the primary attribute differentiating the three clusters.
- Click on *Cluster Model (Clustering)* to see the number of bursts in each cluster. Each cluster contains a sufficient number of instances to suggest the three-cluster model as a possibility.

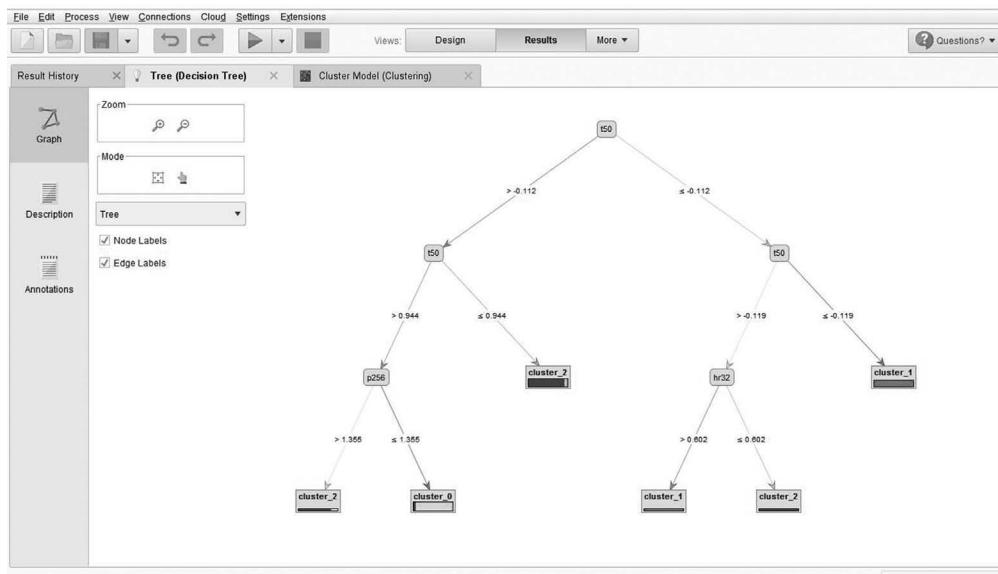


FIGURE 5.59 Decision tree illustrating a gamma-ray burst clustering.

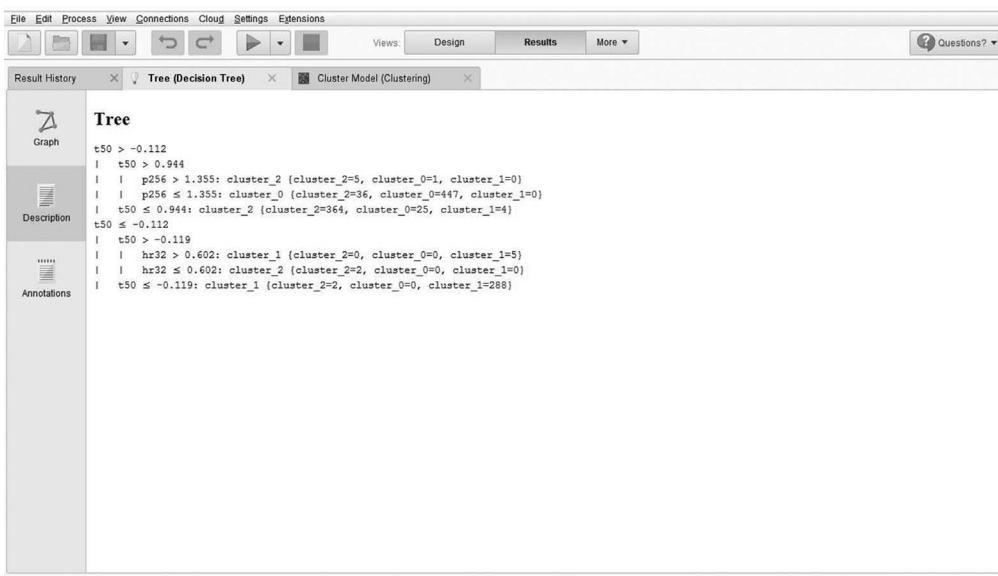


FIGURE 5.60 A descriptive form of a decision tree showing a clustering of gamma-ray burst data.

Here we used RapidMiner's *K*-means algorithm to show you how supervised learning can be applied to help interpret the results of an unsupervised clustering. End-of-chapter exercise 12 shows you how to apply *K*-means to help evaluate a set of input attributes for building a supervised learner model. Several additional uses of unsupervised clustering are highlighted in the chapters that follow.

5.6 ATTRIBUTE SELECTION AND NEAREST NEIGHBOR CLASSIFICATION

We have seen that several data mining techniques, including decision trees and rule induction algorithms, have a built-in attribute selection process. We also know that many data mining methods such as nearest neighbor classifiers and neural networks are unable to differentiate between relevant and irrelevant attributes. This is a problem as models built with several irrelevant attributes almost always perform poorly.

RapidMiner offers several optimization operators for use in situations where attribute selection must be preprocessed. For our example, we focus on RapidMiner's *Forward Selection* operator for attribute selection. Other attribute selection operators of interest include *Backward Elimination* and *Optimize Selection*. End-of-chapter exercise 9 introduces you to backward elimination. Chapter 6 presents the evolutionary approach used by the *Optimize Selection* operator.

Forward selection begins with an empty input attribute list. Input attributes are added to the list based on their performance. The single best-performing attribute starts the process, after which new attributes are added one by one according to how they perform in concert with those attributes currently on the list. This continues until a user-specified maximum number of attributes has been reached or the addition of a new attribute does not improve model performance.

To illustrate forward selection, we apply RapidMiner's nearest neighbor operator (*k*-NN) to the spam data set first introduced in Chapter 4. Recall that the data contain 4601 instances, where each instance is an e-mail message—2788 are valid messages, and the remaining are spam. All 57 input attributes are continuous. The output attribute is nominal, with 0 representing a valid e-mail message. Our goal is to build a classification model that incorporates a few relevant input attributes to achieve an acceptable level of classification correctness. Of primary importance is to minimize the classification of valid e-mails as spam.

To create a benchmark for our experiment, we applied *k*-NN using a training/test set scenario to the spam data. Specifically, we modified the model shown in Figures 5.28 and 5.29 by replacing the decision tree with *k*-NN. In the subprocess, we replaced the customer churn data set with *spam.xlsx* and removed the *Set Role* and *Filter Examples* operators. The test set performance of the benchmark model is displayed in Figure 5.61. Using all 57 input attributes, we saw a test set accuracy of 80.96% with 146 spam e-mails incorrectly classified as valid and 143 valid e-mails determined to be spam. Having the benchmark, we added forward selection to our model.

Forward selection is a nested operator where we are responsible for designing how performance is to be measured. For our experiment, we used *k*-NN not only as the main process classifier but also as the classifier for determining attribute worth. That is, through the

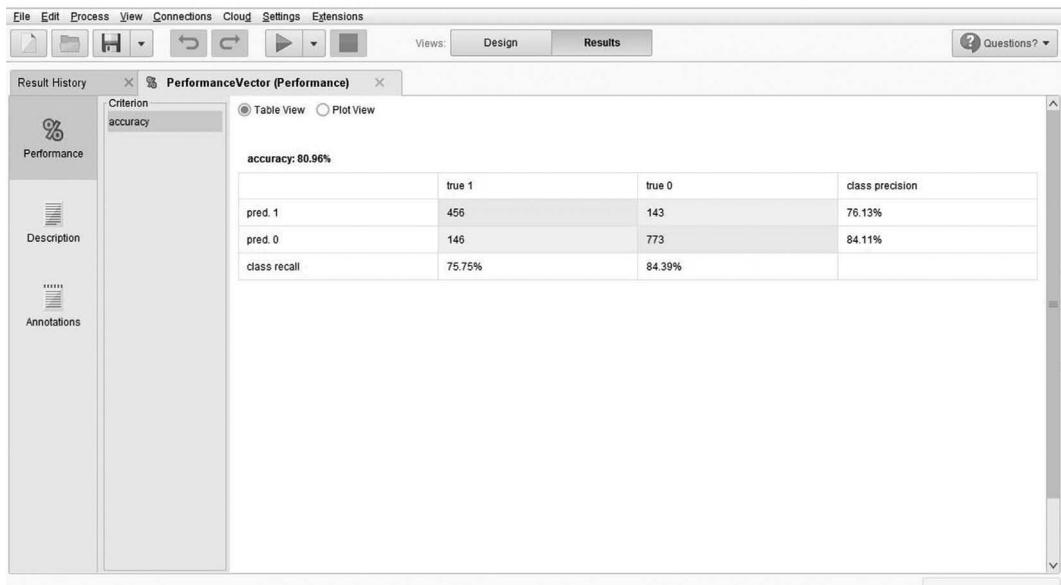


FIGURE 5.61 Benchmark performance for nearest neighbor classification.

forward selection process, k-NN determined which attributes were added to the growing list of selected attributes. Figures 5.62 through 5.64 display our model. Figure 5.64 shows that we chose the same training/test set scenario for individual attribute testing that we used in building the final model. That is, once forward selection is completed, the modified example set passes to the operator that splits the data, and the final model is created and tested.

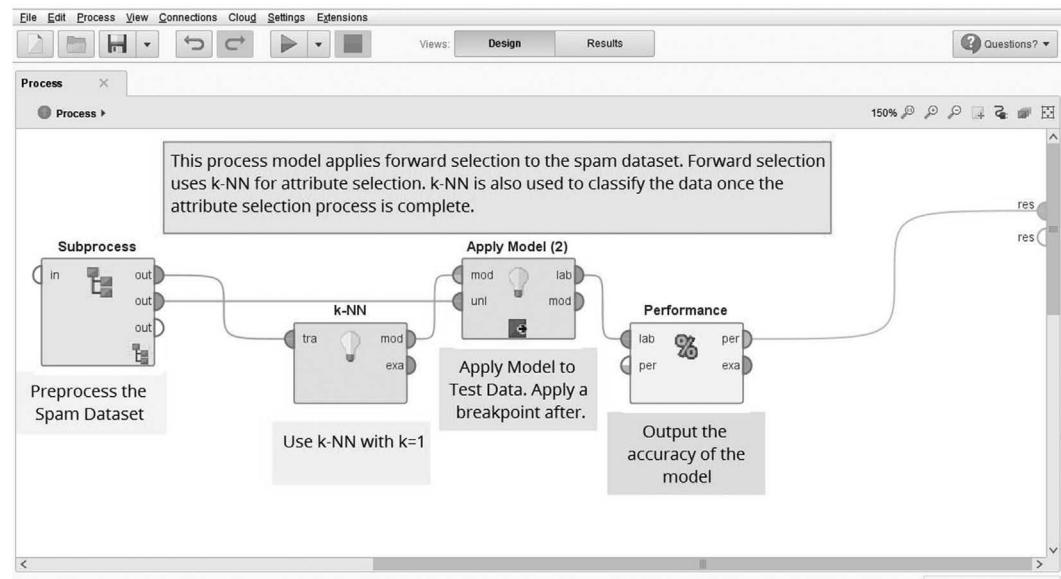


FIGURE 5.62 Main process design for nearest neighbor classification.

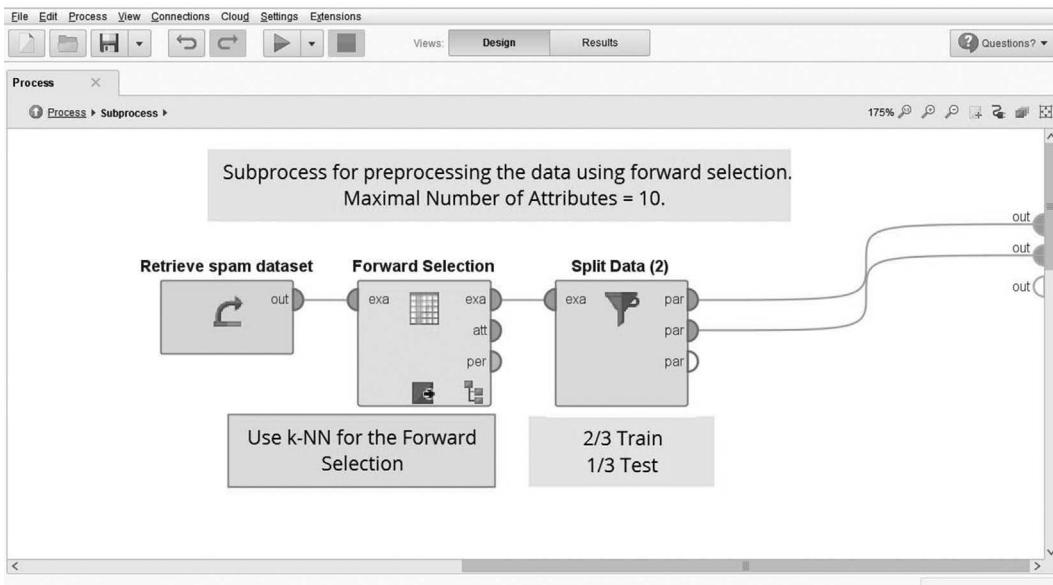


FIGURE 5.63 Subprocess for nearest neighbor classification.

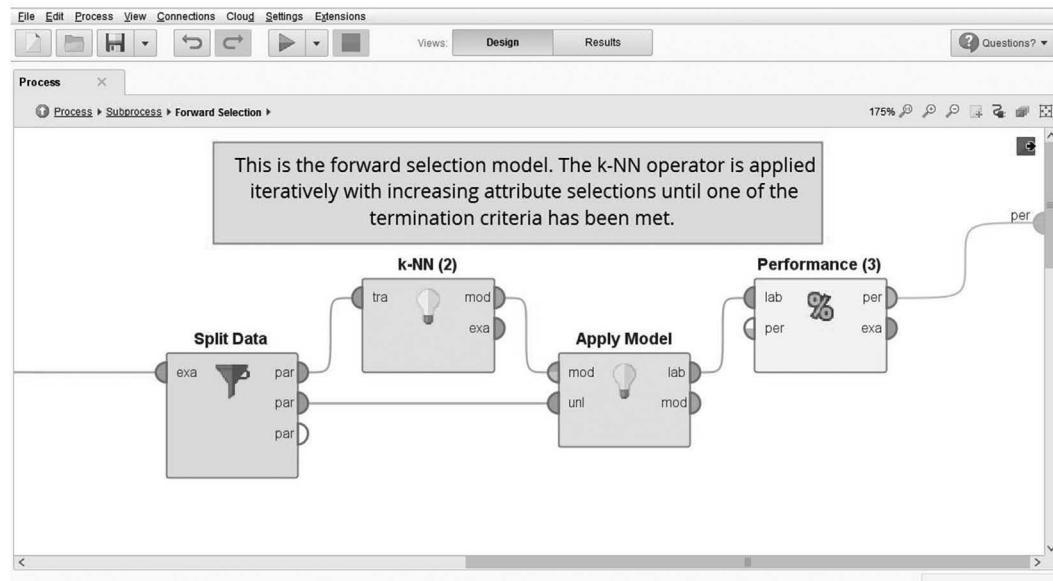


FIGURE 5.64 Forward selection subprocess for nearest neighbor classification.

The performance vector of the final model shown in Figure 5.65 displays a test set performance of 79.31%. Although forward selection has significantly reduced the number of attributes, the decrease in overall model performance together with the model's inability to identify valid emails (84.39% vs. 68.67%) is an unsatisfactory outcome. Exercise 10 offers one possibility for improving model performance.

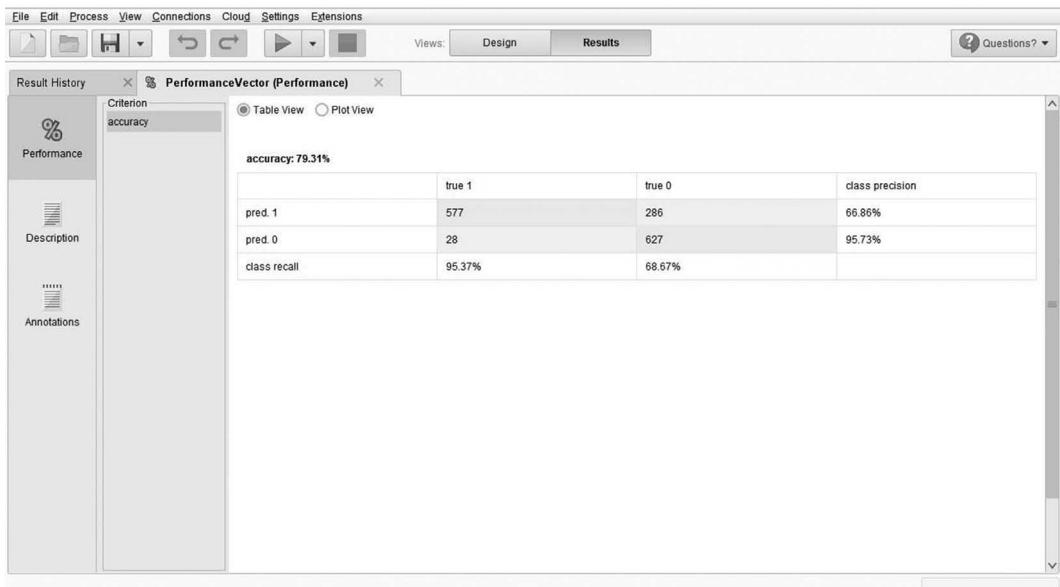


FIGURE 5.65 Performance vector when forward selection is used for choosing attributes.

Forward selection is but one of several attribute selection techniques, none of which are ideal. The counterpart to forward selection is backward elimination, which starts with the entire set of attributes and removes the least effective attributes one at a time. Both forward selection and backward elimination are brute-force techniques that can be computationally expensive. For example, suppose we use forward selection but replace our training/test set scenario with a 10-fold cross-validation. The first attribute selection would require constructing and testing 570 models. We would have 560 models before the second selection is made. The third would require 550 models. A final model of 10 attributes would likely need much more processing time than an extended coffee break. Furthermore, even though the next-best contributing attribute is always selected, there's nothing to guarantee that the sequential attribute selection process used by forward selection will create the best model.

As a final point, using the same classifier for attribute selection and creating the final model is known as a *wrapper* approach. The name comes from the fact that we are wrapping the learning technique into the attribute selection procedure. Another approach is to use an attribute selection method independent of the chosen classifier. For our example, we might try using the gain ratio method for attribute selection frequently used for building decision trees (see exercise 11).

5.7 CHAPTER SUMMARY

This chapter introduced RapidMiner Studio, the community version of RapidMiner's commercial data mining and predictive analytics tool. RapidMiner Studio supports most of the features seen with RapidMiner's commercial product, making it an excellent tool for learning about data mining and analysis. Here we used RapidMiner to show you how to

design problem-solving models from start to finish. However, the community version also provides several templates to use and modify as desired. Templates can oftentimes be a real timesaver when solving complex problems.

Freely downloadable extensions to RapidMiner Studio give access to Weka's modeling methods and attribute selection techniques, operators for analyzing textual data, operators for mining the Web, programming language interfaces, and more. RapidMiner cloud is also available with the studio version. Take time to investigate your options with RapidMiner cloud at <http://docs.rapidminer.com/cloud/>. In the chapters that follow, you will learn much more about RapidMiner's capabilities.

EXERCISES

1. Return to the getting started example shown in Figure 5.16.
 - a. Use the *Set Role* operator to change the label attribute to *Gender*.
 - b. Run your process to display the decision tree. Use the *Description* option to determine how the tree differentiates between male and female credit card customers.
 - c. Display a Histogram Color Chart with *Histogram* = *Gender* and *Color* = *Life Ins Promo*. What does the Histogram tell you?
 - d. Repeat (c) using *Color* = Credit Card Ins.
 - e. Repeat (c) using *Color* = Magazine Promo.
2. Chi-squared automatic interaction detection (CHAID) builds a decision tree by incorporating a statistical measure known as the chi square criterion for attribute selection. As CHAID is unable to process numeric data, all numeric attributes must be discretized during preprocessing. Use RapidMiner's CHAID operator to build a decision tree model from the credit card promotion data set. Use *Discretize by Binning* to discretize age with a bin size of 2. Next, modify the bin size to 3, and finally, use 4 bins. Give the resultant decision tree for each alternative.
3. Use Figure 5.27 to show the computation for precision and recall for the churn class.
4. Install the Weka extension to RapidMiner. Modify the process design in Figure 5.20 by replacing the decision tree operator with Weka's decision tree model W-J48. Compare the resultant performance vector with the one shown in Figure 5.27.
5. For this exercise, you are to modify the design shown in Figures 5.28 and 5.29 as follows:
 - a. Import *cardiologyMixed.xlsx* into your local data folder.
 - b. Replace the customer churn data set with the cardiology data set.
 - c. Modify the *set role* operator so *class* is set as the label attribute.
 - d. Remove the *filter examples* operator as it is not needed.

- e. Create a new subprocess to contain the *Apply Model* and *Performance* vectors. Give your new subprocess the name *results*. The main process design should now show three operators.
 - f. Run your process and report your results. How many healthy patients were classified as sick? How many sick individuals were given a clean bill of health?
6. Choose one of RapidMiner's templates other than *Market Basket Analysis*. Add breakpoints to the template. Examine the output of each operator as the model executes. Write a report about the purpose of the template and how each operator helps fulfill that purpose. Use operator documentation as appropriate.
7. Create a new process that uses the *correlation matrix* operator to create a correlation matrix for the gamma-ray burst data set. Be sure to connect the *mat* port to the *res* port to see the matrix. Which pairs of attributes are highly correlated? Examine each pair of highly correlated attributes with a scatterplot chart.
8. Modify the *K*-means example of Section 5.5 in the following way:
- a. Create a subprocess for preprocessing the data.
 - b. Create a subprocess to hold the operators that follow the clustering operator.
 - c. Replace the decision tree operator with the *Rule Induction* operator.
 - d. Run the process and compare the cluster definitions shown with rule induction with those of the decision tree.
9. Implement the process shown in Figures 5.62 through 5.64 but replace *Forward Selection* with *Backward Elimination*. Set the *maximum number of eliminations* at 10. Set the *stopping behavior* parameter to *with decrease more than*. Set *maximal relative decrease* at 0.03. Summarize your results.
10. k-NN's *k* parameter specifies the number of nearest neighbors used for comparative purposes. With a default of *k* = 1, k-NN places a new instance in the class of its closest neighbor. Implement the forward selection model given in Section 5.6. Experiment with alternative values of *k* for the k-NN operator used within the forward selection subprocess (e.g., *k* = 2, 3, 4, 5). Make a table showing the overall classification accuracy, precision, and recall for each value of *k*. Summarize your results.
11. Implement the process design given in Section 5.6. Within *Subprocess*, set the *sampling type* parameter in *Split Data* to *linear* and set *maximal number of attributes* in *Forward Selection* at 10. Run your process. Next, within the *Forward Selection* subprocess, replace k-NN with the *Decision Tree* operator. Make sure k-NN is still used for building the final model. Run your process. Compare the two performance vectors.

12. For this exercise, you will use unsupervised clustering together with the numerical form of the cardiology patient data to help determine if the input attributes taken as a set are a viable choice for supervised learning. Here is the procedure:

- Add *CardiologyNumerical.xlsx* to the data folder in your local repository. This is the numerical form of the cardiology patient data introduced in Chapter 2. Be sure to declare attribute *class* as binomial and make *class* the label attribute.
- Create the process shown in Figure 5.66. Specifically,
 - For the K-means clustering, set the cluster number at 2 and be sure to check the *add cluster attribute* box.
 - The clustering is written to an Excel file, which will contain a column showing the cluster number (cluster_0 or cluster_1) and the class (0 or 1) of each instance.
 - The multiply operator makes four copies of the output. The output is then filtered. One filter shows instances with *cluster* = *cluster_0* and *class* = 0. A second filter shows all instances with *cluster* = *cluster_0* and *class* = 1. A third filter gives all instances with *cluster* = *cluster_1* and *class* = 0. A fourth filter shows all instances with *cluster* = *cluster_1* and *class* = 1.
- Run your process.

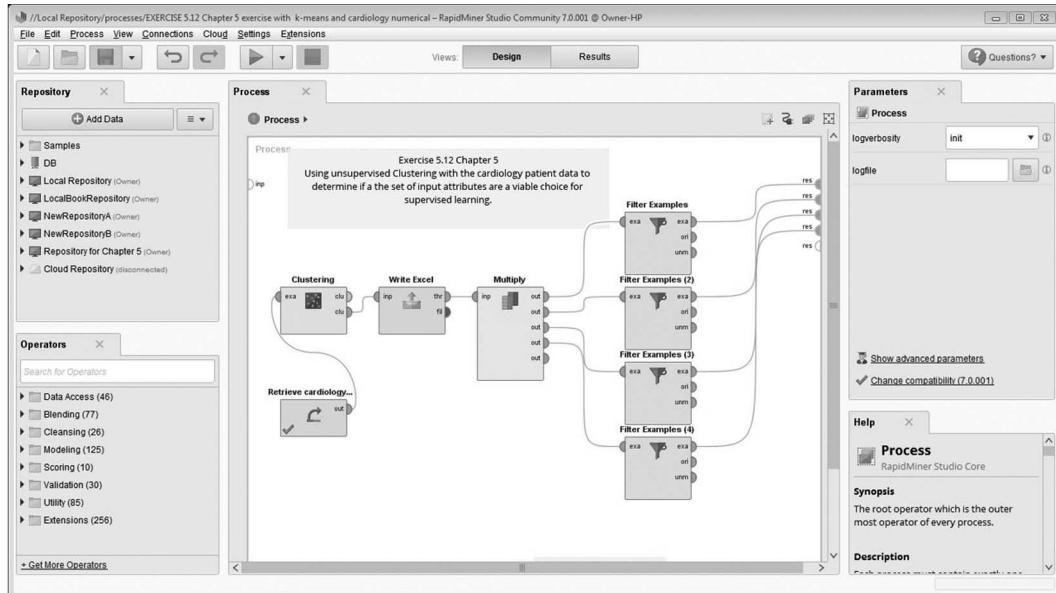


FIGURE 5.66 Unsupervised clustering for attribute evaluation.

- d. Examine the output to determine the number of class 0 and class 1 instances seen in cluster_0 as well as the number of class 0 and class 1 instances found in cluster_1. A best result will show one cluster with a majority of the class 1 instances and the second cluster having a majority of the instances in class 0.
 - e. Record your results by giving the percent of instances from each class contained within the two clusters. Do the instances cluster in a way consistent with their class designation?
13. Repeat exercise 12 using the mixed form of the cardiology patient dataset. As the data set contains mixed data types, set the *K-means* parameter *measure types* to *MixedMeasures*. Modify the filters to reflect the named classes (i.e., replace 0 with Sick and 1 with Healthy). Compare your results with those seen in exercise 12. Is there a difference in how the instances cluster?
14. Repeat exercise 12 using the *creditscreening.xlsx* data set first described in Chapter 4. As the data set contains mixed data types, set the *K-means* parameter *measure types* to *MixedMeasures*. Several attribute values within the data set are flagged with a question mark, indicating a missing value. Preprocess the data with the *Replace Missing Values* operator.
15. Use subgroup discovery to experiment with a data set of your choice. List what you consider to be two rules of most significance. Justify your answer.

The Knowledge Discovery Process

CHAPTER OBJECTIVES

- *Know the seven-step knowledge discovery in databases (KDD) process*
- *Know that data mining is one step of the KDD process*
- *Understand techniques for normalizing, converting, and smoothing data*
- *Understand methods for attribute selection and creation*
- *Know how to implement methods for detecting outliers*
- *Recognize the advantages and disadvantages of methods for dealing with missing data*

In Chapter 1, we described a simple data analytics process model. In Section 6.1 of this chapter we introduce a formal seven-step process model for knowledge discovery. In Sections 6.2 through 6.8, we examine each step of this model in detail. In Section 6.9, we introduce a second knowledge discovery model known as the Cross Industry Standard Process for Data Mining (CRISP-DM).

6.1 A PROCESS MODEL FOR KNOWLEDGE DISCOVERY

In Chapter 1, we introduced the phrase *knowledge discovery in databases (KDD)* as an interactive, iterative procedure that attempts to extract implicit, previously unknown, and potentially useful knowledge from data. As much of today's data is not found in a traditional data warehouse, KDD is most often associated with *knowledge discovery from data*.

Several variations of the KDD process model exist. Variations describe the KDD process from 4 to as many as 12 steps. Although the number of steps may differ, most descriptions show consistency in content. Here we expand the process model displayed in Figure 1.3

to a seven-step approach for knowledge discovery. The seven-step KDD process model is shown in Figure 6.1. A brief description of each step follows:

1. *Goal identification.* The focus of this step is on understanding the domain being considered for knowledge discovery. We write a clear statement about what is to be accomplished. A hypothesis offering a likely or desired outcome can be stated.
2. *Creating a target data set.* With the help of one or more human experts and knowledge discovery tools, we choose an initial set of data to be analyzed.

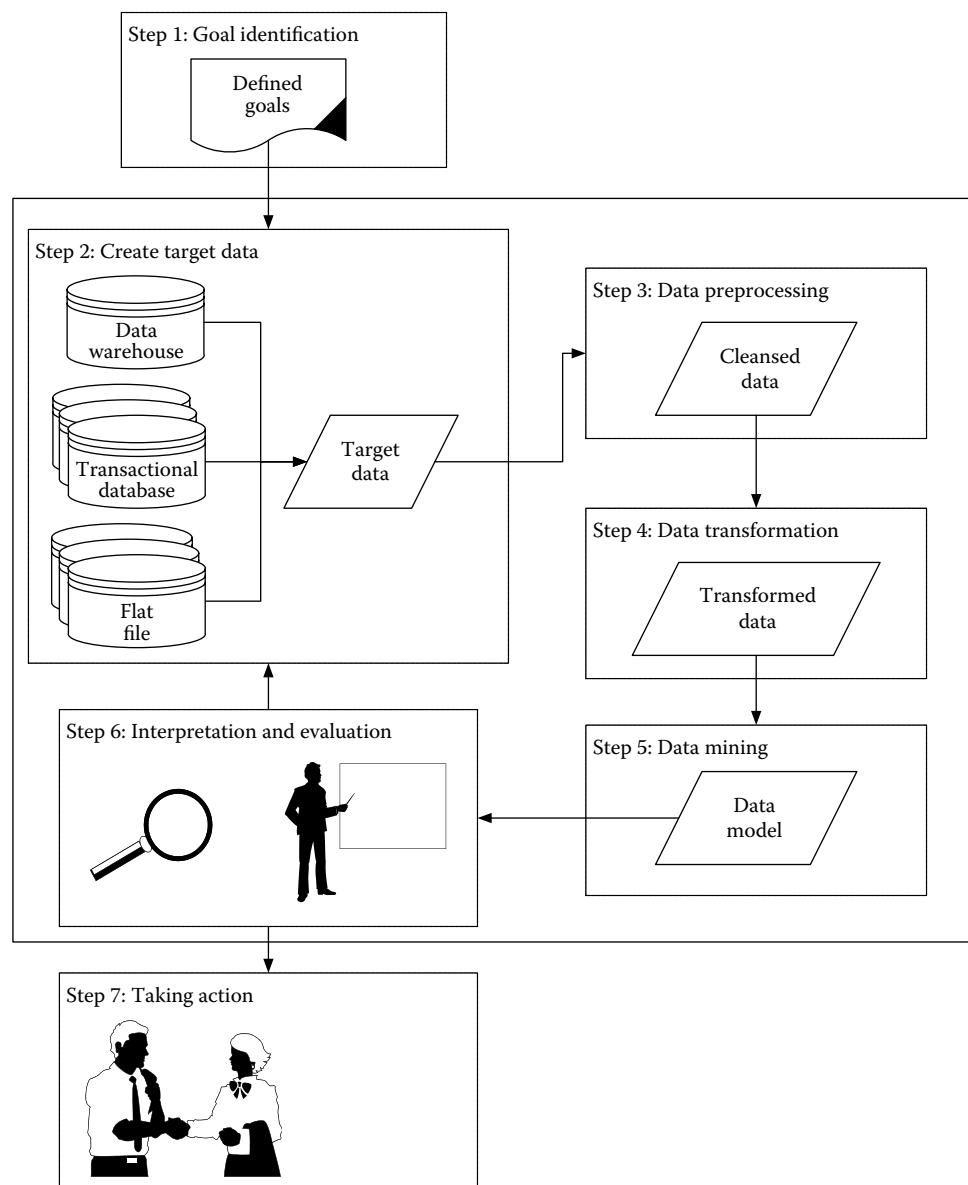


FIGURE 6.1 A seven-step KDD process model.

3. *Data preprocessing.* We use available resources to deal with noisy data. We decide what to do about missing data values and how to account for time-sequence information.
4. *Data transformation.* Attributes and instances are added and/or eliminated from the target data. We decide on methods to normalize, convert, and smooth data.
5. *Data mining.* A best model for representing the data is created by applying one or more data mining algorithms.
6. *Interpretation and evaluation.* We examine the output from step 5 to determine if what has been discovered is both useful and interesting. Decisions are made about whether to repeat previous steps using new attributes and/or instances.
7. *Taking action.* If the discovered knowledge is deemed useful, the knowledge is incorporated and applied directly to appropriate problems.

Weka and RapidMiner are well equipped to perform the tasks defining the KDD process model. As you read Sections 6.2 through 6.8, it is worth your time to investigate the functions and operators available with Weka and RapidMiner for each step of the KDD process model.

6.2 GOAL IDENTIFICATION

A main objective of goal identification is to clearly define what is to be accomplished. This first step is in many ways the most difficult, as decisions about resource allocations as well as measures of success need to be determined. Whenever possible, broad goals should be stated in the form of specific objectives. Here is a partial list of things that should be considered at this stage:

- A clear problem statement is provided as well as a list of criteria to measure success and failure. One or more hypotheses offering likely or desired outcomes may be established.
- The choice of a data mining tool or set of tools is made. The choice of a tool depends on several factors, including the level of explanation required and whether learning is supervised, unsupervised, or a combination of both techniques.
- An estimated project cost is determined. A plan for human resource management is offered.
- A project completion/product delivery date is given.
- Legal issues that may arise from applying the results of the discovery process are taken into account.
- A plan for maintenance of a working system is provided as appropriate. As new data become available, a main consideration is a methodology for updating a working model.

Our list is by no means exhaustive. As with any software project, more complex problems require additional planning. Of major importance are the location, availability, and condition of resource data.

6.3 CREATING A TARGET DATA SET

A viable set of resource data is of primary importance for any data mining project to succeed. Figure 6.1 shows target data being extracted from three primary sources—a data warehouse, one or more transactional databases, or one or several flat files. Many data mining tools require input data to be in a flat file or spreadsheet format. If the original data are housed in a flat file, creating the initial target data is straightforward. Let's examine the other possibilities.

Database management systems (DBMSs) store and manipulate transactional data. The computer programs in a DBMS are able to quickly update and retrieve information from a stored database. The data in a DBMS is often structured using the relational model. A *relational database* represents data as a collection of tables containing rows and columns. Each column of a table is known as an attribute, and each row of the table stores information about one data record. The individual rows are called *tuples*. All tuples in a relational table are uniquely identified by a combination of one or more table attributes.

A main goal of the relational model is to reduce data redundancy so as to allow for quick access to information in the database. A set of normal forms that discourage data redundancy define formatting rules for relational tables. If a relational table contains redundant data, the redundancy is removed by decomposing the table into two or more relational structures. In contrast, the goal of data mining is to uncover the inherent redundancy in data. Therefore, one or more relational join operations are usually required to restructure data into a form amenable for data mining.

To see this, consider the hypothetical credit card promotion database we defined in Table 2.3 of Chapter 2. Recall the following table attributes: *income range*, *magazine promotion*, *watch promotion*, *life insurance promotion*, *credit card insurance*, *gender*, and *age*. The data in Table 2.3 are not a database at all but represent a flat file structure extracted from a database such as the one shown in Figure 6.2. The Acme credit card database contains tables about credit card billing information and orders, in addition to information about credit card promotions. As you can see, the promotional information shown in Table 2.3 is housed in two relational tables within the database. In Chapter 14, we detail the database pictured in Figure 6.2 and show you how the database can be restructured for a data analytics environment.

The possibility also exists of extracting data from multiple databases. If target data are to be extracted from more than one source, the transfer process can be tedious. Consider a simple example where one operational database stores customer gender with the coding *male* = 1 and *female* = 2. A second database stores the gender coding as *male* = M and *female* = F. The coding for *male* and *female* must be consistent throughout all records in the target data, or the data will be of little use. The process of promoting this consistency when transporting the data is a form of *data transformation*. Other types of data transformations are discussed in Section 6.5.

Another possibility for harvesting target data is the data warehouse. Chapter 1 described the data warehouse as a historical database designed specifically for decision support.

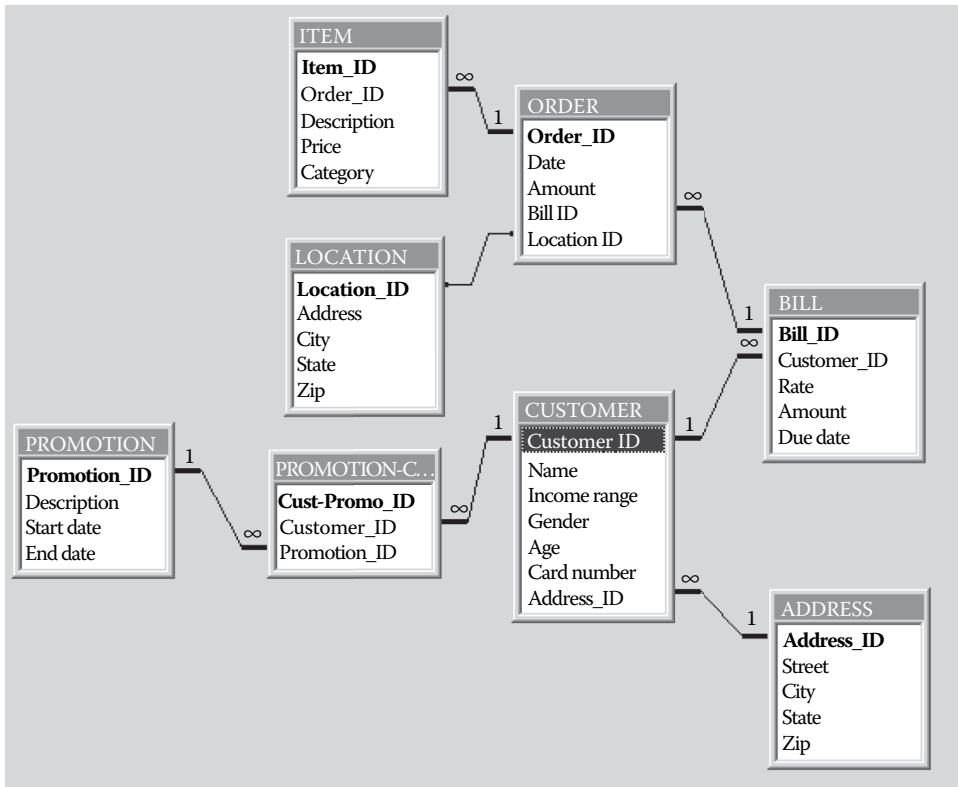


FIGURE 6.2 The Acme credit card database.

In Chapter 14, you will see how a well-structured data warehouse best supports the redundancy required to build learner models through the process of data mining.

A fourth possibility, not shown in the diagram, is when data requires a distributed environment supported by a cluster of servers. This case was briefly described in Chapter 1, where we have, for example, a system like the Hadoop Distributed File System storing data and MapReduce for processing the data and aggregating a final solution. With distributed data, the KDD process model must be supplemented with the added complexities of data distribution and solution aggregation. Lastly, a special case exists with streaming data, where real-time analysis makes data preprocessing extremely difficult at best.

6.4 DATA PREPROCESSING

Most data preprocessing is in the form of data cleaning, which involves accounting for noise and dealing with missing information. Ideally, the majority of data preprocessing takes place before data are permanently stored in a structure such as a data warehouse.

6.4.1 Noisy Data

Noise represents random error in attribute values. In very large data sets, noise can come in many shapes and forms. Common concerns with noisy data include the following:

- How do we find duplicate records?
- How can we locate incorrect attribute values?
- What data smoothing operations should be applied to our data?
- How can we find and process outliers?

6.4.1.1 Locating Duplicate Records

Suppose a certain weekly publication has 100,000 subscribers and 0.1% of all mailing list entries have erroneous dual listings under a variation of the same name (e.g., Jon Doe and John Doe). Therefore, 100 extra publications are processed and mailed each week. At a processing and mailing cost of \$2.00 for each publication, the company spends over \$10,000 each year in unwarranted costs. Ideally, errors such as these are uncovered as data are moved from an operational environment to a data warehouse facility. Automated graphical tools to assist with data cleaning and data movement do exist. However, the responsibility for data transition still lies in the hands of the data warehouse specialist.

6.4.1.2 Locating Incorrect Attribute Values

Finding errors in categorical data presents a problem in large data sets. Some data mining tools offer a summary of frequency values and/or predictability scores for categorical attributes. We should consider attribute values having predictability scores near 0 as error candidates.

A numeric value of 0 for an attribute such as blood pressure or weight is an obvious error. Such errors often occur when data are missing and default values are assigned to fill in for missing items. In some cases, such errors can be seen by examining class mean and standard deviation scores. However, if the data set is large and only a few incorrect values exist, finding such errors can be difficult. Some data analysis tools allow the user to input a valid range of values for numerical data. Instances with attribute values falling outside the valid range limits are flagged as possible error candidates.

6.4.1.3 Data Smoothing

Data smoothing is both a data cleaning and data transformation process. Several data smoothing techniques attempt to reduce the number of values for a numeric attribute. Some classifiers, such as neural networks, use functions that perform data smoothing during the classification process. When data smoothing is performed during classification, the data smoothing is said to be internal. External data smoothing takes place prior to classification. Rounding and computing mean values are two simple external data smoothing techniques. Mean value smoothing is appropriate when we wish to use a classifier that does not support numerical data and would like to retain coarse information about numerical attribute values. In this case, all numerical attribute values are replaced by a corresponding class mean.

Another common data smoothing technique attempts to find atypical (outlier) instances in the data. Outliers often represent errors in the data whereby the items should be corrected

or removed, for instance, a credit card application where applicant age is given as -21. In other cases, it may be counterproductive to remove outliers. For example, in credit card fraud detection, the outliers are those items we are most interested in finding.

Several outlier detection techniques exist, including unsupervised clustering methods, nearest neighbor techniques, and various statistical approaches. Unsupervised outlier detection methods often make the assumption that ordinary instances will cluster together. If definite patterns in the data do not exist, unsupervised techniques will flag an undue number of ordinary instances as outliers. In Chapter 13, you will see how supervised techniques for dealing with rarity can be used for outlier detection. Here we use RapidMiner to show you one approach for detecting candidate outlier instances.

6.4.1.4 Detecting Outliers

RAPIDMINER TUTORIAL

RapidMiner has several operators for detecting outliers. Each takes a slightly different approach to outlier detection. Figure 6.3 shows a simple process model that applies the *Detect Outlier (Distances)* operator to the *diabetes.xlsx* data set found in *datasetsRapidMiner.zip*. The data set contains 768 instances, 500 of which represent individuals who tested negative for diabetes. The data include eight numeric input attributes and a nominal output attribute indicating the outcome of a test for diabetes. Sheet2 of *diabetes.xlsx* offers a description of each attribute found within the data. Let's see if we can find some outliers!

- Create and save a new process.
- Add the *diabetes* data set with attribute *Diabetes* specified as label to your local repository and create the process model shown in Figure 6.3.

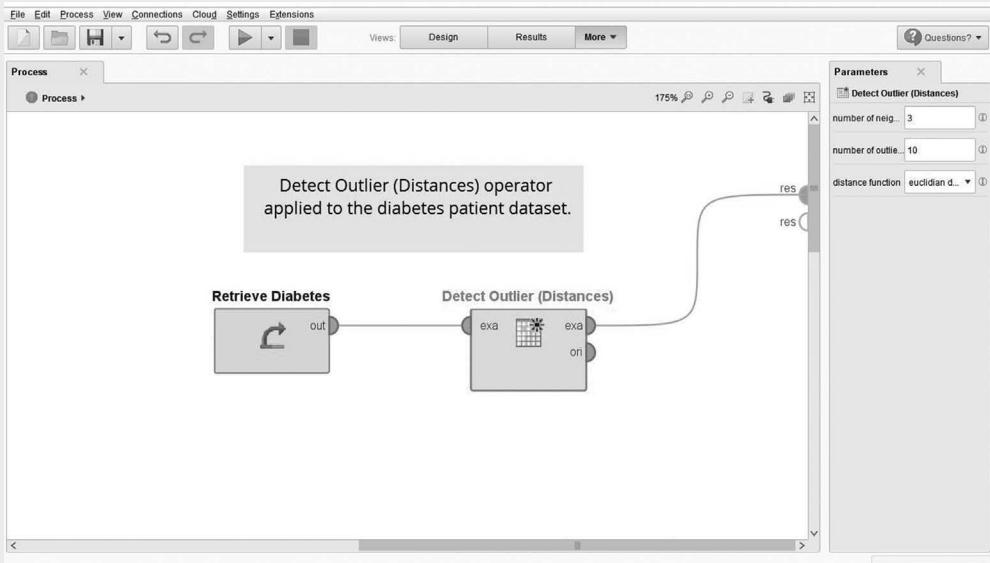
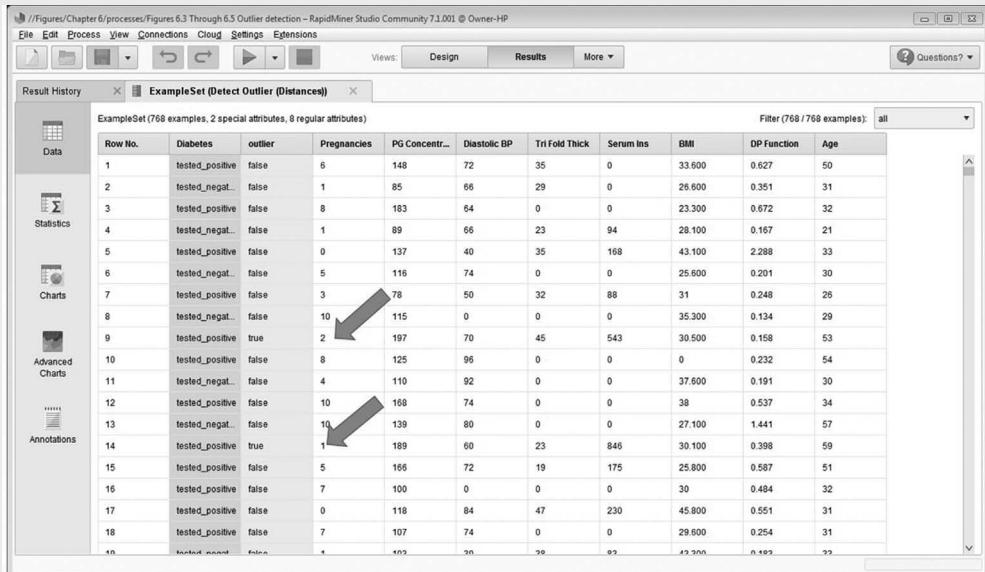


FIGURE 6.3 A process model for detecting outliers.

- The *Detect Outliers (Distances)* operator *distance function* parameter gives us a choice of five distance functions for computing distance scores. Select the *Euclidean Distance* function.
- The *number of outliers* parameter tells us to choose the total number of instances to be designated as outliers. Set the value of this parameter at 10.
- The *number of neighbors* parameter specifies the number of nearest neighbors to be used for determining the outlier instances. Set this value at 3. With the value set at 3, the outlier score for each instance will be based on its three closest neighbors. The 10 instances most distant from their three closest neighbors will be considered outliers.
- Run your process.
- Figure 6.4 shows the first 19 instances of the data set together with an added column indicating whether each instance is an outlier. The arrows in the figure point to the first two outlier instances.
- To see all 10 outliers, click on the *Outlier* column and then click on the *up arrow*.

Figure 6.5 displays the list of the ten outliers.



The screenshot shows a RapidMiner interface with the following details:

- File Edit Process View Connections Cloud Settings Extensions**
- Views: Design Results More**
- Result History** tab is selected.
- ExampleSet (Detect Outlier (Distances))** is displayed.
- Data** view is active.
- Table Headers:** Row No., Diabetes, outlier, Pregnancies, PG Concentr..., Diastolic BP, Tri Fold Thick, Serum Ins, BMI, DP Function, Age.
- Table Rows (Visible):**

Row No.	Diabetes	outlier	Pregnancies	PG Concentr...	Diastolic BP	Tri Fold Thick	Serum Ins	BMI	DP Function	Age
1	tested_positive	false	6	148	72	35	0	33.600	0.627	50
2	tested_negat...	false	1	85	66	29	0	26.600	0.351	31
3	tested_positive	false	8	183	64	0	0	23.300	0.672	32
4	tested_negat...	false	1	89	66	23	94	28.100	0.167	21
5	tested_positive	false	0	137	40	35	168	43.100	2.288	33
6	tested_negat...	false	5	116	74	0	0	25.600	0.201	30
7	tested_positive	false	3	78	50	32	88	31	0.248	26
8	tested_negat...	false	10	115	0	0	0	35.300	0.134	29
9	tested_positive	true	2	197	70	45	543	30.500	0.158	53
10	tested_positive	false	8	125	96	0	0	0	0.232	54
11	tested_negat...	false	4	110	92	0	0	37.600	0.191	30
12	tested_positive	false	10	168	74	0	0	38	0.537	34
13	tested_negat...	false	10	139	80	0	0	27.100	1.441	57
14	tested_positive	true	1	189	60	23	846	30.100	0.398	59
15	tested_positive	false	5	166	72	19	175	25.800	0.587	51
16	tested_positive	false	7	100	0	0	0	30	0.484	32
17	tested_positive	false	0	118	84	47	230	45.800	0.551	31
18	tested_positive	false	7	107	74	0	0	29.600	0.254	31
19	tested_negative	false	4	102	56	26	63	43.300	0.493	22
- Annotations:** Two arrows point to rows 7 and 10, which are highlighted in yellow.

FIGURE 6.4 Two outlier instances from the diabetes patient data set.



FIGURE 6.5 Ten outlier instances from the diabetes patient data set.

Several end-of-chapter exercises ask you to compare the accuracy of models built with and without candidate outlier removal. As we are responsible for determining the total number of outliers, several experiments incorporating a training/test set scenario with differing values for the number of outliers may be needed to correctly identify true outlier instances.

Lastly, since outlier detection operators do not take attribute significance into account, the outliers detected by an operator may not be useful. When using data mining methods such as nearest neighbor classifiers and neural networks that do not have attribute selection built into the modeling process, it is best to first apply an attribute selection technique to the data prior to attempting outlier detection.

6.4.2 Missing Data

Missing data items present a problem that can be dealt with in several ways. In most cases, missing attribute values indicate lost information. For example, a missing value for the attribute *age* certainly indicates a data item that exists but is unaccounted for. However, a missing value for *salary* may be taken as an unentered data item, but it could also indicate an individual who is unemployed. Some data mining techniques are able to deal directly with missing values. However, many classifiers require all attributes to contain a value. The following are possible options for dealing with missing data *before* the data are presented to a data mining algorithm.

- *Discard records with missing values.* This method is most appropriate when a small percent of the total number of instances contain missing data and we can be certain that missing values do indeed represent lost information.

- For real-valued data, replace missing values with the class mean. In most cases, this is a reasonable approach for numerical attributes. Options such as replacing missing numeric data with a zero or some arbitrarily large or small value are generally poor choices.
- Replace missing attribute values with the values found within other highly similar instances. This technique is appropriate for either categorical or numeric attributes.

Some data mining techniques allow instances to contain missing values. Here are three ways that data mining techniques deal with missing data while learning:

1. *Ignore missing values.* Several data mining algorithms, including neural networks and the Bayes classifier (Chapter 11), use this approach.
2. *Treat missing values as equal comparisons.* This approach is dangerous with very noisy data in that dissimilar instances may appear to be very much alike.
3. *Treat missing values as unequal comparisons.* This is a pessimistic approach but may be appropriate. Two similar instances containing several missing values will appear dissimilar.

Finally, a slightly different approach is to use supervised learning to determine likely values for missing data. When the missing attribute is categorical, we designate the attribute with missing values as an output attribute. The instances with known values for the attribute are used to build a classification model. The created model is then summoned to classify the instances with missing values. For numerical data, we can use an estimation mining tool, such as a neural network, and apply the same strategy.

6.5 DATA TRANSFORMATION

Data transformation can take many forms and is necessary for a variety of reasons. We offer a description of some familiar data transformations in the following sections.

6.5.1 Data Normalization

A common data transformation involves changing numeric values so they fall within a specified range. Classifiers such as neural networks do better with numerical data scaled to a range between 0 and 1. Normalization is particularly appealing with distance-based classifiers, because by normalizing attribute values, attributes with a wide range of values are less likely to outweigh attributes with smaller initial ranges. Four common normalization methods include the following:

- *Decimal scaling.* Decimal scaling divides each numerical value by the same power of 10. For example, if we know the values for an attribute range between -1000 and 1000, we can change the range to -1 and 1 by dividing each value by 1000.

- *Min-max normalization.* Min-max is an appropriate technique when minimum and maximum values for an attribute are known. The formula is

$$\text{newValue} = \frac{\text{originalValue} - \text{oldMin}}{\text{oldMax} - \text{oldMin}} (\text{newMax} - \text{newMin}) + \text{NewMin}$$

where *oldMax* and *oldMin* represent the original maximum and minimum values for the attribute in question. *NewMax* and *newMin* specify the new maximum and minimum values. *newValue* represents the transformation of *originalValue*. This transformation is particularly useful with neural networks where the desired range is [0,1]. In this case, the formula simplifies to

$$\text{newValue} = \frac{\text{originalValue} - \text{oldMin}}{\text{oldMax} - \text{oldMin}}$$

- *Normalization using Z-scores.* Z-score normalization converts a value to a standard score by subtracting the attribute mean (μ) from the value and dividing by the attribute standard deviation (σ). Specifically,

$$\text{newValue} = \frac{\text{originalValue} - \mu}{\sigma}$$

This technique is particularly useful when maximum and minimum values are not known.

- *Logarithmic normalization.* The base b logarithm of a number n is the exponent to which b must be raised to equal n . For example, the base 2 logarithm of 64 is 6 because $2^6 = 64$. Replacing a set of values with their logarithms has the effect of scaling the range of values without loss of information.

6.5.2 Data Type Conversion

Many data mining tools, including neural networks and some statistical methods, cannot process categorical data. Therefore, converting categorical data to a numeric equivalent is a common data transformation. On the other hand, some data mining techniques are not able to process numeric data in their original form. For example, most decision tree algorithms discretize numeric values by sorting the data and considering alternative binary splits of the data items.

6.5.3 Attribute and Instance Selection

Classifiers such as decision trees with built-in attribute selection are less likely to suffer from the effects of data sets containing attributes of little predictive value. Unfortunately,

many data mining algorithms such as neural networks and nearest neighbor classifiers are unable to differentiate between relevant and irrelevant attributes. This is a problem, as data mining algorithms do not generally perform well with data containing a wealth of attributes that are not predictive of class membership. Furthermore, it has been shown that the number of training instances needed to build accurate supervised learner models is directly affected by the number of irrelevant attributes in the data. To overcome these problems, we must make decisions about which attributes and instances to use when building our data mining models. The following is a possible algorithm to help us with attribute selection:

-
1. Given N attributes, generate the set S of all possible attribute combinations.
 2. Remove the first attribute combination from set S and generate a data mining model M using these attributes.
 3. Measure the goodness of model M .
 4. Until S is empty,
 - a. Remove the next attribute combination from S and build a data mining model using the next attribute combination in S .
 - b. Compare the goodness of the new model with the saved model M . Call the better model M and save this model as the best model.
 5. Model M is the model of choice.
-

This algorithm will surely give us a best result. The problem with the algorithm lies in its complexity. If we have a total of n attributes, the total number of attribute combinations is $2^n - 1$. The task of generating and testing all possible models for any data set containing more than a few attributes is not possible. Let's investigate a few techniques we can apply.

6.5.3.1 Wrapper and Filtering Techniques

Attribute selection methods are generally divided into filtering and wrapper techniques. *Filtering* methods select attributes based on some measure of quality independent of the algorithm used to build a final model. With *wrapper* techniques, attribute goodness is measured in the context of the learning algorithm used to build the final model. That is, the algorithm is wrapped into the attribute selection process.

One example of a simple attribute filtering technique is to use a data mining algorithm such as a decision tree for attribute selection. The selected attributes are then employed by a competing algorithm such as a nearest neighbor classifier to build a final model. This technique is plausible as the final model will often perform better than the decision tree used for attribute selection.

In Chapter 4, we applied Weka's supervised *AttributeSelection* filter with *Ranker* as the search technique to the spam data set. Recall that *Ranker* ranks the input attributes from most to least predictive of class membership based upon the chosen evaluation method independent of the learning scheme for building the final model.

In Chapter 5, we applied a wrapper technique to the spam data set when we used *forward selection* together with RapidMiner's nearest neighbor operator (k-NN) to select a

best set of input attributes. This was a wrapper approach as k-NN was utilized to build the final learner model.

6.5.3.2 More Attribute Selection Techniques

In addition to applying wrapper and filtering techniques, several other steps can be taken to help determine which attributes to eliminate from consideration:

1. Highly correlated input attributes are redundant. Most data mining tools build better models when only one attribute from a set of highly correlated attributes is designated as an input value. Methods for detecting correlated attributes include scatter-plot diagrams, RapidMiner's *Remove Correlated Attributes* operator, and MS Excel's *CORREL* function.
2. In Chapter 3, we defined predictability scores for categorical data. Any categorical attribute containing a value with a predictability score greater than a chosen threshold can be considered for elimination. This is the case because most domain instances will have the same value for the attribute. As the domain attribute-value predictability score increases, the ability of the value to differentiate the individual classes decreases.
3. When learning is supervised, numerical attribute significance can be determined by comparing class mean and standard deviation scores. In Chapter 7, we show you how to use two of RapidMiner's statistical operators to differentiate between useful and useless numeric attributes.
4. *Principal component analysis* is a statistical technique that looks for and removes possible correlational redundancy within the data by replacing the original attributes with a smaller set of artificial values. You can learn about this technique by reading the documentation for RapidMiner's *PCA* operator.
5. A *self-organizing map* or SOM is a neural network trained with unsupervised learning that can be used for attribute reduction. SOMs are described in Chapters 8 through 10 when we focus on neural network models.

All but the second technique can be applied to both supervised learning and unsupervised clustering. Unfortunately, with unsupervised clustering, numerical attribute significance cannot be computed as predefined classes do not exist. However, we can experiment with subsets of likely attribute choices and use a suitable measure of cluster quality to help us determine a best set of numerical attributes.

6.5.3.3 Genetic Learning for Attribute Selection

An interesting approach to attribute selection makes use of genetic learning. The method is appealing because by incorporating an evaluation function, we eliminate the combinatorial problems seen with the try-everything approach but are still able to achieve satisfactory results. The approach is especially appealing when a wealth of irrelevant attributes exist in the data. The method is best illustrated by an example.

TABLE 6.1 Initial Population for Genetic Attribute Selection

Population Element	Income Range	Magazine Promotion	Watch Promotion	Credit Card Insurance	Gender	Age
1	1	0	0	1	1	1
2	0	0	0	1	0	1
3	0	0	0	0	1	1

Let's consider the credit card promotion database described in Chapter 2. Once again, we assume that the output attribute is *life insurance promotion*. Table 6.1 shows an initial population of three elements.

Each element tells us which attributes to use when building the associated learner model. A 1 indicates that the attribute is an input attribute, and a 0 specifies the attribute as unused. The technique is as follows:

1. Choose appropriate training and test data.
2. Use a random selection process to initialize a population of elements.
3. Build a supervised learner model for each population element. Each model is constructed with the attributes specified by the corresponding element from the population. For example, the learner model for population element 1 uses input attributes *income range*, *credit card insurance*, *gender*, and *age*.
4. Evaluate each element by applying the corresponding model to test data. A possible evaluation function is test set model accuracy.
5. If the termination condition has been met, choose one element from the population to build a final supervised model from the training data.
6. If the termination condition is not satisfied, apply genetic operators to modify one or more population elements and repeat steps 3–5.

Although this technique is guaranteed to converge, the convergence is not necessarily optimal. Because of this, several executions of the algorithm may be necessary to achieve a desired result.

6.5.3.4 Creating Attributes

Attributes of little predictive power can sometimes be combined with other attributes to form new attributes with a high degree of predictive capability. As an example, consider a database consisting of data about stocks. Conceivable attributes include current stock price, 12-month price range, growth rate, earnings, market capitalization, company sector, and the like. The attributes price and earnings are of some predictive value in determining a future target price. However, the ratio of price to earnings (P/E ratio) is known to be more useful. A second created attribute likely to effectively predict a future stock price is the stock P/E ratio divided by the company growth rate. Here are a few transformations commonly applied to create new attributes:

- Create a new attribute where each value represents a ratio of the value of one attribute divided by the value of a second attribute.

- Create a new attribute whose values are differences between the values of two existing attributes.
- Create a new attribute with values computed as the percent increase or percent decrease of two current attributes. Given two values v_1 and v_2 with $v_1 < v_2$, the percent increase of v_2 with respect to v_1 is computed as

$$\text{Percent Increase}(v_2, v_1) = \frac{v_2 - v_1}{v_1}$$

If $v_1 > v_2$, we subtract v_2 from v_1 and divide by v_1 , giving a percent decrease of v_2 with respect to v_1 .

New attributes representing differences and percent increases or decreases are particularly useful with time-series analysis. *Time-series analysis* models changes in behavior over a time interval. For this reason, attributes created by computing differences between one time interval and the next time interval are important. In Chapter 13, you will see how to build models for solving time-series problems.

6.5.3.5 Instance Selection

Data used for the training phase of supervised learning are often randomly chosen from a pool of instances. The only criterion affecting the random process is that instances are selected so as to guarantee representation from each concept class to be learned. As you saw in Chapter 2, decision tree algorithms go a step further by initially choosing a random subset of the selected training instances to build a first classifier. The classifier is then tested on the remaining training instances. Those instances incorrectly classified by the decision tree are added to the subset of training data. The process is repeated until the training set is exhausted or a classifier that correctly classifies all training data is constructed.

An exception to this rule applies to *instance-based classifiers*. Instance-based classifiers do not create generalized classification models. Instead, they save a subset of representative instances from each class. A new instance is classified by comparing its attribute values with the values of saved instances. Unknown instance i is placed in the class whose representative instances are most similar to i . It is obvious that the instances chosen to represent each class determine the predictive accuracy of the model.

Unsupervised clustering can also benefit from instance selection. One technique is to use an outlier detection method. By eliminating the most atypical domain instances, an unsupervised learner is better able to form well-defined clusters. Once quality clusters have been formed, the outliers can then be presented to the clustering system. The clustering model will either form new clusters with the instances or place the instances in existing clusters.

Finally, it's easy to tout the importance of one data mining tool over another. The truth of the matter is that, given the same training data and input attributes, most data mining algorithms misclassify the same instances. The real key to the success of any data mining project is having a set of instances and attributes that clearly represent the domain in question.

6.6 DATA MINING

The experimental and iterative nature of knowledge discovery is most apparent during steps 5 and 6 of the knowledge discovery process. Here is a typical scenario for building a supervised or unsupervised learner model:

1. Choose training and test data from the pool of available instances.
2. Designate a set of input attributes.
3. If learning is supervised, choose one or more attributes for output.
4. Select values for the learning parameters.
5. Invoke the data mining tool to build a generalized model of the data.

Once data mining is complete, the model is evaluated (step 6). If an acceptable result is not seen, the just-described steps may be repeated several times. Because of this, the total number of possible learner models created from one set of data is infinite. Fortunately, the nature of the experimental process combined with the fact that data mining techniques are able to create acceptable models with less-than-perfect data increases our likelihood for success.

6.7 INTERPRETATION AND EVALUATION

The purpose of interpretation and evaluation is to determine whether a learner model is acceptable and can be applied to problems outside the realm of a test environment. If acceptable results are achieved, it is at this stage where acquired knowledge is translated into terms understandable by users.

Interpretation and evaluation can take many forms, some of which include the following:

- *Statistical analysis.* Such analysis is useful for determining if significant differences exist between the performance of various data mining models created using distinct sets of attributes and instances.
- *Heuristic analysis.* Heuristics are rules of thumb that generally give good-enough solutions to problems. Most data mining tools offer numerically computed heuristics to help us decide the degree to which discovered knowledge is of value. One example is the sum of squared error computation associated with the *K*-means algorithm.
- *Experimental analysis.* Neural network techniques as well as the *K*-means algorithm build slightly different models each time they are invoked with the same set of parameters and the same data. Other methods build distinct models with slight variations in data selection or parameter settings. Because of this, experimenting with various attribute or instance choices as well as alternative parameter settings can give markedly different results.

- *Human analysis.* The human component of result analysis reminds us that we are in control of the experimental process. In the final analysis, we must decide whether the knowledge gained from a data mining process can be successfully applied to new problems.

Chapter 2 showed you how to evaluate supervised learner models by incorporating a training/test set scenario. We also showed how lift can help determine the value of supervised models designed for marketing applications. In Chapter 7, we offer several formal evaluation methods. Some of these methods are based on statistical analysis and others that provide a more intuitive approach.

6.8 TAKING ACTION

An ultimate goal of data mining is to apply what has been learned. It is at this point where we see our return on investment. A number of possible actions may result from successful application of the knowledge discovery process:

- Creation of a report or technical article about what has been discovered
- Relocation of retail items for purchase or placement of selected items on sale together
- The mailing of promotional information to a bias sampling of a customer population
- Incorporation of a developed learner model as a front-end system designed to detect fraudulent credit card usage
- Funding of a new scientific study motivated by what has been learned from a knowledge discovery process
- The possibilities are limited only by our ability to gather, preprocess, and effectively analyze data. In the next section, we outline a data mining process model especially designed for the business community

6.9 THE CRISP-DM PROCESS MODEL

CRISP-DM is a product-neutral data mining model developed by a consortium of several companies. The CRISP-DM process model consists of six phases:

1. *Business understanding.* The center of attention is the project objectives and requirements from a business perspective. A data mining problem definition is given, and an initial plan is developed.
2. *Data understanding.* The focus is on data collection and hypothesis formation.
3. *Data preparation.* Tables, records, and attributes are selected. The data are cleansed for the chosen modeling tools.

4. *Modeling.* This phase focuses on selecting and applying one or more data mining modeling tools.
5. *Evaluation.* An analysis of results determines if the developed model achieves the business objectives. A determination about the future use of the model is reached.
6. *Deployment.* If the model achieves the business objectives, a plan of action is developed to apply the model.

Taken together, steps 1 and 2 represent the KDD process of goal identification. Step 3 combines steps 2, 3, and 4 of the KDD process model. Finally, steps 4, 5, and 6 map respectively to steps 5, 6, and 7 of the KDD process model. Although a Crisp-DM website is no longer supported, the model still provides a foundational starting point for many business applications.

6.10 CHAPTER SUMMARY

Knowledge discovery can be modeled as a seven-step process that includes goal identification, target data creation, data preprocessing, data transformation, data mining, result interpretation and evaluation, and knowledge application. A clear statement about what is to be accomplished is a good starting point for successful knowledge discovery. Creating a target data set often involves extracting data from a warehouse, a transactional database, or a distributed environment. Transactional databases do not store redundant data, as they are modeled to quickly update and retrieve information. Because of this, the structure of the data in a transactional database must be modified before data mining can be applied.

Prior to using a data mining tool, the gathered data are preprocessed to remove noise. Missing data are of particular concern because many data mining algorithms are unable to process missing items. In addition to data preprocessing, data transformation techniques can be applied before data mining takes place. Data transformation methods such as data normalization and attribute creation or elimination are often necessary for a best result. An attribute selection technique of particular interest makes use of genetic learning to help us decide on a best choice of attributes.

Once a data mining process has been completed, the results are evaluated. If the results are acceptable, the created model can be applied. If the results are less than optimal, one or more steps of the knowledge discovery process are repeated. Fortunately, an iterative approach involving model creation and model testing will often lead to an acceptable result.

A second knowledge discovery process model that has a proven track record is the CRISP-DM process model. CRISP-DM is a product-neutral model developed specifically for the business community.

6.11 KEY TERMS

- *Attribute filtering.* Attribute filtering methods select attributes based on some measure of quality independent of the algorithm that will be used to build a final model.

- *Data normalization.* A data transformation where numeric values are modified to fall within a specified range.
- *Data preprocessing.* The step of the KDD process that deals with noisy and missing data.
- *Data transformation.* The step of the KDD process that deals with data normalization and conversion as well as the addition and/or elimination of attributes.
- *Decimal scaling.* A data transformation technique for a numeric attribute where each value is divided by the same power of 10.
- *Instance-based classifier.* Any classifier that models data by saving a subset of instances from each class.
- *Logarithmic normalization.* A data transformation method for a numeric attribute where each numeric value is replaced by its base b logarithm.
- *Min-max normalization.* A data transformation method that is used to transform a set of numeric attribute values so they fall within a specified numeric range.
- *Noise.* Random error in data.
- *Outlier.* An instance that by some measure deviates significantly from other instances.
- *Relational database.* A database where data are represented as a collection of tables containing rows and columns. Each column of the table is known as an attribute, and each row of the table stores information about one data record.
- *Time-series analysis.* Any technique that models changes in behavior over a period of time.
- *Tuple.* An individual row of a table in a relational database.
- *Wrapper technique.* An attribution selection method that bases attribute goodness in the context of the learning algorithm used to build the final model.
- *Z-score normalization.* A data normalization technique for a numeric attribute where each numeric value is replaced by its standardized difference from the mean.

EXERCISES

Review Questions

1. Differentiate between the following terms:
 - a. Data cleaning and data transformation
 - b. Internal and external data smoothing
 - c. Decimal scaling and Z-score normalization
 - d. Filter and wrapper attribute selection

2. In Section 6.4, you learned about three basic ways that data mining techniques deal with missing data while learning. Decide which technique is best for the following problems. Explain each choice.
- A model designed to accept or reject credit card applications
 - A model for determining who should receive a promotional flyer in the mail
 - A model designed to determine those individuals likely to develop colon cancer
 - A model to decide whether to drill for oil in a certain region
 - A model for approving or rejecting candidates applying to refinance their home

Data Mining Questions

1. Write a brief summary of the capabilities of RapidMiner's *Remove Duplicates* operator.
2. Compare RapidMiner's operators for handling missing data items by stating a situation when each is a best choice.
3. Write a summary comparing and contrasting RapidMiner's outlier detection operators.
4. Write a two- or three-sentence description of three of Weka's unsupervised instance filters and three of Weka's supervised instance filters.
5. Use RapidMiner's *Decision Tree* operator to build a decision tree for the cardiology patient dataset. Allow the tree to have a maximum of four levels.
 - a. Use the *Detect Outliers (Distances)* operator with *Euclidian distance* as the distance function parameter to detect the top 10 outlier candidates within the cardiology patient data set.
 - b. How many of the detected outlier instances are from each of the two classes?
 - c. Build a process model that retrieves the cardiology data set, and uses the *Detect Outliers (Distances)* followed by the *Filter Examples* operators to remove the top 20 outlier instances from the data. Give the modified data set to the *Decision Tree* operator. As previously, allow the tree a maximum of four levels.
 - d. Compare the two decision trees. Did removing the outlier candidates change the structure of the decision tree?
6. For this exercise, you are to use RapidMiner to create a process model to compare the performance vectors of two process models as follows:
 - a. Retrieve the diabetes data set described in Section 6.4.
 - b. Use the *Multiply* operator to create two copies of the data set.

- c. Apply RapidMiner's *Rule Induction* operator together with 10-fold cross-validation to one copy of the data set. Use the *Apply Model* and *Performance (classification)* operators to obtain a performance vector for your model.
 - d. Use the second copy of the data set to perform the same task as in part (c), but first use the *Detect Outliers (Distances)* operator together with the *Filter Examples* operator to eliminate the top 30 outlier candidates from the data set.
 - e. Compare the performance vectors of the two models. What effect does removing the candidate outliers have on model performance?
7. Repeat Exercise 6 but attempt to achieve a better result by experimenting with at least three alternative values for the parameters of the *Detect Outliers (Distances)* operator. Summarize your findings.

Computational Questions

1. Set up a general formula for a min–max normalization as it would be applied to the attribute *age* for the data in Table 2.3. Transform the data so the new minimum value is 0 and the new maximum value is 1. Apply the formula to determine a transformed value for *age* = 35.
2. Answer the following questions about percent increase and percent decrease.
 - a. The price of a certain stock increases from \$25.00 to \$40.00. Compute the percent increase in the stock price.
 - b. The original price of the stock is \$40.00. The price decreases by 50%. What is the current stock price?
3. You are to apply a base 2 logarithmic normalization to a certain numeric attribute whose current range of values falls between 2300 and 10,000. What will be the new range of values for the attribute once the normalization has been completed?
4. Apply a base 10 logarithmic normalization to the values for attribute *age* in Table 2.3. Use a table to list the original values as well as the transformed values.
5. Use the *CreditCardPromotion.xls* data file together with the initial element population shown in Table 6.1 to perform the first iteration of the genetic algorithm for attribute selection. Use 10 instances for training and the remaining instances as a test set. Use classification correctness on the test data as your fitness function. List the fitness scores for each of the three elements of the initial population.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Formal Evaluation Techniques

CHAPTER OBJECTIVES

- *Determine confidence intervals for model test set accuracy*
- *Use statistical evaluation to compare the accuracy of two or more supervised learner models*
- *Know how unsupervised clustering is used for supervised evaluation*
- *Use supervised learning to evaluate the results of unsupervised data mining*
- *Understand how attribute analysis can be used to evaluate the results of an unsupervised clustering*
- *Evaluate supervised models having numerical output*
- *Know how to use RapidMiner's T-test, analysis of variance (ANOVA), and grouped ANOVA operators*
- *Know how to create and read a Pareto lift chart*

In previous chapters, we showed how test set error rates, confusion matrices, and lift charts can help us evaluate supervised learner models. You also saw how supervised learning can be used to evaluate the results of an unsupervised clustering. In this chapter, we continue our discussion of performance evaluation by focusing on formal evaluation methods for supervised learning and unsupervised clustering. Most of the methods introduced in this chapter are of a statistical nature. The advantage of this approach is that it permits us to associate a level of confidence with the outcome of our data mining experiments.

We emphasize the practical application of standard statistical and nonstatistical methods rather than the theory behind each technique. Our goal is to provide the necessary tools to enable you to develop a clear understanding of which evaluation techniques are appropriate for your data mining applications. The methods presented here are enough

to meet the needs of most interested readers. However, Appendix B provides additional material for the reader who desires a more complete treatment of statistical evaluation techniques.

In Section 7.1, we highlight the component parts of the data mining process that are responsive to an evaluation. In Section 7.2, we provide an overview of several foundational statistical concepts such as mean and variance scores, standard error computations, data distributions, populations and samples, and hypothesis testing. Section 7.3 offers a method for computing test set confidence intervals for classifier error rates. Section 7.4 shows you how to employ classical hypothesis testing together with test set error rates to compare the classification accuracy of competing models. Section 7.5 offers several methods for evaluating the results of an unsupervised clustering. In Section 7.6, we show you how to evaluate supervised learner models having numerical output. In Section 7.7, we use RapidMiner's *T-test* and *ANOVA* operators to compare the performance vectors of competing models. Section 7.8 demonstrates how two of RapidMiner's operators help with attribute selection. Section 7.9 shows you how to use RapidMiner's *Create Lift Chart* operator. As you read and work through the examples of this chapter, keep in mind that a best evaluation is accomplished by applying a combination of statistical, heuristic, experimental, and human analyses.

7.1 WHAT SHOULD BE EVALUATED?

Figure 7.1 shows the major components used to create and test a supervised learner model. All elements contribute in some way to the performance of a created model. When a model fails to perform as expected, an appropriate strategy is to evaluate the effect every component has on model performance. The individual elements of Figure 7.1 have each been a topic of discussion in one or more of the previous chapters. The following is a list of the components shown in the figure together with additional considerations for evaluation.

1. *Supervised model.* Supervised learner models are usually evaluated on test data. Special attention may be paid to the cost of different types of misclassification. For example, we might be willing to use a loan application model that rejects borderline individuals who would likely pay off a loan provided that the model does not accept strong candidates for loan default. In this chapter, we add to your evaluation toolbox by showing you how to compute test set error rate confidence intervals for supervised models having categorical output.

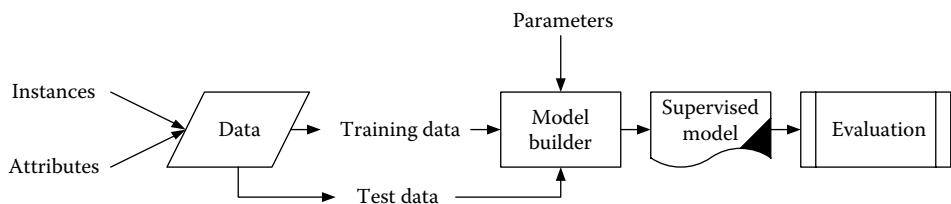


FIGURE 7.1 Components for supervised learning.

2. *Training data.* If a supervised learner model shows a poor test set accuracy, part of the problem may lie with the training data. Models built with training data that do not represent the set of all possible domain instances or contain an abundance of atypical instances are not likely to perform well. A best preventative measure is to randomly select training data, making sure the classes contained in the training data are distributed as they are seen in the general population. The procedure for ensuring an appropriate distribution of data is known as *stratification*.
3. *Attributes.* Attribute evaluation focuses on how well the attributes define the domain instances. In Section 7.8, we examine statistical methods for attribute evaluation.
4. *Model builder.* It has been shown that supervised learner models built with alternative learning techniques tend to show comparable test set error rates. However, there may be situations where one learning technique is preferred over another. For example, neural networks tend to outperform other supervised techniques when the training data contain a wealth of missing or noisy data items. In Section 7.4, we show you how to decide if two supervised learner models built with the same training data show a significant difference in test set performance.
5. *Parameters.* Most data mining models support one or more user-specified learning parameters. Parameter settings can have a marked effect on model performance. The technique used to compare supervised learner models built with different data mining techniques can also be applied to compare models constructed with the same data mining method but alternate settings for one or more learning parameters.
6. *Test set evaluation.* The purpose of test data is to offer a measure of future model performance. Test data should be selected randomly, with stratification applied as appropriate.

Figure 7.1 also applies to unsupervised clustering, with the exception that we are without test data containing instances of known classification. In Section 7.5, we review unsupervised evaluation techniques and explore additional methods for evaluating an unsupervised clustering.

7.2 TOOLS FOR EVALUATION

Statistics are a part of our everyday lives. The results of statistical studies help us make decisions about how to invest our hard-earned money, when to retire, where to place our gambling bets, and even whether to have a certain surgical procedure. The following are several interesting statistical findings:

- Sixty percent of the 116,000,000 households in the United States have credit card debt of \$6000 or more.
- Fifty percent of all adults in the United States are not invested in the stock market.

- Seventy percent of all felons come from homes without a father.
- One in 700,000 deaths is caused by dog bite.
- One in six men will get prostate cancer during their lifetime.
- The average age when a woman becomes a widow is 55.
- Approximately 70% of financial professionals do not extend their education beyond initial licensing and continuing education requirements.
- One in five adults over 65 have been victimized by financial swindle.

Sometimes, the results of statistical studies must be interpreted with a degree of caution. For example, the average age when a woman becomes a widow may be 55; however, the *median* age when a woman is widowed is likely to be higher as well as more informative.

Findings such as those just listed are often gathered through a process of random sampling. The first statement fits this category. It is simply too difficult to poll each and every American family to determine the amount of household credit card debt. Therefore, experts poll a sample of individual households and report findings in terms of the general population along with a margin of error. As you will see, we can apply the techniques developed to conduct statistical studies to our data mining problems. The advantage of a statistical approach is that it allows us to associate levels of confidence with the outcome of our data mining experiments.

Before investigating several useful statistics for evaluating and comparing data mining models, we review the fundamental notions of mean, variance, and population distributions.

7.2.1 Single-Valued Summary Statistics

A population of numerical data is uniquely defined by a mean, a standard deviation, and a frequency or probability distribution of values occurring in the data. The *mean*, or average value, denoted by μ , is computed by summing the data and dividing the sum by the number of data items.

Whereas the mean designates an average value, the *variance* (σ^2) represents the amount of dispersion about the mean. To calculate the variance, we first compute the sum of squared differences from the mean. This is accomplished by subtracting each data value from the mean, squaring the difference, and adding the result to an accumulating sum. The variance is obtained by dividing the sum of squared differences by the number of data items. The *standard deviation*, denoted by σ , is simply the square root of the variance.

When computing a mean, variance, or standard deviation score for a sampling of data, symbol designations change. We adopt the following notation for sample mean and variance scores:

$$\text{Sample mean} = \bar{X}$$

$$\text{Sample variance} = V$$

The mean and variance are useful statistics for summarizing data. However, two populations can display very similar mean and variance scores yet show a marked variation between their individual data items. Therefore, to allow for a complete understanding of the data, knowledge about the distribution of data items within the population is necessary. With a small amount of data, the data distribution is easily obtainable. However, with large populations, the distribution is often difficult to determine.

7.2.2 The Normal Distribution

A fundamentally important data distribution that is well understood is the *normal distribution*, also known as the Gaussian curve or the normal probability curve. Several useful statistics have been developed for populations showing a normal distribution.

The normal, or bell-shaped, curve was discovered by accident in 1733 by the French mathematician Abraham de Moivre while solving problems for wealthy gamblers. The discovery came while recording the number of heads and tails during a coin-tossing exercise. For his experiment, he repeatedly tossed a coin 10 times and recorded the average number of heads. He found that the average as well as the most frequent number of heads tossed was five. Six and four heads appeared with the same frequency and were the next most frequently occurring numbers. Next, three and seven heads occurred equally often, followed by two and eight, and so on. Since Moivre's initial discovery, many phenomena such as measures of reading ability, height, weight, intelligence quotients, and job satisfaction ratings, to name a few, have been found to be distributed normally. The general formula defining the normal curve for continuous data is uniquely determined by a population mean and standard deviation and can be found in Appendix B.

A graph of the normal curve is displayed in Figure 7.2. The x -axis shows the arithmetic mean at the center in position 0. The integers on either side of the mean indicate the number of standard deviations from the mean. To illustrate, if data are normally distributed, approximately 34.13% of all values will lie between the mean and one standard deviation above the mean. Likewise, 34.13% of all values are seen between the mean and one

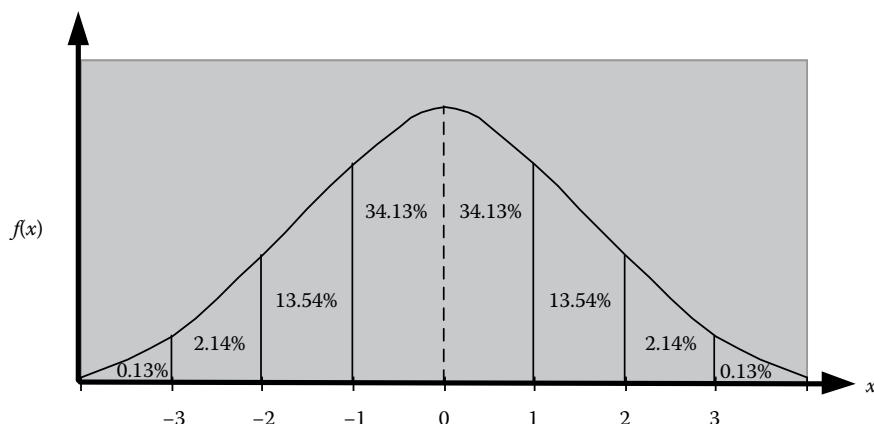


FIGURE 7.2 A normal distribution.

standard deviation below the mean. That is, we can expect approximately 68.26% of all values to lie within one standard deviation on either side of the mean score.

As an example, suppose the scores on a test are known to be normally distributed with a mean of 70 and a standard deviation of 5. Knowing this, we can expect 68.26% of all students to have a test score somewhere between 65 and 75. Likewise, we should see over 95% of the student scores falling somewhere between 60 and 80. Stated another way, we can be 95% confident that all student test scores lie between two standard deviations above and below the mean score of 70.

As most data are not normally distributed, you may wonder as to the relevance of this discussion to data mining. After all, even if one attribute is known to be normally distributed, we must deal with instances containing several numeric values, most of which are not likely to be distributed normally. Our discussion of the normal distribution serves two purposes. First, some data mining models assume numeric attributes to be normally distributed. We discuss one such model in Chapter 11. More importantly, as you will see in the remaining sections of this chapter, we can use the properties of the normal distribution to help us evaluate the performance of our data mining models.

7.2.3 Normal Distributions and Sample Means

Most interesting populations are quite large, making experimental analysis extremely difficult. For this reason, experiments are often performed on subsets of data randomly selected from the total population. Figure 7.3 shows three sample data sets each containing three elements that have been taken from a population of 10 data items.

When sampling from a population, we cannot be sure that the distribution of values in the sample is normal. This is the case even if the population is normally distributed.

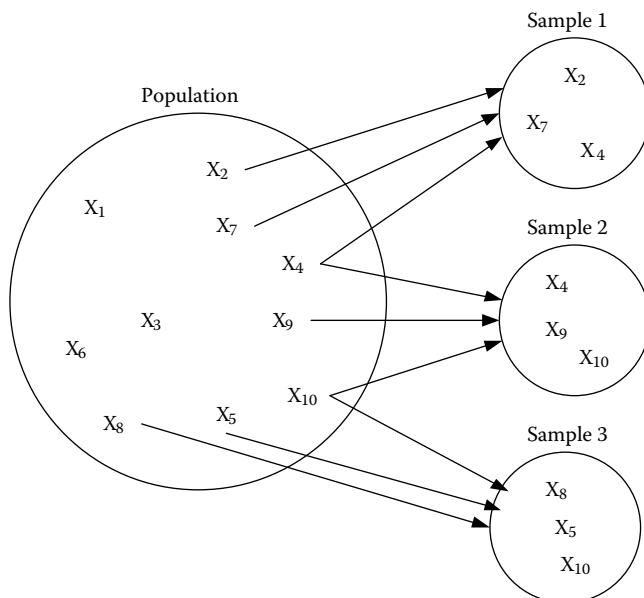


FIGURE 7.3 Random samples from a population of 10 elements.

However, there is a special situation where we are guaranteed a normal distribution. Specifically,

CENTRAL LIMIT THEOREM

For a given population, a distribution of means taken from random sets of independent samples of equal size are distributed normally.

To better understand the importance of the central limit theorem, let's consider the problem of determining the average American household credit card debt. There are approximately 116,000,000 American households. We have neither the time nor the resources to poll each and every household. Therefore, we sample a random subset of 10,000 homes to obtain an average household credit card debt figure. We generalize our findings by reporting the obtained value as the average amount of American household credit card debt. An obvious question is, "How confident can we be that the average computed from the sample data is an accurate estimate of the average household credit card debt for the general population?"

To help answer this question, suppose we repeat this experiment several times, each time recording the average household credit card debt for a new random sample of 10,000 homes. The central limit theorem tells us that the average values we obtain from the repeated process are normally distributed. Stated in a formal manner, we say that any one of the obtained sample means is an unbiased estimate of the mean for the general population. Also, the average of sample means taken over all possible samples of equal size is exactly equal to the population mean!

We now know that the average credit card debt computed from the initial sample mean of 10,000 households is an unbiased estimate of the average for the population. We still do not have a confidence in the computed average value. Although we cannot unequivocally state our computed value as an exact average debt figure for the general population, we can use the sample variance to easily obtain a confidence interval for the computed average value.

First, we estimate the variance of the population. The population variance is estimated by v/n , where v is the sample variance and n is the number of sample instances. Next, we compute the *standard error*. The standard error (SE) is simply the square root of the estimated population variance. The formula for computing standard error is given as

$$SE = \sqrt{v/n} \quad (7.1)$$

As the population of sample means is normally distributed and the standard error is an estimate of the population variance, we can make the following claim:

Any sample mean will vary less than plus or minus two standard errors from the population mean 95% of the time.

For the household credit card debt problem, this means that we can be 95% confident that the actual average household credit card debt lies somewhere between two standard errors above and two standard errors below the computed sample mean. For our example, suppose the average household debt for our sample of 10,000 households is \$6000 with a standard error of 100. Our statement tells us that we can be 95% certain that the actual average American household credit card debt for the general population lies somewhere between \$5800 and \$6200.

We can use this technique as well as the hypothesis testing model introduced in the next section to help us compute confidence intervals for test set error rates, compare the classification error rates of two or more data mining models, and determine those numerical attributes best able to differentiate individual classes or clusters.

7.2.4 A Classical Model for Hypothesis Testing

You will recall that a hypothesis is an educated guess about the outcome of some event. Hypothesis testing is commonplace in the fields of science and medicine as well as in everyday situations dealing with politics and government. Let's take a look at a standard experimental design using a hypothetical example from the field of medicine.

Suppose we wish to determine the effects of a new drug, treatment X, that was developed to treat the symptoms associated with an allergy to house dust. For the experiment, we randomly choose two groups from a select population of individuals who suffer from the allergy. To perform the experiment, we pick one group as the *experimental group*. The second group is designated as the *control group*. Treatment X is applied to the experimental group, and a placebo in the form of a sugar pill is distributed to the control group. We take care to make sure individual patients are not aware of their group membership. The average increase or decrease in the total number of allergic reactions per day is recorded for each patient in both groups. After a period of time, we apply a statistical test to determine if a significant difference in the measured parameter is seen between the two groups of patients.

A usual procedure is to state a hypothesis to be tested about the outcome of the experiment. Typically, the outcome is stated in the form of a *null hypothesis*. The null hypothesis takes a negative point of view in that it asserts that any relationship found as a result of the experiment is due purely to chance. For our example, the null hypothesis declares that the outcome will show no significant difference between the two groups of patients. A plausible null hypothesis for our experiment is as follows:

There is no significant difference in the mean increase or decrease of total allergic reactions per day between patients in the group receiving treatment X and patients in the group receiving the placebo.

Notice that the null hypothesis specifies no significant difference rather than no significant improvement. The reason for this is that we have no a priori guarantee that treatment X will not cause an adverse reaction and worsen the symptoms associated with the allergy.

Once the experiment is performed and the data showing the results of the experiment have been gathered, we test if the outcome shows a significant difference between the two groups of patients. A classical model to test for a significant difference between the mean scores of a measured parameter such as the one for our experiment is one form of the well-known *t-test* given as

$$T = \frac{|\bar{X}_1 - \bar{X}_2|}{\sqrt{(\nu_1/n_1 + \nu_2/n_2)}} \quad (7.2)$$

where

T is the test statistic

\bar{X}_1 and \bar{X}_2 are sample means for the independent samples

ν_1 and ν_2 are variance scores for the respective means

n_1 and n_2 are corresponding sample sizes

The term in the numerator is the absolute difference between the two sample means. To test for a significant difference between the mean scores, the difference is divided by the standard error for the distribution of mean differences. As the two samples are independent, the standard error is simply the square root of the sum of the two variance scores associated with the two sample means. To be 95% confident that the difference between two means is not due to chance, the value for T in the equation must be greater than or equal to 2 (see Figure 7.2). The model is valid because, like the distribution of means taken from sets of independent samples of equal size, the distribution of differences between sample means is also normal.

A 95% level of confidence still leaves room for error. With hypothesis testing, two general types of error are defined. A *type 1 error* is seen when a true null hypothesis is rejected. A *type 2 error* is observed when a null hypothesis that should have been rejected is accepted. These two possibilities are shown in the confusion matrix of Table 7.1. For our experiment, a type 1 error would have us believing that treatment X has a significant impact on the average number of allergic reactions to dust when it does not. The degree of risk we are willing to take in committing a type 1 error is known as *statistical significance*. A type 2 error would state that treatment X does not affect the mean number of allergic reactions when indeed it does.

A requirement of the *t-test* described previously is that each mean is computed from an independent data set. Fortunately, many forms for the *t-test* exist. With data mining, the

TABLE 7.1 A Confusion Matrix for the Null Hypothesis

	Computed Accept	Computed Reject
Accept Null Hypothesis	True accept	Type 1 error
Reject Null Hypothesis	Type 2 error	True reject

usual case is to compare models with the help of a single test set of data. In Section 7.4, you will see how we employ a slight modification of the *t*-test given in Equation 7.2 to situations where we are limited to one test set.

7.3 COMPUTING TEST SET CONFIDENCE INTERVALS

Training and test data may be supplemented by *validation data*. One purpose of validation data is to help us choose one of several models built from the same training set. Once trained, each model is presented with the validation data, whereby the model showing a best classification correctness is chosen as the final model. Validation data can also be used to optimize the parameter settings of a supervised model so as to maximize classification correctness.

Once a supervised learner model has been validated, the model is evaluated by applying it to the test data. The most general measure of model performance is *classifier error rate*. Specifically,

$$\text{classifier error rate } (E) = \frac{\text{number of test set errors}}{\text{number of test set instances}} \quad (7.3)$$

The purpose of a test set error rate computation is to give an indication as to the likely future performance of the model. How confident can we be that this error rate is a valid measure of actual model performance? To answer this question, we can use the standard error statistic to calculate an error rate confidence interval for model performance. To apply the standard error measure, we treat classifier error rate as a sample mean. Although the error rate is actually a proportion, if the number of test set instances is sufficiently large (say $n > 100$), the error rate can represent a mean value.

To determine a confidence interval for a computed error rate, we first calculate the standard error associated with the classifier error rate. The standard error together with the error rate is then used to compute the confidence interval. The procedure is as follows:

-
1. Consider a test set sample S of size n and error rate E .
 2. Compute the sample variance as

$$\text{variance}(E) = E(1 - E)$$

3. Compute the standard error (SE) as the square root of $\text{variance}(E)$ divided by n .
 4. Calculate an upper bound for the 95% confidence interval as $E + 2(SE)$.
 5. Calculate a lower bound for the 95% confidence interval as $E - 2(SE)$.
-

Let's look at an example to better understand how the confidence interval is computed. Suppose a classifier shows a 10% error rate when applied to a random sample of 100 test set instances. We set $E = 0.10$ and compute the sample variance as

$$\text{variance } (0.10) = 0.10 (1 - 0.10) = 0.09$$

With a test set of 100 instances, the standard error computation is

$$SE = \sqrt{(0.09/100)} = 0.03$$

We can be 95% confident that the actual test set error rate lies somewhere between two standard errors below 0.10 and two standard errors above 0.10. This tells us that the actual test set error rate falls between 0.04 and 0.16, which gives a test set accuracy between 84% and 96%.

If we increase the number of test set instances, we are able to decrease the size of the confidence range. Suppose we increase the test set size to 1000 instances. The standard error becomes

$$SE = \sqrt{(0.09/1000)} \approx .0095$$

Making the same computations as in the previous example, the test set accuracy range is now between 88% and 92%. As you can see, the size of the test data set has a marked effect on the range of the confidence interval. This is to be expected, because as the size of the test data set becomes infinitely large, the standard error measure approaches 0. Three general comments about this technique are as follows:

1. The confidence interval is valid only if the test data have been randomly chosen from the pool of all possible test set instances.
2. Test, training, and validation data must represent disjoint sets.
3. If possible, the instances in each class should be distributed in the training, validation, and test data as they are seen in the entire data set.

Recall that Chapter 2 introduced cross-validation and bootstrapping as two techniques we can use when ample test data are not available. Chapter 2 also showed that test set error rate is but one of several considerations when determining the value of a supervised learner model. To emphasize this point, let's assume that an average of 0.5% of all credit card purchases are fraudulent. A model designed to detect credit card fraud that always states a credit card purchase is valid will show a 99.5% accuracy. However, such a model is worthless as it is unable to perform the task for which it was designed. In contrast, a model that correctly detects all cases of credit card fraud at the expense of showing a high rate of false-positive classifications is of much more value.

One way to deal with this issue is to assign weights to incorrect classifications. With the credit card example, we could assign a large weight to incorrect classifications that allow a fraudulent card to go undetected and a smaller weight to the error of incorrectly identifying a credit card as fraudulent. In this way, a model will have its classification error rate increase if it shows a bias toward allowing fraudulent card usage to go undetected. In the next section, we show you how test set classification error rate can be employed to compare the performance of two supervised learner models.

7.4 COMPARING SUPERVISED LEARNER MODELS

We can compare two supervised learner models constructed with the same training data by applying the classical hypothesis testing paradigm. Our measure of model performance is once again test set error rate. Let's state the problem in the form of a null hypothesis. Specifically,

There is no significant difference in the test set error rate of two supervised learner models, M_1 and M_2 , built with the same training data.

Three possible test set scenarios are as follows:

1. The accuracy of the models is compared using two independent test sets randomly selected from a pool of sample data.
2. The same test set data are employed to compare the models. The comparison is based on a pairwise, instance-by-instance computation.
3. The same test data are used to compare the overall classification correctness of the models.

From a statistical perspective, the most straightforward approach is the first one, as we can directly apply the t statistic described in Section 7.2. This approach is feasible only if an ample supply of test set data is available. With large data sets, the prospect of extracting independent test set data is real. However, with smaller-sized data, a single test set may be the only possibility.

When the same test set is applied to the data, one option is to perform an instance-by-instance pairwise matching of the test set results. This approach is described in Appendix B. Here we describe a simpler technique that compares the overall classification correctness of two models. The method can be applied to the case of two independent test sets (scenario 1) or a single test set (scenario 3). The most general form of the statistic for comparing the performance of two classifier models M_1 and M_2 is

$$T = \frac{|E_1 - E_2|}{\sqrt{q(1-q)(1/n_1 + 1/n_2)}} \quad (7.4)$$

where

E_1 = The error rate for model M_1

E_2 = The error rate for model M_2

$q = (E_1 + E_2)/2$

n_1 = the number of instances in test set A

n_2 = the number of instances in test set B

Notice that $q(1 - q)$ is a variance score computed using the average of the two error rates. With a single test set of size n , the formula simplifies to

$$T = \frac{|E_1 - E_2|}{\sqrt{q(1-q)(2/n)}} \quad (7.5)$$

With either Equation 7.4 or 7.5, if the value of $T \geq 2$, we can be 95% confident that the difference in the test set performance between M_1 and M_2 is significant.

7.4.1 Comparing the Performance of Two Models

Let's look at an example. Suppose we wish to compare the test set performance of learner models M_1 and M_2 . We test M_1 on test set A and M_2 on test set B. Each test set contains 100 instances. M_1 achieves an 80% classification accuracy with set A, and M_2 obtains a 70% accuracy with test set B. We wish to know if model M_1 has performed significantly better than model M_2 . The computations are as follows:

- For model M_1 : $E_1 = 0.20$
- For model M_2 : $E_2 = 0.30$
- q is computed as

$$(0.20 + 0.30)/2 = 0.25$$

- The combined variance $q(1 - q)$ is

$$0.25(1.0 - 0.25) = 0.1875$$

- The computation for P is

$$T = \frac{|0.20 - 0.30|}{\sqrt{0.1875(1/100 + 1/100)}}$$

$$T \approx 1.633$$

As $T < 2$, the difference in model performance is not considered to be significant. We can increase our confidence in the result by switching the two test sets and repeating the experiment. This is especially important if a significant difference is seen with the initial test set selection. The average of the two values for T is then used for the significance test.

7.4.2 Comparing the Performance of Two or More Models

The various forms of the *t*-test are limited to comparing pairs of competing models. Using multiple *t*-tests is one way around this problem. For example, six *t*-tests can be used to compare four competing models. Unfortunately, in the world of statistics, it is a well-known fact that performing multiple *t*-tests increases the chance of a type I error. Because of this, a one-way ANOVA is often used in situations when two or more model comparisons are needed.

A one-way ANOVA uses a single factor to compare the effects of one treatment/independent variable on a continuous dependent variable. For example, suppose we wish to test for a significant difference in the number of hours of TV watched by individuals in three age groups:

- $Age \leq 20$
- $Age > 20$ and $age \leq 40$
- $Age > 40$

For this example, the dependent variable is number of hours of TV watched per week, and the independent variable is *age group*.

The ANOVA uses the *F* statistic to test for significant differences between the groups defined by the independent variable. The *F* statistic is computed as the ratio of between-group variance divided by within-group variance. For our example, the computed *F* ratio represents the variance in the number of hours of TV watched between the three age groups relative to the variance in the number of hours watched within each of the three groups.

The *F* ratio increases as between-group variance increases when compared to within-group variance. The larger the *F* ratio, the greater the likelihood of a significant difference between the means of the tested groups. A table of critical values determines if an *F* ratio is statistically significant.

When the ANOVA is applied to test for significant differences in the performance of two or several data mining models, the model becomes the independent variable. Each tested model is one instance of the independent variable. The dependent variable is model error rate.

Although the ANOVA is the stronger test, it is important to note that when three or more models are tested, the *F* statistic tells whether any of the models differ significantly in performance but does not indicate exactly where these differences lie. In Section 7.8, we show you how RapidMiner's implementations of these statistics work together to compare the performance vectors of competing supervised learner models.

Finally, the *t*-test and the ANOVA are known as parametric statistical tests as they make certain assumptions about the parameters (mean and variance) they use to describe the characteristics of the data. Two such assumptions are that the choice of one sample does not affect the chances of any other sample being included and that the data are taken from normally distributed populations having equal variances. However, the tests are often used even when these assumptions cannot be completely verified.

7.5 UNSUPERVISED EVALUATION TECHNIQUES

Supervised learning and unsupervised clustering complement one another in that each approach can be applied to evaluate the opposite strategy. In this section, we review both possibilities and offer additional evaluation methods for unsupervised clustering.

7.5.1 Unsupervised Clustering for Supervised Evaluation

When unsupervised clustering is adopted for supervised evaluation, the data instances selected for supervised training are presented to an unsupervised clustering technique with the output attribute flagged as display-only. If the instances cluster into the predefined classes contained in the training data, a supervised learner model built with the training data is likely to perform well. If the instances do not cluster into their known classes, a supervised model constructed with the training data is apt to perform poorly.

Although this approach is straightforward, depending on the clustering technique, it may be several iterations before an evaluation can be made. For example, suppose we decide to utilize the K -means algorithm for the evaluation. The clusters formed by an application of the K -means algorithm are highly affected by the initial selection of cluster means. For a single iteration, the algorithm is likely to experience a less-than-optimal convergence. With a nonoptimal convergence, a single cluster could contain a mixture of instances from several classes. This would in turn give us a false impression about the efficacy of the domain for supervised learning.

As the unsupervised evaluation is based solely on the training data, a quality clustering is no guarantee of acceptable performance on test set instances. For this reason, the technique complements other evaluation methods and is most valuable for identifying a rationale for supervised model failure. As a general rule, we see an inverse relationship between the value of this approach and the total number of predefined classes contained in the training data.

7.5.2 Supervised Evaluation for Unsupervised Clustering

In Chapter 2, we showed you how supervised learning can help explain and evaluate the results of an unsupervised clustering. This method is particularly appealing because it is independent of the technique used for the unsupervised clustering. Let's review the procedure:

1. Designate each formed cluster as a class.
2. Build a supervised learner model by choosing a random sampling of instances from each class.
3. Test the supervised learner model with the remaining instances.

As many unsupervised techniques lack an explanation facility, the supervised classification helps explain and analyze the formed clusters. A slight variation of this technique is to build the supervised model with the first n prototype instances from each cluster. Following this procedure allows us to observe model performance from the perspective of the instances that best define each cluster.

7.5.3 Additional Methods for Evaluating an Unsupervised Clustering

Unsupervised evaluation is often a two-step process. The first evaluation is internal and involves repeating the clustering process until we are satisfied that any user-defined parameters hold optimal values.

The second evaluation is external and independent of the model creating the clusters. The following is a short list of several additional external evaluation methods.

1. *Use a supervised learning approach as described in the previous section, but designate all instances as training data.* This variation is appropriate if a complete explanation about between-cluster differences is of primary importance. A supervised learner capable of generating production rules is particularly appealing for purposes of explanation.
2. *Apply an alternative technique's measure of cluster quality.*
3. *Create your own measure of cluster quality.* Agglomerative clustering is an unsupervised technique that merges instances at each step of the clustering process until a terminating condition is satisfied. In Chapter 12, we describe agglomerative clustering along with two heuristic techniques to help determine when the merge process should terminate. These heuristics can be modified to make evaluative comparisons between different clusterings created by the same or alternative clustering models.
4. *Perform a between-cluster attribute-value comparison.* If between-cluster attribute values differ significantly, we conclude that between-cluster instance differences are also significant.

As you can well imagine, this list is by no means exhaustive! We encourage you to use this list as a starting point for developing your own toolkit for unsupervised evaluation.

7.6 EVALUATING SUPERVISED MODELS WITH NUMERIC OUTPUT

Thus far, our discussion has been limited to models giving categorical output. However, we can use the techniques just described to help us evaluate models whose output is numerical. To apply the aforementioned techniques, we need a performance measure for models giving numerical output that is equivalent to classifier error rate. Several such measures have been defined, most of which are based on differences between actual and computed output. In each case, a smallest possible difference represents a best scenario. We will limit our discussion to three common methods for evaluating numeric output: the *mean squared error*, the *root mean squared error*, and the *mean absolute error*.

The *mean squared error* (mse) is the average squared difference between actual and computed output, as shown in Equation 7.7.

$$\text{mse} = \frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \dots + (a_i - c_i)^2 + \dots + (a_n - c_n)^2}{n} \quad (7.7)$$

where for the i th instance,

a_i = actual output value

c_i = computed output value

The *root mean squared error* (rms) is simply the square root of the mean squared error. By applying the square root, we reduce the dimensionality of the mse to that of the actual error computation. The backpropagation neural network model described in Chapter 9 uses rms as a measure of network convergence. Notice that for values less than 1.0, rms > mse.

The *mean absolute error* (mae) finds the average absolute difference between actual and computed output values. An advantage of mae is that it is less affected by large deviations between actual and computed output values. In addition, mae maintains the dimensionality of the error value. Equation 7.8 shows the formula.

$$\text{mae} = \frac{|a_1 - c_1| + |a_2 - c_2| + \dots + |a_n - c_n|}{n} \quad (7.8)$$

In Chapter 2, we applied a backpropagation neural network to the credit card promotion data where *life insurance promotion* was designated as the output attribute. Table 7.2 repeats the actual and computed output values for the experiment and displays the absolute and squared error output scores for each data instance. Although the values in the table are the result of applying training rather than test set data to the trained network, the

TABLE 7.2 Absolute and Squared Error (Output Attribute = Life Insurance Promotion)

Instance Number	Actual Output	Computed Output	Absolute Error	Squared Error
1	0.0	0.024	0.024	0.0005
2	1.0	0.998	0.002	0.0000
3	0.0	0.023	0.023	0.0005
4	1.0	0.986	0.014	0.0002
5	1.0	0.999	0.001	0.0000
6	0.0	0.050	0.050	0.0025
7	1.0	0.999	0.001	0.0000
8	0.0	0.262	0.262	0.0686
9	0.0	0.060	0.060	0.0036
10	1.0	0.997	0.003	0.0000
11	1.0	0.999	0.001	0.0000
12	1.0	0.776	0.224	0.0502
13	1.0	0.999	0.001	0.0000
14	0.0	0.023	0.023	0.0005
15	1.0	0.999	0.001	0.0000

example is useful for illustrative purposes. The computations for mse, mae, and rms using the table data are as follows:

$$\text{mse} = \frac{(0.0 - 0.024)^2 + (1.0 - 0.998)^2 + \dots + (1.0 - 0.999)^2}{15}$$

$$\approx \frac{0.1838}{15} \approx 0.0123$$

$$\text{mae} = \frac{|0.0 - 0.024| + |1.0 - 0.998| + \dots + |1.0 - 0.999|}{15}$$

$$\approx \frac{0.6899}{15} \approx 0.0459$$

To compute rms, we simply take the square root of 0.0123. The computation gives an rms of 0.1107.

Finally, if your interests in numeric model testing lie beyond our discussion here, formulas and examples for computing test set confidence intervals as well as comparing supervised learner models having numeric output can be found in Appendix B.

7.7 COMPARING MODELS WITH RAPIDMINER

RapidMiner offers two operators for comparing the performance vectors of competing supervised models. The *T-Test* operator is used to determine if there is a statistically significant difference between the performance vectors of two competing models. The ANOVA operator is used to test for significant differences between two or more competing models. Let's take a look at an example illustrating the use of both operators.

RAPIDMINER TUTORIAL

The T-Test and ANOVA Operators

Our example compares three decision tree models using the mixed form of the cardiology patient data set. Each decision tree is given a unique value for the maximum depth parameter. The differing values will affect the level of generalization and the classification accuracy of each tree. The maximum depth for the first decision tree is set at 1 giving a decision tree with a single (root) node. The maximum depth for the second tree is set at 5, and the maximum depth for the third tree is 20. Here's how to create the process model:

- Create and save a new process.
- Locate and drag the *Subprocess*, *T-Test*, and *ANOVA* operators into the main process window and make the connections shown in Figure 7.4.

As there are three competing models, the *T-Test* to *res* connection will display the results of the three *t*-tests in table form. The *ANOVA sig* to *res* connection outputs a table with the

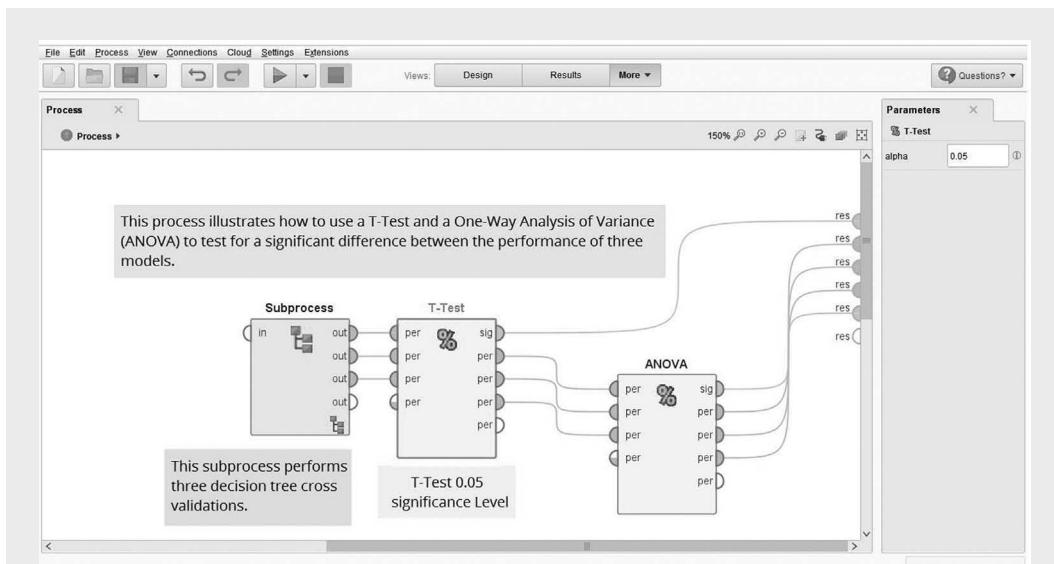


FIGURE 7.4 A process model for comparing three competing models.

results of the ANOVA test. The *per-to-res* connections output the performance vectors of the three models.

- Be sure the *alpha* parameter for the *T-Test* operator is set at its default value of 0.05.
- Double-click on the *Subprocess* operator and use three 10-fold cross-validations to create the subprocess shown in Figure 7.5. The output of the *Subprocess* operator is the performance vectors for the competing models.
- Use Figure 7.6 as the template for constructing the three decision tree cross-validations. Set the maximum tree depths as shown in Figure 7.6.
- Run your process.

The *T-Test* operator in Figure 7.4 takes the output of the performance vectors as input and tests for a significant difference between the performance vectors of each pair of competing models. Figure 7.7 displays the output of the *T-Test* operator. The output shows the average accuracy of the models as the first value in each row as well as the first value in columns B through D. The average accuracy for each respective model is 54.5% (maximum depth = 1), 71.6% (maximum depth = 5), and 76.5% (maximum depth = 20).

To determine if a significant difference is seen in model performance, we look to the values in the matrix. Specifically, comparing the model whose average accuracy is 54.5% with the model showing an average accuracy of 71.6% (arrow 1–5), we see a value of 0.00. This value represents the comparative level of significance. As 0.00 is less than the significance level of 0.05, the two models differ significantly in their performance. This is also the case in the comparison between the trees with maximum depth 1 and maximum depth 20. Lastly, comparing the two models each having an accuracy greater than 71% gives a significance value of 0.145 (arrow 5–20). Since 0.145 > 0.05, the performance vectors corresponding to these models are not significantly different.

The results of the ANOVA test are given in Figure 7.8. The figure offers several statistics of potential interest, but the term of immediate importance is *Prob*, with value 0.000. *Prob* is the

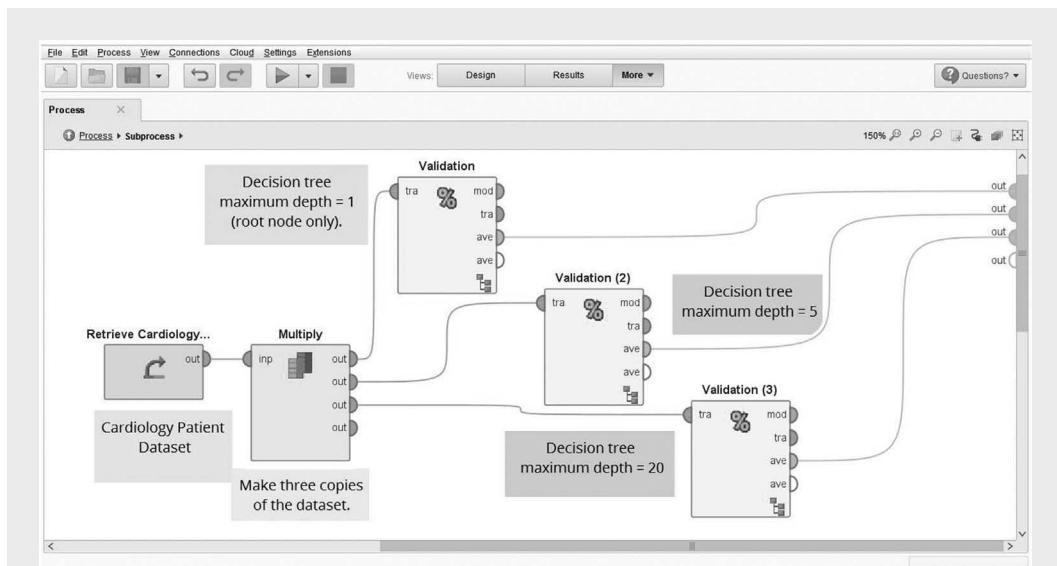


FIGURE 7.5 Subprocess for comparing three competing models.

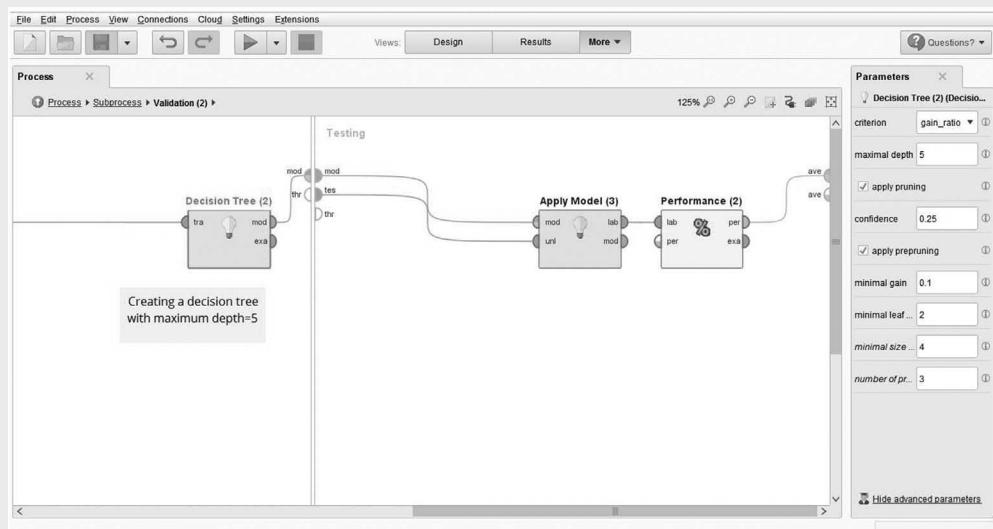


FIGURE 7.6 Cross-validation test for a decision tree with maximum depth = 5.

significance value associated with the ANOVA test. This value tells us that we can be close to 100% (0.000) certain that the difference seen between the models is not due to chance. However, the output does not tell us exactly where the difference lies.

In summary, when comparing the performance of two models, either the *t*-test or ANOVA suffices. When more than two models are part of the experiment, both tests are useful. The multiple *t*-tests give us the details about where the differences lie, and the ANOVA helps confirm that any significant findings are valid and not due to one or several type 1 errors.

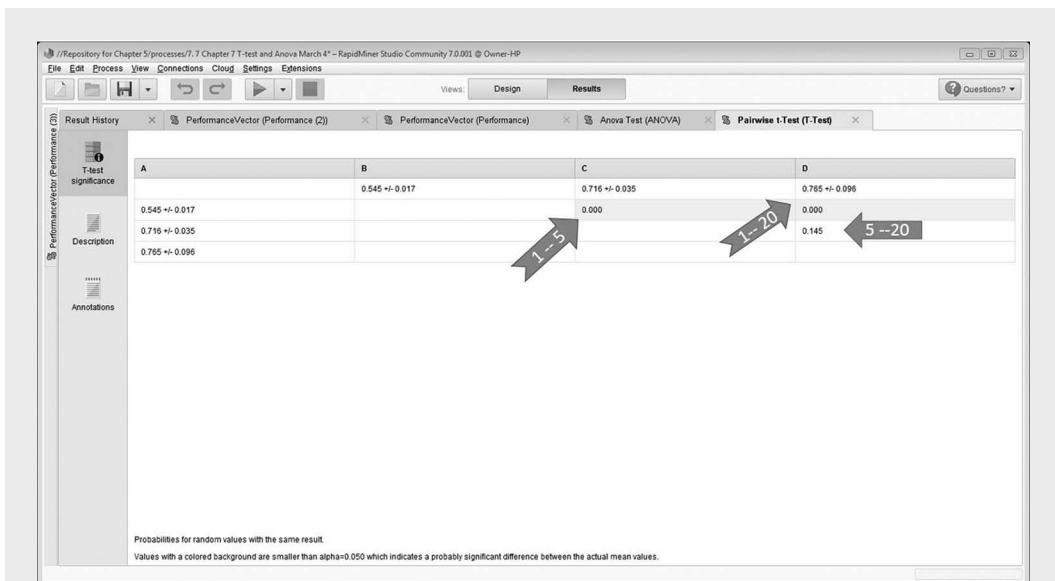
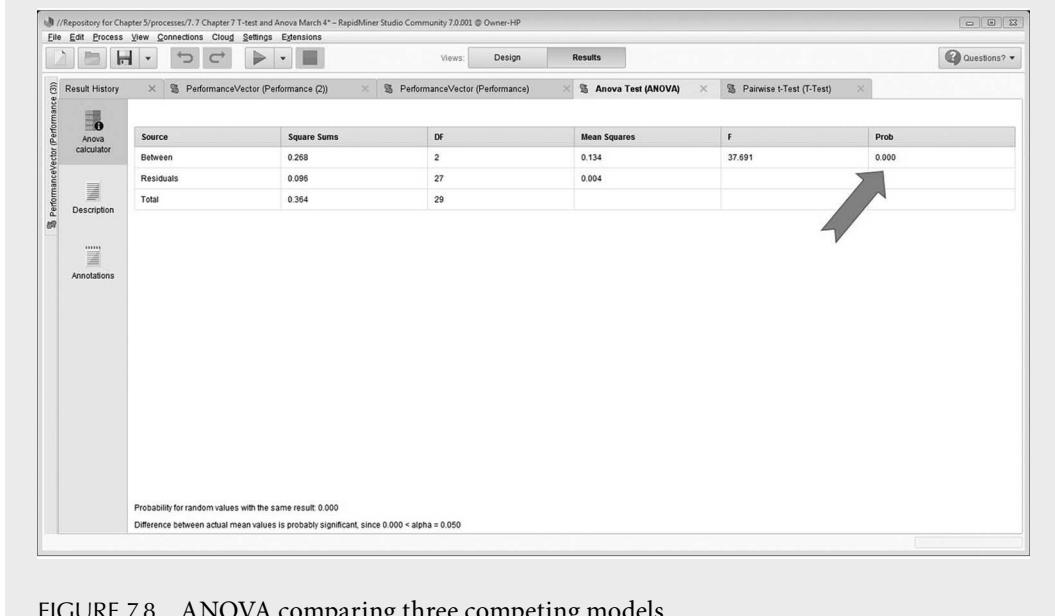
FIGURE 7.7 A matrix of *t*-test scores.

FIGURE 7.8 ANOVA comparing three competing models.

7.8 ATTRIBUTE EVALUATION FOR MIXED DATA TYPES

In previous chapters, we highlighted the importance of attribute selection and offered several attribute selection techniques. We emphasized how eliminating correlated and redundant attributes can improve model performance. In this section, we broaden our

discussion about attribute evaluation by showing how two of RapidMiner's statistical operators help select useful numerical attributes as well as uncover numerical attributes of little predictive value.

Figure 7.9 offers a process model incorporating the cardiology patient data set to demonstrate the *Grouped ANOVA* and *ANOVA Matrix* operators. The parameter settings shown in the figure—*anova attribute* and *group by attribute*—are for the *Grouped ANOVA* operator.

Given a categorical grouping attribute and a numeric attribute, the *Grouped ANOVA* operator tests for a statistically significant difference between the group means of the chosen numeric attribute. For our example, we chose *maximum heart rate* as the numeric attribute and *class* as the grouping attribute. Figure 7.10 shows the result of this comparison. The value of 0.000 for *Prob* tells us that we can be more than 99% certain that a significant difference exists when the healthy and sick class means for *maximum heart rate* are compared. Replacing *maximum heart rate* by *cholesterol* (not shown) gives a value for *Prob* of 0.137. As this value is not significant, this supports removing *cholesterol* from the data set prior to building a supervised learner model.

The *ANOVA matrix* operator uses the *Grouped ANOVA* operator to make this comparison for all possible combinations of categorical (grouping attribute) and numeric attribute pairs. The output when this operator is applied to the cardiology patient data is given in Figure 7.11. Table values less than or equal to 0.05 are significant. Darkened cells highlight nonsignificant relationships.

Close examination of the figure tells us that all numeric input attributes except *cholesterol* show a significant difference when the grouping attribute is *class* (last column in

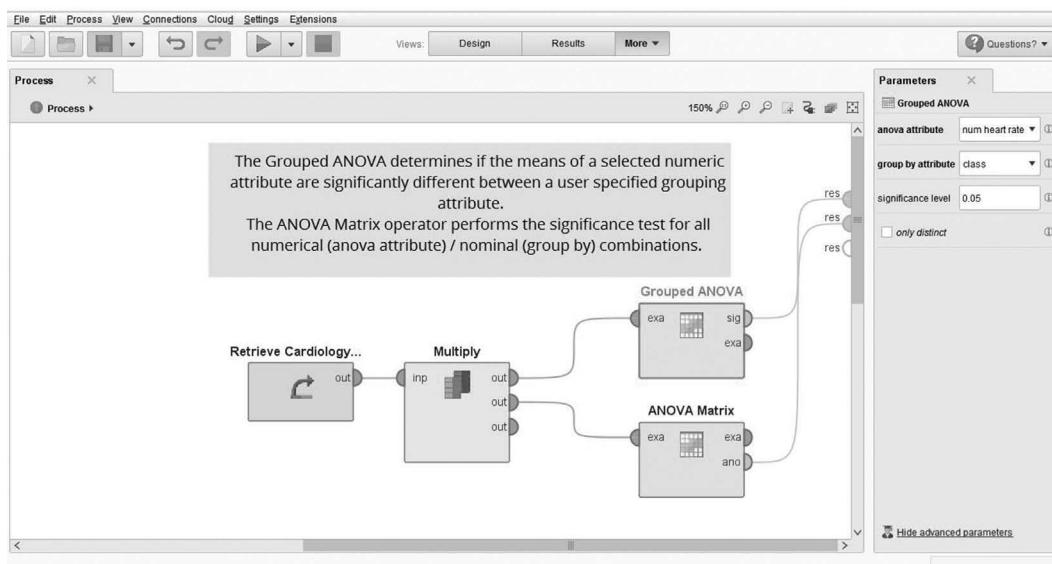


FIGURE 7.9 ANOVA operators for comparing nominal and numeric attributes.

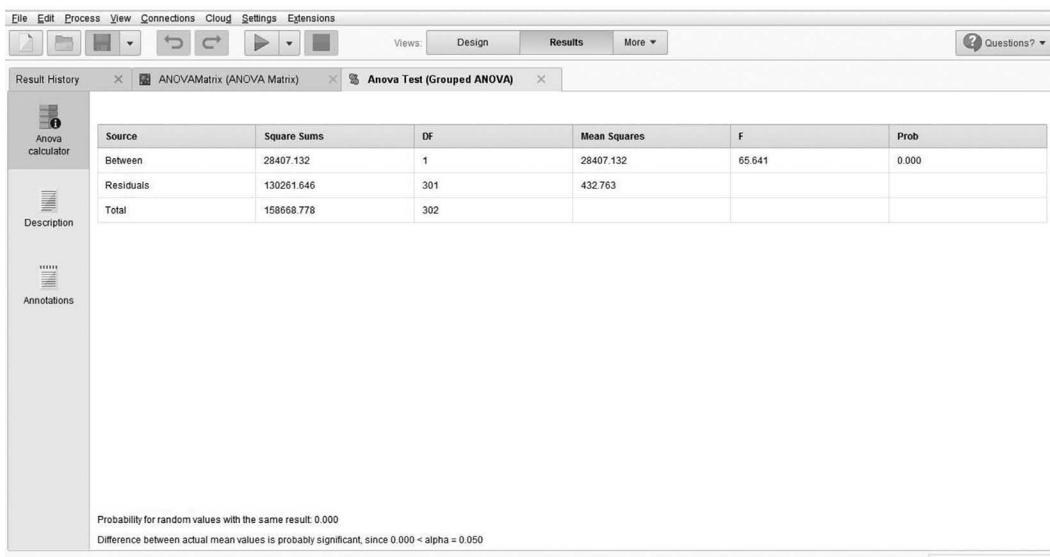


FIGURE 7.10 The grouped ANOVA operator comparing class and maximum heart rate.

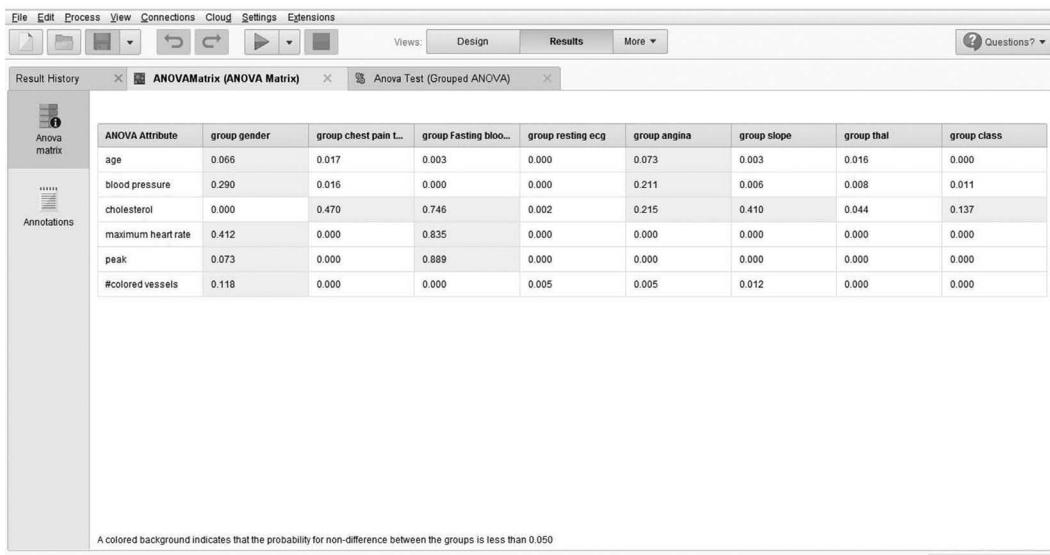


FIGURE 7.11 The ANOVA matrix operator for the cardiology patient data set.

Figure 7.11). All attributes other than *cholesterol* should be considered for initial inclusion when building a supervised learner model. Beyond this, the next step is to look for correlated values between the various numeric attributes and eliminate redundancies as appropriate. The ideal case is when uncorrelated numeric input attributes show significant between-class mean differences.

7.9 PARETO LIFT CHARTS

RapidMiner's *Create Lift Chart* operator provides us with the opportunity to create lift charts based on Pareto plots of discretized confidence values. The original confidence scores are obtained by applying a data mining model to a specific data set. A Pareto lift chart is a three- rather than two-dimensional graph, which can make it a challenge to interpret. An example best illustrates the operator.

Figures 7.12 and 7.13 give the process model for our example. The *PreProcess* operator reads the familiar customer churn data set, eliminates the 97 instances of unknown outcome, and splits the remaining data for training (two-thirds) and testing (one-third). A decision tree is constructed, and the model is tested.

Figure 7.12 shows the input to the *Create Lift Chart* operator as the labeled examples given by the output of the *Apply Model* operator (Figure 7.14) and the created decision tree model. The *Apply Model* operator provides the classification of each instance together with needed confidence scores.

The *Create Lift Chart* operator parameter settings are given in Figure 7.12 as follows:

- target class = churn
- binning type = simple
- number of bins = 10
- automatic number of digits = checked
- show bar labels = checked

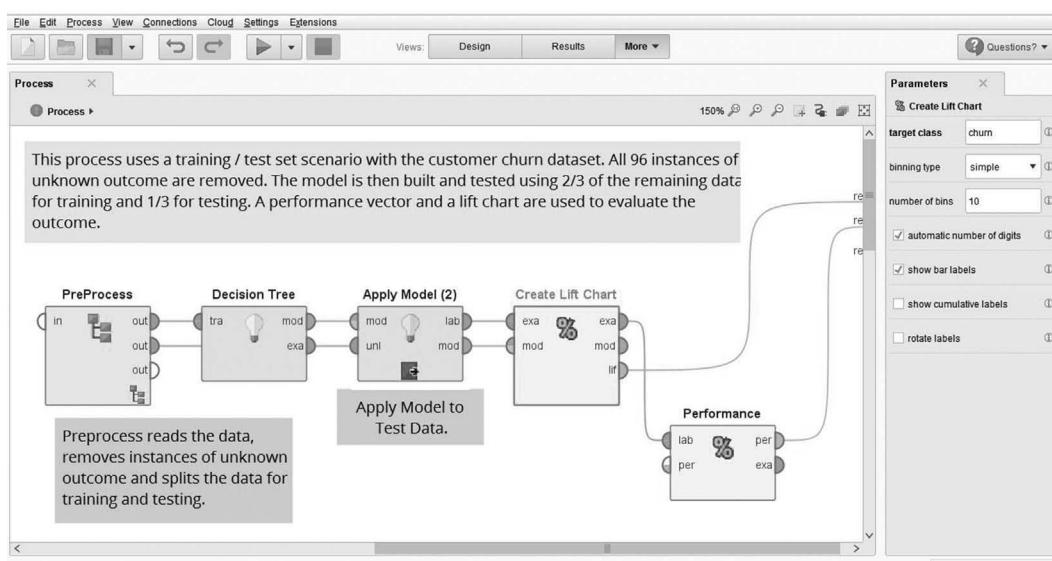


FIGURE 7.12 A process model for creating a lift chart.

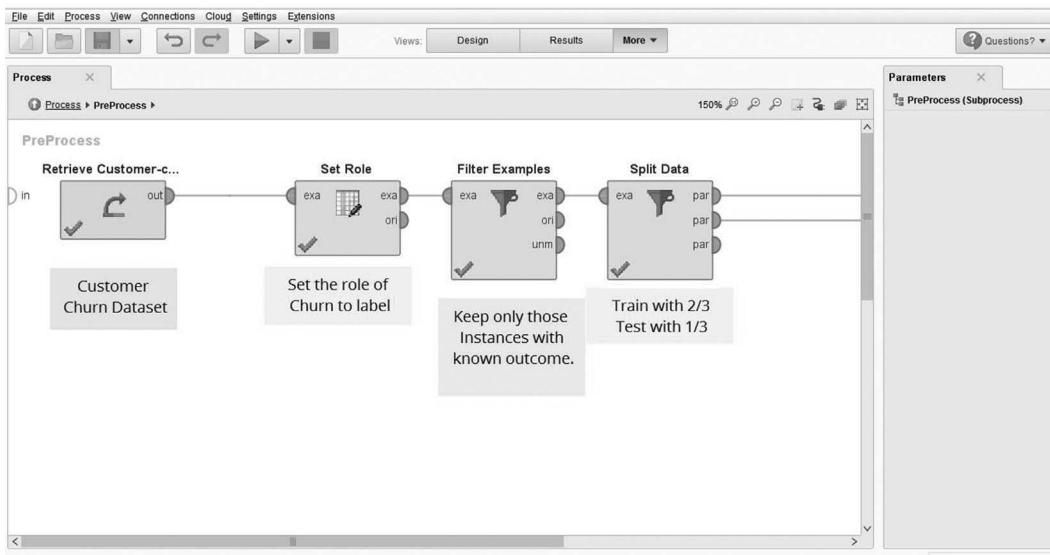


FIGURE 7.13 Preprocessing the customer churn data set.

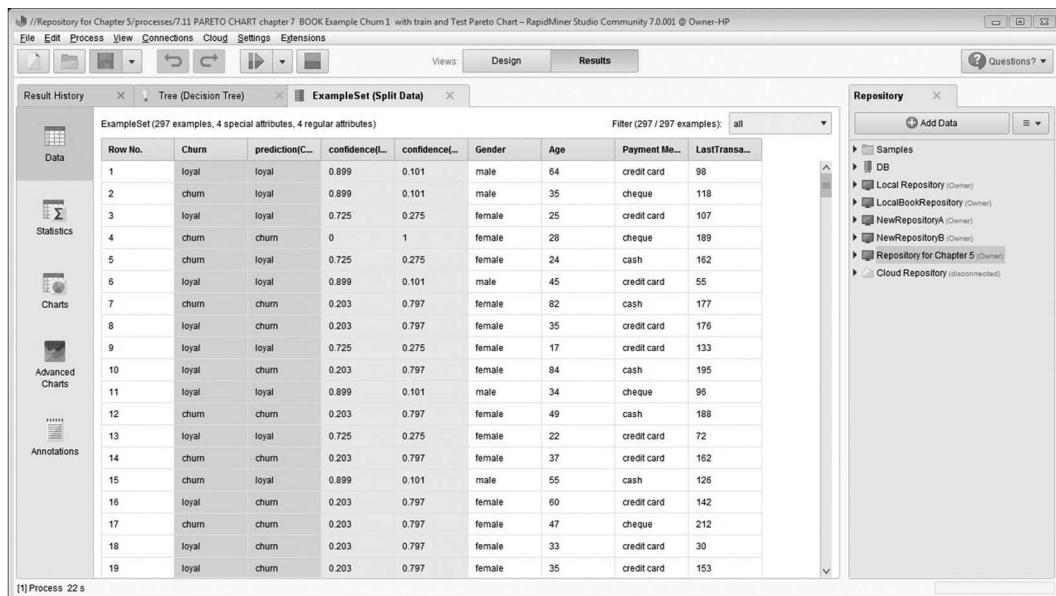


FIGURE 7.14 Output of the Apply Model operator for the customer churn data set.

Target class = churn tells the operator to create the graph relative to the value *Churn = churn*. *Binning type* together with *number of bins* specifies the maximal number of bins to be used for the confidence scores. *Automatic number of digits* and *show bar labels* default to *true*.

For our example, the chart helps us decide on an appropriate cutoff confidence value for identifying likely churners. Once this score is determined, we use our model on new data

in order to predict likely churners. Those instances predicted to churn with a confidence score at or above the cutoff will receive special attention in an attempt to prevent churning.

The test set performance vector in Figure 7.15 tells us our model correctly classified 80 churners. Also, 33 churners were incorrectly classified as loyal customers, and 20 loyal customers were determined to have churned. The Pareto lift chart resulting from applying the decision tree model to the test data is displayed in Figure 7.16. Let's look at the chart in more detail:

- We see four bins on the horizontal axis, where each bin represents a range for model confidence.
- To find the number of churners in the test data, add the values 4, 76, 13, and 20, as shown in the numerator portion of the fractions at the top of each interval. This gives 113, which is exactly the number of true churners within the test data. The sum of the denominators of each fraction is 297, which represents the total number of test instances.
- The leftmost bin tells us that five instances within the given confidence range were classified as churners. Four were correctly classified, and one loyal customer was incorrectly designated as having churned.
- If we include all instances classified as having churned with a confidence of 0.73 or greater, we capture 80 ($76 + 4$) of the 113 churners and misclassify 20 loyal customers as having churned. If we include the confidence interval of 0.19 to 0.28, we quickly see that although we add 13 to our total churn count, our model also makes 28 new misclassifications.

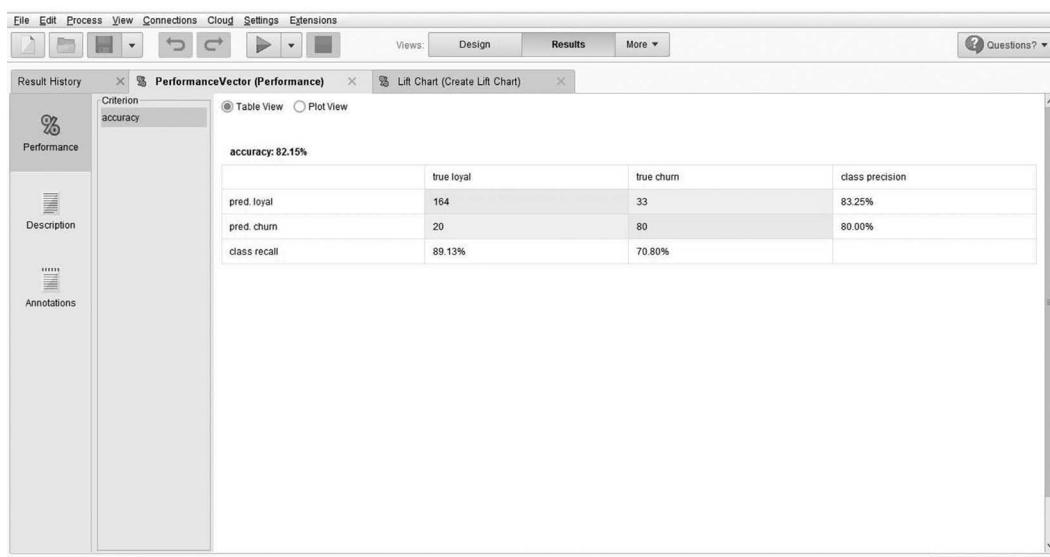


FIGURE 7.15 Performance vector for customer churn.

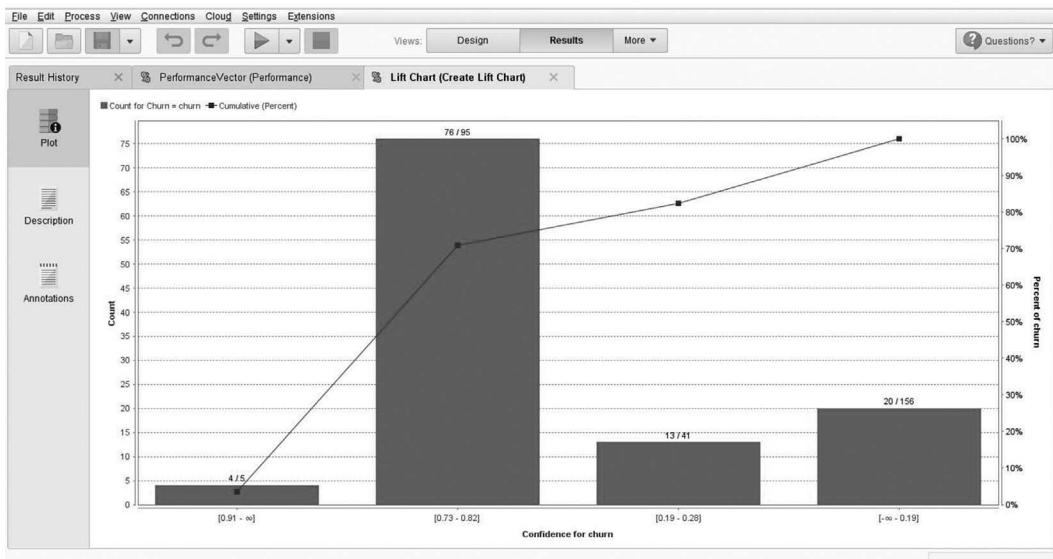


FIGURE 7.16 A Pareto lift chart for customer churn.

- The left vertical axis shows us the total number of correctly classified churners within each confidence range.
- Finally, the right vertical axis is used to interpret the four connected points. To see this, consider the point residing above and midway between the confidence interval 0.19 to 0.28. This positioning of this point tells us that if we include the instances within the confidence interval of 0.19 to 0.28, we select approximately 82.3% of all churners. Likewise, selecting only those with a churn confidence of 0.73 or more captures 70.8% ($80/113 * 100$) of all test set churners.

For our example, the Pareto lift chart visually displays the change in model performance seen as we begin to assign the 33 true churners that have been incorrectly classified as loyal to the class of churned customers. The Pareto lift charts are a very useful visual analytics tool but are also highly sensitive to small changes in parameter settings. We strongly encourage you to further explore the many ways to apply the *Create Lift Chart* operator to your data mining applications.

7.10 CHAPTER SUMMARY

A model's performance is influenced by several components, including training data, input attributes, learner technique, and learner parameter settings to name just a few. Each component that in some way dictates model performance is a candidate for evaluation.

Supervised model performance is most often evaluated with some measure of test set error. For models having categorical output, the measure is test set error rate computed as the ratio of test set errors to total test set instances. For models whose output is numeric, the error measure is usually the mean squared error, the root mean squared error, or the mean absolute error.

Although most data are not normally distributed, the distribution of sample means taken from a set of independent samples of the same size is distributed normally. We can take advantage of this fact by treating test set error rate as a sample mean and applying the properties of normal distributions to compute test set error confidence intervals. We can also apply classical hypothesis testing to compare test set error values for two or more supervised learner models. These statistical techniques offer us a means to associate measures of confidence with the output of our data mining sessions.

Unsupervised clustering techniques often support their own internal evaluation criteria. As many unsupervised techniques offer minimal explanation about the nature of the formed clusters, the evaluation of an unsupervised clustering should include an explanation about what has been discovered. Supervised learning can help explain and evaluate the results of an unsupervised clustering. Another effective evaluative procedure is to perform a between-cluster, attribute-value comparison to determine if the instances contained within the alternative clusters differ significantly. RapidMiner provides several statistical analysis tools to help us formally evaluate the goodness of our data mining models.

7.11 KEY TERMS

- *Classifier error rate.* The number of test set errors divided by the number of test set instances.
- *Control group.* In an experiment, the group not receiving the treatment being measured. The control group is used as a benchmark for measuring change in the group receiving the experimental treatment.
- *Experimental group.* In a controlled experiment, the group receiving the treatment whose effect is being measured.
- *Mean.* The average of a set of numerical values.
- *Mean absolute error.* Given a set of instances each with a specified numeric output, the mean absolute error is the average of the sum of absolute differences between the classifier-predicted output for each instance and the actual output.
- *Mean squared error.* Given a set of instances each with a specified numeric output, the mean squared error is the average of the sum of squared differences between the classifier-predicted output and actual output.
- *Normal distribution.* A distribution of data is considered normal if a frequency graph of the data shows a bell-shaped or symmetrical characteristic.
- *Null hypothesis.* The hypothesis of no significant difference.
- *Root mean squared error.* The square root of the mean squared error.
- *Sample data.* Individual data items drawn from a population of instances.

- *Standard deviation.* The square root of the variance.
- *Standard error.* The square root of a sample variance divided by the total number of sample instances.
- *Statistical significance.* The degree of risk we are willing to take in committing a type 1 error.
- *Stratification.* Selecting data in a way so as to ensure that each class is properly represented in both the training set and the test set.
- *Type 1 error.* Rejecting a null hypothesis when it is true.
- *Type 2 error.* Accepting a null hypothesis when it is false.
- *Validation data.* A set of data that is applied to optimize parameter settings for a supervised model or to help choose from one of several models built with the same training data.
- *Variance.* The average squared deviation from the mean.

EXERCISES

Review Questions

1. Differentiate between the following terms:
 - a. Validation data and test set data
 - b. Type 1 and type 2 error
 - c. Control group and experimental group
 - d. Mean squared error and mean absolute error
2. For each of the following scenarios, state the type 1 and type 2 error. Also, decide whether a model that commits fewer type 1 or type 2 errors would be a best choice. Justify each answer.
 - a. A model for predicting if it will snow
 - b. A model for selecting customers likely to purchase a television
 - c. A model to decide likely telemarketing candidates
 - d. A model to predict whether a person should have back surgery
 - e. A model to determine if a tax return is fraudulent
3. A measure of categorical attribute significance can be computed by multiplying an attribute-value predictiveness and predictability score. What are the advantages and disadvantages of this measure of categorical attribute significance?

4. When a population is normally distributed, we can be more than 95% confident that any value chosen from the population will be within two standard deviations of the population mean. What is the confidence level that any value will fall within three standard deviations on either side of the population mean?
5. Read the description given for RapidMiner's *T-Test alpha* parameter.
 - a. Summarize the description.
 - b. What does the statement "the null hypothesis can never be proven" mean?
6. Write a short description of RapidMiner's *Sample*, *Sample (bootstrap)*, and *Sample (stratified)* operators.

Data Mining Questions—RapidMiner

1. Create a process using a data set of your choice to illustrate the three options available with RapidMiner's *Sample* operator. Attach a note to each operator that clearly describes what your example illustrates. Run your process and provide screenshots of your output.
2. Use the cardiology patient data set to illustrate the *Bootstrap Sampling* operator and the *Stratified Sampling* operator. Attach a note to each operator that clearly describes what your example illustrates. Run your process and provide screenshots of your output.
3. Implement the process model described in Section 7.8 but set the maximum depth for the decision trees at 2, 4, and 10. Also, modify the process so you can see the performance vector for each model.
 - a. Does the *t*-test show any significant differences in model performance?
 - b. Does the ANOVA help confirm the results of the *t*-tests?
 - c. Use the mikro value within each performance vector to compute the 95% confidence error (or accuracy) interval for each model.
4. Implement the process model given in Section 7.9 illustrating the Pareto lift chart but change the target class parameter to *loyal*. Take a screenshot of the chart and summarize what the chart is telling you.
5. Implement the process model in Section 7.8 using the credit screening data set. Also, modify the process so you can see the performance vector for each model.
 - a. Does the *t*-test show any significant differences in model performance?
 - b. Does the ANOVA help confirm the results of the *t*-tests?
 - c. Use the mikro value within each performance vector to compute the 95% confidence error (or accuracy) interval for each model.

III

Building Neural Networks



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Neural Networks

CHAPTER OBJECTIVES

- *Know how input and output data conversions are performed for neural networks*
- *Understand how feed-forward neural networks learn through backpropagation*
- *Know how genetic learning is applied to train feed-forward neural networks*
- *Know how a Kohonen self-organizing neural network performs an unsupervised clustering*
- *List the strengths and weaknesses of neural networks*

Neural networks continue to grow in popularity within the business, scientific, and academic worlds. This is because neural networks have a proven track record in predicting both continuous and categorical outcomes. An excellent overview of neural network applications is given by Widrow et al. (1994).

Although several neural network architectures exist, we limit our discussion to two of the more popular structures. For supervised classification, we examine feed-forward neural networks trained with backpropagation or genetic learning. For unsupervised clustering, we discuss Kohonen self-organizing maps.

Section 8.1 introduces some of the basic concepts and terminology for neural networks. Neural networks require numeric input values ranging between 0 and 1 inclusive. As this can be a problem for some applications, we discuss neural network input and output issues in detail. In Section 8.2, we offer a conceptual overview of how supervised and unsupervised neural networks are trained. Neural networks have been criticized for their inability to explain their output. Section 8.3 looks at some of the techniques that have been developed for neural network explanation. Section 8.4 offers a list of general strengths and weaknesses found with all neural networks. Section 8.5 presents detailed examples of how backpropagation and self-organizing neural networks are trained. If your interests do not

lie in a precise understanding of how neural networks learn, you may want to skip Section 8.5.

8.1 FEED-FORWARD NEURAL NETWORKS

Neural networks offer a mathematical model that attempts to mimic the human brain. Knowledge is often represented as a layered set of interconnected processors. These processor nodes are frequently referred to as *neurodes* so as to indicate a relationship with the neurons of the brain. Each node has a weighted connection to several other nodes in adjacent layers. Individual nodes take the input received from connected nodes and use the connection weights together with a simple evaluation function to compute output values.

Neural network learning can be supervised or unsupervised. Learning is accomplished by modifying network connection weights while a set of input instances is repeatedly passed through the network. Once trained, an unknown instance passing through the network is classified according to the value(s) seen at the output layer.

Figure 8.1 shows a fully connected *feed-forward* neural network structure together with a single input instance [1.0,0.4,0.7]. Arrows indicate the direction of flow for each new instance as it passes through the network. The network is *fully connected* because nodes at one layer are connected to all nodes in the next layer.

The number of input attributes found within individual instances determines the number of input-layer nodes. The user specifies the number of hidden layers as well as the number of nodes within a specific hidden layer. Determining a best choice for these values is a matter of experimentation. In practice, the total number of hidden layers is usually restricted to two. Depending on the application, the output layer of the neural network may contain one or several nodes.

8.1.1 Neural Network Input Format

The input to individual neural network nodes must be numeric and fall in the closed interval range [0,1]. Because of this, we need a way to numerically represent categorical data. We also require a conversion method for numerical data falling outside the [0,1] range.

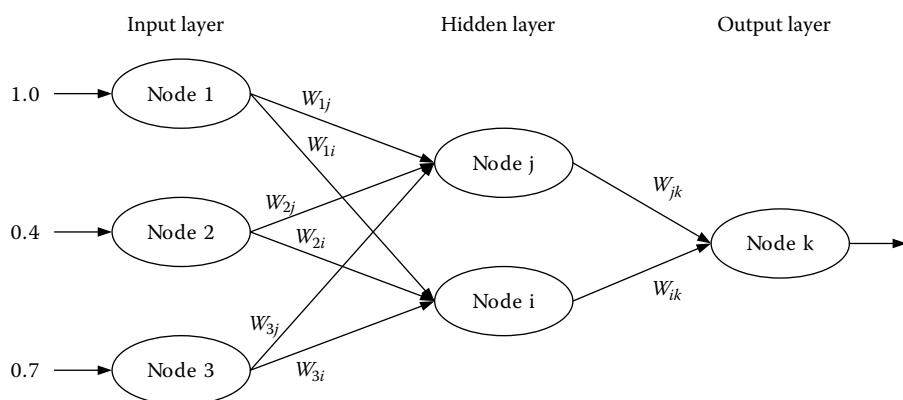


FIGURE 8.1 A fully connected feed-forward neural network.

There are several choices for categorical data conversion. A straightforward technique divides the interval range into equal-size units. To illustrate, consider the attribute *color* with possible values of red, green, blue, and yellow. Using this method, we might make the following assignments: red = 0.00, green = 0.33, blue = 0.67, and yellow = 1.00. Although this technique can be used, it has an obvious pitfall. The modification incorporates a measure of distance not seen prior to the conversion. This is shown in our example in that the distance between red and green is less than the distance between red and yellow. Therefore, it appears as though the color red is more similar to green than it is to yellow.

A second technique for categorical-to-numerical conversion requires one input node for each categorical value. Once again, consider the attribute *color* with the four values assigned as in the previous example. By adding three additional input nodes for *color*, we can represent the four colors as follows: red = [1,0,0,0], green = [0,1,0,0], blue = [0,0,1,0], and yellow = [0,0,0,1]. Using this scheme, each categorical value becomes an attribute with possible values 0 or 1. It is obvious that this method eliminates the bias seen with the previous technique.

Now let's consider the conversion of numerical data to the required interval range. Suppose we have the values 100, 200, 300, and 400. An obvious conversion method is to divide all attribute values by the largest attribute value. For our example, dividing each number by 400 gives the following converted values: 0.25, 0.5, 0.75, and 1.0. The problem with this method is that we cannot take advantage of the entire interval range unless we have at least some values close to 0. A slight modification of this technique offers the desired result, as shown in Equation 8.1.

$$\text{newValue} = \frac{\text{originalValue} - \text{minimumValue}}{\text{maximumValue} - \text{minimumValue}} \quad (8.1)$$

where

newValue is the computed value falling in the [0,1] interval range

originalValue is the value to be converted

minimumValue is the smallest possible value for the attribute

maximumValue is the largest possible attribute value

Applying the formula to the previous values gives us 0.0, 0.33, 0.66, and 1.0.

A special case exists when maximum values cannot be determined. One possible solution is to use an arbitrarily large value as a divisor. Once again, dividing by an arbitrary number leaves us with the possibility of not covering the entire interval range. Finally, highly skewed data may cause less-than-optimal results unless variations of these techniques are applied. A common approach with skewed data is to take the base 2 or base 10 logarithm of each value before applying one of the previous transformations.

8.1.2 Neural Network Output Format

The output nodes of a neural network represent continuous values in the [0,1] range. However, the output can be transformed to accommodate categorical class values. To illustrate, suppose we wish to train a neural network to recognize new credit card customers

likely to take advantage of a special promotion. We design our network architecture with two output-layer nodes, node 1 and node 2. During training, we indicate a correct output for customers that have taken advantage of previous promotions as 1 for the first output node and 0 for the second output node. A correct output for customers that traditionally do not take advantage of promotions is designated by a node 1, node 2 combination of 0 and 1, respectively. Once trained, the neural network will recognize a node 1, node 2 output combination of 0.9, 0.2 as a new customer likely to take advantage of a promotion.

This method has certain disadvantages in that node output combinations such as 0.2, 0.3 have no clear classification. Various approaches have been proposed for this situation. One method suggests the association of certainty factors with node output values. A popular method uses a special test data set to help with difficult-to-interpret output values. This method also allows us to build a neural network with a single output-layer node even when the output is categorical. An example illustrates the method.

Suppose we decide on a single output node for the credit card customer example just discussed. We designate 1 as an ideal output for customers likely to take advantage of a special promotion and 0 for customers likely to pass on the offer. Once we have trained the network, we can be confident about classifying an output value of 0.8 as a customer likely to take advantage of the promotion. However, what do we do when the output value is 0.45? The special test data set helps with our dilemma. Prior to applying the network to unknown instances, we present the test set to the trained network and record the output values for each test instance. We then apply the network to the unknown instances. When unknown instance x shows an uncertain output value v , we classify x with the category shown by the majority of test set instances clustering at or near v .

Finally, when we wish to use the computed output of a neural network for prediction, we have another problem. Let's assume a network has been trained to help us predict the future price of our favorite stock. As the output of the network gives a result between 0 and 1, we need a method to convert the output into an actual future stock price.

Suppose the actual output value is 0.35. To determine the future stock price, we need to undo the original [0,1] interval conversion. The process is simple. We multiply the training data range of the stock price by 0.35 and add the lowest price of the stock to this result. If the training data price range is \$10.00 to \$100.00, the computation is

$$(90.00)(0.35) + \$10.00$$

This gives a predicted future stock price of \$41.50. All commercial and some public domain neural network packages perform these numeric conversions to and from the [0,1] interval range. This still leaves us with the responsibility of making sure all initial input is numeric.

8.1.3 The Sigmoid Evaluation Function

The purpose of each node within a feed-forward neural network is to accept input values and pass an output value to the next higher network layer. The nodes of the input layer pass input attribute values to the hidden layer unchanged. Therefore, for the input instance

shown in Figure 8.1, the output of node 1 is 1.0, the output of node 2 is 0.4, and the output of node 3 is 0.7.

A hidden or output-layer node n takes input from the connected nodes of the previous layer, combines the previous layer's node values into a single value, and uses the new value as input to an evaluation function. The output of the evaluation function is the output of the node, which must be a number in the closed interval [0,1].

Let's look at an example. Table 8.1 shows sample weight values for the neural network of Figure 8.1. Consider node j . To compute the input to node j , we determine the sum total of the multiplication of each input weight by its corresponding input-layer node value. That is,

$$\text{input to node } j = (0.2)(1.0) + (0.3)(0.4) + (-0.1)(0.7) = 0.25$$

Therefore, 0.25 represents the input value for node j 's evaluation function.

The first criterion of an evaluation function is that the function must output values in the [0,1] interval range. A second criterion is that as x increases, $f(x)$ should output values close to 1. In this way, the function propagates activity within the network. The *sigmoid function* meets both criteria and is often used for node evaluation. The sigmoid function is computed as

$$f(x) = \frac{1}{1+e^{-x}} \quad (8.2)$$

where

e is the base of natural logarithms approximated by 2.718282

Figure 8.2 shows the graph of the sigmoid function. Notice that values of x less than 0 provide little output activation. For our example, $f(0.25)$ evaluates to 0.562, which represents the output of node j .

TABLE 8.1 Initial Weight Values for the Neural Network shown in Figure 8.1

W_{lj}	W_{li}	W_{2j}	W_{2i}	W_{3j}	W_{3i}	W_{jk}	W_{ik}
0.20	0.10	0.30	-0.10	-0.10	0.20	0.10	0.50

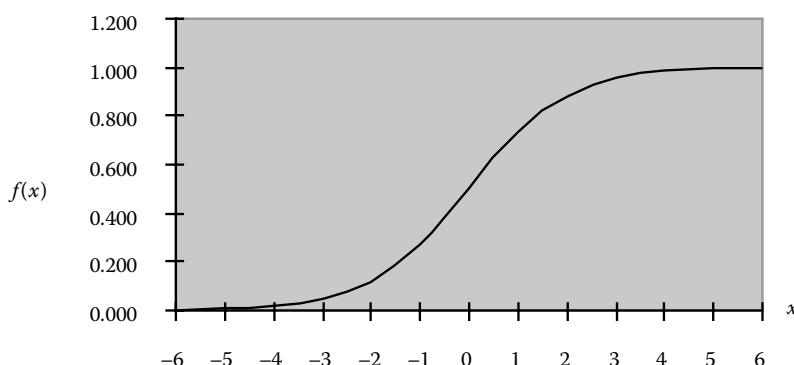


FIGURE 8.2 The sigmoid evaluation function.

Finally, it is important to note that a *bias* or *threshold* node is often times associated with each hidden and output-layer node. Threshold nodes differ from regular nodes in that their link values change during the learning process in the same manner as all other network weights but their input is fixed at a constant value (usually 1). The purpose and functioning of threshold nodes is made clear in Chapters 9 and 10, where we investigate Weka's neural network function and RapidMiner's neural net operator.

8.2 NEURAL NETWORK TRAINING: A CONCEPTUAL VIEW

In this section, we discuss two methods for training feed-forward networks and one technique for unsupervised neural net clustering. Our discussion is limited in that we do not detail how the algorithms work. For most readers, the discussion here is enough to satisfy basic curiosities about neural network learning. However, for the more technically inclined individual, Section 8.5 offers specific details about how neural networks learn.

8.2.1 Supervised Learning with Feed-Forward Networks

Supervised learning involves both training and testing. During the training phase, training instances are repeatedly passed through the network while individual weight values are modified. The purpose of changing the connection weights is to minimize training set error between predicted and actual output values. Network training continues until a specific terminating condition is satisfied. The terminating condition can be convergence of the network to a minimum total error value, a specific time criterion, or a maximum number of iterations.

8.2.1.1 Training a Neural Network: Backpropagation Learning

Backpropagation learning is most often used to train feed-forward networks. For each training instance, backpropagation works by first feeding the instance through the network and computing the network output value. Recall that one output value is computed for each output-layer node. To illustrate backpropagation learning, we use Figure 8.1 together with the input instance shown in the figure.

We previously determined that the computed output for the instance in Figure 8.1 is 0.52. Now suppose that the target output associated with the instance is 0.65. Obviously, the absolute error between the computed and target values is 0.13. However, a problem is seen when we attempt to determine why the error occurred. That is, we do not know which of the network connection weights is to blame for the error. It is possible that changing just one of the weights will provide us with a better result the next time the instance passes through the network. It is more likely that the problem lies with some combination of two or more weight values. Still another possibility is that the error is, to some degree, the fault of every network connection associated with the output node.

The backpropagation learning algorithm assumes this last possibility. For our example, the output error at node k is propagated back through the network, and all eight of the associated network weights change value. The amount of change seen with each connection weight is computed with a formula that makes use of the output error at node k, individual node output values, and the derivative of the sigmoid function. The formula has a

way of smoothing the actual error value so as not to cause an overcorrection for any one training instance.

Given enough iterations, the backpropagation learning technique is guaranteed to converge. However, there is no guarantee that the convergence will be optimal. Therefore, several applications of the algorithm may be necessary to achieve an acceptable result.

8.2.1.2 Training a Neural Network: Genetic Learning

Genetic learning is also a choice for training feed-forward neural nets. The general idea is straightforward. As usual, we first randomly initialize a population of elements where each element represents one possible set of network connection weights. Table 8.2 displays a set of plausible population elements for the network architecture seen in Figure 8.1. Once the element population is initialized, the neural network architecture is populated with the weights of the first element. The training data are passed through the created network once, and the output error for each training instance is recorded. After all training instances have passed through the network, an average mean squared or absolute error is computed. The computed average is the fitness score for the first population element. This process repeats for each element of the population.

As soon as the entire element population has been used to build and train a network structure, some combination of crossover, mutation, and selection is applied, and the process repeats. Network convergence is seen when at least one element of the population builds a network that achieves an acceptable fitness score. When training is complete, the network is given the weights associated with the element having a best fitness score. As with backpropagation learning, this technique is guaranteed to converge. However, there is no guarantee that the convergence will be optimal. Once again, using several applications of the genetic learning algorithm offers a best chance of achieving an acceptable result.

8.2.2 Unsupervised Clustering with Self-Organizing Maps

Teuvo Kohonen (1982) first formalized neural network unsupervised clustering in the early 1980s when he introduced Kohonen feature maps. His original work focused on mapping images and sounds. However, the technique has been effectively used for unsupervised clustering. Kohonen networks are also known as self-organizing maps (SOMs).

Kohonen networks support two layers. The input layer contains one node for each input attribute. Nodes of the input layer have a weighted connection to all nodes in the output layer. The output layer can take any form but is commonly organized as a two-dimensional

TABLE 8.2 A Population of Weight Elements for the Network in Figure 8.1

Population Element	W_{1j}	W_{1i}	W_{2j}	W_{2i}	W_{3j}	W_{3i}	W_{jk}	W_{ik}
1	0.20	2.10	0.30	-0.10	-3.10	2.20	0.10	0.50
2	0.14	1.38	0.19	0.25	-4.17	0.27	0.11	0.54
3	0.20	0.10	0.38	-3.16	-0.16	0.24	0.12	6.53
4	0.23	0.10	0.39	-2.18	-0.17	0.26	5.15	0.54

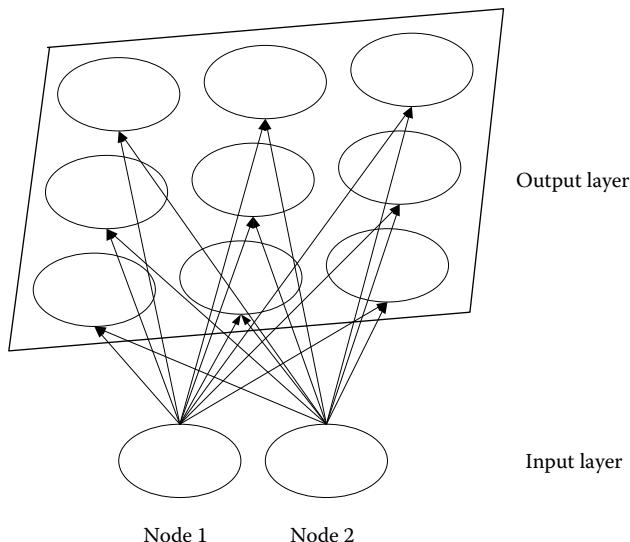


FIGURE 8.3 A 3×3 Kohonen network with two input-layer nodes.

grid. Figure 8.3 shows a simple Kohonen network with two input-layer and nine output-layer nodes.

During network learning, input instances are presented to each output-layer node. When an instance is presented to the network, the output node whose weight connections most closely match the input instance *wins* the instance. The node is rewarded by having its weights changed to more closely match the instance. At first, neighbors to the winning node are also rewarded by having their weight connections modified to more closely match the attribute values of the current instance. However, after the instances have passed through the network several times, the size of the neighborhood decreases until finally, only the winning node gets rewarded.

Each time the instances pass through the network, the output-layer nodes keep track of the number of instances they win. The output nodes winning the most instances during the last pass of the data through the network are saved. The number of output-layer nodes saved corresponds to the number of clusters believed to be in the data. Finally, those training instances clustered with deleted nodes are once again presented to the network and classified with one of the saved nodes. The nodes, together with their associated training set instances, characterize the clusters in the data set. Alternatively, test data may be applied, and the clusters formed by these data are then analyzed to help determine the meaning of what has been found.

8.3 NEURAL NETWORK EXPLANATION

A major disadvantage seen with the neural network architecture is a lack of understanding about what has been learned. Here we summarize four techniques for addressing this disadvantage. One possibility for neural network explanation is to transform a network architecture into a set of rules. Algorithms designed to extract rules from a

neural network typically involve removing weighted links that minimally affect classification correctness. Unfortunately, rule extraction methods have met with limited success.

Sensitivity analysis is a second technique that has been successfully applied to gain insight into the effect individual attributes have on neural network output. There are several variations to the approach. The general process consists of the following steps:

-
1. Divide the data into a training set and a test data set.
 2. Train the network with the training data.
 3. Use the test set data to create a new instance I . Each attribute value for I is the average of all attribute values within the test data.
 4. For each attribute,
 - a. Vary the attribute value within instance I and present the modification of I to the network for classification.
 - b. Determine the effect the variations have on the output of the neural network.
 - c. The relative importance of each attribute is measured by the effect of attribute variations on network output.
-

A sensitivity analysis allows us to determine a rank ordering for the relative importance of individual attributes. However, the approach does not offer an explicit set of rules to help us understand more about what has been learned.

A third technique that has merit as a generalized explanation tool for unsupervised clustering is the *average member technique*. With this method, the average or most typical member of each class is computed by finding the average value for each class attribute in each class (versus the average value of all instances as done for sensitivity analysis).

A fourth and more informative alternative is to apply the method described in Chapter 2 where supervised learning is employed to interpret the results of an unsupervised clustering. We utilized this method in Chapter 5 in combination with the K -means algorithm. Here is the procedure as it applies to unsupervised neural network learning:

-
1. Perform any data transformations necessary to prepare the data for the unsupervised neural network clustering.
 2. Present the data to the unsupervised network model.
 3. Call each cluster created by the neural network a class and assign each cluster an arbitrary class name.
 4. Use the newly formed classes as training instances for a decision tree algorithm.
 5. Examine the decision tree to determine the nature of the concept classes formed by the clustering algorithm.
-

This method holds an advantage over the average member technique in that we are offered a generalization about differences as well as similarities of the formed clusters. As an alternative, a rule generator can be employed to describe each cluster in detail.

8.4 GENERAL CONSIDERATIONS

The process of building a neural network is both an art and a science. A reasonable approach is to conduct several experiments while varying attribute selections and learning parameters. The following is a partial list of choices that affect the performance of a neural network model:

- What input attributes will be used to build the network?
- How will the network output be represented?
- How many hidden layers should the network contain?
- How many nodes should there be in each hidden layer?
- What condition will terminate network training?

There are no right answers to these questions. However, we can use the experimental process to help us achieve desired results. In Chapters 9 and 10, you will learn how to better answer these questions by experimenting with Weka's and RapidMiner's neural net software tools. Here we provide a list of strengths and weaknesses for the neural network approach to knowledge discovery.

Strengths

- Neural networks work well with data sets containing large amounts of noisy input data. Neural network evaluation functions such as the sigmoid function naturally smooth input data variations caused by outliers and random error.
- Neural networks can process and predict numeric as well as categorical outcome. However, categorical data conversions can be tricky.
- In Chapter 13, we show how neural networks can be used for applications that require a time element to be included in the data.
- Neural networks have performed consistently well in several domains.
- Neural networks can be used for both supervised classification and unsupervised clustering.

Weaknesses

- Probably the biggest criticism of neural networks is that they lack the ability to explain their behavior.
- Neural network learning algorithms are not guaranteed to converge to an optimal solution. With most types of neural networks, this problem can be dealt with by manipulating various learning parameters.

- Neural networks can be easily trained to work well on the training data but poorly on test data. This often happens when the training data are passed through the network so many times that the network is not able to generalize when presented with previously unseen instances. This problem can be monitored by consistently measuring test set performance.

8.5 NEURAL NETWORK TRAINING: A DETAILED VIEW

Here we provide detailed examples of how two popular neural network architectures modify their weighted connections during training. In the first section, we provide a partial example of backpropagation learning and state a general form of the backpropagation learning algorithm. In the second section, we show you how Kohonen SOMs are used for unsupervised clustering.

8.5.1 The Backpropagation Algorithm: An Example

Backpropagation is the training method most often used with feed-forward networks. Backpropagation works by making modifications in weight values starting at the output layer and then moving backward through the hidden layers. The process is best understood with an example. We will follow one pass of the backpropagation algorithm using the neural network of Figure 8.1, the input instance shown in the figure, and the initial weight values from Table 8.1.

Let's assume the target output for the specified input instance is 0.68. The first step is to feed the instance through the network and determine the computed output for node k. We apply the sigmoid function to compute all output values, as shown in the following calculations.

$$\text{Input to node } j = (0.2)(1.0) + (0.3)(0.4) + (-0.1)(0.7) = 0.250$$

$$\text{Output from node } j = 0.562$$

$$\text{Input to node } i = (0.1)(1.0) + (-0.1)(0.4) + (0.2)(0.7) = 0.200$$

$$\text{Output from node } i = 0.550$$

$$\text{Input to node } k = 0.1 * 0.562 + 0.5 * 0.550 = 0.331$$

$$\text{Output from node } k = 0.582$$

Next, we compute the observed error at the output layer of the network. The output-layer error is computed as

$$\text{Error } (k) = (T - O_k)[f'(x_k)] \quad (8.3)$$

where

T = the target output

O_k = the computed output of node k

$(T - O_k)$ = the actual output error

$f'(x_k)$ = the first-order derivative of the sigmoid function
 x_k = the input to the sigmoid function at node k

Equation 8.3 shows that the actual output error is multiplied by the first-order derivative of the sigmoid function. The multiplication scales the output error, forcing stronger corrections at the point of rapid rise in the sigmoid curve. The derivative of the sigmoid function at x_k conveniently computes to $O_k(1 - O_k)$. Therefore,

$$\text{Error}(k) = (T - O_k) O_k (1 - O_k) \quad (8.4)$$

For our example, $\text{Error}(k)$ is computed as

$$\text{Error}(k) = (0.65 - 0.582)(0.582)(1 - 0.582) = 0.017$$

Computing the output errors for hidden-layer nodes is a bit more intuitive. The general formula for the error at node j is

$$\text{Error}(j) = \left(\sum_k \text{Error}(k) W_{jk} \right) f'(x_j) \quad (8.5)$$

where

$\text{Error}(k)$ = the computed output error at node k
 W_{jk} = the weight associated with the link between node j and output node k
 $f'(x_j)$ = the first-order derivative of the sigmoid function
 x_j = the input to the sigmoid function at node j

As in Equation 8.3, $f'(x_j)$ evaluates to $O_j(1 - O_j)$.

Notice that the computed error is summed across all output nodes. For our example, we have a single output node. Therefore,

$$\text{Error}(j) = (0.017)(0.1)(0.562)(1 - 0.562) = 0.00042$$

We leave the computation of $\text{Error}(i)$ as an exercise.

The final step in the backpropagation process is to update the weights associated with the individual node connections. Weight adjustments are made using the *delta rule* developed by Widrow and Hoff (Widrow and Lehr 1995). The objective of the delta rule is to minimize the sum of the squared errors, where error is defined as the distance between computed and actual output. We will give the weight adjustment formulas and illustrate the process with our example. The formulas are as follows:

$$w_{jk} (\text{new}) = w_{jk} (\text{current}) + \Delta w_{jk} \quad (8.6)$$

where Δw_{jk} is the value added to the current weight value.

Finally, Δw_{jk} is computed as

$$\Delta w_{jk} = (r)[Error(k)](O_j) \quad (8.7)$$

where

r = the learning rate parameter with $1 > r > 0$

$Error(k)$ = the computed error at node k

O_j = the output of node j

Here are the parameter adjustments for our example with $r = 0.5$.

- $\Delta w_{jk} = (0.5)(0.017)(0.562) = 0.0048$

The updated value for $w_{jk} = 0.1 + 0.0048 = 0.1048$

- $\Delta w_{1j} = (0.5)(0.00042)(1.0) = 0.0002$

The updated value for $w_{1j} = 0.2 + 0.0002 = 0.2002$

- $\Delta w_{2j} = (0.5)(0.00042)(0.4) = 0.000084$

The updated value for $w_{2j} = 0.3 + 0.000084 = 0.300084$

- $\Delta w_{3j} = (0.5)(0.00042)(0.7) = 0.000147$

The updated value for $w_{3j} = -0.1 + 0.000147 = -0.099853$

We leave adjustments for the links associated with node i as an exercise. Now that we have seen how backpropagation works, we state the general backpropagation learning algorithm.

1. Initialize the network.
 - a. Create the network topology by choosing the number of nodes for the input, hidden, and output layers.
 - b. Initialize weights for all node connections to arbitrary values between -1.0 and 1.0.
 - c. Choose a value between 0 and 1.0 for the learning parameter.
 - d. Choose a terminating condition.
2. For all training set instances,
 - a. Feed the training instance through the network.
 - b. Determine the output error.
 - c. Update the network weights using the previously described method.
3. If the terminating condition has not been met, repeat step 2.
4. Test the accuracy of the network on a test data set. If the accuracy is less than optimal, change one or more parameters of the network topology and start over.

The terminating condition can be given as a total number of passes (also called *epochs*) of the training data through the network. Alternatively, network termination can be determined by the degree to which learning has taken place within the network. A generalized form of the *root mean squared error* (rms) introduced in Chapter 7 is often used as a standard measure of network learning. The general formula to calculate rms is given as the square root of the following value.

$$\sqrt{\frac{\sum_n \sum_i (T_{in} - O_{in})^2}{ni}} \quad (8.8)$$

where

n = the total number of training set instances

i = the total number of output nodes

T_{in} = the target output for the n th instance and the i th output node

O_{in} = the computed output for the n th instance and i th output node

As you can see, the rms is simply the square root of the average of all instance output error values. A common criterion is to terminate the backpropagation learning when the rms is less than 0.10. Variations of the just-stated approach exist. One common variation is to keep track of training data errors but wait to update network weight connections only after all training instances have passed through the network. Regardless of the methodology, several iterations of the process are often necessary for an acceptable result.

8.5.2 Kohonen Self-Organizing Maps: An Example

To see how unsupervised clustering is accomplished, we consider the input-layer nodes and two of the output-layer nodes for the Kohonen feature map shown in Figure 8.3. The situation is displayed in Figure 8.4.

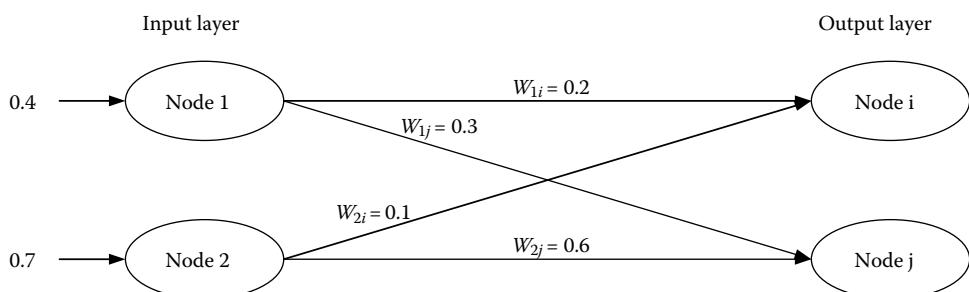


FIGURE 8.4 Connections for two output-layer nodes.

Recall that when an instance is presented to the network, a score for classifying the instance with each of the output-layer nodes is computed. The score for classifying a new instance with output node j is given by

$$\sqrt{\sum_i (n_i - w_{ij})^2} \quad (8.9)$$

where n_i is the attribute value for the current instance at input node i , and w_{ij} is the weight associated with the i th input node and output node j . That is, the output node whose weight vectors most closely match the attribute values of the input instance is the winning node.

Let's use Equation 8.9 to compute the score for the two output nodes shown in Figure 8.4 using the input instance [0.4,0.7]. The score for inclusion with output node i is:

$$\sqrt{(0.4 - 0.2)^2 + (0.7 - 0.1)^2} = 0.632$$

Likewise, the score for inclusion with output node j is computed as

$$\sqrt{(0.4 - 0.3)^2 + (0.7 - 0.6)^2} = 0.141$$

As you can see, node j is the winner as its weight vector values are more similar to the input values of the presented instance. As a result, the weight vectors associated with the output node are adjusted so as to reward the node for winning the instance. The following formula is used to adjust the values of the weight vectors:

$$w_{ij}(\text{new}) = w_{ij}(\text{current}) + \Delta w_{ij} \quad (8.10)$$

where

$$\begin{aligned}\Delta w_{ij} &= r(n_i - w_{ij}) \\ 0 < r < 1\end{aligned}$$

The winning output node has its weight vectors adjusted in the direction of the values contained within the new instance. For our example, with the learning rate parameter $r = 0.5$, the weight vectors of node j are adjusted as follows:

- $\Delta w_{1j} = (0.5)(0.4 - 0.3) = 0.05$
- $\Delta w_{2j} = (0.5)(0.7 - 0.6) = 0.05$
- $w_{1j}(\text{new}) = 0.3 + 0.05 = 0.35$
- $w_{2j}(\text{new}) = 0.6 + 0.05 = 0.65$

Output-layer nodes within a specified neighborhood of the winning node also have their weights adjusted using the same formula. A square grid typically defines the neighborhood. The center of the grid contains the winning node. The size of the neighborhood as well as the learning rate r is specified when training begins. Both parameters are decreased linearly over the span of several iterations. Learning terminates after a preset number of iterations or after instance classifications do not vary from one iteration to the next.

To complete the clustering, the output nodes have their connection weights fixed, and all but the n most populated output nodes are deleted. After this, the original training data or a previously unseen test data set is fed through the output layer one last time. If the original training data are used, those instances previously associated with a deleted node move to one of the appropriate remaining clusters. Finally, the clusters formed by the training or test data are analyzed possibly using one or both of the unsupervised evaluation methods described previously in order to determine what has been discovered.

8.6 CHAPTER SUMMARY

A neural network is a parallel computing system of several interconnected processor nodes. The input to individual network nodes is restricted to numeric values falling in the closed interval range $[0,1]$. Because of this, categorical data must be transformed prior to network training.

Developing a neural network involves first training the network to carry out the desired computations and then applying the trained network to solve new problems. During the learning phase, training data are used to modify the connection weights between pairs of nodes so as to obtain a best result for the output node(s).

The feed-forward neural network architecture is commonly used for supervised learning. Feed-forward neural networks contain a set of layered nodes and weighted connections between nodes in adjacent layers.

Feed-forward neural networks are often trained using a backpropagation learning scheme. Backpropagation learning works by making modifications in weight values starting at the output layer and then moving backward through the hidden layers of the network. Genetic learning can also be applied to train feed-forward networks.

The self-organizing Kohonen neural network architecture is a popular model for unsupervised clustering. A self-organizing neural network learns by having several output nodes compete for the training instances. For each instance, the output node whose weight vectors most closely match the attribute values of the input instance is the winning node. As a result, the winning node has its associated input weights modified to more closely match the current training instance. When unsupervised learning is complete, output nodes winning the most instances are saved. After this, test data are applied, and the clusters formed by the test set data are analyzed to help determine the meaning of what has been found.

A central issue surrounding neural networks is their inability to explain what has been learned. Despite this, neural networks have been successfully applied to solve problems in both the business and scientific worlds. Although we have discussed the most popular neural network models, several other architectures and learning rules have been developed. Jain et al. (1996) provide a good starting point for learning more about neural networks.

8.7 KEY TERMS

- *Average member technique.* An unsupervised clustering neural network explanation technique where the most typical member of each cluster is computed by finding the average value for each class attribute.
- *Backpropagation learning.* Backpropagation is a training method used with many feed-forward networks. Backpropagation works by making modifications in weight values starting at the output layer and then moving backward through the hidden layer.
- *Bias or threshold nodes.* Nodes associated with hidden- and output-layer nodes. Bias or threshold nodes differ from regular nodes in that their input is a constant value. Corresponding link weights change during the learning process in the same manner as all other network weights.
- *Delta rule.* A neural network learning rule designed to minimize the sum of squared errors between computed and target network output.
- *Epoch.* One complete pass of the training data through a neural network.
- *Feed-forward neural network.* A neural network architecture where all weights at one layer are directed toward nodes at the next network layer. Weights do not cycle back as inputs to previous layers.
- *Fully connected.* A neural network structure where all nodes at one layer of the network are connected to all nodes in the next layer.
- *Kohonen network.* A two-layer neural network used for unsupervised clustering.
- *Neural network.* A parallel computing system consisting of several interconnected processors.
- *Neurode.* A neural network processor node. Several neurodes are connected to form a complete neural network structure.
- *Sensitivity analysis.* A neural network explanation technique that allows us to determine a rank ordering for the relative importance of individual attributes.
- *Sigmoid function.* One of several commonly used neural network evaluation functions. The sigmoid function is continuous and outputs a value between 0 and 1.

EXERCISES

Review Questions

1. Draw the nodes and node connections for a fully connected feed-forward network that accepts three input values, and has one hidden layer of five nodes and an output layer containing four nodes.

2. Section 8.1 describes two methods for categorical data conversion. Explain how you would use each method to convert the categorical attribute *income range* with possible values 10–20K, 20–30K, 30–40K, 40–50K... 0–100K to numeric equivalents. Which method is most appropriate?
3. The average member technique is sometimes used to explain the results of an unsupervised neural network clustering.
 - a. List the advantages and disadvantages of this approach.
 - b. Do you see similarities between this explanation technique and the *K*-means algorithm?
 - c. A major issue with the *K*-means algorithm is its inability to guarantee an optimal solution. How might the average member technique be incorporated into the *K*-means algorithm to help with this problem?

Computational Questions

1. We have trained a neural network to predict the future price of our favorite stock. The 1-year stock price range is a low of \$20 and a high of \$50.
 - a. Use Equation 8.1 to convert the current stock price of \$40 to a value between 0 and 1.
 - b. Suppose we apply the network model to predict a new price for some time period in the future. The neural network gives an output price value of 0.3. Convert this value to a predicted price we can understand.
2. Consider the feed-forward network in Figure 8.1 with the associated connection weights shown in Table 8.1. Apply the input instance [0.5,0.2,1.0] to the feed-forward neural network. Specifically,
 - a. Compute the input to nodes i and j.
 - b. Use the sigmoid function to compute the initial output of nodes i and j.
 - c. Use the output values computed in part (b) to determine the input and output values for node k.

Building Neural Networks with Weka

CHAPTER OBJECTIVES

- *Perform supervised neural network learning with Weka's MultiLayerPerceptron function*
- *Know how nominal attributes are transformed into real-valued equivalents*
- *Understand the importance of attribute preprocessing prior to neural network learning*
- *Know that Weka's neural network function can build models when the output attribute is either categorical or numeric*
- *Understand that Weka's neural network function is able to build models when instances contain some missing input data*
- *Know how to save and apply a neural network model to new data of unknown origin*
- *Perform unsupervised clustering with Weka's implementation of the Kohonen Self-Organizing map*

In this chapter, we introduce Weka's *MultiLayerPerceptron (MLP)* function for supervised learning and Weka's *SelfOrganizingMap* algorithm for unsupervised clustering. In Section 9.1, we describe the data sets used for illustrating Weka's neural network function as well as a five-step approach for building feed-forward neural networks. These data sets are also utilized in Chapter 10, where the focus is on RapidMiner's neural network operator. In Section 9.2, we use the *exclusive-OR* (XOR) function to illustrate how MLP builds feed-forward neural networks when the output attribute is numeric. In Section 9.3, we again use the XOR function but this time to show how to build feed-forward networks when the output attribute is categorical. In Section 9.4, we employ the satellite image data set to revisit attribute selection and to review the process of applying saved learner models to new data of unknown outcome. In Section 9.5, we use Weka's implementation of the clustering

algorithm described in Chapter 8 to cluster instances of individuals who tested positive or negative for diabetes. We encourage you to make use of the MS PowerPoint file *Screens_09.ppt*, which clearly depicts what you will see as you work through the tutorials.

9.1 DATA SETS FOR BACKPROPAGATION LEARNING

Here we describe the data sets used for our experiments. The first data set offers a simple yet interesting example.

9.1.1 The Exclusive-OR Function

Most of us are familiar with the basic logical operators. Common operators include *and*, *or*, *implication*, *negation*, and *exclusive OR* (XOR). The definition of the XOR function is shown in Table 9.1. We can think of the XOR function as defining two classes. One class is denoted by the two instances with XOR function output equal to 1. The second class is given by the two instances with function output equal to 0.

Figure 9.1 offers a graphical interpretation of the output. The *x*-axis represents values for *input 1*, and the *y*-axis denotes values for *input 2*. The instances for the class with XOR equal to 1 are denoted in Figure 9.1 by an *A*. Likewise, instances for the class with XOR equal to 0 are denoted with a *B*. The XOR function is of particular interest because, unlike the other logical operators, the XOR function is not *linearly separable*. This is seen in Figure 9.1, as we cannot draw a straight line to separate the instances in class *A* from those in class *B*.

TABLE 9.1 Exclusive-OR Function

Input 1	Input 2	XOR
1	1	0
0	1	1
1	0	1
0	0	0

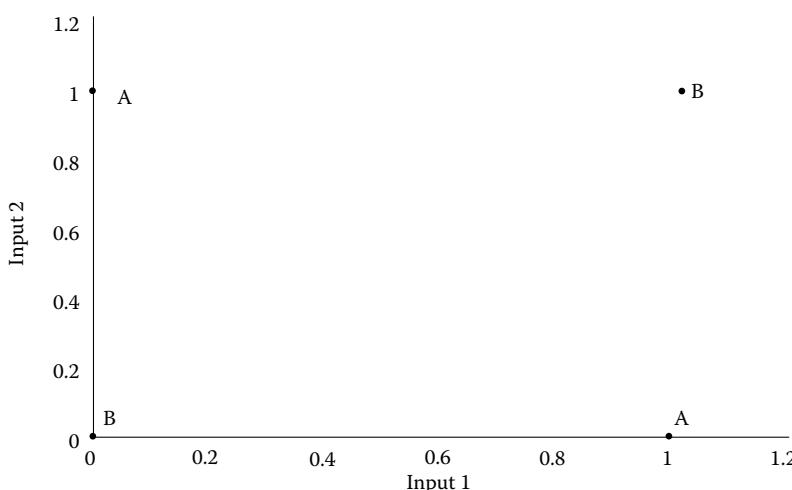


FIGURE 9.1 Graph of the XOR function.

	A	B	C	D	E	F	G	H
1	Input 1	Input 2	XOR					
2	1	1	0					
3	1	0	1					
4	0	1	1					
5	0	0	0					
6								
7								
8								
9								
10								

FIGURE 9.2 XOR training data.

The first neural networks, known as *perceptron networks*, consisted of an input layer and a single output layer. The XOR function caused trouble for these early networks because they are able to converge only when presented with problems having linearly separable solutions. The development of the backpropagation network architecture, which is able to model nonlinear problems, contributed to a renewed interest in neural network technology.

Figure 9.2 shows the training data for the XOR function in an Excel spreadsheet. The top row gives the attribute names. Recall that all neural network input must be real-valued. The XOR function represents the output attribute. With MLP, the output attribute can be either numeric or categorical.

9.1.2 The Satellite Image Data Set

The second data set contains pixels representing a digitized satellite image of a portion of the earth's surface. The training and test data consist of 300 pixels for which *ground truth* has been established. Ground truth of a satellite image is established by having a person on the ground measure the same thing the satellite is trying to measure (at the same time). The answers are then compared to help evaluate how well the satellite instrument is doing its job.

These data have been classified into 15 categories: Urban, Agriculture 1, Agriculture 2, Turf/Grass, Southern Deciduous, Northern Deciduous, Coniferous, Shallow Water, Deep Water, Marsh, Shrub Swamp, Wooded Swamp, Dark Barren, Barren 1, and Barren 2. Each category contains approximately 20 instances. Each pixel is represented by six numeric values consisting of the multispectral reflectance values in six bands of the electromagnetic spectrum: blue (0.45–0.52 m), green (0.52–0.60 m), red (0.63–0.069 m), near infrared (0.76–0.90 m), and two middle infrared (1.55–1.75 and 2.08–2.35 m).

An interesting aspect of this data set is the fact that there are 15 output classes. One way to define the output layer is to use 4 nodes to model the 15 classes, since having 4 nodes with binary values offers 16 possibilities. However, a better approach is to use one output node for each class, giving a total of 15 output-layer nodes. In this way, each class is represented by exactly one output node, which can be treated as a categorical or numeric value. Figure 9.3 shows a single representative instance from each of the 15 classes. The class output sequence was arbitrarily chosen.

	A	B	C	D	E	F	G	H	I	J	K	L
1	blue	green	red	nred	ir1	ir2	class					
2	104	43	49	65	80	43	urban					
3	82	34	30	35	78	23	agriculture1					
4	84	34	32	19	76	24	agriculture2					
5	83	37	35	45	6	33	turf_grass					
6	98	41	63	81	55	77	s_deciduous					
7	93	35	54	69	29	60	n_deciduous					
8	78	29	28	67	64	25	coniferous					
9	75	24	21	12	9	4	shallow_water					
10	78	26	24	15	11	6	deep_water					
11	88	34	42	46	73	40	marsh					
12	82	30	33	42	71	33	shrub_swamp					
13	82	31	36	49	68	30	flooded_swamp					
14	108	51	78	69	15	77	dark_barren					
15	125	58	81	82	74	95	br_barren1					
16	148	66	88	74	23	78	br_barren2					

FIGURE 9.3 Satellite image data.

Now that we have identified the data sets for our experiments, it's time to show you how to build models that perform backpropagation learning. We follow a simple five-step approach:

1. Identify the goal.
2. Prepare the data to be mined.
3. Define the network architecture.
4. Watch the network train.
5. Read and interpret summary results.

When using neural network architectures, it is important to remind ourselves that neural networks are unable to determine attribute relevance. Although backpropagation networks can still do well if some irrelevant attributes are present, we are more likely to achieve an acceptable outcome when attribute relevance is addressed in step 2.

9.2 MODELING THE EXCLUSIVE-OR FUNCTION: NUMERIC OUTPUT

MLP applies backpropagation learning to build feed-forward neural networks. Let's build a backpropagation network using the aforementioned five-step approach to model the XOR function.

Step 1: Identify the Goal

The goal for the XOR problem is straightforward. Given two input values where each value is either a 1 or a 0, we want our network to output the value of the XOR function.

Step 2: Prepare the Data

MLP uses the previously described sigmoid function for node evaluation. The function processes both numeric and nominal (categorical) input attributes. In the case of a categorical attribute, by default the network creates a separate input node for each possible input value. Output attributes can also be either numeric or categorical. If an output attribute is numeric, the network requires but one output node. However, if the output attribute is nominal, each value of the output attribute is assigned an output node. Once the network has been built, an unknown instance is classified with the class whose corresponding output node shows the largest numeric value.

We first consider the Weka file *XOR.arff* where the output attribute is numeric. Open Weka's Explorer, and while in preprocess mode, load *XOR.arff*. Your screen will be similar to Figure 9.4. Be sure that the class attribute is designated as XOR.

Step 3: Define the Network Architecture

To define the network architecture, we must move from preprocess mode to classify mode. Click on *Choose*, then *functions*, and finally, *MultiLayerPerceptron*. Be sure to highlight the *Use training set* radio button as the *Test option*. This is the right choice as the training data contain all possible values presented to the network.

At this point, unless we wish to use default values for the number of network layers and epochs, defining the network architecture requires us to make several choices. Using the default network settings is almost always a poor choice. Given this, click

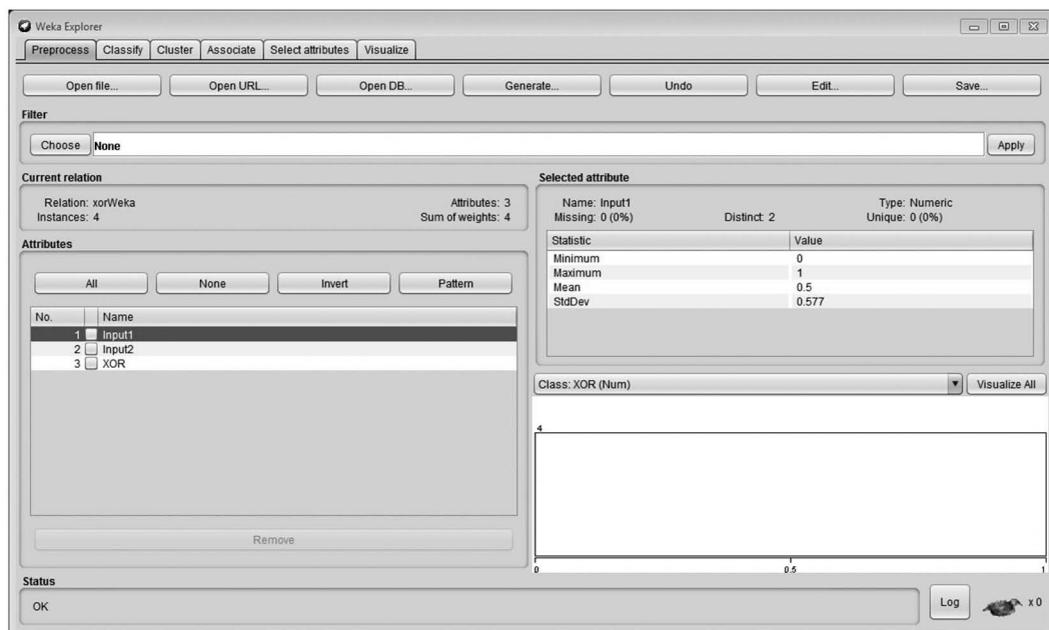


FIGURE 9.4 Weka four graphical user interfaces (GUIs) for XOR training.

anywhere on the *Choose* command line to display the *GenericObjectEditor*. Your screen will appear similar to that shown in Figure 9.5. Click on *capabilities* to learn about this classifier's limitations. Click on *more* to read about all of the options available with MLP. We have used arrows to highlight five of the parameters that are of particular interest. Let's examine each of these parameters in greater detail.

- The *GUI* parameter allows us to visualize the network prior to and during the training process. The default value is false for good reason. If for example, with a 10-fold cross-validation and a setting of *true* for the *GUI* parameter, each fold will present us with a newly created network, where we must respond with a click on *accept* to continue. Also, it is important to note that when the *GUI* parameter is *true*, we can terminate network training or testing prior to completion by simply clicking *accept*. For our experiment, we wish to see the structure of the created network(s), so let's set the *GUI* parameter to *true*.
- The *hiddenLayers* parameter allows us to specify the total number of nodes in each hidden layer and the total number of hidden layers. The value 4 for this parameter gives us one hidden layer containing four nodes. The value 4,3 for this

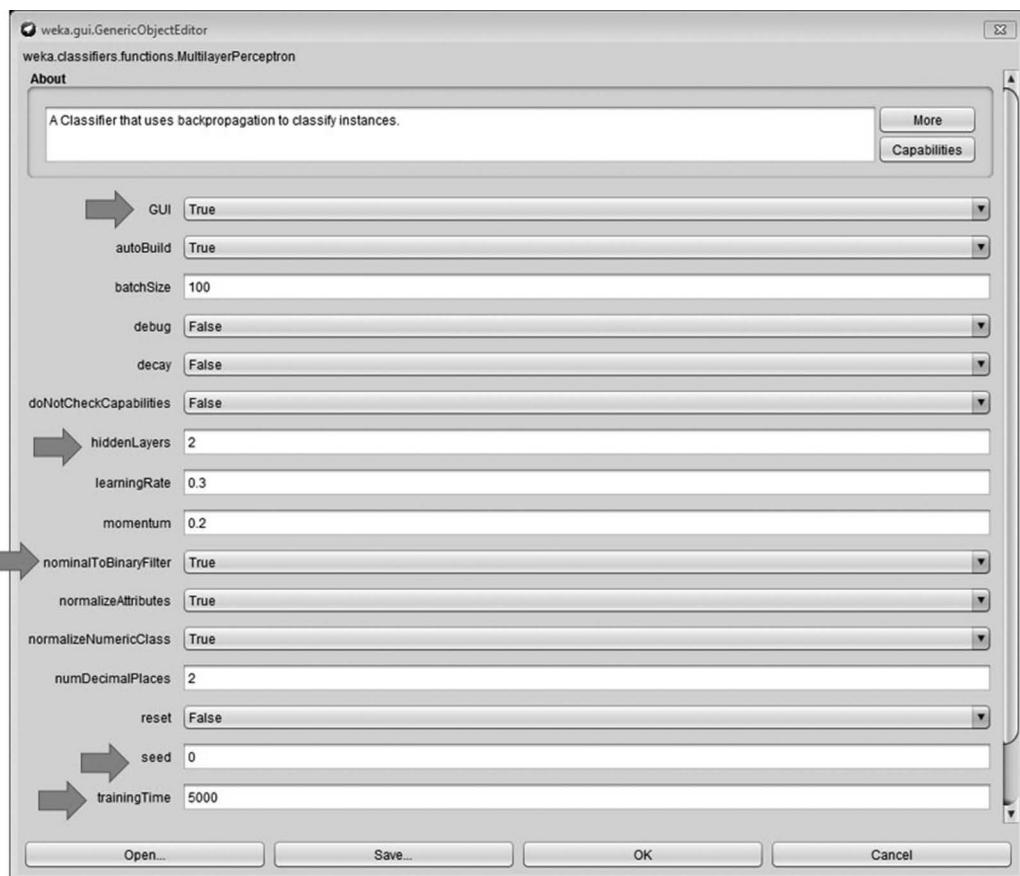


FIGURE 9.5 Backpropagation learning parameters.

parameter gives us two hidden layers, the first hidden layer having 4 nodes and the second hidden layer showing 3 nodes. Deciding on the number of hidden layers and the number of nodes within each hidden layer is both an art and a science. For our first example, give this field a value of 2 to specify one hidden layer with two nodes.

- The *nominalToBinaryFilter* creates a separate input node for each possible nominal input value. This is designed to improve performance when nominal data are present. To illustrate, if our initial attribute is *income* with nominal values *high*, *middle*, and *low*, the filter will create one input node for each of the three values. That is, the income attribute will actually generate three input nodes. If an input instance has a value of *high* for the income attribute, the input node representing *high* within the neural network will receive a value of 1. The nodes representing *middle* and *low* income receive values of 0.
- The *seed* parameter offers us the opportunity to vary the initial values for the network weights. The initial weight assignments can affect the resultant accuracy of the network.
- The *trainingTime* parameter allows us to specify the total number of passes (epochs) of the training data through the network. As we have limited the network to one hidden layer containing two nodes, the number of epochs required for an acceptable convergence could be large. To accommodate this, set the *trainingTime* parameter to 5000 and click *OK*.
- Finally, as the output is numeric we want to see specific values for the output attribute. To have the output values for each instance displayed click on *More options*, mouse to *output predictions*, choose *plain text*, and click *OK*.

Step 4: Watch the Network Train

With the parameters set, click *start* to begin network training. The graphical representation shown in Figure 9.6 that defines the network architecture is displayed. Figure 9.6 shows that the current parameter settings for *Epoch* and *Error per Epoch* are both 0. That is, although the network architecture is displayed, network training has yet to begin. Click *start* to watch the network train. As our training set contains but four instances, the training phase quickly terminates. Notice that the value for *Epoch* is now 5000, indicating that the training cycle is complete. In addition, the *Error per Epoch* is 0, indicating that the network has learned the XOR function. To continue, click *accept*.

Step 5: Read and Interpret Summary Results

Figure 9.7 displays the results of the data mining session. If you did not modify the *seed* parameter, your results will be as those seen in the figure. Interpreting this output can be confusing. Figure 9.8 shows the output weights as they would appear on the trained network. *Node 0* is the lone output node. The *node 1* and *node 2* values

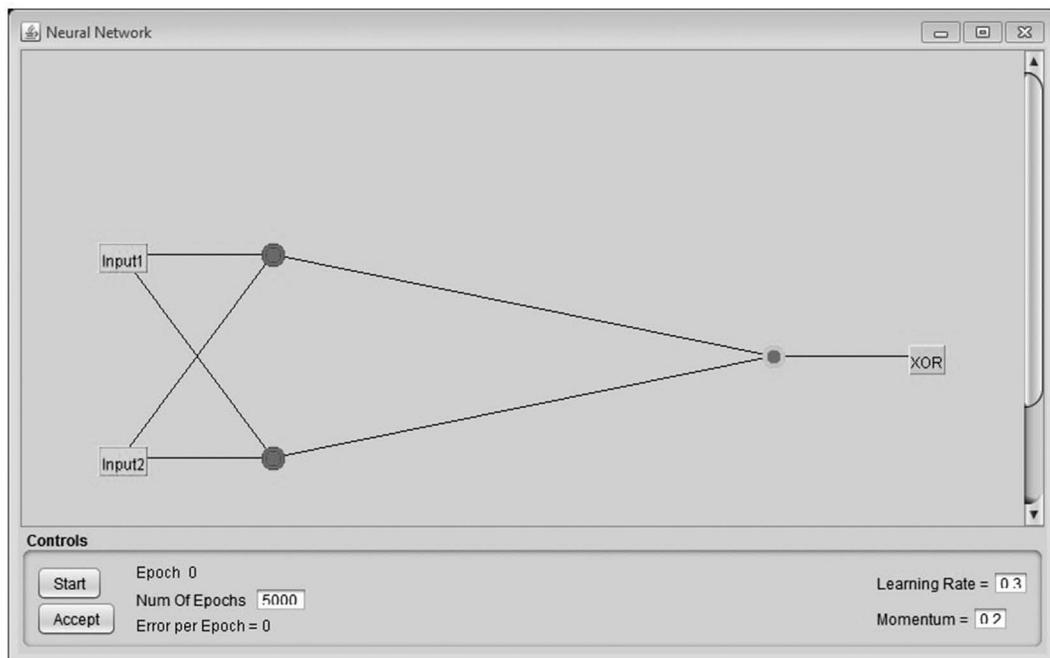


FIGURE 9.6 Architecture for the XOR function.

Linear Node 0

Inputs Weights

Threshold	1.1186668049316135
Node 1	-3.267948652302308
Node 2	3.483851281989351

Sigmoid Node 1

Inputs Weights	
Threshold	0.7295727209179403
Attrib Input1	-1.950632684944084
Attrib Input2	2.0571299630872546

Sigmoid Node 2

Inputs Weights	
Threshold	-4.519783472921987
Attrib Input1	-2.588817970900386
Attrib Input2	4.082564088088655

inst#	actual	predicted	error
1	0	0	0
2	1	1	0
3	1	1	0
4	0	0	0

FIGURE 9.7 XOR training output.

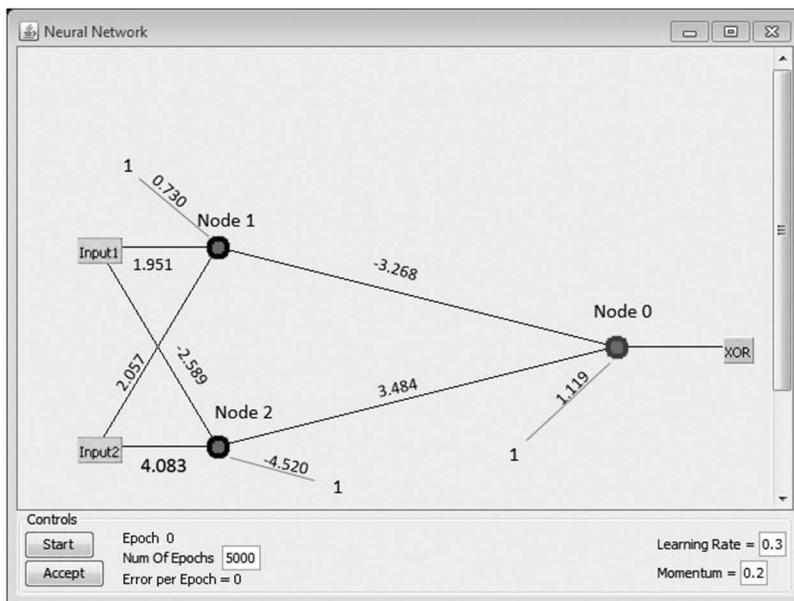


FIGURE 9.8 Network architecture with associated connection weights.

listed under the heading *Linear Node 0* represent the connection weights of *node 1* to *node 0* and *node 2* to *node 0*. Similarly, under *Sigmoid Node 1*, we see the respective connection weights between *input1* and *node 1* as well as *input2* and *node 1*. The same scheme applies for the values under *Sigmoid Node 2*.

Figure 9.8 also shows an additional weighted link associated with each node. The weights are the rounded *threshold* values shown in Figure 9.7. The threshold serves a purpose similar to the constant term in a regression equation and functions as another connection weight between two nodes. The 1 at one end of the link represents a node whose output is always 1. The threshold values change during the learning process in the same manner as all other network weights.

In the simplest terms, the *threshold* or *bias* can be thought of as a value added to the associated node. To see this, consider Figure 9.8 and *Node 1* having *threshold* = 0.730 together with the instance showing *input1* = 1 and *input2* = 0. The rounded activation value given to *Node 1* for processing by the sigmoid function will be $(1.951 * 1 + 2.057 * 0 + 0.730) = 2.681$. Similar reasoning pertains to the threshold values associated with *Node 0* and *Node 2*.

Returning to Figure 9.7, we see identical values in the actual and predicted columns. Scroll your output to see that the mean absolute error and root mean squared error are zero. As the training data represent all possible values for the XOR function, we can be assured that—given correct input values—our network will always correctly identify the value of the function. As you will see, this is the exception rather than the rule.

9.3 MODELING THE EXCLUSIVE-OR FUNCTION: CATEGORICAL OUTPUT

Having investigated the case where the output of the XOR function is listed as real, it's time to proceed to the case where the XOR output attribute is defined to be categorical. To see this, use your favorite editor to open *XORC.arff*. Notice that the XOR attribute is now defined categorically by @attribute XOR {0, 1}. This contrasts the previous example, where the *XOR.arff* file shows XOR as @attribute XOR real. Next, load *XORC.arff* into the Weka environment. Make sure that XOR is designated as the output attribute and make *use training set* the test option.

For the first experiment, let's verify that MLP without a hidden layer cannot learn the XOR function. In this case, we are defining a simple perceptron network. To do this, we click on the command line to once again display the ObjectEditor parameters given in Figure 9.5. Be sure to set the *GUI* parameter to *true*. Change the value of *hiddenLayers* to 0. Change the *trainingTime* value to 50,000. This should provide ample opportunity for the algorithm to converge. Once these changes are made, click *start* to observe the network configuration.

The configuration is given in Figure 9.9. Click *start* to begin training. Notice that after 50,000 epochs, the network still shows an error value greater than 0.25. Click *accept*. The output is displayed in Figure 9.10. As the output attribute is categorical, we now have a confusion matrix. It is clear that all four instances have been classified as belonging to the class whose XOR output is 1. This gives us a classification correctness of only 50%. This outcome clearly supports the need for a hidden layer to model the XOR function.

For our second experiment with the *XORC* file, let's revisit steps 3 through 5 of our five-step process. First, highlight the *Use training set* radio button. Next, make sure the *GUI* parameter is set to *true*. Set the *hiddenLayers* parameter to 4 and the *trainingTime* value to 2000. Click *start* to see the network architecture shown in Figure 9.11. Notice that we now have two output nodes, one for class 1 and the second for class 2. Click *start* to begin

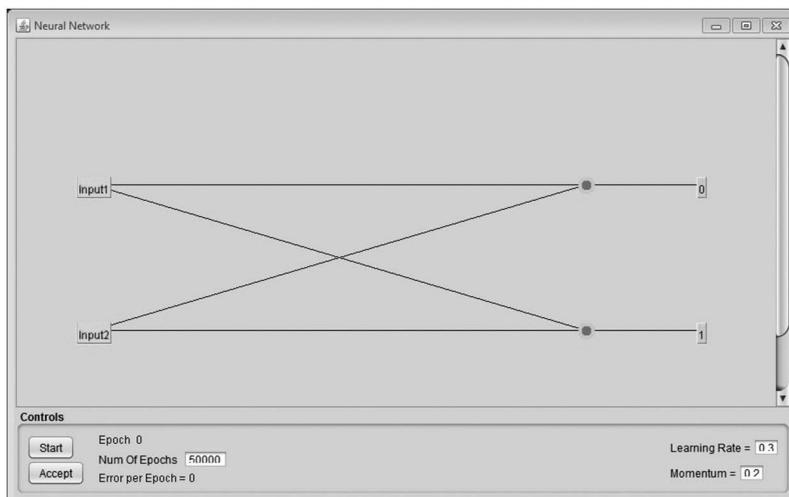


FIGURE 9.9 XOR network architecture without a hidden layer.

Root mean squared error	0.5001
Relative absolute error	99.9999 %
Root relative squared error	100.021 %
Total Number of Instances	4

==== Detailed Accuracy By Class ====

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.000	0.000	0.000	0.000	0.000	0.000	0.500	0.750	0
1.000	1.000	0.500	1.000	0.667	0.000	0.500	0.583	1
Weighted Avg.	0.500	0.500	0.250	0.500	0.333	0.000	0.500	0.667

==== Confusion Matrix ====

a b <-- classified as
 0 2 | a = 0
 0 2 | b = 1

FIGURE 9.10 Confusion matrix for XOR without a hidden layer.

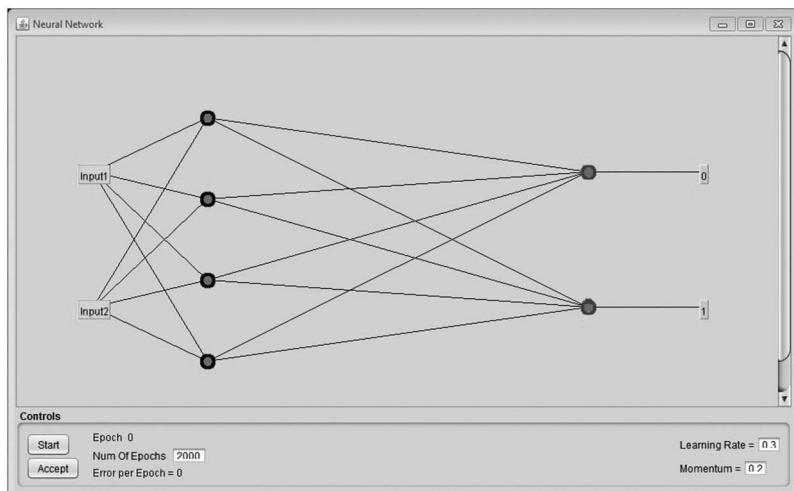


FIGURE 9.11 XOR with hidden layer and categorical output.

training. The output shown in Figure 9.12 tells us that the network correctly identifies the XOR operator. Notice that in this case, the *mean absolute error* (mae) and rms are both acceptable values, but they are not 0. Trying to obtain values of 0 for mae and rms may be an exercise in futility. To see this, let's bump the *hiddenLayers* parameter to 10 and the *trainingTime* to 100,000. This gives us two hidden layers each of size 10. The large number of epochs allows us to actually watch the epoch count change value. Train the network and examine the output. In fact, these modifications make matters worse! This emphasizes the fact that picking appropriate values for the parameters is of critical importance. It is

Correctly Classified Instances	4	100	%
Incorrectly Classified Instances	0	0	%
Kappa statistic	1		
Mean absolute error	0.0782		
Root mean squared error	0.0811		
Relative absolute error	15.6429 %		
Root relative squared error	16.2198 %		
Total Number of Instances	4		

==== Detailed Accuracy By Class ====

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
0	1.000	0.000	1.000	1.000	1.000	1.000	1.000	0
1	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000

==== Confusion Matrix ====

a b <-- classified as

2 0 | a = 0

0 2 | b = 1

FIGURE 9.12 XOR confusion matrix and categorical output.

oftentimes the case that fewer in terms of hidden layers and hidden-layer nodes gives a better result.

9.4 MINING SATELLITE IMAGE DATA

It's time to move to the 15-class satellite image data set to gain additional insight into training neural networks. The satellite image data set is represented by 300 instances and the 15 classes shown in Figure 9.3. Once again, we follow the five-step procedure for back-propagation learning.

Step 1: Identify the Goal

Our goal is to build a neural network model that can be used to monitor land cover changes in the region defined by the data set. Once a network architecture is determined to be acceptable, the model will be used to monitor for significant changes in the specified region. We will accept a model that shows a test set accuracy greater than 95%. Upon achieving the desired accuracy, the entire data set will be used to build the final model.

Step 2: Prepare the Data

Open Weka's Explorer, and while in preprocess mode, load *Sonar.arff*. Be sure that the *class* attribute is designated as *class*. With the exception of *turf_grass* with

21 instances and deep_water with 19 instances, all classes have 20 instances evenly split between the first and second half of the data set. This makes a case for using half of the data for training and the remaining half for testing.

When building neural network models, it is important to understand the importance of attribute selection. This is the case as, unlike models such as decision trees where attribute selection is part of the modeling algorithm, neural network algorithms are unable to determine *a priori* the importance of any given input attribute. Therefore, it is our job to use the preprocessing phase to make appropriate input attribute selections. In our first two examples, attribute preprocessing was trivial as the XOR is a well-defined function where all instances and attributes are relevant. However, with the satellite image data, this is not the case. On the other hand, neural network algorithms are quite resilient in that they are oftentimes able to build useful models even when attribute selection is not optimal. As we saw in Chapter 4, Weka has several filtering algorithms that deal with attribute selection for supervised learner models. We also saw that an effective filtering technique is to use InfoGainAttributeEval as the Attribute Evaluator together with Ranker as the search method. In exercise 3, you are asked to experiment with this data set and InfoGainAttributeEval to determine if any of the input attributes should be eliminated. For our example, we use all six input attributes.

Step 3: Define the Network Architecture

Move from preprocess mode to classify mode. Next, click on *choose*, then *functions*, and finally, *multilayerperceptron*. Be sure to highlight the *Percentage Split* testing option. Insert 50 as the value for percentage split. This divides the data set so that 150 instances are used for training and the remaining 150 are used for testing. Open the command line options screen. As an initial experiment, set *GUI* to *true*, *hidden-Layers* to 0, and *trainingTime* to 5000.

Step 4: Watch the Network Train

Given 15 classes and only 150 training instances, a natural expectation is a low test data classification correctness. Let's find out. Click *start* to see the network architecture given in Figure 9.13. Click *start* to begin network training. When training is complete, click *accept*. Notice that once again, we see the GUI! The reason for this is the interactive nature of the GUI. Although the network has been trained, it has yet to be tested on the remaining 150 instances. We must again click *start*, wait for testing to complete, and click *accept*.

Step 5: Read and Interpret Summary Results

Scrolling the output, we see a 92.67% classification correctness on the test data! This is a strong indication that within the context of the six input dimensions, the data are almost linearly separable. Figure 9.14 offers the resultant confusion matrix. The matrix tells us that 8 of the 11 misclassifications were incorrectly classified marsh instances.

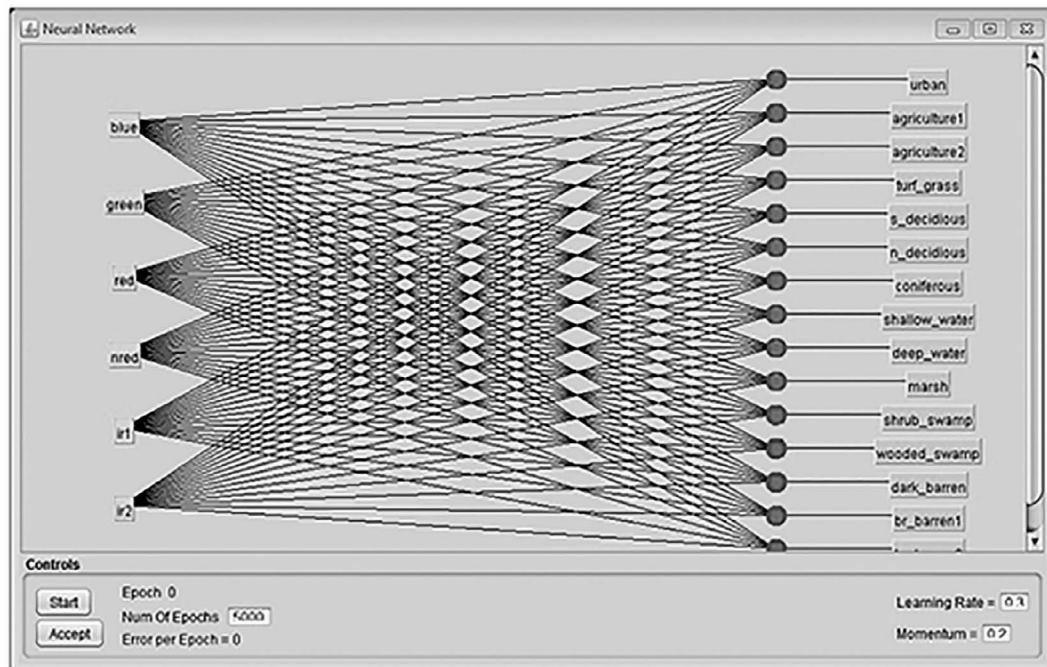


FIGURE 9.13 Satellite image data network architecture.

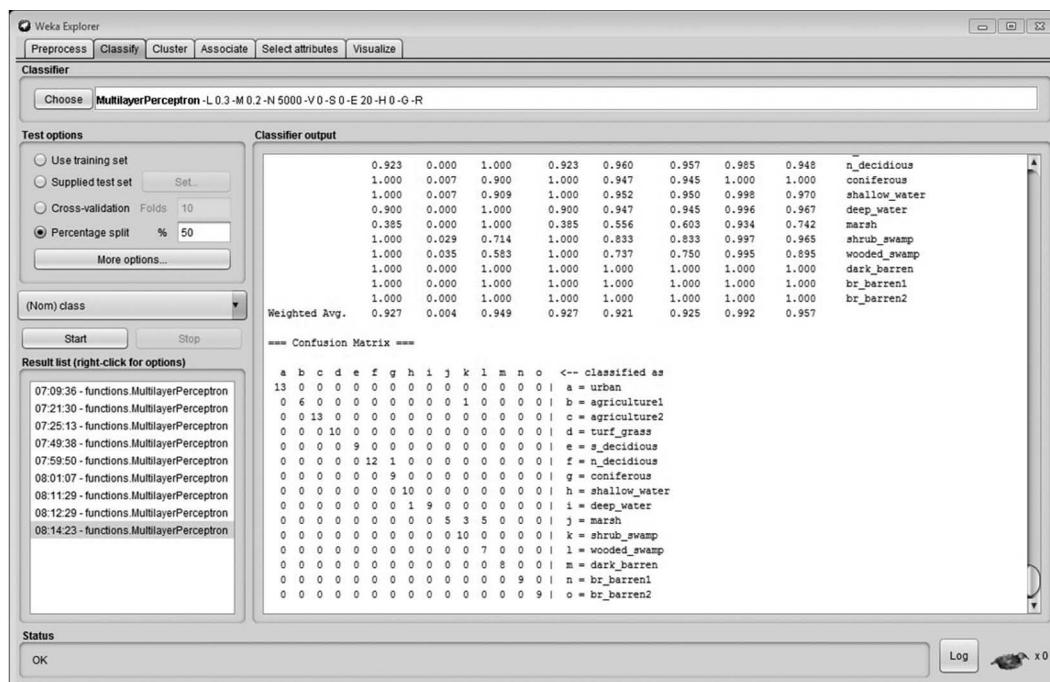


FIGURE 9.14 Confusion matrix for satellite image data.

We can do even better by adding a single hidden layer of 10 nodes. Doing so results in a 96% test set correctness with all but six instances and all class marsh instances correctly classified. These results offer strong support for our goal.

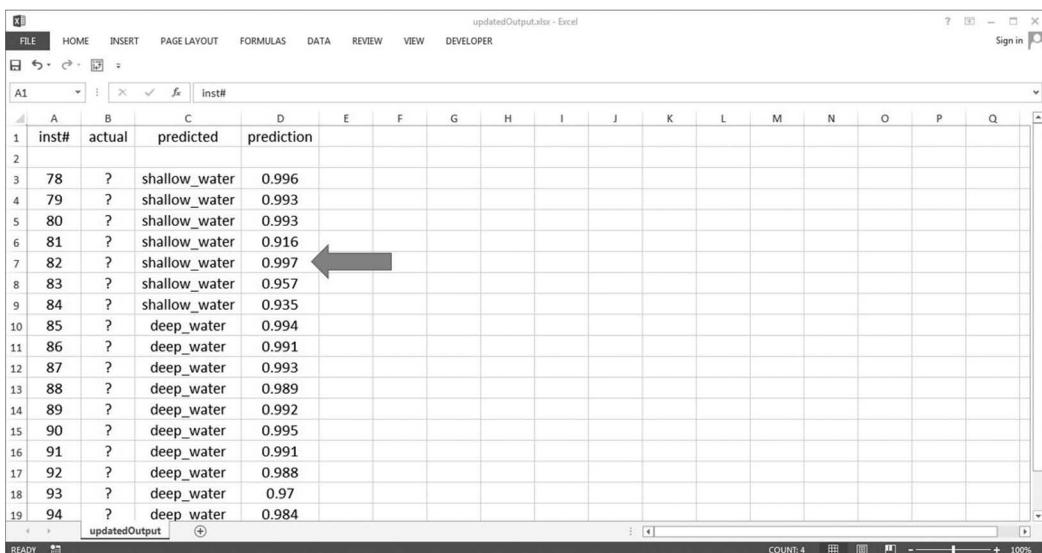
Given that we are satisfied that we have achieved our goal of creating a good MLP model architecture for classifying land cover, the next step is to use all of the data to create and save a final model. This model will be used in the future to classify the same regions after specified time periods in order to determine if the land cover of any of the regions has changed. Let's review the process. We must first create and save the final model.

- Use all of the instances within *Sonar.arff* to create a final model—your Test option should be *Use training set*. Prior to training, left-click on *More Options* and check the box for *Output Model*.
- When the final model is created, it is listed in the section titled *Result list (right-click for options)*. The model has the name *functions.MultiLayerPerceptron*. Right-click on the model name.
- Left-click on *Save Model* to save the model to a desired destination folder. The model is saved as a Java object file with a .model extension.
- Close Weka.

We can now use the saved model to classify unknown data. To do this, we use the file *SonarMissingClass.arff*. Our assumption here is that this file contains a new satellite image of the same region with the class attribute showing a question mark rather than the name of the class. We use this file to illustrate the process of applying the saved model to unclassified data. Let's see if there are any changes in ground cover within this new image!

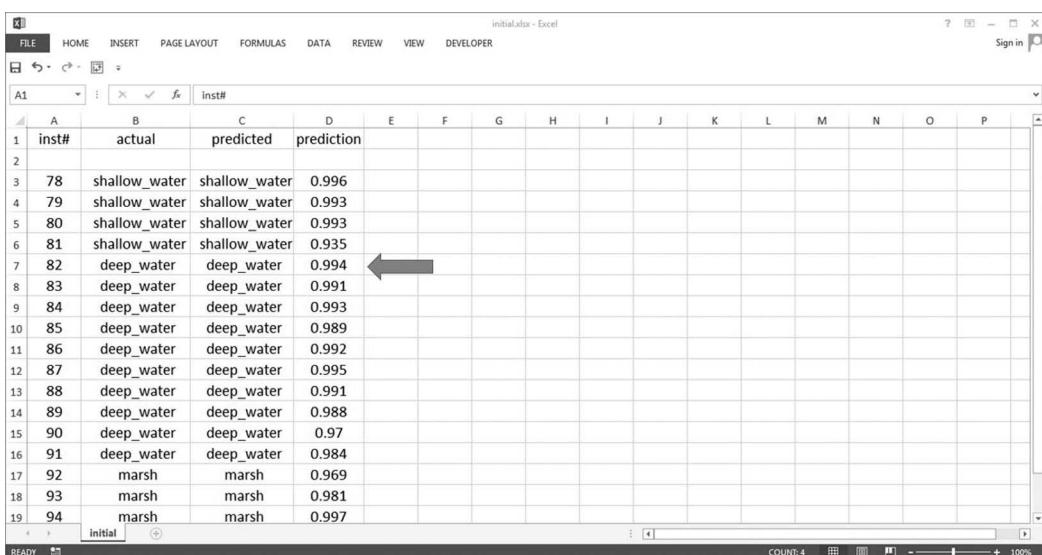
- Open Weka's Explorer and load *SonarMissingClass.arff*.
- Move from preprocess mode to classify mode.
- Right-click in the white space area of the *Result-List (right-click for options)*, choose the *Load Model* option. Find the previously saved model. Load your file but first be sure to add the .model extension.
- Click on *more options*. Click *Choose* to the right of *output predictions* and select *plain text*. Click *OK*.
- Under *Test options*, highlight the *Supplied Test set* radio button, click on *set* and then on *open file*, and type *SonarMissingClass.arff* in the specified location. Be sure to add the file extension!
- Mouse back to the *Result-List* area and right-click on the model name. Click on *Re-evaluate model on current test set* to obtain the classifications. Scroll the output window to see the results. The resultant output buffer can be saved for further investigation with a right-click on the model name and mousing to *save result buffer*.

Scroll your output until it shows the instances in Figure 9.15. For each instance, we are given a predicted class together with a prediction probability. Figure 9.16 shows the original classification for these instances. Notice that instances 82 through 84, which were once deep-water areas, are now classified as belonging to the shallow-water class. This situation requires further investigation as it could be due to excessive irrigation, lack of rain, or even changes in the underlying structure of the earth.



inst#	actual	predicted	prediction
78	?	shallow_water	0.996
79	?	shallow_water	0.993
80	?	shallow_water	0.993
81	?	shallow_water	0.916
82	?	shallow_water	0.997
83	?	shallow_water	0.957
84	?	shallow_water	0.935
85	?	deep_water	0.994
86	?	deep_water	0.991
87	?	deep_water	0.993
88	?	deep_water	0.989
89	?	deep_water	0.992
90	?	deep_water	0.995
91	?	deep_water	0.991
92	?	deep_water	0.988
93	?	deep_water	0.97
94	?	deep_water	0.984

FIGURE 9.15 Updated class assignment for instances 78 through 94 of the satellite image data set.



inst#	actual	predicted	prediction
78	shallow_water	shallow_water	0.996
79	shallow_water	shallow_water	0.993
80	shallow_water	shallow_water	0.993
81	shallow_water	shallow_water	0.935
82	deep_water	deep_water	0.994
83	deep_water	deep_water	0.991
84	deep_water	deep_water	0.993
85	deep_water	deep_water	0.989
86	deep_water	deep_water	0.992
87	deep_water	deep_water	0.995
88	deep_water	deep_water	0.991
89	deep_water	deep_water	0.988
90	deep_water	deep_water	0.97
91	deep_water	deep_water	0.984
92	marsh	marsh	0.969
93	marsh	marsh	0.981
94	marsh	marsh	0.997

FIGURE 9.16 Initial classification for pixel instances 78 through 94 of the satellite image data set.

Finally, Chapter 4 offered an alternative to the aforementioned technique whereby all of the data are contained in one file. Recall that the downside to this technique is that the model must be recreated using the same parameters each time before it is used.

9.5 UNSUPERVISED NEURAL NET CLUSTERING

SelfOrganizingMap is Weka's implementation of the Kohonen unsupervised clustering algorithm presented in Chapter 8. As *SelfOrganizingMap* is not part of Weka's core installation package, it must be installed. This is easily accomplished by following the procedure explained in Appendix A. Once installed, we can use *SelfOrganizingMap* to cluster the instances of the diabetes data set described in Chapter 6. Let's get started!

Recall that the diabetes data set contains 768 instances, 268 of which represent individuals who tested positive for diabetes. The data include eight numeric input attributes and a nominal output attribute indicating the outcome of a test for diabetes. Let's use the *Classes to clusters* option to help determine how well the input attributes differentiate the two classes. Here is the process:

- Load the *diabetes.arff* data set into Weka's Explorer.
- Click on Cluster (arrow 1, Figure 9.17) then on *Choose* to locate and load *SelfOrganizingMap*.
- Highlight the *Classes to clusters evaluation* radio button (arrow 2, Figure 9.17).
- Click anywhere on the *Choose* command line to invoke the *GenericObjectEditor* (arrow 3, Figure 9.17).

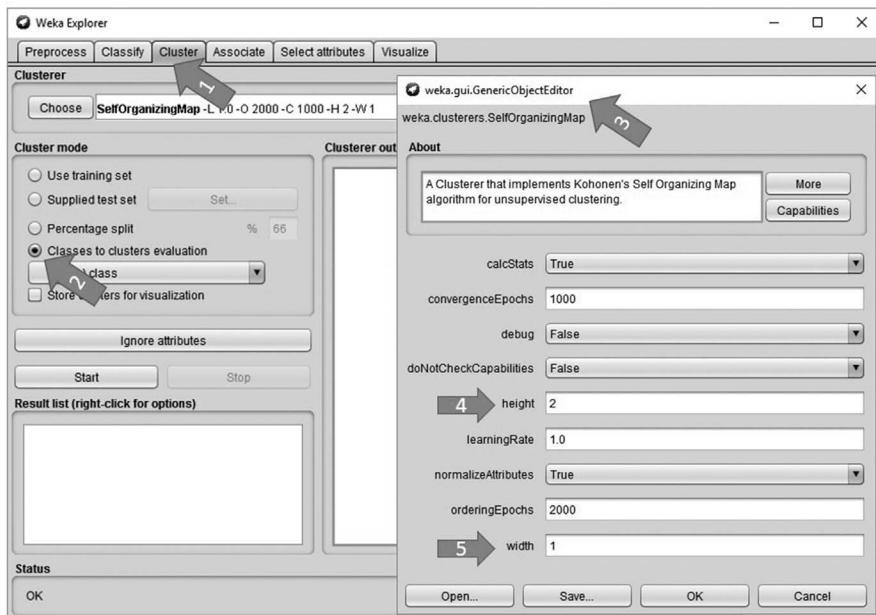


FIGURE 9.17 Parameter settings for Weka's *SelfOrganizingMap*.

```

Figure 9 - WordPad
File Home View
Courier New 16 A A
B I U X X A A
Font Paragraph Insert Editing
Picture Paint Date and time Insert object Find Replace Select all
Clustered Instances
0 520 ( 68%)
1 248 ( 32%)

Class attribute: class
Classes to Cluster?
0 1 but clustered
382 118 | assigned to cluster tested_negative
138 130 | tested_positive

Cluster 0 <-- tested_negative
Cluster 1 <-- tested_positive

Incorrectly clustered instances : 256.0

```

FIGURE 9.18 Applying Weka’s *SelfOrganizingMap* to the diabetes data set.

The parameters are displayed in the right portion of Figure 9.17. You can obtain information about all of the parameters with a click on Capabilities. We are specifically interested in height (arrow 4) and width (arrow 5). The values for these parameters specify the final dimensions of the Kohonen map and therefore determine the total number of clusters. A setting of *height* = 2 and *width* = 2, gives us four clusters.

- As our experiment calls for two clusters, set *height* = 2 and *width* = 1. Click *start*.

The first part of the output displays within-cluster mean and standard deviation values for each attribute. Scroll your output until it appears as in Figure 9.18. The bottom of the figure tells us that there are 256 incorrectly clustered instances.

The 520-instance cluster represents the negative test class as the majority of its instances showed a negative test result. The 248-instance cluster denotes the positive test class as 130 of its 248 instances tested positive for diabetes. Over 47% (118) of the instances in the positive cluster showed a negative test result. Further, over 50% (138) of the positive instances clustered with the negative class. This result suggests the need for further experimentation in order to determine if these attributes are a viable option for differentiating the classes. End of chapter exercise 10 asks you to perform several additional experiments with this data set.

9.6 CHAPTER SUMMARY

Backpropagation learning can be implemented as a five-step procedure that consists of identifying the goal, preparing the data, defining the network architecture, watching the network train, and interpreting summary results. The Weka MLP neural network tool allows us to build supervised models for estimation, classification, and prediction. MLP processes both numeric and categorical input attributes. In the case of a categorical attribute, the network creates a separate input node for each possible input value. Output attributes can be either categorical or numeric. To design the network architecture, we must

decide on the number of hidden layers, the number of nodes within each hidden layer, and the number of training epochs. Several additional parameters can be adjusted to help us build accurate supervised neural network models. End-of-chapter exercises 4 and 5 are of particular interest as they emphasize the importance of attribute selection prior to neural network learning. Weka's *SelfOrganizingMap* algorithm is based on the two-layer Kohonen unsupervised clustering architecture. As with MLP, *SelfOrganizingMap* can be used to cluster data sets having missing values and accepts both categorical and numeric attribute types.

9.7 KEY TERMS

- *Linearly separable*. Two classes, A and B, are said to be linearly separable if a straight line can be drawn to separate the instances of class A from the instances of class B.
- *Perceptron neural network*. A simple feed-forward neural network architecture consisting of an input layer and a single output layer.

EXERCISES

1. Create a backpropagation network to model one or more of the three logical operators shown in the following table. Because these operators are linearly separable, you might hypothesize that unlike the XOR operator, we don't need to use a hidden layer as part of the network architecture. Run experiments to affirm or reject this hypothesis. The simplest method for creating your ARFF file(s) is to use Notepad or WordPad to modify *XORC.arff*. Change the name in the @relation line and modify the instance values to reflect the chosen logical operator. Save the modified file with a new name, load the file into Weka, mouse to *classify*, and highlight *Use training set* as the Test Option.

Input	Input 2	And	Or	Implication
1	1	1	1	1
0	1	0	1	1
1	0	0	1	0
0	0	0	0	1

2. Create an ARFF file that represents the logical complement (negation) operator. The complement operator takes a single input value, and outputs 1 when the input is 0 and 0 when the input is 1. When you create your file, be sure to specify the output attribute as categorical. Use MLP to build a backpropagation network to model logical complement. Denote *Use training set* as the Test Option. How many epochs are needed to train the network without a hidden layer? If you add a hidden layer, does it take fewer epochs to train the network?
3. For this exercise, you will use the credit card promotion database *CreditCardPromotion.arff* and backpropagation learning to build a model for estimating the likelihood of a new credit card customer accepting a life insurance promotion. Make the assumption

that the model built with the training data has been previously tested. Here is the procedure:

- a. Copy the *CreditCardPromotion.arff* file. Give the new file the name *ccpromo.arff*.
- b. Add the six instances listed in the following table representing new customers to the end of *ccpromo.arff*. As these instances represent new customers, use question marks to specify the cells for *magazine promotion*, *watch promotion*, and *life insurance promotion*.
- c. Load the modified file into the Explorer. As the model is for predictive purposes, use *remove* to delete the *magazine promotion* and *watch promotion* attributes.
- d. Move to *classify* and choose *MultiLayerPerceptron*. Set the parameters for epochs at 5000. Allow default values for all other parameters. Specify *Use training set* as the Test Option. Recall that only those instances that have a known value for the output attribute will be used for training.
- e. Make *LifeInsPromo* the output attribute. Go to *more options* and Choose *Plain text* for *Output predictions*.
- f. Report the predicted *lifeInsPromo* value and prediction confidence for each of the six previously unknown instances.

Income Range	Magazine Promotion	Watch Promotion	Life Ins Promo	Credit Card Ins	Gender	Age
30–40K	?	?	?	No	Female	33
40–50K	?	?	?	Yes	Male	42
20–30K	?	?	?	No	Female	19
30–40K	?	?	?	No	Male	50
40–50K	?	?	?	Yes	Female	48
30–40K	?	?	?	No	Female	31

4. Apply InfoGainAttributeEval as the evaluator together with the Ranker search technique to the *Sonar.arff* data set to determine the effect that attribute elimination has on MLP's classification correctness. Make a table showing the classification accuracy of models built with the best five input attributes followed by the best four attributes down to the best single attribute. Use a twofold cross-validation for each experiment together with an epoch value of 5000. Allow default values for all other parameters. Compare your results with those seen when all six input attributes are employed. Include in your analysis whether a statistically significant difference is seen between the classification correctness of each model when compared to the accuracy of the model using all six input attributes.
5. Use InfoGainAttributeEval and the Ranker search technique together with the *Spam.arff* data set to determine the effect attribute elimination has on MLP's classification correctness. First, use 10-fold cross-validation with the entire data set and record the

classification accuracy. After this, apply the Ranker search technique and delete all but the top 10 attributes. Finally, use the five best attributes to obtain a third classification correctness value. Is there a significant difference in classification accuracy between each result? Summarize your findings. Be sure to comment on differences between the percentages of type 1 and type 2 errors seen with each model. Make a general statement about what you have discovered.

6. Replace J48 with MLP and work the examples given in Chapter 4 Section 4.6 that show how to use Weka's cost/benefit analysis tool.
7. Use the *customer-churn-data-KN.arff* data set and follow the procedure described in Section 9.4 to create and save a neural network model for this data set. Apply the saved model to the *customer-churn-data-UNK.arff* file. Be sure to display the prediction probabilities.
8. Load an ARFF data set of your choice for neural net learning with MLP. Once *Multilayer Perceptron* has been selected, click the *choose* panel to bring up the generic object editor. Experiment with different values for the *learningRate* and *momentum* parameters. Report your results.
9. Use a data set of your choice to create a neural network and experiment with adding and deleting network nodes. Summarize your findings.
10. For this exercise, you will use Weka's SelfOrganizingMap to help evaluate the input attributes of the *diabetes.arff* data set. Here is the technique:
 - a. While in preprocess mode, use InfoGainAttributeEval and Ranker to rank the attributes of the *diabetes.arff* data set in terms of their predictive value.
 - b. Eliminate the least predictive attribute and designate Cluster mode as Classes to clusters evaluation.
 - c. Cluster the instances. Record the number and percent of incorrectly clustered instances as well as the number of positive instances that clustered with the negative cluster.
 - d. Eliminate the least predictive attribute and repeat (c) until the final clustering uses the single most predictive attribute.
 - e. Use a 10-fold cross-validation together with J48 to classify the instances using all eight input attributes. Record the error rate and the number of positive instances classified with the negative class. Classify the instances a second time, but this time use only those input attributes associated with the best clustering.
 - f. Use equation 7.5 to determine if there is a significant difference between the error rates obtained in (e).



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Building Neural Networks with RapidMiner

CHAPTER OBJECTIVES

- *Perform supervised neural network learning with RapidMiner’s Neural Net operator*
- *Understand that the Neural Net operator requires numeric input attributes and nominal output*
- *Understand that missing attribute values must be resolved prior to neural network learning*
- *Know that nominal input attributes must be transformed into real-valued equivalents prior to neural network learning*
- *Know how to build, save, and apply a neural net model to data whose outcome is unknown*
- *Perform unsupervised clustering with RapidMiner’s Self-Organizing Map Operator*

In this chapter, we introduce RapidMiner’s *Neural Net* operator for supervised learning and RapidMiner’s *Self-Organizing Map* (SOM) operator for unsupervised clustering. The workflow-based nature of RapidMiner gives us a great deal of flexibility in our experiments with these operators. For our first two experiments, we use the data sets defined in Section 9.12 of Chapter 9. In Section 10.1, we use the *Neural Net* operator to model the exclusive-OR function. In Section 10.2, the *Neural Net* operator together with cross-validation is applied to satellite image data. In Section 10.3, we build, save, and apply a neural net model designed to predict customer churn. In Section 10.4, we use SOM to cluster the instances of the cardiology patient data set.

Before we begin, it is important to note that the output attribute for the *Neural Net* operator must be categorical. Also, the *Neural Net* and SOM operators require numeric input, and missing values are not allowed.

Here we use a simple five-step approach for supervised neural net learning.

1. Identify the goal.
2. Prepare the data to be mined.
3. Define the network architecture.
4. Train the network.
5. Read and interpret summary results.

10.1 MODELING THE EXCLUSIVE-OR FUNCTION

Step 1: Identify the Goal

Our goal is straightforward. Given two input values where each value is either a 1 or a 0, we want to build a neural network model to output the value of the XOR function defined in Figure 9.2.

Step 2: Prepare the Data

Preparing the data is a simple matter of loading the data and making sure that the attributes are correctly identified. Let's begin!

- Open RapidMiner to create, name, and save a new process.
- Load *XORFunction.xlsx* into your local repository and move a copy of the file into your main process area.
- If needed, use the *Set Role* operator to check and modify the role of each attribute. *target role* for the XOR attribute must be *label*. *input1* and *input2* must have a *target role* of *regular*.
- Connect the output of the *Retrieve* (or *Set Role*) operator to *res* and run your process. Highlight *Statistics* to view a summary of the data. The result is displayed in Figure 10.1.

The *Neural Net* operator uses the sigmoid function for node evaluation during training. Model testing is done with the training data as the training data contain all possible data set instances.

Step 3: Define the Network Architecture

- Return to the *Design* view.
- Drag the *Neural Net*, *Multiply*, *Apply Model*, and *Performance* (classification) operators into the main process window. Make the operator connections as shown in Figure 10.2.
- Click on the *Neural Net* operator so your screen shows the neural net parameters seen in Figure 10.2.

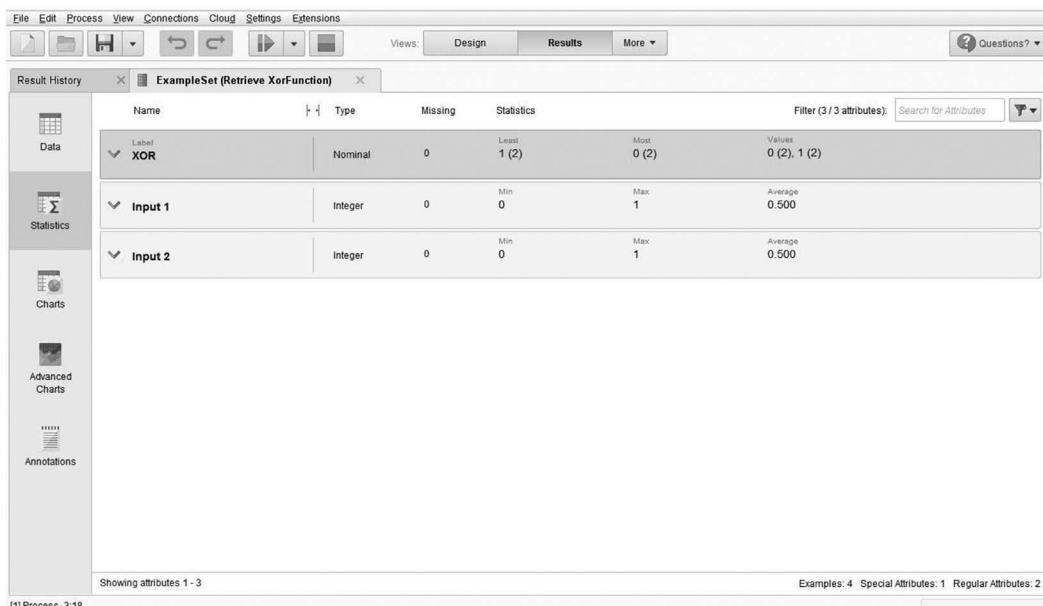


FIGURE 10.1 Statistics for the XOR function.

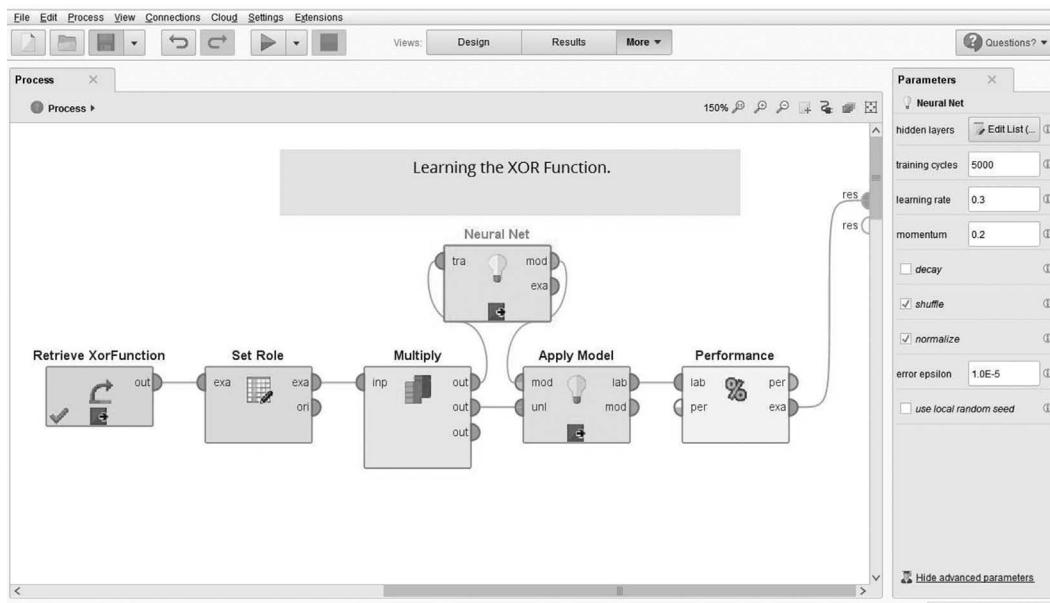


FIGURE 10.2 Main process for learning the XOR function.

Here we detail a subset of the *Neural Net* parameters that are of particular interest to us. You can learn about the other parameters by reading the synopsis given in the lower right-hand portion of your screen.

- The *hidden layers* parameter defines the structure of the network. This parameter allows us to specify the total number of nodes in each hidden layer as well as the total number of hidden layers. To see this, click the *Edit List* box associated with the *hidden layers* parameter. Your screen will appear as in Figure 10.3.

Notice that *hidden layer sizes* defaults to -1 . If unchanged, this gives us a neural network of one hidden layer. The number of nodes in this hidden layer is computed as $(\text{number of attributes} + \text{number of classes})/2 + 1$. As an alternative, we can specify one or several hidden layers as well as the size of each hidden layer. Let's do just that!

- Use the *Edit Parameter List* frame displayed in Figure 10.3 to define a single hidden layer having five nodes. Provide a hidden layer name of your choosing and set the *hidden layer sizes* value at 5. Figure 10.4 shows the result of this action where the hidden layer name is *one*.
- Click *Apply*.
- The *training cycles* parameter specifies the total number of passes (epochs) of the training data through the network. Set the *training cycles* parameter at 5000.

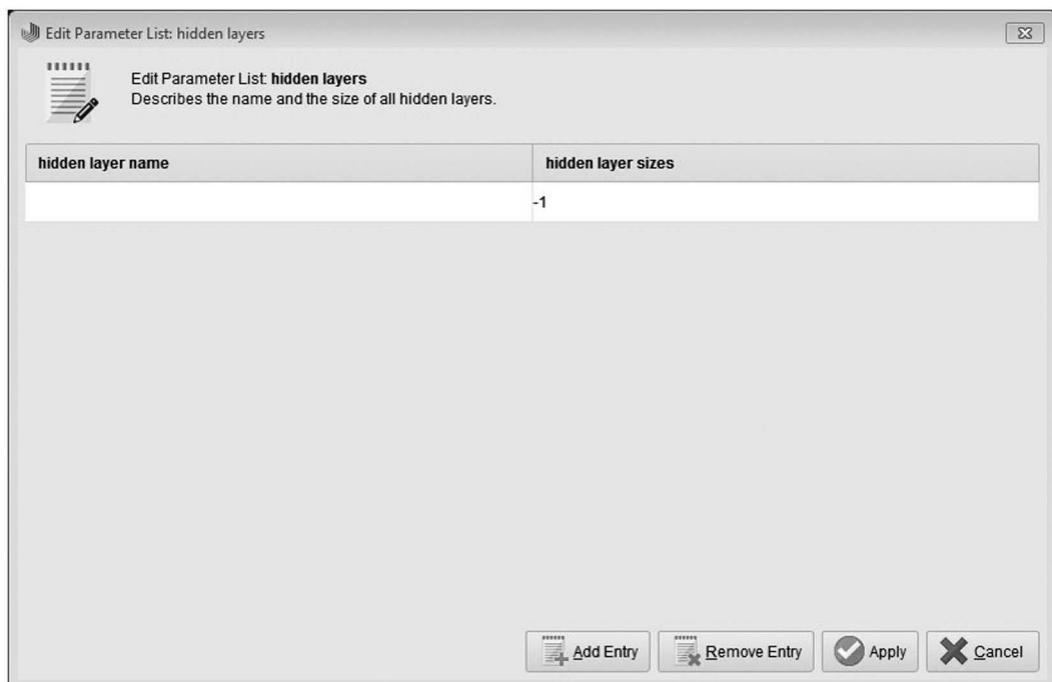


FIGURE 10.3 Default settings for the *hidden layers* parameter.

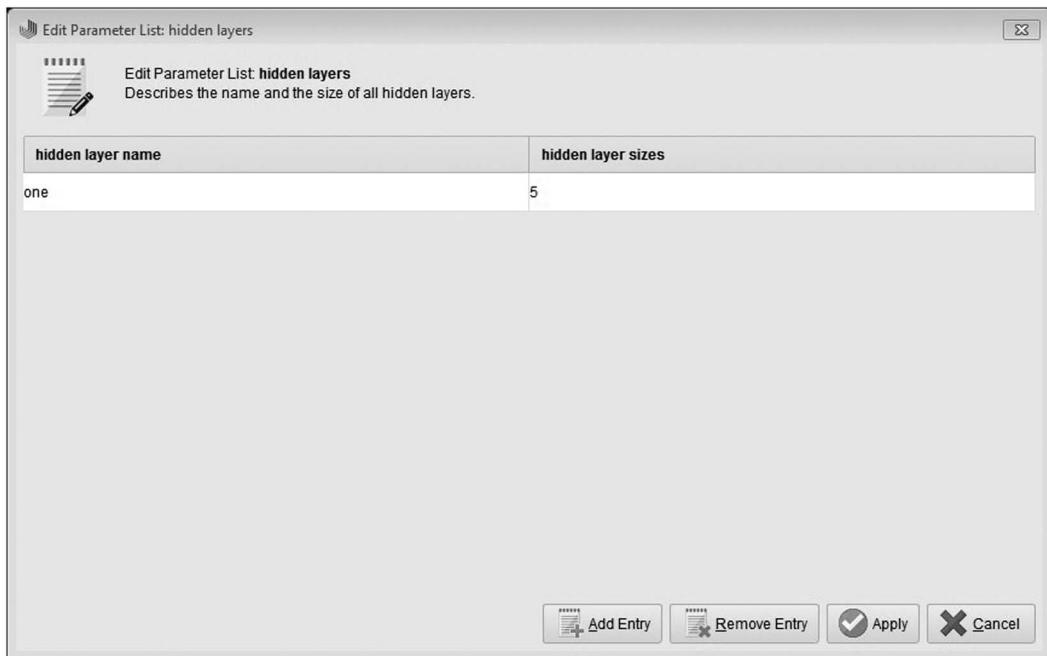


FIGURE 10.4 A single hidden layer of five nodes.

- Check the *shuffle* box parameter to have the data shuffled prior to network training. This is of particular importance when the data are sorted.
- Check the *normalize* box, which calls for the data to be scaled to a range between -1 and 1 inclusive. As the network uses the sigmoid function, this box should always be checked.
- Click on the *Performance* operator and scroll the parameters list to check the *accuracy*, *absolute error*, and *root mean squared error* parameters. Figure 10.5 gives the parameters to check.
- Set a *Breakpoint After* in the *Neural Net* and *Apply Model* operators.

Step 4: Train the Network

- Click the run process arrow to train the network.

The neural network architecture is displayed in Figure 10.6. The first thing we notice is three rather than two input-layer nodes and six rather than five hidden-layer nodes. The two extra nodes are the *bias* or *threshold* nodes described in Chapter 8. The threshold serves a purpose similar to the constant term in a regression equation and functions as another connection weight between two nodes. The input to the two threshold nodes is always 1. The threshold values change during the learning process in the same manner as all other network weights.

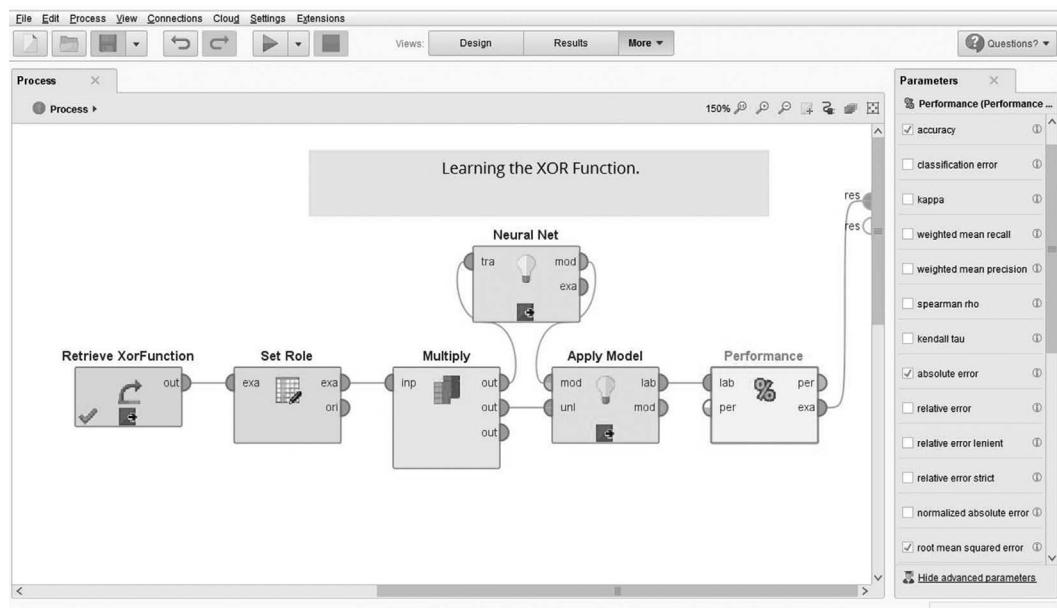


FIGURE 10.5 Performance parameters for neural network learning.

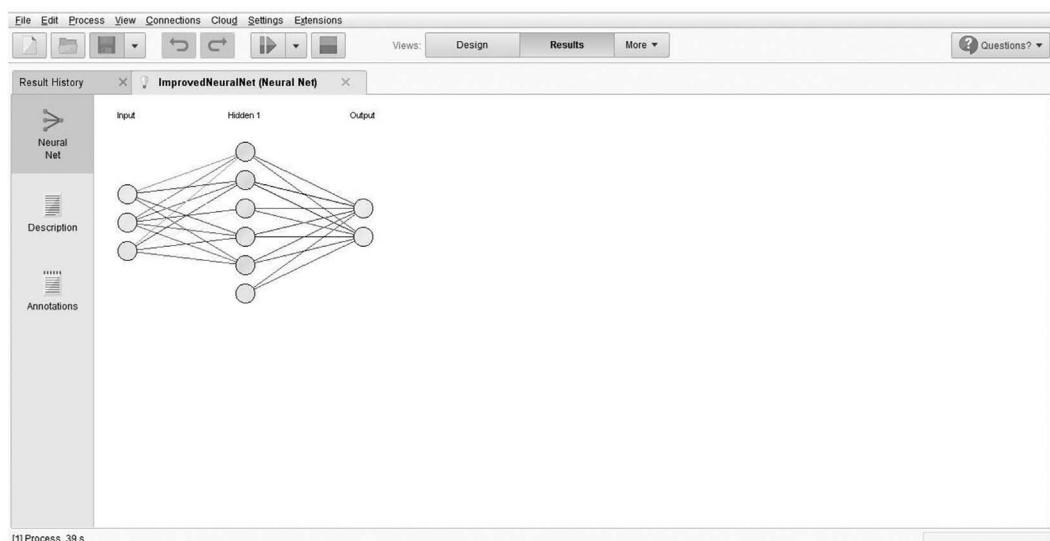


FIGURE 10.6 Neural network architecture for the XOR function.

- Click on the lower output layer node as in Figure 10.7 to see the *Hidden 1 to Output layer* connection weights for this node.
- You can obtain a listing of all connection weights by clicking on *Description* located on the left side of your process window.
- Click the run arrow to resume your process.

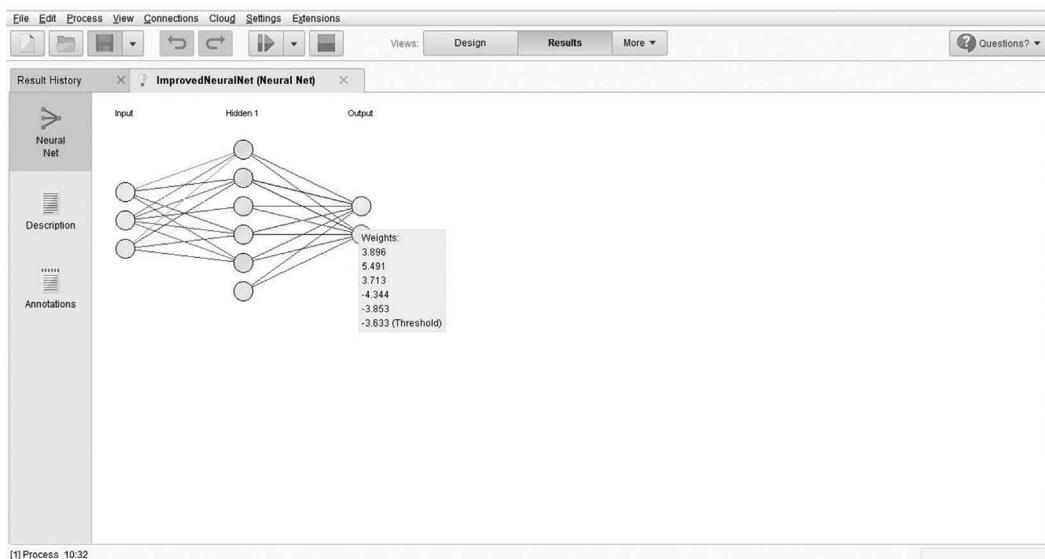


FIGURE 10.7 Hidden-to-output layer connection weights.

Step 5: Read and Interpret Summary Results

The output in Figure 10.8 shows the result of the *Breakpoint After* within the *Apply Model* operator. The figure shows actual and predicted values as well as confidence scores for each prediction. For example, when both inputs are 1, we see a predicted XOR value of 0. The associated prediction confidence is 0.967.

The screenshot shows the RapidMiner interface with the 'Results' tab selected. A table titled 'ExampleSet (Multiply)' displays the results for four examples. The columns are 'Row No.', 'XOR', 'prediction(XOR)', 'confidence(0)', 'confidence(1)', 'Input 1', and 'Input 2'. The data is as follows:

Row No.	XOR	prediction(XOR)	confidence(0)	confidence(1)	Input 1	Input 2
1	0	0	0.967	0.033	1	1
2	1	1	0.028	0.972	1	0
3	1	1	0.015	0.985	0	1
4	0	0	0.980	0.020	0	0

On the left sidebar, under the 'Data' category, there are icons for 'Statistics', 'Charts', 'Advanced Charts', and 'Annotations'. The status bar at the bottom left indicates '[1] Process 13:41'.

FIGURE 10.8 Prediction confidence values for the XOR function.

- Click the run process to display the performance vector as in Figure 10.9.

The performance vector in Figure 10.9 tells us that the network identifies the XOR operator with 100% accuracy. Recall that we also requested absolute and root mean squared error values. To see the absolute error, *highlight* absolute error in the *Criterion* window.

The result of this action is shown in Figure 10.10. The absolute error of 0.024 is negligible. It is worth noting that the error will be marginally different if the

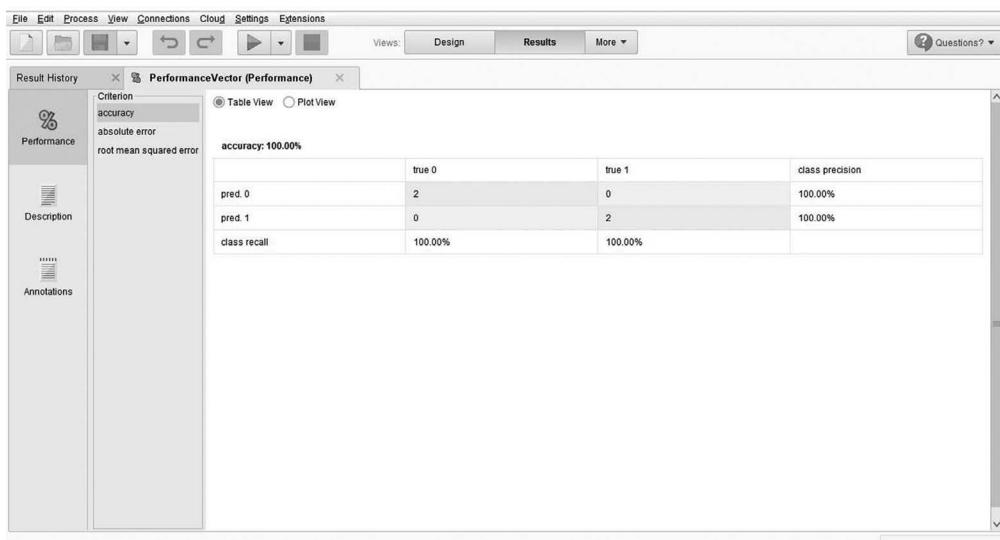


FIGURE 10.9 Performance vector for the XOR function.

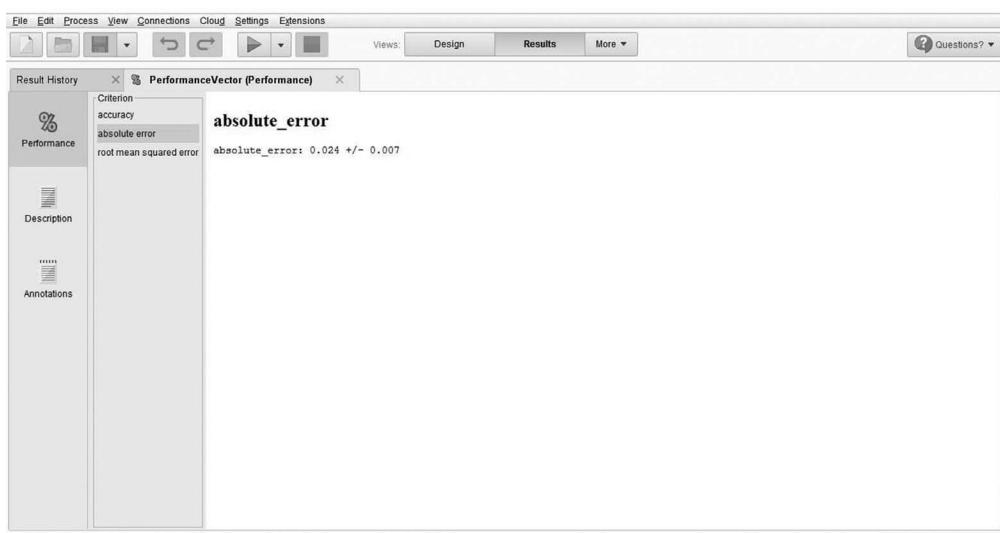


FIGURE 10.10 Absolute error value for the XOR function.

parameter *use local seed* is checked prior to network training. This parameter allows you to choose the seed for initially randomizing the connection weights. Alternative seed values will cause the network to converge with varying connection weights and differing accuracy levels. In the majority of cases, differences in model accuracy will not be significant.

Take time to experiment with alternate parameter settings especially for the training cycles and hidden layers parameters. For example, see what happens when you change *training cycles* from its current value of 5000 to 500, 1000, and finally, 1500. Furthermore, instead of one hidden layer of five nodes, try one or two hidden layers with various numbers of nodes. Do your results show that fewer hidden layers and hidden-layer nodes give a better result?

10.2 MINING SATELLITE IMAGE DATA

Let's use the knowledge gained from our first example to build a neural network model for classifying the satellite image data described in Section 9.1.2. The satellite image data set is represented by 300 instances and 15 classes.

Step 1: Identify the Goal

Our goal is to build a neural network model to monitor changes in the region defined by the data set. Once a network architecture is determined to be acceptable, the model will be used to monitor for significant changes in the specified region. We will accept a model that shows a test set accuracy greater than 95%. Upon achieving the desired accuracy, the entire data set will be used to build the final model.

Step 2: Prepare the Data

To prepare the data, the data set has been evenly divided so the first and second 150 instances contain the same number of class instances. As in this case, a twofold cross-validation is a viable choice for our experiment. Let's get started!

- Open RapidMiner to create, name, and save a new process.
- Add *Sonar.xlsx* to your repository.
- Drag *Sonar.xlsx* into the main process window. If necessary, use *Set Role* to make sure that *class* is a *label* attribute.
- Connect the output of the *Set Role* or *Retrieve Sonar* operator to *res* and run your process. Highlight *Statistics* to view a partial list of summary data as in Figure 10.11.
- Return to the *Design View*.

Step 3: Define the Network Architecture

Let's use cross-validation for estimating model performance.

- Load the *X-Validation* operator into the main process window.

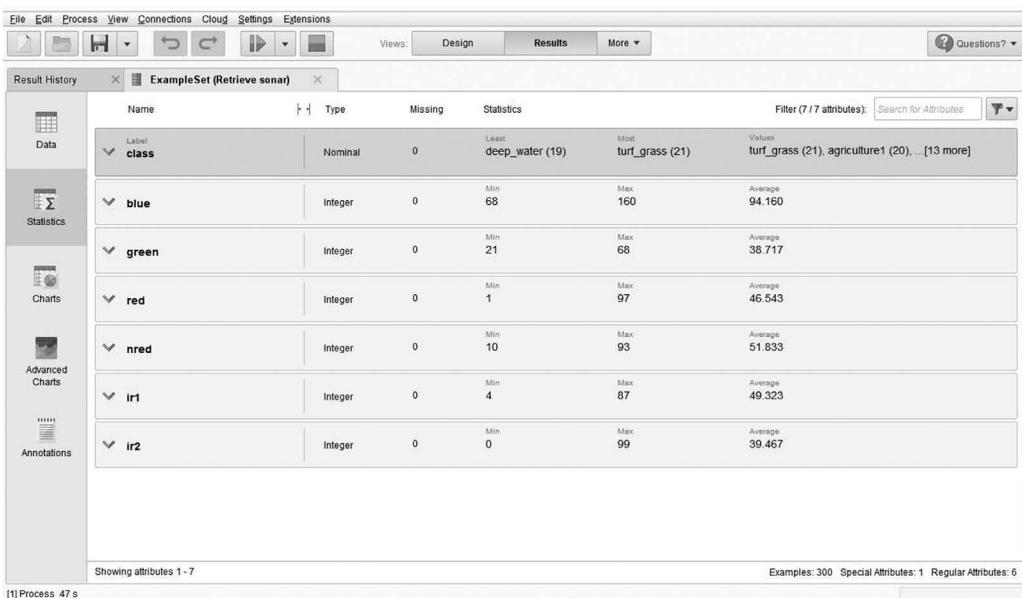


FIGURE 10.11 Attribute declarations for the satellite image data set.

- Connect the *out* port from *Retrieve Sonar* (or *Set Role*) to the *tra* port in *X-Validation*. Connect the *ave* port in *X-Validation* to *res*.
- Place a 2 in the box representing the number of validations. Make the *sampling type* linear sampling as this will partition the data without changing its order. Figure 10.12 displays the main process window.

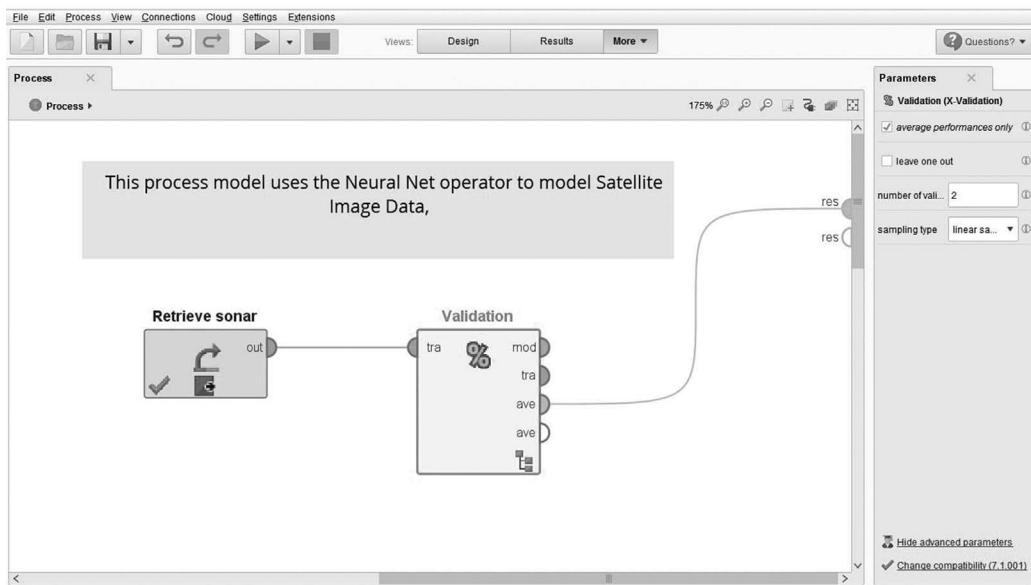


FIGURE 10.12 Main process for mining the satellite image data set.

- Double click the *X-Validation* operator to display the training and testing subprocess windows.
- Load the *Neural Net* operator into the training subprocess window. Use the default parameter settings and add a *Breakpoint After*.
- Load the *Apply Model* and *Performance* (classification) operators. Within the *Performance* operator, check *accuracy* and *absolute error*. Make all appropriate connections. Your display should appear as in Figure 10.13.

Step 4: Train the Network

- Click the run process arrow to invoke the *Breakpoint After* within the neural net operator. This presents the network structure displayed in Figure 10.14.

Notice that the number of hidden-layer nodes defaulted to 12 regular nodes $[(7 + 15)/2 + 1]$ plus the threshold node.

- Resume your process. As we are using twofold cross-validation, the network architecture is displayed a second time.
- Resume your process to see the performance vector in Figure 10.15.

Step 5: Read and Interpret Summary Results

The performance vector shows an impressive result given that it took only 500 training iterations to achieve a classification accuracy above 95%. This result is a strong indication that the data set is very well defined. However, knowing that neural

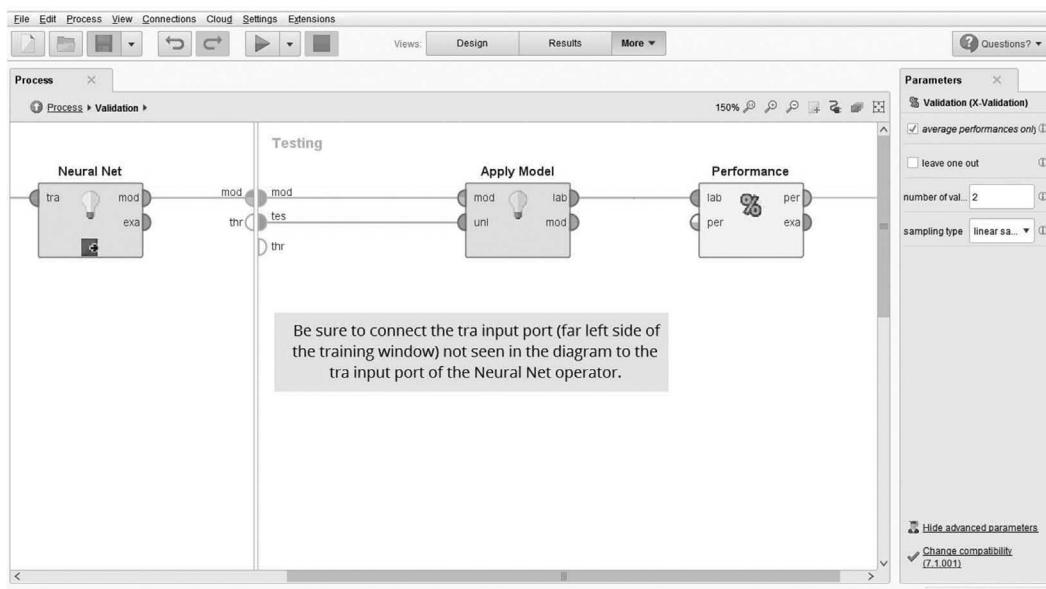


FIGURE 10.13 Subprocesses for satellite image data set.

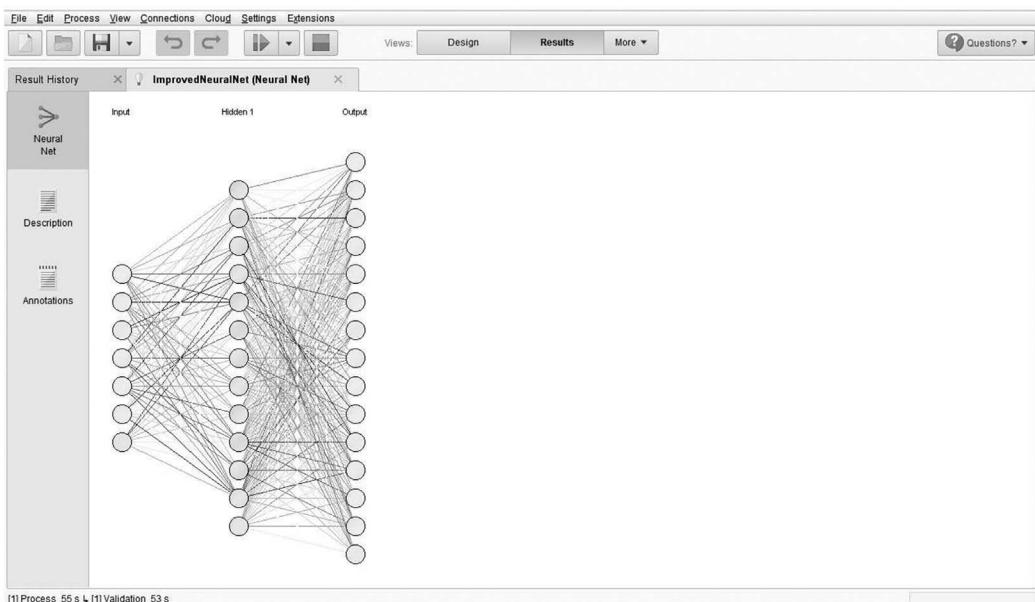


FIGURE 10.14 Network architecture for the satellite image data set.

	accuracy: 96.33% +/- 1.00% (mikro: 96.33%)												
	true urban	true agri...	true agric...	true turf...	true s_d...	true n_d...	true confit...	true shal...	true dee...	true mar...	true shr...	true woo...	pred. dark...
pred. urb...	20	0	0	0	0	0	0	0	0	0	0	0	0
pred. agr...	0	19	0	0	0	0	0	0	0	0	0	0	0
pred. agr...	0	0	20	0	0	0	0	0	0	0	0	0	0
pred. turf...	0	0	0	21	0	0	0	0	0	0	0	0	0
pred. s_d...	0	0	0	0	20	0	0	0	0	0	0	0	0
pred. n_d...	0	0	0	0	0	19	0	0	0	0	0	0	0
pred. confit...	0	0	0	0	0	0	20	0	0	0	0	0	0
pred. shal...	0	0	0	0	0	0	0	20	4	0	0	0	0
pred. dee...	0	0	0	0	0	0	0	0	15	0	0	0	0
pred. mar...	0	0	0	0	0	0	0	0	0	17	0	0	0
pred. shr...	0	1	0	0	0	0	0	0	0	2	20	0	0
pred. woo...	0	0	0	0	1	0	0	0	0	1	0	20	0
pred. dar...	0	0	0	0	0	0	0	0	0	0	0	0	20
pred. br...	0	0	0	0	0	0	0	0	0	0	0	0	0
pred. br...	0	0	0	0	0	0	0	0	0	0	0	0	0
class rec...	100.00%	95.00%	100.00%	100.00%	100.00%	95.00%	100.00%	100.00%	78.95%	85.00%	100.00%	100.00%	100.00%

FIGURE 10.15 Performance vector for the satellite image data set.

network algorithms are not able to determine a best set of input attributes, it may be possible to do even better. Let's repeat steps 3, 4, and 5 to determine if we can find one or several pairs of highly correlated input attributes.

- Modify your main process window so it appears as in Figure 10.16, where the *Correlation Matrix* and *Remove Correlated Attributes* operators have been added.

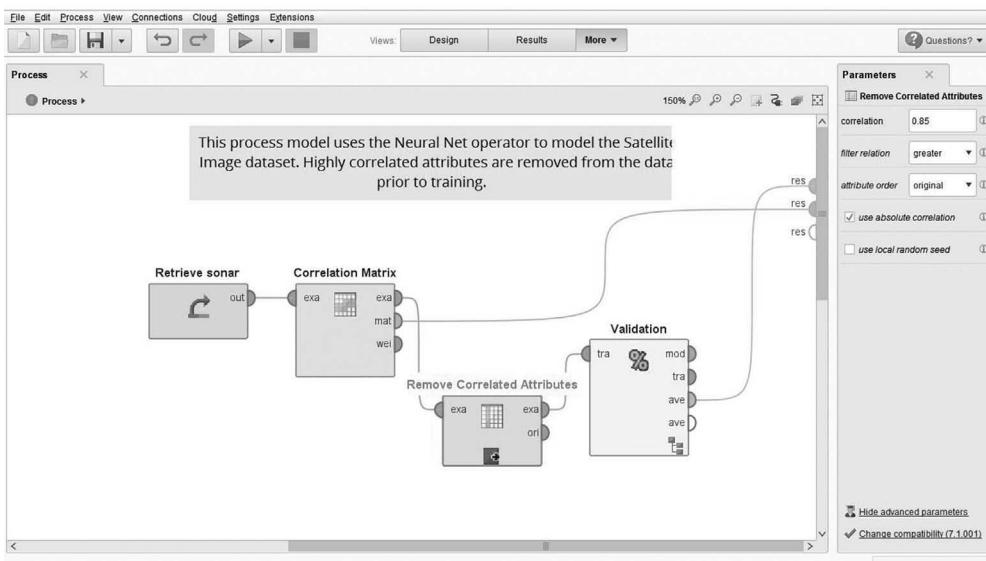


FIGURE 10.16 Removing correlated attributes from the satellite image data set.

- Click the *Remove Correlated Attributes* operator, insert a *Breakpoint After*, and set the correlation value at 0.85. If two input attributes have a correlation value of 0.85 or above, one will be removed.
- Run your process.

Figure 10.17 shows the result of the *Breakpoint After* within *Remove Correlated Attributes*. We see that input attributes *green* and *red* have been eliminated.

The screenshot shows the RapidMiner interface with the "Result History" tab selected. The table is titled "ExampleSet (300 examples, 1 special attribute, 4 regular attributes)". The table has columns: Row No., class, blue, nred, ir1, and ir2. The data shows rows from 1 to 19, with the first 10 rows belonging to the "urban" class and the next 9 to "agriculture1". The "blue" column values range from 104 to 118. The "nred" column values range from 65 to 72. The "ir1" and "ir2" columns show lower values, mostly between 21 and 47. The "Annotations" column on the left indicates which rows contain the attributes removed by the "Remove Correlated Attributes" operator.

Row No.	class	blue	nred	ir1	ir2
1	urban	104	65	80	43
2	urban	115	68	83	43
3	urban	106	47	76	42
4	urban	106	63	79	41
5	urban	104	66	76	41
6	urban	105	65	79	44
7	urban	103	58	81	45
8	urban	118	62	80	46
9	urban	112	54	79	47
10	urban	115	72	87	47
11	agriculture1	82	35	78	23
12	agriculture1	81	37	76	22
13	agriculture1	82	34	75	22
14	agriculture1	82	35	77	21
15	agriculture1	81	37	73	21
16	agriculture1	78	31	74	22
17	agriculture1	81	35	76	22
18	agriculture1	79	33	75	20
19	agriculture1	81	29	77	21

FIGURE 10.17 Green and red have been removed from the satellite image data set.

The screenshot shows the KNIME interface with the 'Results' view selected. On the left, there is a sidebar with icons for Data, Pairwise Table, Charts, and Annotations. The main area displays a correlation matrix table titled 'Correlation Matrix (Correlation Matrix)'.

Attributes	blue	green	red	nred	ir1	ir2
blue	1	0.958	0.867	0.687	-0.071	0.630
green	0.958	1	0.869	0.725	-0.050	0.633
red	0.867	0.859	1	0.729	-0.070	0.762
nred	0.687	0.725	0.729	1	0.078	0.688
ir1	-0.071	-0.050	-0.070	0.078	1	-0.016
ir2	0.630	0.633	0.762	0.688	-0.016	1

FIGURE 10.18 Correlation matrix for the satellite image data set.

- Resume the process to observe the correlation matrix (Figure 10.18). The performance vector (not shown) displays a classification accuracy of 97%.

This result implies that we can eliminate one-third of the input attributes and maintain the same level of classification correctness! Furthermore, Figure 10.18 verifies the high correlations between *green* and *blue* (0.958) and *red* and *blue* (0.867).

At this point, the entire data set can be incorporated to build and save a final model for classifying the same region after specified time periods. When significant changes such as, for example, increased or decreased water levels are identified, appropriate action can be taken.

Lastly, this example used the *X-Validation* operator to help test model accuracy. However, in some situations, the *X-Prediction* operator is more appropriate. The *X-Prediction* operator is similar to *X-Validation* except that instead of a performance vector, it produces a labeled example set. End-of-chapter exercise 3 investigates the use of the *X-Prediction* operator. After applying either *X-Validation* or *X-Prediction*, it becomes obvious that breakpoints are best avoided. In the next section, we review the procedure for saving and applying a model such as the one developed here.

10.3 PREDICTING CUSTOMER CHURN

For our final example with the *Neural Net* operator, we return to the customer churn data set, *customer-churn-data.xlsx*. Recall that the data set contains 996 instances, 900 whose outcome (loyal or churn) is known. Our job is to build a neural network model with the

help of the 900 instances of known outcome to predict likely churners. We then apply the model to the 96 instances whose outcomes are to be determined. First, we state the goal.

Step 1: Identify the Goal

Our goal is to build a predictive neural network model using the 900 known instances from the customer churn data set able to correctly identify 75% of all churners and at the same time misclassify fewer than 15% of all loyal customers.

Steps 2 and 3:

Use Figures 10.19 through 10.21 to design the process model that prepares the data and defines the network architecture.

- The *Nominal to Numerical* operator is needed to convert nominal attributes *payment method* and *gender* to numeric equivalents. Be sure to specify *attribute filter type* as all.
- The *Filter Examples* operator is used to eliminate instances whose outcome contains a question mark. Use this operator to add a filter stating that churn *does not equal?*
- Check the *shuffle* and *normalize* parameters for the *Neural Net* operator. Use default settings for *training cycles*, *learning rate*, and *momentum*.
- For the cross-validation, set the number of validations at 10.

Step 4: Train the Network

- Return to the main window and run your process.

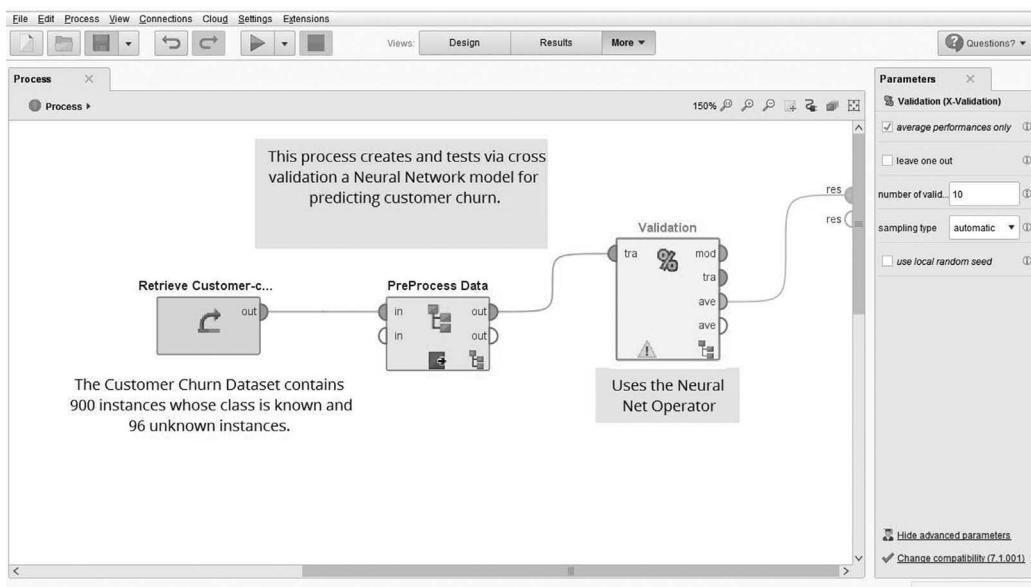


FIGURE 10.19 Neural network model for predicting customer churn.

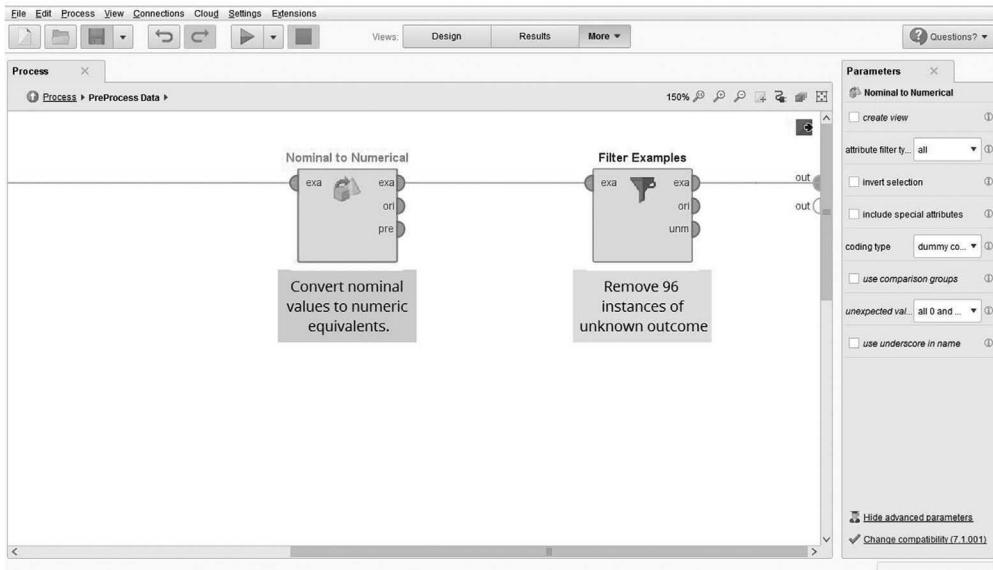


FIGURE 10.20 Preprocessing the customer churn data.

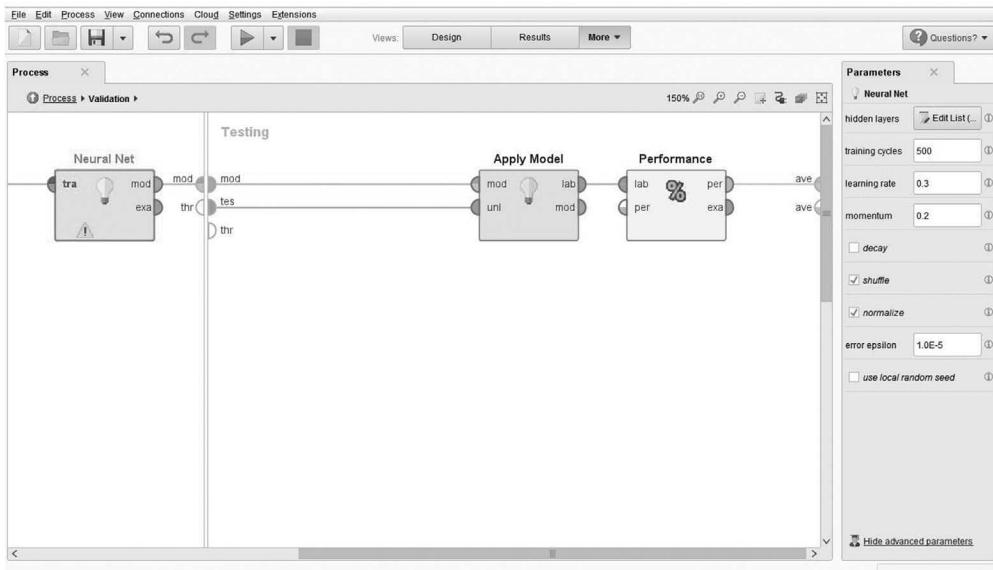


FIGURE 10.21 Cross-validation subprocess for customer churn.

Step 5: Read and Interpret Summary Results

The resultant performance vector (Figure 10.22) shows a classification accuracy of 84.44% with 72 churners deemed loyal and 68 loyal customers declared as churners. The model correctly identifies 77.64% of the churning customers and misclassifies less than 13% of all loyal customers. As the *shuffle* parameter was checked, your results may differ slightly from those shown in Figure 10.22.

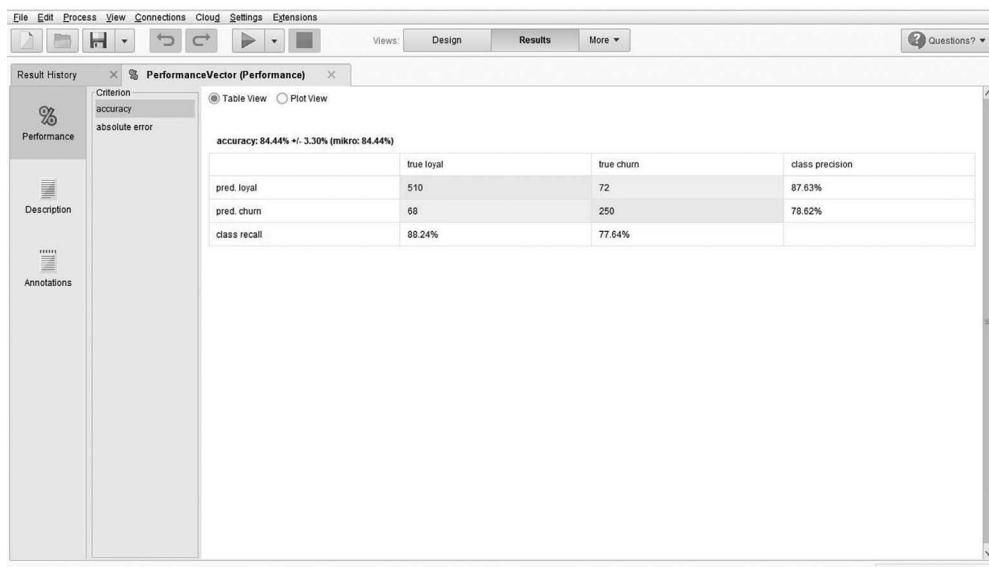


FIGURE 10.22 Performance vector for customer churn.

Given that we have met the stated goal, our next step is to use all of the known instances to create and save a final model. Figure 10.23 shows the desired process model. Here are the details:

- Use the *Write Model* operator with *output type* set at *Binary*. Choose a destination folder for the model and run your process.
- Check to make sure your model has been written to the specified location.

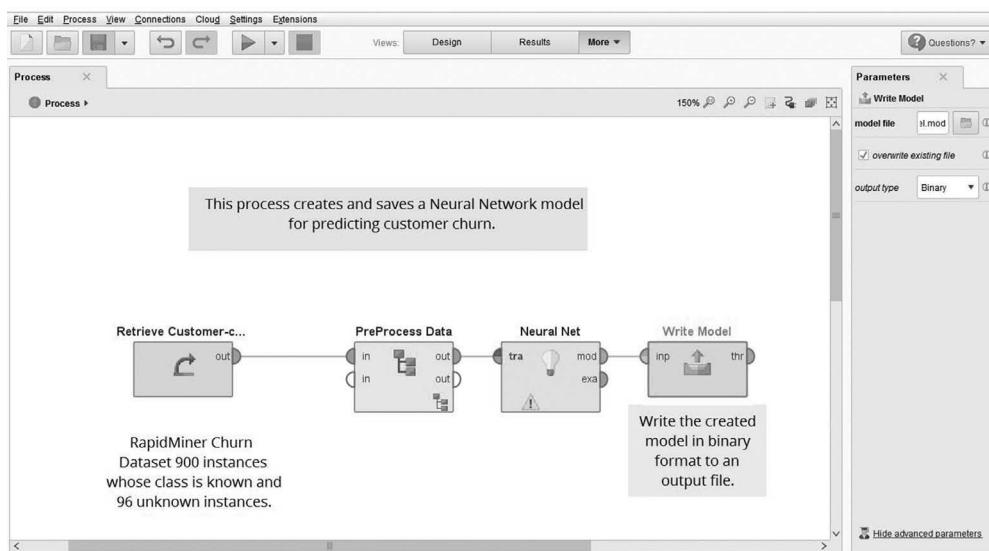


FIGURE 10.23 Process for creating and saving a neural network model.

The neural network model is now ready to be applied to the customer data whose churn status is unknown! Here's how.

- Implement the design in Figure 10.24. This design generates a process that reads the created model and applies it to the 96 instances whose outcomes are unknown. The subprocess should be identical to the one shown in Figure 10.20, except that

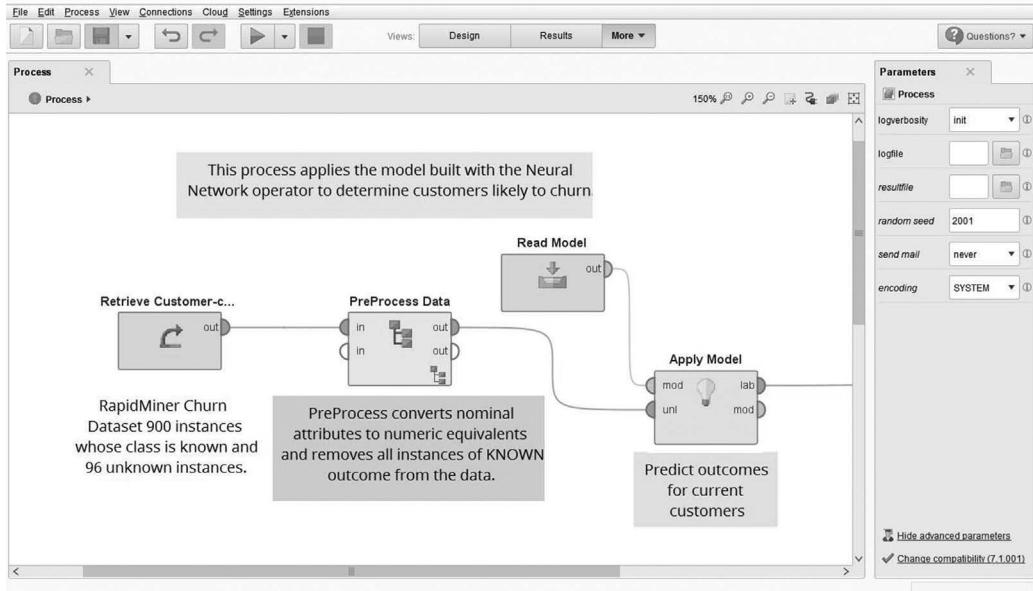


FIGURE 10.24 Process model for reading and applying a neural network model.

This screenshot shows the 'Result History' view in RapidMiner, specifically the 'ExampleSet (Filter Examples)' tab. The table displays 96 examples with various attributes. The columns include Row No., Churn, prediction(Churn), confidence(Lab), confidence(Churn), Gender = male, Gender = female, Payment Method, Age, and Last Transaction. A filter at the top right shows 'Filter (96 / 96 examples): all'. The data table is as follows:

Row No.	Churn	prediction(Churn)	confidence(Lab)	confidence(Churn)	Gender = male	Gender = female	Payment Method	Age	LastTransa...
1	?	churn	0.193	0.807	0	1	1	0	39
2	?	churn	0.204	0.795	0	1	0	1	53
3	?	churn	0.213	0.787	0	1	1	0	33
4	?	churn	0.126	0.874	0	1	1	0	194
5	?	loyal	0.694	0.306	1	0	0	1	81
6	?	churn	0.192	0.808	0	1	0	1	153
7	?	loyal	0.968	0.032	1	0	1	0	146
8	?	churn	0.192	0.808	0	1	1	0	54
9	?	churn	0.192	0.808	0	1	1	0	102
10	?	churn	0.250	0.750	0	1	1	0	58
11	?	loyal	0.970	0.030	1	0	1	0	176
12	?	loyal	0.967	0.033	1	0	1	0	45
13	?	churn	0.196	0.804	0	1	1	0	150
14	?	churn	0.196	0.804	0	1	0	1	144
15	?	loyal	0.973	0.127	1	0	0	0	82
16	?	churn	0.428	0.572	0	1	1	0	36
17	?	loyal	0.763	0.237	1	0	0	1	91
18	?	loyal	0.853	0.147	0	1	1	0	72
19	?	loyal	0.886	0.114	1	0	0	1	158

FIGURE 10.25 Neural network output for predicting customer churn.

the *Filter Examples* operator must be modified so it eliminates all instances of known rather than unknown outcome.

- Run your process to see the result displayed in Figure 10.27.

Figure 10.25 displays predictions for the first few instances. Having this, we can examine the probability values associated with those individuals likely to churn. We can use these values as confidence factors to sort the subset of likely churners. This result helps us choose the subset of individuals to offer incentives for remaining loyal customers.

10.4 RAPIDMINER'S SELF-ORGANIZING MAP OPERATOR

RapidMiner lists its SOM operator as an attribute set reduction and transformation tool. However, we can also set SOM's parameters to implement the unsupervised clustering algorithm described in Chapter 8. Here we use the cardiology patient data set of Chapter 2 to outline the process.

- Load the *cardiologyMixed* data set, and the *Nominal to Numerical* and SOM operators into the main process window. Click on the SOM operator.
- Set *number of dimensions* at 1.

This tells SOM to use the input attributes to create a one-dimensional model to characterize the data. This single dimension represents the cluster attribute.

- Set *net size* at 2. Net size gives the total number of clusters to be formed. As the heart patient data set contains two classes, this is a viable choice.
- Set *training rounds* at 300. Training rounds give the total number of times the data are passed through the network.

Once these changes are made, your screen will appear as in Figure 10.26.

- Click the run process to see the result displayed in Figure 10.27.

Figure 10.27 gives us the class associated with each instance and its corresponding cluster number. The ideal case will show all instances of the healthy class contained in one of the two clusters and all instances of the sick class in the other cluster. The degree to which the clusters differentiate between the classes is an indicator of the degree of success we will have using the input attributes to build a supervised learner model.

To obtain a clearer understanding about the clusters formed, we have several options.

- We can click the *class* or *SOM_0* heading to sort the data in order to get a visual read of how well the instances cluster relative to their assigned class.
- We can use the *Write Excel* operator to write our results into an Excel spreadsheet in order to analyze how well the clusters separate the classes.

312 ■ Data Mining

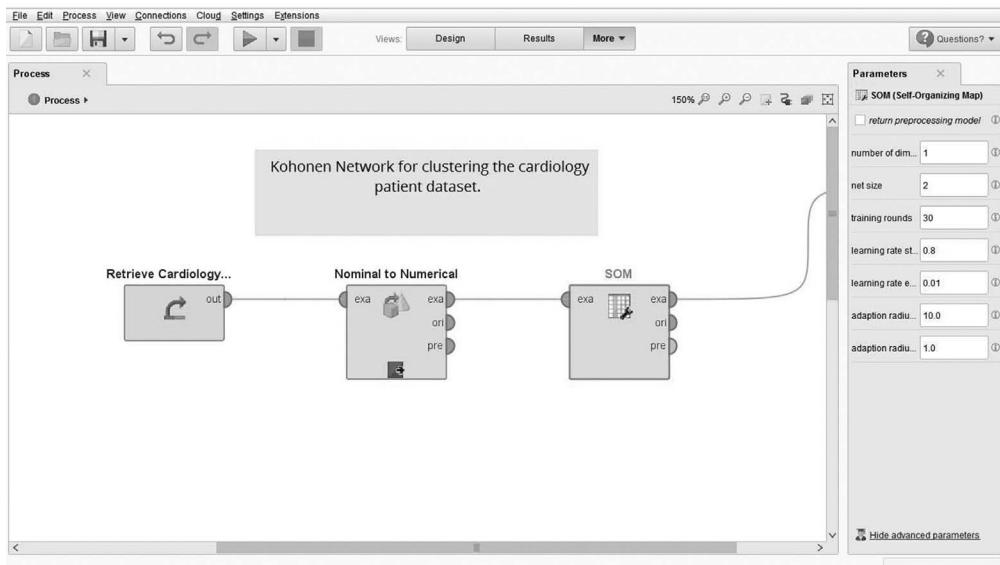


FIGURE 10.26 SOM process model for the cardiology patient data set.

ExampleSet (SOM)			
ExampleSet (/Local Repository/data/XOR Function Ap 22)			
ExampleSet (/NewRepositoryA/data/XORFunction)			
Data	Row No.	class	SOM_0
Statistics	1	Sick	0
Charts	2	Healthy	1
Advanced Charts	3	Healthy	0
Annotations	4	Sick	1
	5	Sick	0
	6	Healthy	1
	7	Sick	1
	8	Sick	0
	9	Healthy	0
	10	Sick	1
	11	Sick	0
	12	Healthy	0
	13	Healthy	0
	14	Healthy	0
	15	Sick	1
	16	Healthy	1
	17	Healthy	0
	18	Healthy	1
	19	Sick	1
	20	Healthy	1

FIGURE 10.27 Clustered instances of the cardiology patient data set.

- An interesting activity is to follow writing the output to an Excel file with these steps:
 - Open the Excel file and copy the SOM_0 column.
 - Open the original cardiology patient data set and form a new column using the copied data. You now have a match of the original data set relative to the clustering.

- Load the modified cardiology file into RapidMiner and designate SOM_0 as the label attribute.
- Use one of the decision tree operators to build a model from the data.

This model will help you determine the attributes that best define the formed clusters. End-of-chapter exercise 10 challenges you to investigate this last possibility.

10.5 CHAPTER SUMMARY

RapidMiner's *Neural Network* operator uses backpropagation learning to build supervised models for estimation, classification, and prediction. To design the network architecture, we can modify default settings for the number of hidden layers, the number of hidden-layer nodes, and the number of training epochs. Several additional parameters can be adjusted to help us build accurate neural net models.

RapidMiner's *Self-Organizing Map* is listed as an *attribute set reduction* and *transformation* tool. With the proper parameter settings, we are able to use SOM to model the traditional unsupervised neural network clustering described in Chapter 8.

Attribute preprocessing is of special significance with neural networks as the algorithms they use are unable to determine attribute relevance. Several of RapidMiner's attribute preprocessing operators are of particular interest including operators that eliminate correlated and irrelevant attributes.

EXERCISES

1. Create a backpropagation neural network to model one of the three logical operators shown in the following table. Use the *Multiply* operator together with the *Apply Model* and *Performance* operators to test the network. Run experiments with one or more of these operators using one and two hidden layers. What effect does adding an extra hidden layer have on your results?

Input	Input 2	And	Or	Implication
1	1	1	1	1
0	1	0	1	1
1	0	0	1	0
0	0	0	0	1

2. Create a backpropagation neural network to model logical complement (negation). Logical complement outputs 1 when the input is 0, and 0 when the input is 1. Use the *Multiply* operator together with the *Apply Model* and *Performance* operators to test the network. How many epochs are needed to train the network? What effect does adding a second hidden layer have on the number of epochs required to create an accurate model?

3. Make the needed modifications to the Sonar example described in 10.2 in order to replace the *X-Validation* operator with *X-Prediction*. Describe situations when each operator is a better choice with these data.
4. Add a hidden layer to the XOR example to train a neural network with one hidden layer of 5 nodes and a second hidden layer of 10 nodes. First try 500 and then 5000 epochs. What happens to the accuracy of the network? Why do you think this is the case?
5. Create and save a new process. Load *CardiologyMixed.xlsx* with output attribute *class* into the main process area. Build and test a model for these data using the *Neural Net*, *Set Role*, *Apply Model*, *Performance (classification)*, *Cross Validation*, *Nominal to Numerical*, *Correlation*, and *Remove Correlated Attributes* operators. Summarize your results. Highlight the correlation matrix and performance vector in your summary.
6. Return to the customer churn example in Section 10.3. Attempt to achieve a better classification model by modifying the parameters of the *Neural Net* operator. Apply your model to the instances whose outcome is unknown. Summarize your results.
7. For this exercise, you are to use the *Neural Net* operator to experiment with the spam (*spam.xlsx*) data set.
 - a. Use a training/test-set scenario with default settings for the *Neural Net* operator to obtain an initial classification accuracy.
 - b. Modify the parameters of the *Neural Net* operator and investigate removing highly correlated attributes in an attempt to obtain a better result. Focus on building a model to minimize the number of valid e-mails incorrectly classified as spam.
 - c. Attempt to improve your results with *Forward Selection* or *Backward Elimination*. Try to minimize the number of valid e-mails incorrectly classified as spam.
 - d. Summarize the results.
8. Load a data set of your choice for neural net learning. Experiment with different values for the *Neural Net* operator parameters. Report your results.
9. Implement the clustering model given in Figure 10.26. Add the *Write Excel* operator to write the output to an Excel file. Examine the file to determine the number of healthy and sick patients associated with each cluster. Compute the percent of healthy and sick instances within each cluster. To what extent did the clustering differentiate the classes?
10. Use SOM to experiment with the cardiology patient data set by following the steps outlined in Section 10.4 for building a decision tree to summarize the formed clusters. What attributes are used for the top three levels of the decision tree?

IV

Advanced Data Mining Techniques



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Supervised Statistical Techniques

CHAPTER OBJECTIVES

- *Understand how the Bayes classifier is able to build supervised models for datasets having categorical data, numeric data, or a combination of both data types*
- *Know how support vector machines use nonlinear mappings to create linear models*
- *Understand when linear regression is an appropriate data mining technique*
- *Know how regression and model trees are used to build nonlinear classification models*
- *Know that logistic regression can be used to build supervised learner models for data sets having a binary outcome*

In this chapter, we provide a detailed overview of several statistical supervised data mining techniques. It is not necessary for you to study each technique in detail to obtain an overall understanding of this field. For this reason, each section of this chapter is self-contained.

In Section 11.1, you learn how the Bayes classifier builds supervised models for both categorical and real-valued data. Section 11.2 introduces support vector machines for supervised learning. Section 11.3 covers simple and multiple linear regression. Regression trees and model trees are the topics of Section 11.4. In Section 11.5, we discuss logistic regression and how it is applied to build supervised learner models for data sets with a binary outcome. We supplement our discussion with tutorials and several end-of-chapter exercises.

11.1 NAÏVE BAYES CLASSIFIER

The *naïve Bayes classifier* offers a simple yet powerful supervised classification technique. The classifier is termed naïve as it assumes the independence of all input attributes, and that numeric attribute values are normally distributed. Naïve Bayes usually performs very

well even when these assumptions are known to be false. The classifier is based on *Bayes' theorem*, which is stated as

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E)} \quad (11.1)$$

where H is a hypothesis to be tested and E is the evidence associated with the hypothesis.

From a classification viewpoint, the hypothesis is the dependent variable and represents the predicted class. The evidence is determined by values of the input attributes. $P(H|E)$ is the *conditional probability* that H is true given evidence E . $P(H)$ is an *a priori probability*, which denotes the probability of the hypothesis before the presentation of any evidence. Conditional and a priori probabilities are easily computed from the training data. The classifier is best understood with an example.

11.1.1 Naïve Bayes Classifier: An Example

Consider the data in Table 11.1, which is a subset of the credit card promotion database defined in Chapter 2. For our example, we use *gender* as the output attribute whose value is to be predicted. Table 11.2 lists the distribution (counts and ratios) of the output attribute values for each input attribute. To demonstrate, Table 11.2 tells us that four males took

TABLE 11.1 Data for Bayes Classifier

Magazine Promotion	Watch Promotion	Life Insurance Promotion	Credit Card Insurance	Gender
Yes	No	No	No	Male
Yes	Yes	Yes	Yes	Female
No	No	No	No	Male
Yes	Yes	Yes	Yes	Male
Yes	No	Yes	No	Female
No	No	No	No	Female
Yes	Yes	Yes	Yes	Male
No	No	No	No	Male
Yes	No	No	No	Male
Yes	Yes	Yes	No	Female

TABLE 11.2 Counts and Probabilities for Attribute *Gender*

Gender	Magazine Promotion		Watch Promotion		Life Insurance Promotion		Credit Card Insurance	
	Male	Female	Male	Female	Male	Female	Male	Female
Yes	4	3	2	2	2	3	2	1
No	2	1	4	2	4	1	4	3
Ratio: yes/total	4/6	3/4	2/6	2/4	2/6	3/4	2/6	1/4
Ratio: no/total	2/6	1/4	4/6	2/4	4/6	1/4	4/6	3/4

advantage of the magazine promotion, and that these four males represent two-thirds of the total male population. As a second example, Table 11.2 indicates that three of the four female data set instances purchased the magazine promotion.

Let's use the data in Table 11.2 together with the Bayes classifier to perform a new classification. Consider the following new instance to be classified:

Magazine Promotion = Yes

Watch Promotion = Yes

Life Insurance Promotion = No

Credit Card Insurance = No

Gender = ?

We have two hypotheses to be tested. One hypothesis states that the credit card holder is male. The second hypothesis sees the instance as a female card holder. In order to determine which hypothesis is correct, we apply naïve Bayes to compute a probability for each hypothesis. The general equation for computing the probability that the individual is a male customer is

$$P(\text{gender} = \text{male}|E) = \frac{P(E|\text{gender} = \text{male})P(\text{gender} = \text{male})}{P(E)} \quad (11.2)$$

Let's start with the conditional probability $P(E|\text{gender} = \text{male})$. This probability is computed by multiplying the conditional probability values for each piece of evidence. This is possible if we make the assumption that the evidence is independent. The overall conditional probability is the product of the following four conditional probabilities:

$$P(\text{magazine promotion} = \text{yes}|\text{gender} = \text{male}) = 4/6$$

$$P(\text{watch promotion} = \text{yes}|\text{gender} = \text{male}) = 2/6$$

$$P(\text{life insurance promotion} = \text{no}|\text{gender} = \text{male}) = 4/6$$

$$P(\text{credit card insurance} = \text{no}|\text{gender} = \text{male}) = 4/6$$

These values are easily obtained as they can be read directly from Table 11.2. Therefore, the conditional probability for $\text{gender} = \text{male}$ is computed as

$$\begin{aligned} P(E|\text{gender} = \text{male}) &= (4/6)(2/6)(4/6)(4/6) \\ &= 8/81 \end{aligned}$$

The a priori probability, denoted in Equation 11.2 as $P(\text{gender} = \text{male})$, is the probability of a male customer without knowing the promotional offering history of the instance. In this case, the a priori probability is simply the fraction of the total population that is male. As there are six males and four females, the a priori probability for $\text{gender} = \text{male}$ is 3/5.

Given these two values, the numerator expression for Equation 11.2 becomes

$$(8/81)(3/5) \cong 0.0593$$

We now have

$$P(\text{gender} = \text{male}|E) \cong 0.0593/P(E)$$

Next, we compute the value for $P(\text{gender} = \text{female}|E)$ using the formula

$$P(\text{gender} = \text{female}|E) = \frac{P(E|\text{gender} = \text{female})P(\text{gender} = \text{female})}{P(E)} \quad (11.3)$$

We first compute the conditional probability with values obtained directly from Table 11.2. Specifically,

$$P(\text{magazine promotion} = \text{yes}|\text{gender} = \text{female}) = 3/4$$

$$P(\text{watch promotion} = \text{yes}|\text{gender} = \text{female}) = 2/4$$

$$P(\text{life insurance promotion} = \text{no}|\text{gender} = \text{female}) = 1/4$$

$$P(\text{credit card insurance} = \text{no}|\text{gender} = \text{female}) = 3/4$$

The overall conditional probability is

$$\begin{aligned} P(E|\text{gender} = \text{female}) &= (3/4)(2/4)(1/4)(3/4) \\ &= 9/128 \end{aligned}$$

As there are four females, the a priori probability for $P(\text{gender} = \text{female})$ is 2/5. Therefore, the numerator expression for Equation 11.3 becomes

$$(9/128)(2/5) \cong 0.0281$$

We now have

$$P(\text{gender} = \text{female}|E) \cong 0.0281/P(E)$$

Finally, we need not concern ourselves with $P(E)$, because it represents both the probability of the evidence when $gender = male$ and when $gender = female$. Therefore, because $0.0593 > 0.0281$, the naïve Bayes classifier tells us the instance is most likely a male credit card customer.

11.1.2 Zero-Valued Attribute Counts

A significant problem with the naïve Bayes technique is when one of the counts for an attribute value is 0. For example, suppose the number of females with a value of *no* for *credit card insurance* is 0. In this case, the numerator expression for $P(gender = female|E)$ will be 0. This means that the values for all other attributes are irrelevant because any multiplication by 0 gives an overall 0 probability value.

To solve this problem, we add a small constant, k , to the numerator (n) and denominator (d) of each computed ratio. Therefore, each ratio of the form n/d becomes

$$\frac{n+(k)(p)}{d+k} \quad (11.4)$$

where

k is a value between 0 and 1 (usually 1)

p is chosen as an equal fractional part of the total number of possible values for the attribute (for example, if an attribute has two possible values, p will be 0.5)

Let's use this technique to recompute the conditional probability $P(E|gender = female)$ for our previous example. With $k = 1$ and $p = 0.5$, the conditional probability of the evidence given $gender = female$ computes to

$$\frac{(3+0.5) \times (2+0.5) \times (1+0.5) \times (3+0.5)}{5 \times 5 \times 5 \times 5} \approx 0.0176$$

11.1.3 Missing Data

Fortunately, missing data items are not a problem for the Bayes classifier. To demonstrate, consider the following instance to be classified by the model defined in Table 11.2.

Magazine Promotion = Yes

Watch Promotion = Unknown

Life Insurance Promotion = No

Credit Card Insurance = No

Gender = ?

As the value of *watch promotion* is unknown, we can simply ignore this attribute in our conditional probability computations. By doing so, we have

$$P(E|gender = male) = (4/6)(4/6)(4/6) = 8/27$$

$$P(E|gender = female) = (3/4)(1/4)(3/4) = 9/64$$

$$P(gender = male|E) \cong 0.1778/P(E)$$

$$P(gender = female|E) \cong 0.05625/P(E)$$

As you can see, the effect is to give a probability value of 1.0 to the *watch promotion* attribute. This will result in a larger value for both conditional probabilities. However, this is not a problem, because both probability values are equally affected.

11.1.4 Numeric Data

Numeric data can be dealt with in a similar manner provided that the probability density function representing the distribution of the data is known. If a particular numerical attribute is normally distributed, we use the standard probability density function shown in Equation 11.5.

$$f(x) = 1/(\sqrt{2\pi}\sigma)e^{-(x-\mu)^2/2\sigma^2} \quad (11.5)$$

where

e = the exponential function

μ = the class mean for the given numerical attribute

σ = the class standard deviation for the attribute

x = the attribute value

Although this equation looks quite complicated, it is very easy to apply. To demonstrate, consider the data in Table 11.3. This table displays the data in Table 11.1 with an added column containing numerical attribute *age*.

Let's use this new information to compute the conditional probabilities for the *male* and *female* classes for the following instance.

Magazine Promotion = Yes

Watch Promotion = Yes

Life Insurance Promotion = No

Credit Card Insurance = No

Age = 45

Gender = ?

TABLE 11.3 Addition of Attribute Age to the Bayes Classifier Data Set

Magazine Promotion	Watch Promotion	Life Insurance Promotion	Credit Card Insurance	Age	Gender
Yes	No	No	No	45	Male
Yes	Yes	Yes	Yes	40	Female
No	No	No	No	42	Male
Yes	Yes	Yes	Yes	30	Male
Yes	No	Yes	No	38	Female
No	No	No	No	55	Female
Yes	Yes	Yes	Yes	35	Male
No	No	No	No	27	Male
Yes	No	No	No	43	Male
Yes	Yes	Yes	No	41	Female

For the overall conditional probabilities, we have

$$P(E|gender = male) = (4/6)(2/6)(4/6)(4/6) [P(age = 45|gender = male)]$$

$$P(E|gender = female) = (3/4)(2/4)(1/4)(3/4) [P(age = 45|gender = female)]$$

To determine the conditional probability for *age* given *gender = male*, we assume *age* to be normally distributed and apply the probability density function. We use the data in Table 11.3 to find the mean and standard deviation scores. For the class *gender = male* we have $\sigma = 7.69$, $\mu = 37.00$, and $x = 45$. Therefore the probability that *age* = 45 given *gender = male* is computed as

$$P(age = 45|gender = male) = 1/(\sqrt{2\pi} 7.69) e^{-(45-37.00)^2/2(7.69)^2}$$

Making the computation, we have

$$P(age = 45|gender = male) \cong 0.030$$

To determine the conditional probability for *age* given *gender = female*, we substitute $\sigma = 7.77$, $\mu = 43.50$, and $x = 45$. Specifically,

$$P(age = 45|gender = female) = 1/(\sqrt{2\pi} 7.77) e^{-(45-43.50)^2/2(7.77)^2}$$

Making the computation, we have

$$P(age = 45|gender = female) \cong 0.050$$

We can now determine the overall conditional probability values:

$$P(E|gender = male) = (4/6) (2/6) (4/6) (4/6) (0.030) \cong .003$$

$$P(E|gender = female) = (3/4) (2/4)(1/4) (3/4) (0.050) \cong .004$$

Finally, applying Equation 11.1 we have

$$P(gender = male|E) \cong (.003) (0.60)/P(E) \cong .0018/P(E)$$

$$P(gender = female|E) \cong (.004) (0.40)/P(E) \cong .0016/P(E)$$

Once again, we ignore $P(E)$ and conclude that the instance belongs to the *male* class.

11.1.5 Implementations of the Naïve Bayes Classifier

RapidMiner and Weka offer several tools based on Bayes' theorem. Here we focus on the naïve Bayes classifier as it closely parallels the above discussion. For a tutorial using RapidMiner's Naïve Bayes operator, please see the highlighted section "RapidMiner's Naïve Bayes Operator." For the Weka tutorial, please see the highlighted section "Weka's Naïve Bayes Classifier."

11.1.6 General Considerations

Naïve Bayes offers a simple probability-based approach that accepts both numeric and categorical input data and easily handles missing data. Naïve Bayes tends to work well even when underlying assumptions about the data are not true. Specific assumptions include input attribute independence and the normal distribution of numeric attributes. However, if we know the distribution of an individual numeric attribute, we can use it to replace Equation 11.5. Also, as the naïve Bayes data model is a simple table, model update is trivial.

One problem with naïve Bayes is that a single conditional probability with a value close to zero can dramatically affect outcome attribute values. Another shortcoming is the assumption that all input attributes are of equal importance. Our chances of doing well with naïve Bayes often improve significantly when attribute selection takes place prior to model building.

11.2 SUPPORT VECTOR MACHINES

Support vector machines (SVMs) provide a unique approach for classifying both linearly separable and nonlinear data. The first SVM algorithm was developed by Vladimir Vapnik and Alexey Chervonenkis in the 1960s. However, SVMs drew general interest only after their formal introduction to the research community in 1992 (Boser

RAPIDMINER TUTORIAL

RapidMiner's Naïve Bayes Operator

Let's use the customer churn dataset to examine RapidMiner's Naïve Bayes operator. Here is the procedure.

- Create and save a new process.
- Retrieve the customer churn data set from your repository and create the process model illustrated in Figures 11.1 and 11.2.
- Place a *Breakpoint After* within the Naïve Bayes operator.
- Run your process to invoke the breakpoint and click on *Distribution Table* to see Figure 11.3.

The distribution table gives the conditional probability values for each categorical attribute. To illustrate, $P(\text{gender} = \text{male}|\text{loyal})$ is 0.769, and $P(\text{payment method} = \text{credit card}|\text{churn})$ is 0.545. Conditional probabilities for numeric attributes can be computed by applying Equation 11.5 using the provided mean and standard deviation values.

Notice that each categorical attribute has a value named *unknown* whose conditional probability is 0. To make nonzero conditional probability scores a possibility, we must check the *Laplace correction* parameter box shown in the upper right corner of Figure 11.1. This parameter determines if a small nonzero conditional probability should be assigned to *unknown* in order to avoid the possibility of zero-valued probability scores. You can learn more about this parameter by reading the documentation.

Lastly, for any instance, the information in the distribution table and the *a priori* values for $\text{Churn} = \text{loyal}$ (578/900) and $\text{Churn} = \text{churn}$ (322/900) gives us what we need to manually compute the probability of customer churn. Fortunately, we have the Naïve Bayes operator to take care of this for us!

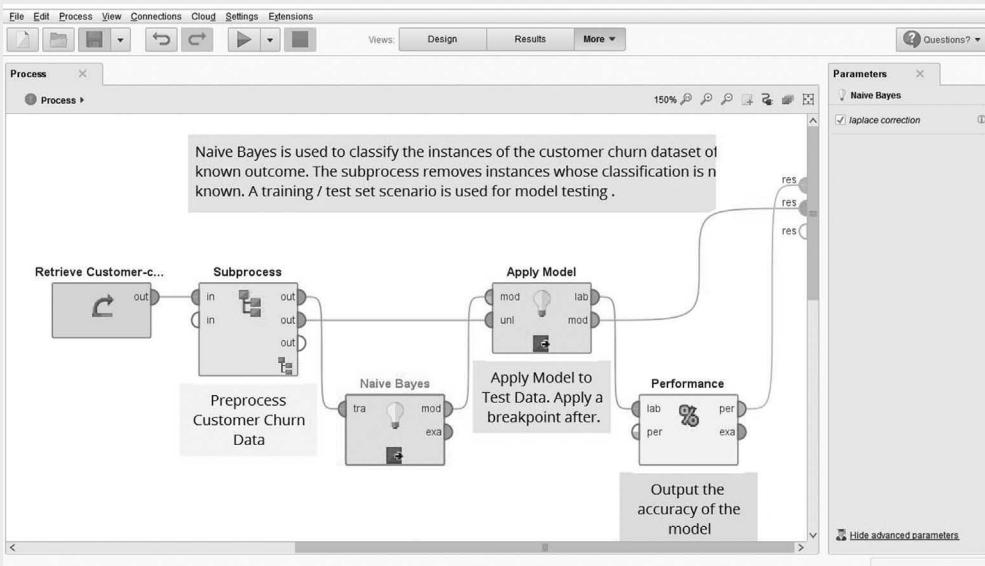


FIGURE 11.1 RapidMiner's naïve Bayes operator.

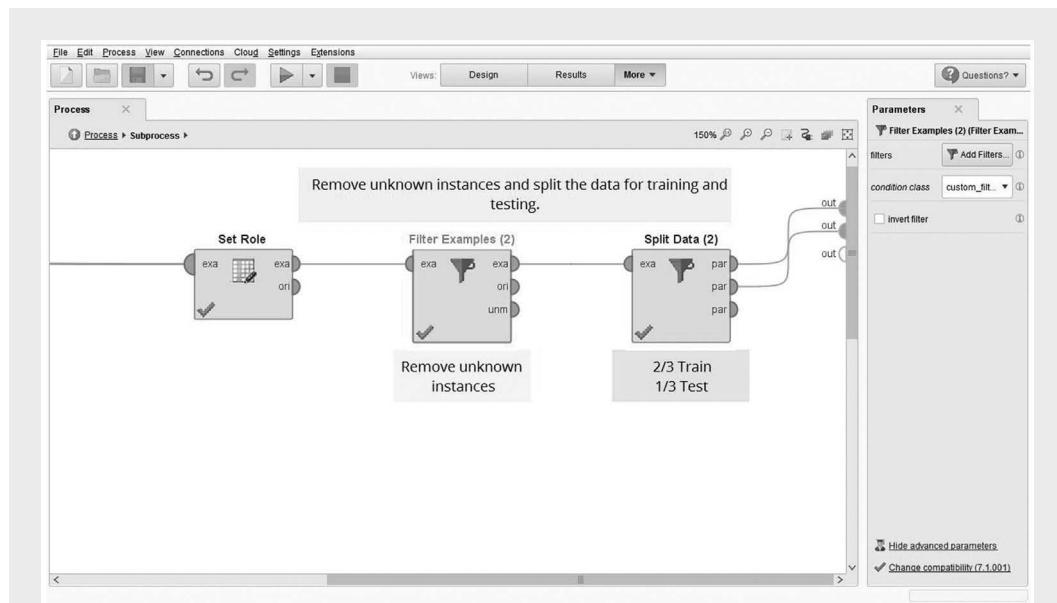


FIGURE 11.2 Subprocess for applying naïve Bayes to customer churn data.

The screenshot shows the KNIME interface with a results view titled "SimpleDistribution (Naïve Bayes)". The results are presented in a table format. The table has four columns: "Attribute", "Parameter", "loyal", and "churn". The table rows are as follows:

Description	Attribute	Parameter	loyal	churn
Charts	Gender	value=male	0.769	0.163
	Gender	value=female	0.231	0.837
	Gender	value=unknown	0.000	0.000
Distribution Table	Age	mean	43.322	51.440
	Age	standard deviation	18.761	18.148
Annotations	Payment Method	value=credit card	0.703	0.545
	Payment Method	value=cheque	0.038	0.086
	Payment Method	value=cash	0.259	0.368
	Payment Method	value=unknown	0.000	0.000
	LastTransaction	mean	100.041	130.498
	standard deviation	41.413	45.482	

FIGURE 11.3 Naïve Bayes Distribution Table for customer churn data.

- Click resume to see the performance vector shown in Figure 11.4.

The performance vector tells us that 32 customers who churned were predicted as loyal, and 23 loyal customers were classified as churning. The first data mining exercise asks you to use the *Naïve Bayes* operator to perform additional experiments with this data set.

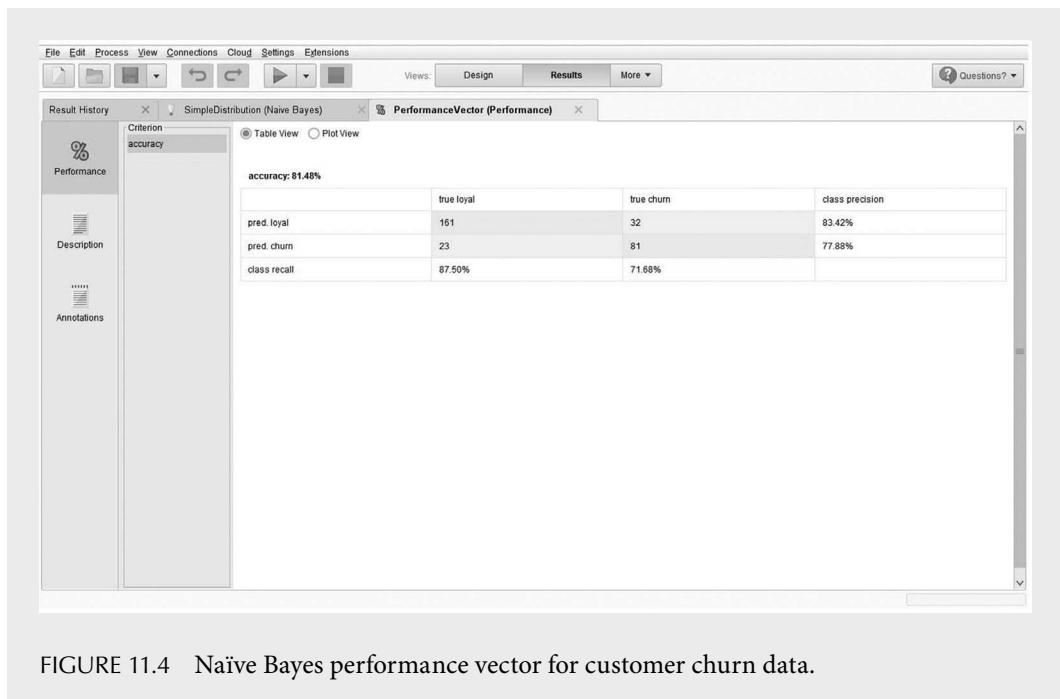


FIGURE 11.4 Naïve Bayes performance vector for customer churn data.

et al., 1992). SVMs have been applied in several areas including time series analysis, bioinformatics, textual data mining, and speech recognition. Unlike backpropagation neural networks and other techniques that often create locally optimal models, SVMs are able to provide globally optimal solutions. In addition, because SVMs use the most difficult-to-classify instances as the basis for their predictions, they are less likely to overfit the data.

The basic idea of how an SVM algorithm works is conceptually simple, but the mathematics can be a challenge. To help remove much of the mystery behind SVMs, we take an example-based approach.

Although SVMs can be used for both classification and numeric prediction, our focus is on SVMs that create binary classification models. Those who desire to delve deeper into the more theoretical aspects of SVMs can find a wealth of information both in textbooks (Vapnik, 1998, 1999) and on the Web.

Key to the workings of an SVM is the hyperplane. A hyperplane is simply a subspace one dimension less than its ambient space. To see this, consider Figure 11.8, which displays a two-dimensional space with linearly separable classes. One class is represented by stars, and the second, by circles. In two dimensions, a hyperplane is a straight line. The figure shows two such hyperplanes labeled h_1 and h_2 . Clearly, there is an infinite number of hyperplanes dividing the classes. So, which hyperplane is the best choice? It turns out that it's the hyperplane showing the greatest separation between the class instances. This

WEKA TUTORIAL

Weka's Naïve Bayes Classifier

Let's use the credit card promotion dataset to examine Weka's implementation of the naïve Bayes classifier. The procedure follows:

- Load *CreditCardPromotion.arff* into the Explorer.
- While in preprocess mode, go to the lower right portion of your display screen and set *LifeInsPromo* (life insurance promotion) as the class attribute. Place a check in the *Gender* attribute box. Your screen will appear as in Figure 11.5.

The visualization makes it clear that the majority of female card holders accepted the life insurance promotion, whereas the majority of male card holders did not.

- Mouse to Classify mode, and in the box under the More options box, change the output attribute from age to *LifeInsPromo*.
- Go to the Choose command line to locate and load *NaiveBayes*.
- Click on More options, mouse to *Output Predictions*, click *Choose*, and then click on *plain text*.
- Use a 15-fold cross-validation as the test option. For each iteration of the cross-validation, 14 instances will be used for model building and 1 instance for testing. Click start.
- Scroll your display screen until it appears similar to Figure 11.6.

Figure 11.6 represents the final model built from all 15 instances. To avoid the possibility of zero-valued conditional probabilities with categorical data, Weka's naïve Bayes

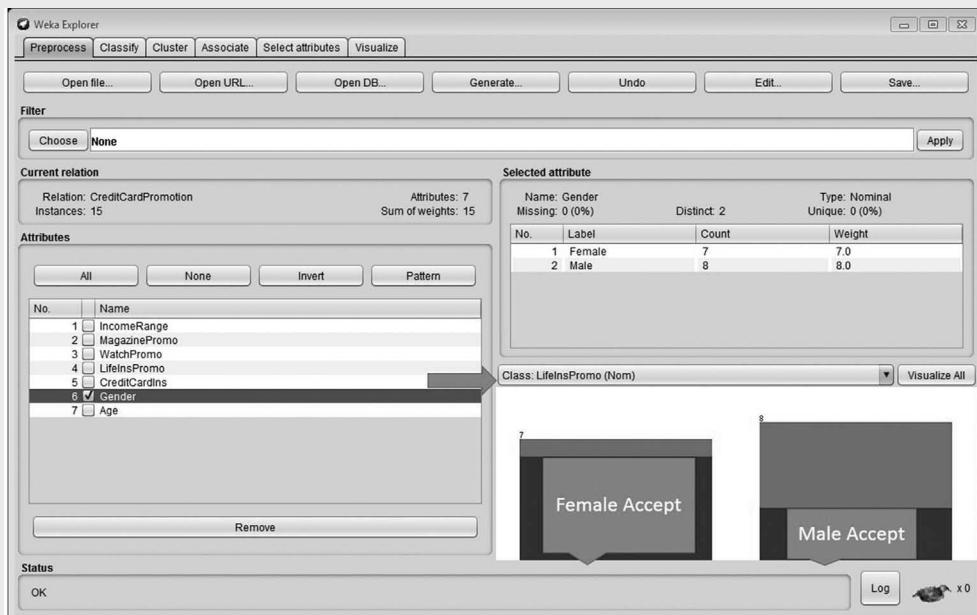


FIGURE 11.5 Life insurance promotion by gender.

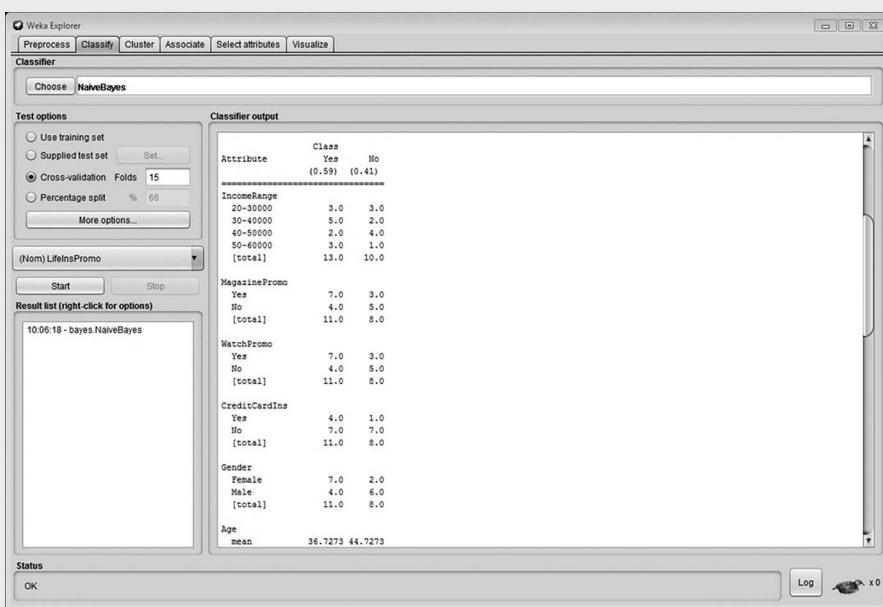


FIGURE 11.6 Naïve Bayes model with output attribute = LifeInsPromo.

classifier uses a variation of Equation 11.4 known as the *Laplace* correction parameter, where 1 is added to each categorical attribute value count.

- Scroll your screen until it appears as in Figure 11.7.

Here we see actual and predicted outcome values together with corresponding prediction probabilities. As only 15 instances make up our data set, the 66.67% classification correctness score is not surprising.

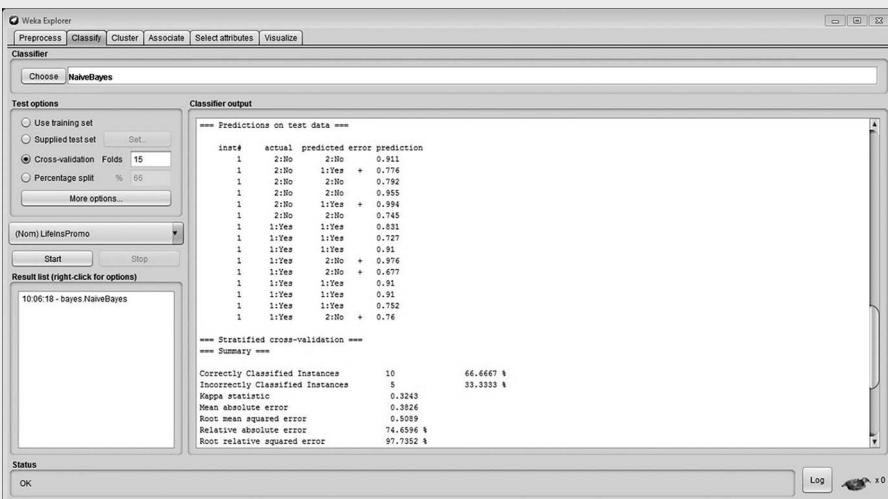


FIGURE 11.7 Predictions for the life insurance promotion.

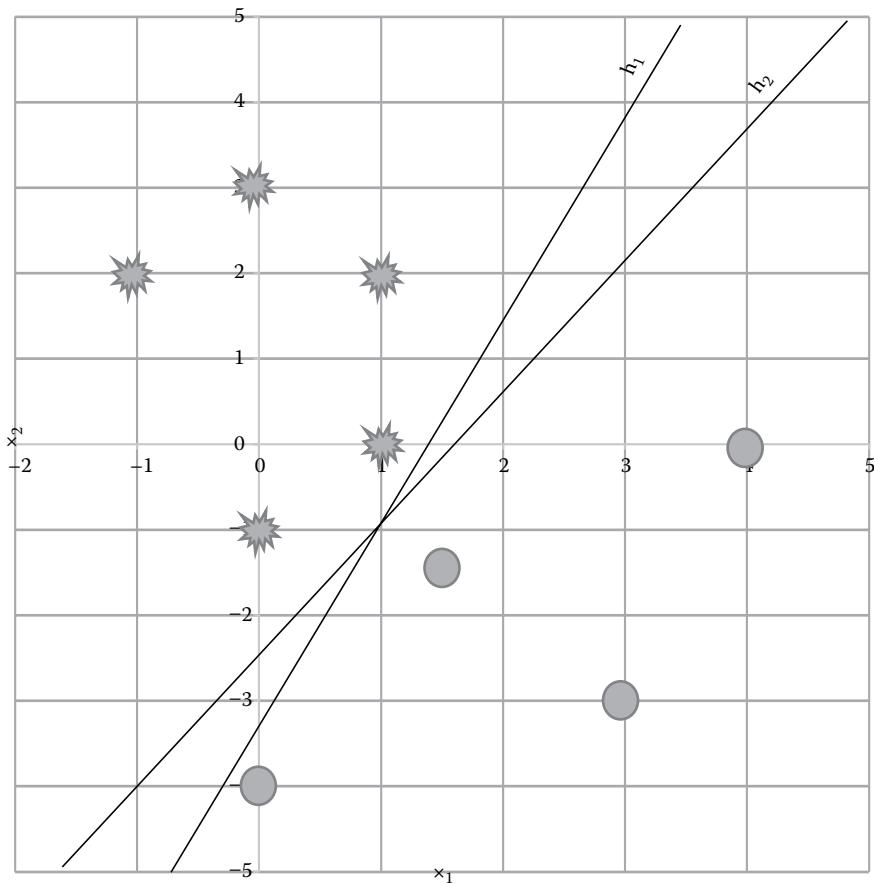


FIGURE 11.8 Hyperplanes separating the circle and star classes.

particular hyperplane is unique and has a special name. It is known as the *maximum margin hyperplane* (MMH). Given a data set having two classes, the job of the SVM algorithm is to find the MMH. Here's how it's done.

An SVM uses the instances lying on the outer marginal boundaries between the classes to find the MMH. To see this, consider Figure 11.9, which shows two hyperplanes each passing through the instances defining the outer margin boundaries for their respective classes. The hyperplane labeled h_1 passes through the two boundary instances for the star class, and h_2 passes through the single boundary instance for the circle class. These boundary instances are known as *support vectors*. The support vectors associated with the MMH have the best chance of differentiating between the classes as they lie along the largest marginal instance gap. The SVM algorithm builds its model using only these critical support vectors. All other class instances are irrelevant. Here is a general description of the process used by the SVM algorithm:

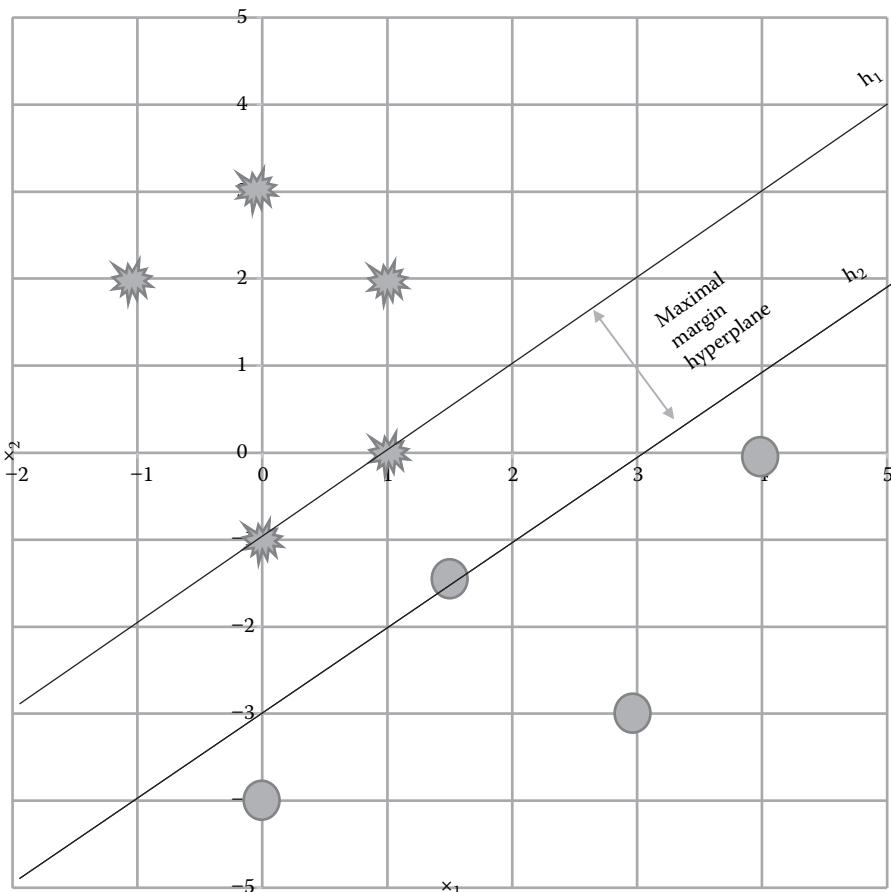


FIGURE 11.9 Hyperplanes passing through their respective support vectors.

- Determine if the instances defining the classes are linearly separable in the current dimension. Provided that the classes are separated by a linear boundary, the support vectors are used to establish the MMH.
- If the classes are not linearly separable, map the instances into consecutively higher dimensions until linear separability is achieved. Once again, the support vectors found in the new higher dimension determine the MMH.

Finding the support vectors and MMH is a problem in constrained quadratic optimization, the details of which are beyond the scope of this book. The important point is that given certain constraints, the MMH and support vectors can be determined in a reasonable amount of time.

In the next section, we present a simple two-dimensional example with linearly separable data. We follow our example with a discussion of the nonlinear case where we

detail the dimensional mapping process. We complete our study of SVMs with general considerations and a demonstration of the SVM software available with Weka and RapidMiner.

11.2.1 Linearly Separable Classes

When discussing SVMs, it is useful to reference instances as vectors. A *vector* is a quantity having both direction and magnitude. Instances can be displayed horizontally as row vectors or vertically as a column vectors. We denote vectors in boldface type.

One vector operation of particular importance is the *dot product*. The dot product can be thought of as a measure of similarity. Algebraically, given two vectors of equal length, the dot product of two vectors is the sum of the products of their corresponding entries. For example, given $\mathbf{v} = (v_1, v_2)$ and $\mathbf{w} = (w_1, w_2)$, then $\mathbf{v} \cdot \mathbf{w} = (v_1w_1 + v_2w_2)$. It is common to see the previous equation written as $\mathbf{v} \cdot \mathbf{w}^T = (v_1w_1 + v_2w_2)$, where \mathbf{v} is the row vector and \mathbf{w}^T is the transpose of row vector \mathbf{w} . This information together with Figure 11.9 gives us what we need for our first example!

The coordinate axes in Figure 11.9 are labeled x_1 and x_2 rather than x and y . The rationale is that x_1 and x_2 are both independent, whereas the class associated with each vector is dependent. Here, each vector instance is a member of one of two classes. The class marked by stars contains instances $(0, -1), (0, 3), (1, 0), (1, 2)$, and $(-1, 2)$. The circle class shows instances $(0, -4), (1.5, -1.5), (3, -3)$, and $(4, 0)$. We have already visually determined the classes to be linearly separable. Our goal is to find the MMH separating the classes. To find the MMH, we must locate the support vectors. For our simple example, this task is easy.

Figure 11.9 shows the two support vectors for the star class to be $(1, 0)$ and $(0, -1)$. The figure also shows $(1.5, -1.5)$ as the lone support vector for the circle class. If there is any doubt about which instances represent support vectors, we can use simple Euclidean distance to either determine or verify our choices. For this example, the Euclidean distance between $(1.5, -1.5)$ and $(1, 0)$ is $\sqrt{2.5}$ as is the distance between $(1.5, -1.5)$ and $(0, -1)$. As all other between-class vector distances are greater than $\sqrt{2.5}$, only these three vectors are relevant to the SVM algorithm.

The class with support vectors $(1, 0)$ and $(0, -1)$ clearly lies above the margin of separation. By convention, this class is denoted as the positive class and is given a label of 1. The support vector $(1.5, -1.5)$ represents the remaining class and lies below the class margin. Therefore, $(1.5, -1.5)$ is a negative example. Convention tells us to give this class a label of -1 . Although this labeling scheme is standard, choosing the values to represent each respective class is arbitrary other than having all support vectors on the upper part of the margin get the same positive value and all vectors on the lower margin receive the corresponding negative value.

Next, establishing the MMH requires a general form for a two-dimensional equation. We have two choices. Here is a standard representation of a linear equation in two dimensions with a constant (bias) term included.

$$w_0 + w_1x_1 + w_2x_2 = 0 \quad (11.6)$$

where

x_1 and x_2 are attribute values

w_0 , w_1 , and w_2 are parameters to be learned

However, if the support vectors are modified by adding a term to account for the bias, we can also write the MMH in terms of its support vectors in the following way:

$$\sum a_i \mathbf{x}_i \cdot \mathbf{x} = 0 \quad (11.7)$$

where

a_i is the learned parameter associated with the i th support vector

\mathbf{x}_i is the i th support vector

\mathbf{x} is the vector instance to be classified

Equation 11.7 better suits our needs for determining the MMH. To use equation 11.7, we first supplement each support vector with a bias of 1 to account for the constant term in the equation. To match Equation 11.6, we add the bias as the first term. The modified class 1 support vectors now appear as $\mathbf{sv}_1 = (1, 1, 0)$ and $\mathbf{sv}_2 = (1, 0, -1)$. The modified support vector for the class labeled -1 is $\mathbf{sv}_3 = (1, 1.5, -1.5)$. Knowing that \mathbf{x} in Equation 11.7 represents an instance to be classified, the assignment of $a_0 = w_0$, $a_1 = w_1$, and $a_2 = w_2$ confirms the equivalency of Equations 11.6 and 11.7.

We need one more equation, specifically, a vector representation for the parameters making up the MMH — w_0 , w_1 , and w_2 . The parameters can be given in vector notation as

$$\mathbf{w} = \sum a_i \mathbf{sv}_i \quad (11.8)$$

where the

a_i values are to be learned

\mathbf{sv}_i represents the i th support vector

With three support vectors and the above equations, determining the MMH is a matter of solving three equations in three unknowns.

Consider the augmented support vector $\mathbf{x} = (1, 1, 0)$. Using Equation 11.7 and the fact that $(1, 1, 0)$ lies on the hyperplane represented by positive 1, it must satisfy this condition.

$$a_1(1, 1, 0) \cdot (1, 1, 0)^T + a_2(1, 0, -1) \cdot (1, 1, 0)^T + a_3(1, 1.5, -1.5) \cdot (1, 1, 0)^T = 1$$

In a like manner, $(1, 0, -1)$ must satisfy

$$a_1(1, 1, 0) \cdot (1, 0, -1)^T + a_2(1, 0, -1) \cdot (1, 0, -1)^T + a_3(1, 1.5, -1.5) \cdot (1, 0, -1)^T = 1$$

Finally, $(1, 1.5, -1.5)$ is on the hyperplane of negative examples, so it must satisfy

$$a_1(1, 1, 0) \cdot (1, 1.5, -1.5)^T + a_2(1, 0, -1) \cdot (1, 1.5, -1.5)^T + a_3(1, 1.5, -1.5) \cdot (1, 1.5, -1.5)^T = -1$$

Computing the dot products, we have

$$2a_1 + a_2 + 2.5a_3 = 1$$

$$a_1 + 2a_2 + 2.5a_3 = 1$$

$$2.5a_1 + 2.5a_2 + 5.5a_3 = -1$$

Solving the system of equations gives us

$$a_1 = 2 \quad a_2 = 2 \quad a_3 = -2$$

With values for a_1 , a_2 , and a_3 , we use Equation 11.8 to determine w_0 , w_1 , and w_2 .

$$(w_0, w_1, w_2) = 2(1, 1, 0) + 2(1, 0, -1) - 2(1, 1.5, -1.5)$$

This gives us

$$w_0 = 2, \quad w_1 = -1, \quad \text{and} \quad w_2 = 1.$$

Therefore, the equation for the MMH is

$$2 - x_1 + x_2 = 0 \tag{11.9}$$

Figure 11.10 displays the equation of the MMH together with the equations for the two boundary hyperplanes. Using the general form of the equations for the decision boundaries, it can be shown that the distance d between the two hyperplanes is given by $2/\|w\|$, where $\|w\|$ is the Euclidean norm of w defined as the square root of the dot product $w \cdot w$. That is,

$$\|w\| = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2} \tag{11.10}$$

During training, the support vector algorithm begins by estimating the parameters of the two decision boundaries under certain constraints. One obvious constraint is the maximization of this margin. For our example, Equation 11.10 tells us that the width of the maximal margin = $\sqrt{w_1^2 + w_2^2}$, which computes to $\sqrt{2}$.

Knowing the support vectors and the MMH, we can apply our model to new instances for classification. Two alternatives present themselves. We can use the MMH by placing the x_1 and x_2 values of an unclassified instance into Equation 11.9. If the result is greater than 0, we classify the instance with class label 1. If the value is negative, it is placed with the class

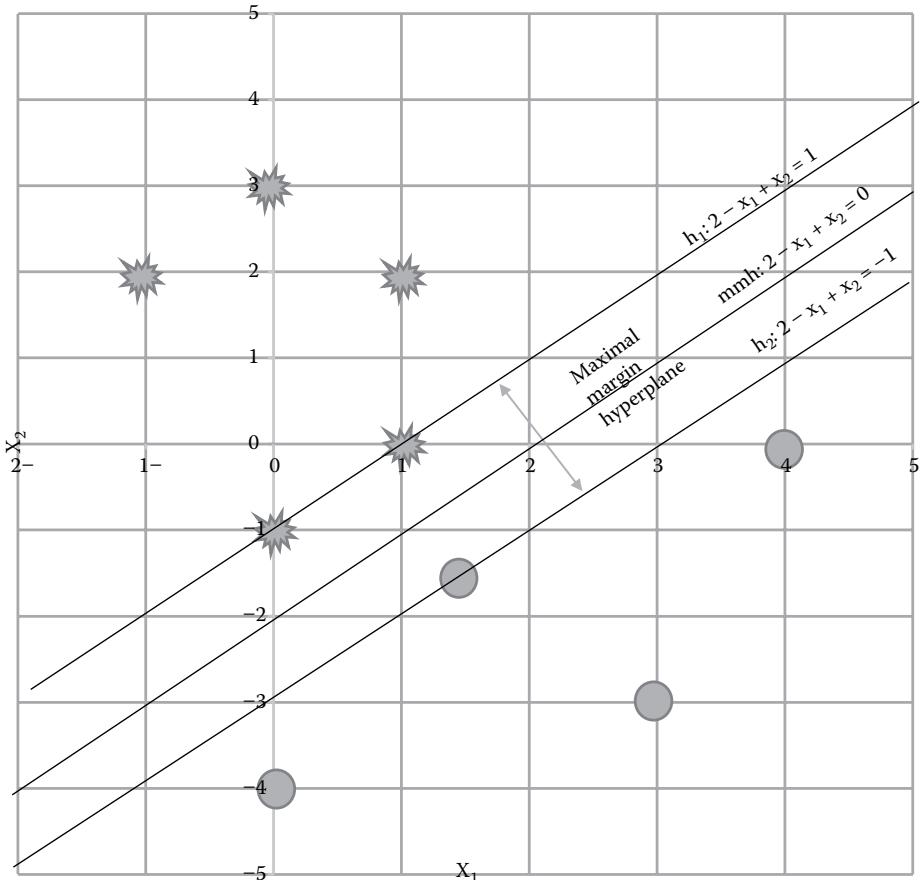


FIGURE 11.10 Maximal margin hyperplane separating the star and circle classes.

labeled -1 . Alternatively, we can use the equation for the MMH in terms of the support vectors, but this time with the bias included as the constant term. Specifically,

$$y = b + \sum a_i \mathbf{x}_i \cdot \mathbf{x} \quad (11.11)$$

where

a_i is the learned parameter associated with the i th support vector

b is the learned constant term

\mathbf{x}_i is the i th support vector

\mathbf{x} is the vector instance to be classified

If y is positive, we classify an unknown instance with class label 1 . Likewise, if y proves to be negative, the instance is associated with the class labeled -1 . If $y = 0$, the instance lies on the MMH and is classified with the most commonly occurring class. Let's use Equation 11.11 to determine the classification of the instance $(0.5, 0)$.

To make our computation, we remove the bias term from our support vectors and include the value 2 for b . Applying the previously computed values for a_1 , a_2 , and a_3 , we have

$$y = 2 + 2[(1, 0) \cdot (0.5, 0)^T] + 2[(0, -1) \cdot (0.5, 0)^T] - 2[(1.5, -1.5) \cdot (0.5, 0)^T]$$

This gives us a value of 1.5 for y , which tells us that $(0.5, 0)$ lies above the MMH and is classified with the vectors labeled 1. Using Equation 11.11 to classify $(0, -5)$ gives a value of -3 , telling us $(0, -5)$ is classified with the class labeled -1 .

Using Equation 11.11 to classify support vector $(1, 0)$ gives a value of 1. This is expected as $(1, 0)$ lies on hyperplane h_1 . Likewise, applying Equation 11.11 to $(1.5, -1.5)$ offers a result of -1 . Using the model to classify $(3, 1)$ gives a value of 0. This tells us that $(3, 1)$ is on the MMH.

Before we proceed to the nonlinear case, it is obvious that a linear model is preferred especially if an acceptable level of accuracy is seen. That said, it is worth noting that if the classes are almost linearly separable in the current dimension, a slight modification of the SVM algorithm gives it the capability of creating a linear model.

11.2.2 The Nonlinear Case

The strength of SVMs lies in their ability to classify in clearly nonlinear domains. When the instances in the current dimensional space are not linearly separable, an SVM applies a nonlinear mapping, thereby transforming the instances into a new, likely higher dimension. An unacceptable degree of complexity can enter the picture unless the transformation is accomplished with a mapping function having certain special characteristics. Let's see why this is the case.

Consider the two dimensional vectors \mathbf{x} and \mathbf{z} with respective attribute values (x_1, x_2) and (z_1, z_2) . Suppose that in an attempt to achieve linearity, our function transforms the current two-dimensional space by applying the mapping

$$(x_1, x_2) \rightarrow (x_1, x_2, x_1^2, x_1 x_2, x_2^2)$$

Prior to this mapping, the dot product computation for each vector pair requires one multiplication and one addition. In the new space, the dot product of each vector pair \mathbf{x} and \mathbf{z} entails five multiplications and four additions. Specifically,

$$\mathbf{x} \cdot \mathbf{z} = (x_1 z_1, x_2 z_2, x_1^2 z_1^2, x_1 x_2 z_1 z_2, x_2^2 z_2^2)$$

Once a model has been established, model application also requires the same dot product computations. However, given the right type of mapping these added computations can be completely avoided! The process requires using a special mapping function known as a *kernel function*. For a function K to qualify as a kernel function, the mapping function ϕ must be chosen such that

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \quad (11.12)$$

To illustrate a function with this property, consider the mapping ϕ with $\mathbf{x} = (x_1, x_2)$

$$\phi(x_1, x_2) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

Computing the dot product of $\phi(x_1, x_2)$ and $\phi(z_1, z_2)$ and factoring gives us

$$\phi(x_1, x_2) \cdot \phi(z_1, z_2) = (1 + x_1z_1 + x_2z_2)^2$$

This tells us that if we assign

$$K(\mathbf{x}, \mathbf{z}) = (1 + x_1z_1 + x_2z_2)^2 \quad (11.13)$$

the property stated in Equation 11.12 is satisfied and K qualifies as a kernel function. More importantly, it tells us that the dot product computations can be performed in the original space rather than in the higher dimension! The general form of Equation 11.13 is known as the polynomial kernel of degree n and is given as

$$K(\mathbf{x}, \mathbf{z}) = (1 + x_1z_1 + x_2z_2)^n \quad (11.14)$$

In general, every location within the training algorithm where the dot product representation $\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ appears can be replaced by $K(\mathbf{x}, \mathbf{z})$. This is true for both model training and application. In addition to the polynomial function, several other kernel functions have been studied, some of which are built into the SVM models of both Weka and RapidMiner.

11.2.3 General Considerations

Even with the use of a kernel function, SVMs cannot compete with the classification times seen with decision trees and several other techniques. However, their accuracy in difficult-to-classify domains makes SVMs a valuable classification tool. Of particular interest is the use of SVMs in finding anomalies in big data.

Added difficulties with SVMs include questions about computational complexity, dimensionality (finding a minimal dimension for achieving linear separation), choosing appropriate attributes, and mapping functions. SVMs are also limited to two class problems with numeric input.

The last issues have been addressed in that we know how categorical input attributes can be transformed to numeric values by simply making each attribute value a binary attribute. Also, the two-class limitation has been dealt with in several ways. A common method known as *one-versus-all* builds a classifier model for each class. For n classes, we have n models. An unknown instance is classified with the class of closest association. The *one-vs-one* method builds a classifier for each pair of classes. Given n classes, a total of $n(n-1)/2$ classes are constructed. An unknown instance is classified with the class getting the most votes.

WEKA TUTORIAL

Support Vector Machines—Weka

SMO is a supervised learner that automatically transforms nominal attributes into binary values and globally replaces missing values with attribute means. It also normalizes all attributes by default. SMO invokes the polynomial kernel discussed above, but others can be used. The one-versus-one method is applied in domains with more than two classes. Let's look at our example.

- Load SVMexample.arff into the Explorer. This data set contains the nine instances used for the linearly separable example above.
- While in Preprocess mode, place a check mark in the class attribute. Your screen will appear as in Figure 11.11.
- Move to Classify mode. Use Choose to locate SMO in the functions folder.
- Change the Cross-validation value to 9. The result is seen in Figure 11.12.

Recall that we did not normalize our linearly separable data prior to creating the MMH of Figure 11.9. Therefore, in order to compare the output given by SMO with our example, we must inhibit data normalization. Here's how.

- Click SMO in the Choose box to invoke the GUI object editor. Change the *filter type* to No normalization/standardization—Figure 11.13. Click OK.
- Click start to see the equation displayed in Figure 11.14. As expected, the output is identical to Equation 11.9.

Figure 11.14 makes it clear that the linear model created by SMO is identical to Equation 11.9.

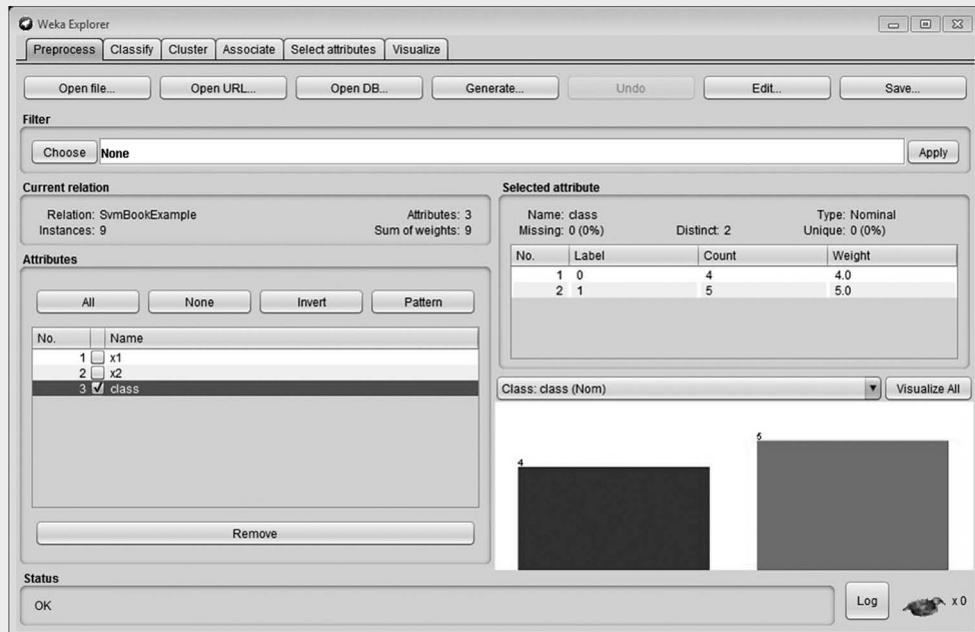


FIGURE 11.11 Loading the nine instances of Figure 11.8 into the Explorer.

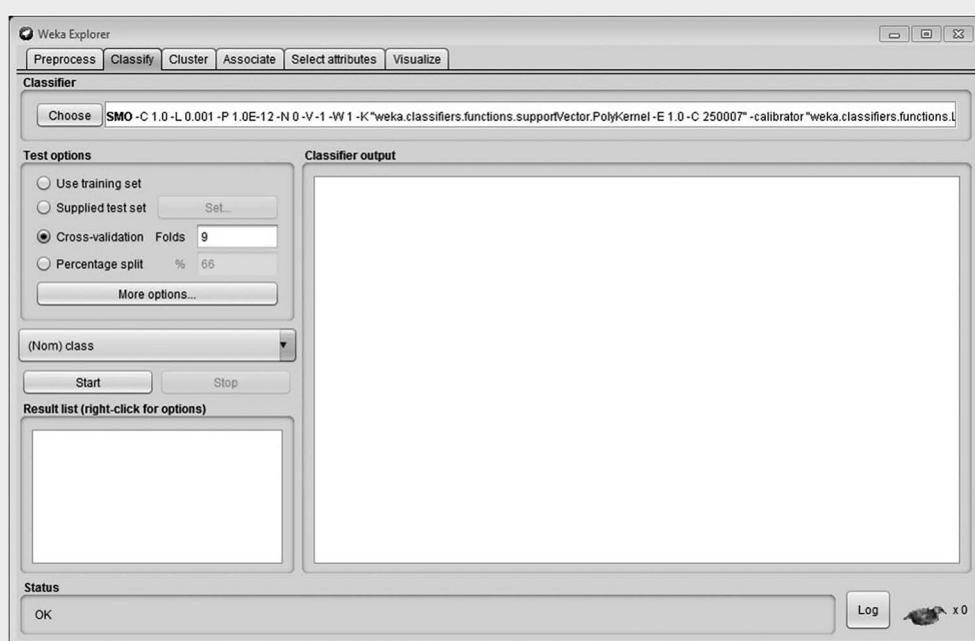


FIGURE 11.12 Invoking SMO model.

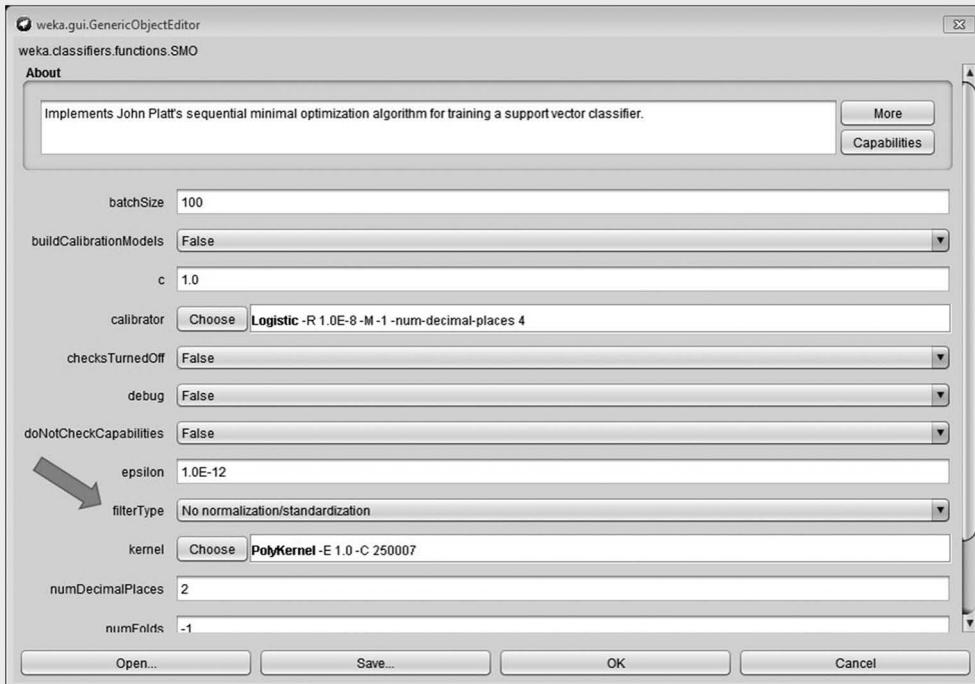


FIGURE 11.13 Disabling data normalization/standardization.

SMO

Kernel used:

Linear Kernel: $K(x,y) = \langle x,y \rangle$

Classifier for classes: 0, 1

BinarySMO

Machine linear: showing attribute weights, not support vectors.

$$\begin{array}{r} -1 * x_1 \\ + 1 * x_2 \\ + 2 \end{array}$$

Number of kernel evaluations: 40 (73.856% cached)

FIGURE 11.14 The SMO-created MMH for the data shown in Figure 11.8.

It is important to point out that data normalization is the rule. However, as the input attributes for our example are of the same scale, data normalization is not needed. In fact, unnecessary normalization/standardization with small data sets can be detrimental. This is the case here as the model resulting from the normalized data actually results in several misclassifications. You will learn much more about applying SMO while exploring the end-of-chapter exercises.

11.2.4 Implementations of Support Vector Machines

Weka implements John Platt's (1998) sequential minimum optimization algorithm (SMO) and a support vector model for regression (SMOreg). RapidMiner provides several SVM models, including one that uses an evolutionary approach for solving the SVM optimization problem. We first use the data from our linearly separable example above to illustrate Weka's SMO algorithm. We then illustrate one of RapidMiner's SVM operators in the highlighted section titled: *Support Vector Machines–RapidMiner*. Let's get started!

11.3 LINEAR REGRESSION ANALYSIS

Statistical *regression* is a supervised technique that generalizes a set of numeric data by creating a mathematical equation relating one or more input attributes to a single output attribute. With *linear regression*, we attempt to model the variation in a dependent variable as a linear combination of one or more independent variables. A linear regression equation is of the form

$$f(x_1, x_2, x_3, \dots, x_n) = a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n + c \quad (11.15)$$

where $x_1, x_2, x_3, \dots, x_n$ are independent variables and $a_1, a_2, a_3, \dots, a_n$ and c are constants. $f(x_1, x_2, x_3, \dots, x_n)$ represents the dependent variable and is often shown simply as y . In general, linear

RAPIDMINER TUTORIAL

Support Vector Machines—RapidMiner

RapidMiner provides several SVM operators. Here we apply Stefan Rueping's implementation of mySVM to the cardiology patient data set. As mySVM is based on Vapnik's original model, it represents an excellent starting point for examining the support vector approach.

Figure 11.15 displays the main process for our illustration. To follow our example,

- Implement the operators in Figure 11.15. Be sure to use the *Performance* (classification) operator.
- Nominal to Numerical is necessary as SVM only accepts numeric input. Data normalization is preferred as value ranges widely vary between the individual attributes. Our example uses a 2/3 training 1/3 test set scenario (Chapter 5, Figure 5.22). The polynomial kernel is used; however, the kernel degree of 1 makes a linear model. The C parameter is a complexity constant that defaults to 0. Higher values of C allow for more misclassifications. You can learn more about this and other SVM parameters in the documentation.
- Place a *Breakpoint After* within the Normalize, SVM, and Apply Model operators.
 - Run your process.

Figure 11.16 shows the normalized example set. Figure 11.17 gives us the attribute weights. Figure 11.18 gives us actual and predicted class values. Figure 11.19 displays the performance vector, which shows an 84.47% test set accuracy. End-of-chapter exercise 3 asks you to perform this experiment using cross validation rather than a training-test set scenario.

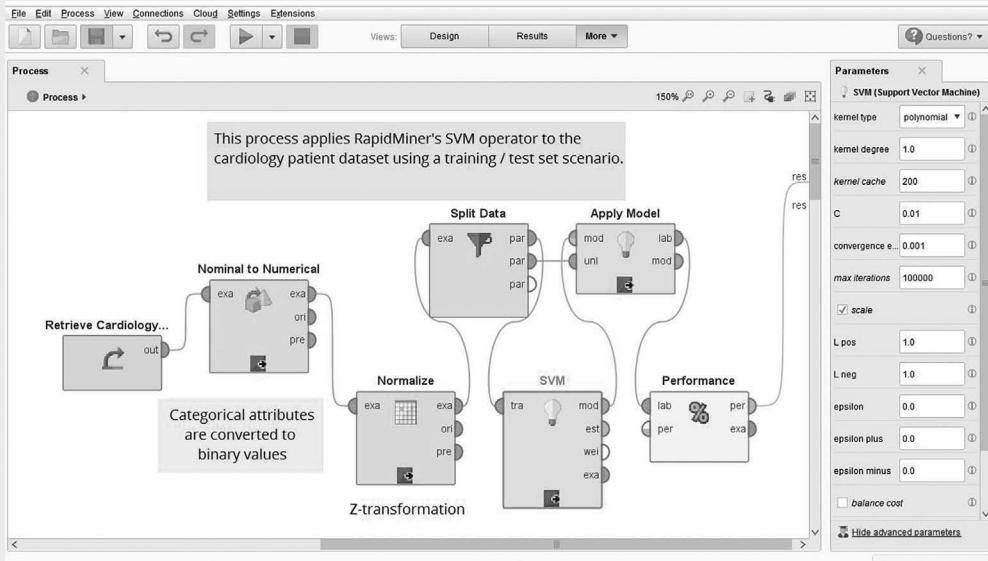


FIGURE 11.15 Applying mySVM to the cardiology patient data set.

Row No.	class	sex = Male	sex = Female	chest pain L.	chest pain L.	chest pain L.	chest pain L.	Fasting blo...	Fasting blo...	resting ecg ...	resting ecg ...	angina = true	angina
1	Sick	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028	-1.002	1.433	-1.433
2	Healthy	0.680	-0.680	-0.944	2.246	-0.286	-0.634	0.417	-0.417	-0.969	0.995	-0.695	0.695
3	Healthy	0.680	-0.680	-0.944	-0.444	3.483	-0.634	0.417	-0.417	1.028	-1.002	1.433	-1.433
4	Sick	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028	-1.002	-0.695	0.695
5	Sick	0.680	-0.680	1.056	-0.444	-0.286	-0.634	-2.390	2.390	1.028	-1.002	1.433	-1.433
6	Healthy	-1.466	1.466	-0.944	-0.444	3.483	-0.634	-2.390	2.390	1.028	-1.002	-0.695	0.695
7	Sick	0.680	-0.680	-0.944	2.246	-0.286	-0.634	0.417	-0.417	1.028	-1.002	-0.695	0.695
8	Sick	0.680	-0.680	-0.944	-0.444	-0.286	1.573	0.417	-0.417	1.028	-1.002	-0.695	0.695
9	Healthy	0.680	-0.680	-0.944	-0.444	3.483	-0.634	-2.390	2.390	1.028	-1.002	-0.695	0.695
10	Sick	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028	-1.002	1.433	-1.433
11	Sick	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028	-1.002	1.433	-1.433
12	Healthy	0.680	-0.680	-0.944	-0.444	-0.286	1.573	0.417	-0.417	-0.969	0.995	-0.695	0.695
13	Healthy	-1.466	1.466	-0.944	2.246	-0.286	-0.634	0.417	-0.417	1.028	-1.002	-0.695	0.695
14	Healthy	0.680	-0.680	-0.944	2.246	-0.286	-0.634	0.417	-0.417	-0.969	0.995	-0.695	0.695
15	Sick	-1.466	1.466	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028	-1.002	-0.695	0.695
16	Healthy	-1.466	1.466	1.056	-0.444	-0.286	-0.634	0.417	-0.417	-0.969	0.995	1.433	-1.433
17	Healthy	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	-0.969	0.995	-0.695	0.695
18	Healthy	-1.466	1.466	-0.944	2.246	-0.286	-0.634	0.417	-0.417	1.028	-1.002	-0.695	0.695
19	Sick	0.680	-0.680	-0.944	-0.444	-0.286	1.573	-2.390	2.390	1.028	-1.002	1.433	-1.433

FIGURE 11.16 Normalized cardiology patient data.

Total number of Support Vectors: 200
Bias (offset): 0.356
w[sex = Male] = -0.060
w[sex = Female] = 0.060
w[chest pain type = Asymptomatic] = -0.192
w[chest pain type = Abnormal Angina] = 0.090
w[chest pain type = Angina] = 0.079
w[chest pain type = Notang] = 0.087
w[Fasting blood sugar <120 = false] = 0.014
w[Fasting blood sugar <120 = true] = -0.014
w[resting ecg = Nyp] = -0.001
w[resting ecg = Normal] = 0.024
w[angina = true] = -0.098
w[angina = false] = 0.098
w[slope = Flat] = -0.109
w[slope = Up] = 0.112
w[slope = Down] = -0.004
w[thal = Revl] = -0.124
w[thal = Normal] = 0.187
w[thal = Fixl] = 0.011
w[age] = -0.034
w[blood pressure] = -0.078
w[cholesterol] = -0.002
w[maximum heart rate] = 0.124
w[peak] = -0.099
w[#colored vessels] = -0.290

FIGURE 11.17 Equation of the MMH for the cardiology patient data set.

File Edit Process View Connections Cloud Settings Extensions

Views Design Results Questions?

Result History ExampleSet (Split Data)

Data Statistics Charts Advanced Charts Annotations

ExampleSet (103 examples, 4 special attributes, 24 regular attributes)

Filter (103 / 103 examples): all

Row No.	class	prediction(...)	confidence(...)	confidence(...)	sex = Male	sex = Female	chest pain L...	chest pain L...	chest pain L...	chest pain L...	Fasting bloo...	Fasting bloo...	restin
1	Sick	Sick	0.848	0.152	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028
2	Healthy	Healthy	0.160	0.840	0.680	-0.680	-0.944	2.246	-0.286	-0.634	0.417	-0.417	-0.969
3	Sick	Healthy	0.268	0.732	0.680	-0.680	-0.944	2.246	-0.286	-0.634	0.417	-0.417	1.028
4	Sick	Sick	0.528	0.472	0.680	-0.680	-0.944	-0.444	-0.286	1.573	0.417	-0.417	1.028
5	Sick	Sick	0.851	0.149	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028
6	Sick	Sick	0.511	0.489	-1.466	1.466	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028
7	Healthy	Healthy	0.234	0.766	-1.466	1.466	-0.944	2.246	-0.286	-0.634	0.417	-0.417	1.028
8	Sick	Sick	0.560	0.440	0.680	-0.680	-0.944	-0.444	-0.286	1.573	-2.390	2.390	1.028
9	Sick	Sick	0.777	0.223	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	-0.969
10	Sick	Sick	0.809	0.191	-1.466	1.466	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028
11	Healthy	Healthy	0.387	0.613	0.680	-0.680	-0.944	-0.444	3.483	-0.634	0.417	-0.417	-0.969
12	Healthy	Healthy	0.226	0.774	0.680	-0.680	-0.944	-0.444	-0.286	1.573	-2.390	2.390	-0.969
13	Sick	Healthy	0.228	0.772	-1.466	1.466	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028
14	Sick	Sick	0.681	0.319	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028
15	Healthy	Healthy	0.231	0.769	-1.466	1.466	-0.944	-0.444	-0.286	1.573	-2.390	2.390	1.028
16	Healthy	Healthy	0.483	0.517	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	-0.969
17	Sick	Healthy	0.330	0.670	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028
18	Sick	Sick	0.682	0.318	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	-0.969
19	Sick	Sick	0.814	0.186	0.680	-0.680	1.056	-0.444	-0.286	-0.634	0.417	-0.417	1.028

[1] Process 2:57

FIGURE 11.18 Actual and predicted output for the cardiology patient data.

File Edit Process View Connections Cloud Settings Extensions

Views Design Results Questions?

Result History PerformanceVector (Performance)

Performance Description Annotations

Criterion accuracy

accuracy: 84.47%

	true Sick	true Healthy	class precision
pred. Sick	43	2	95.55%
pred. Healthy	14	44	75.86%
class recall	75.44%	95.65%	

FIGURE 11.19 Performance vector for the cardiology patient data.

regression is a favorite among statisticians and tends to work well when the relationship between the dependent and independent variables is nearly linear.

11.3.1 Simple Linear Regression

The simplest form of the linear regression equation allows but a single independent variable as the predictor of the dependent variable. This type of regression is appropriately named *simple linear regression*. The regression equation is written in *slope–intercept form*. Specifically,

$$y = ax + b \quad (11.16)$$

where x is the independent variable and y depends on x . The constants a and b are computed via supervised learning by applying a statistical criterion to a data set of known values for x and y . The graph of Equation 11.16 is a straight line with slope a and y -intercept b .

A common statistical measure used to compute a and b is the *least-squares criterion*. The least-squares criterion minimizes the sum of squared differences between actual and predicted output values. Deriving a and b via the least-squares method requires a knowledge of differential calculus. Therefore, we simply state the formulas for computing a and b . For a total of n instances, we have

$$b = \frac{\Sigma xy}{\Sigma x^2} \quad a = \frac{\Sigma y}{\Sigma n} - \frac{b \Sigma y}{n} \quad (11.17)$$

Simple linear regression is easy both to understand and to apply. However, the fact that it allows but a single independent variable makes the technique of limited use for most data mining applications.

11.3.2 Multiple Linear Regression

The least-squares criterion is also used to create linear regression equations of more than two variables. With Equation 11.15 written as

$$y = \sum a_i x_i + c$$

and i varying from 1 to n , the regression algorithm determines c and the a_i values so as to minimize—over all instances—the sum of squared differences between actual and predicted values for y .

11.3.2.1 Linear Regression—Weka

We applied Weka's linear regression model to *RegressionExample.arff*, whose instances are identical to the nine-instance file used in Section 11.2 with SMO. The only exception is that the class attribute has been declared numeric. Figure 11.20 displays the outcome of our experiment. Although these data are linearly separable, a mean absolute error of 0.43 is less than optimal. This is likely due to the small amount of provided data.

```
class = -0.1536 * x1 +
       0.1362 * x2 + 0.7555
```

==== Predictions on test data ====

inst#	actual	predicted	error
1	0	0.934	0.934
1	0	0.678	0.678
1	1	0.551	-0.449
1	0	-0.185	-0.185
1	1	1.287	0.287
1	1	0.835	-0.165
1	1	1.257	0.257
1	1	0.524	-0.476
1	0	0.376	0.376

Correlation coefficient	0.5078
Mean absolute error	0.4229

FIGURE 11.20 A linear regression model for the instances of Figure 11.8.

11.3.2.2 Linear Regression—RapidMiner

As a second experiment, we applied RapidMiner's linear regression operator to the gamma-ray burst data set first introduced in Chapter 5. Figure 11.21 shows that the Set Role operator is used to set t90 as the output attribute. Further, Set Role is used to declare the *burst* attribute as an id as it is a unique identifier. Figure 11.22 displays the linear regression operator together with the apply model and performance operators. A breakpoint was set in *Apply Model* to allow us to investigate individual predictions.

At the first breakpoint, Figure 11.23 shows a partial list of actual and predicted t90 values. When the process completes, Figure 11.24 gives us the coefficients for the linear regression equation and added statistical information. Click on *Description* to limit the output to the linear regression equation. The regression equation shows that attribute t50 is the attribute of greatest significance in determining the output. This is the case as t50 is also a measure of burst length and thereby correlates linearly with t90. This is made clear in Figure 11.25, which gives a scatterplot diagram of their relationship. Figure 11.26 displays minimal values for the root mean squared and absolute error values. This result is not surprising given the relationship between t50 and t90. Lastly, we saw a similar outcome when we performed this same experiment using Weka's linear regression model. As you will see in the end-of-chapter exercises, removing t50 from the experiment results in a decidedly different result.

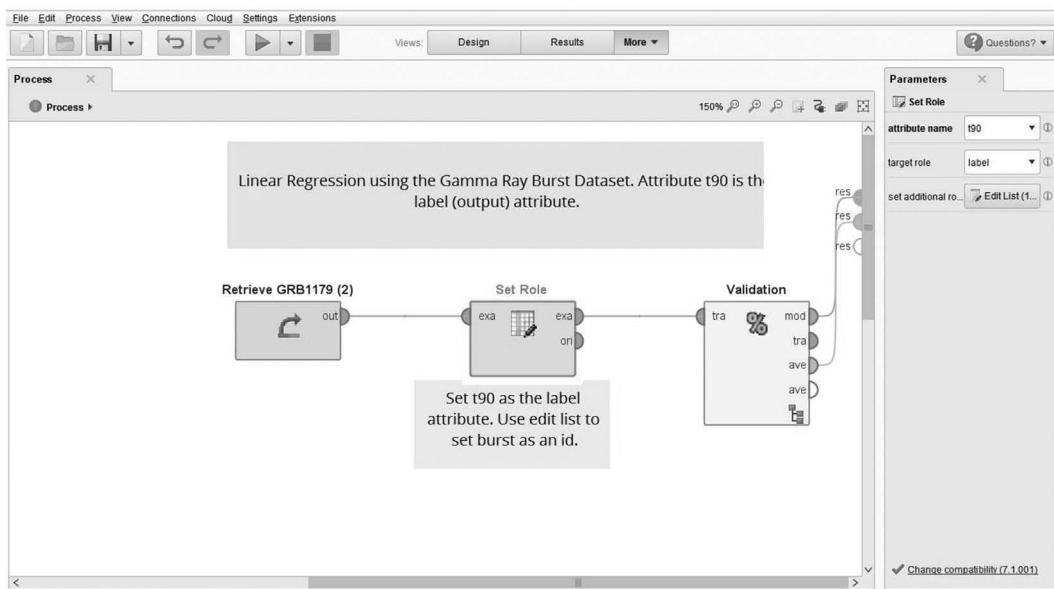


FIGURE 11.21 Main process window for applying RapidMiner’s linear regression operator to the gamma-ray burst data set.

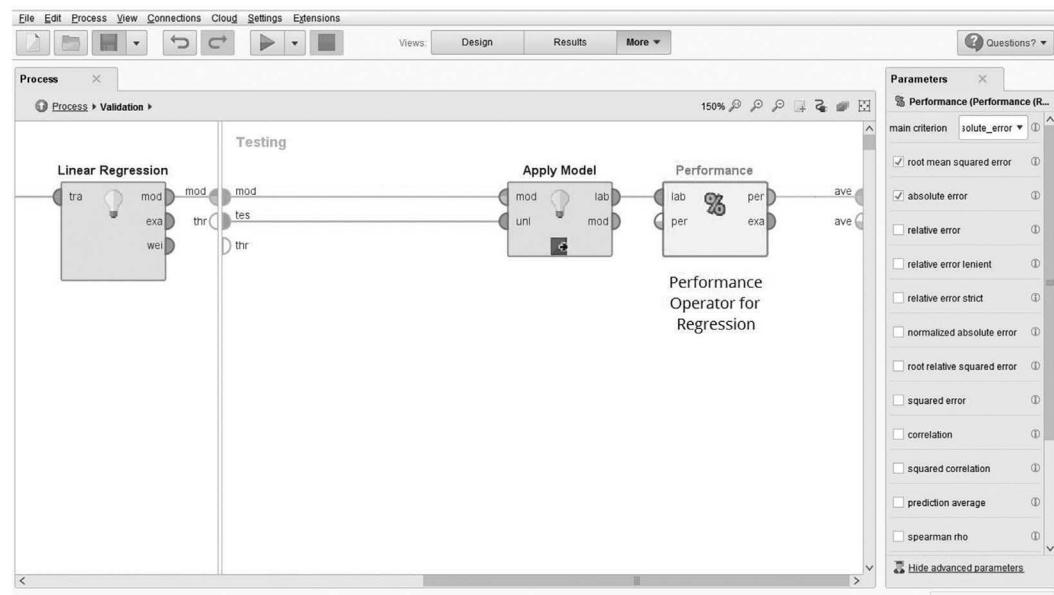


FIGURE 11.22 Subprocess windows for the Gamma Ray burst experiment.

File Edit Process View Connections Cloud Settings Extensions

Views: Design Results More

Result History ExampleSet (Validation)

Filter (118 / 118 examples): all

Data Statistics Charts Advanced Charts Annotations

ExampleSet (118 examples, 2 special attributes, 6 regular attributes)

Row No.	t90	prediction(t..)	burst	p256	fl	hr32	hr321	t50
1	1.424	1.248	1924	0.067	-5.252	0.473	0.219	0.802
2	0.940	1.014	3072	0.028	-5.676	0.439	0.170	0.599
3	0.859	1.031	803	0.072	-5.761	0.464	0.296	0.645
4	1.085	1.151	1452	-0.107	-5.913	0.412	0.276	0.797
5	1.415	1.442	3040	0.230	-5.502	0.410	0.155	1.047
6	1.493	1.495	2985	-0.058	-5.771	0.372	0.177	1.137
7	1.589	1.562	1580	0.082	-5.322	0.534	0.356	1.181
8	1.478	1.344	3805	-0.089	-5.709	0.351	0.172	0.968
9	1.427	1.460	3015	0.244	-5.232	0.530	0.316	1.049
10	1.257	1.212	3168	-0.295	-5.608	0.566	0.190	0.819
11	1.356	1.477	1447	0.241	-5.063	0.529	0.361	1.047
12	1.813	1.619	2505	-0.164	-5.644	0.457	0.228	1.277
13	1.822	1.608	3903	-0.133	-5.546	0.553	0.184	1.250
14	1.526	1.340	1446	-0.316	-5.564	0.528	0.306	0.965
15	1.005	0.877	2087	0.323	-5.503	0.547	0.312	0.440
16	1.807	1.603	3193	-0.185	-5.395	0.481	0.223	1.221
17	1.559	1.562	548	0.300	-5.211	0.371	0.175	1.137
18	1.546	1.511	2664	-0.266	-5.343	0.380	0.137	1.096
19	1.120	1.066	3637	0.431	-5.386	0.519	0.257	0.626

[1] Process 47 s [1] Validation 47 s

FIGURE 11.23 Linear regression—actual and predicted output for the gamma-ray burst data set.

File Edit Process View Connections Cloud Settings Extensions

Views: Design Results More

Result History PerformanceVector (Performance) LinearRegression (Linear Regression)

Data Description Annotations

Attribute	Coefficient	Std. Error	Std. Coefficient	Tolerance	t.Stat	p-Value	Code
fl	0.140	0.010	0.113	0.591	13.558	0	****
hr32	-0.034	0.069	-0.011	0.841	-0.494	0.622	
hr321	-0.105	0.062	-0.037	0.826	-1.702	0.089	*
t50	0.892	0.009	0.883	0.442	96.661	0	****
(Intercept)	1.308	0.067	?	?	19.505	0	****

FIGURE 11.24 Summary statistics and the linear regression equation for the gamma-ray burst data set.

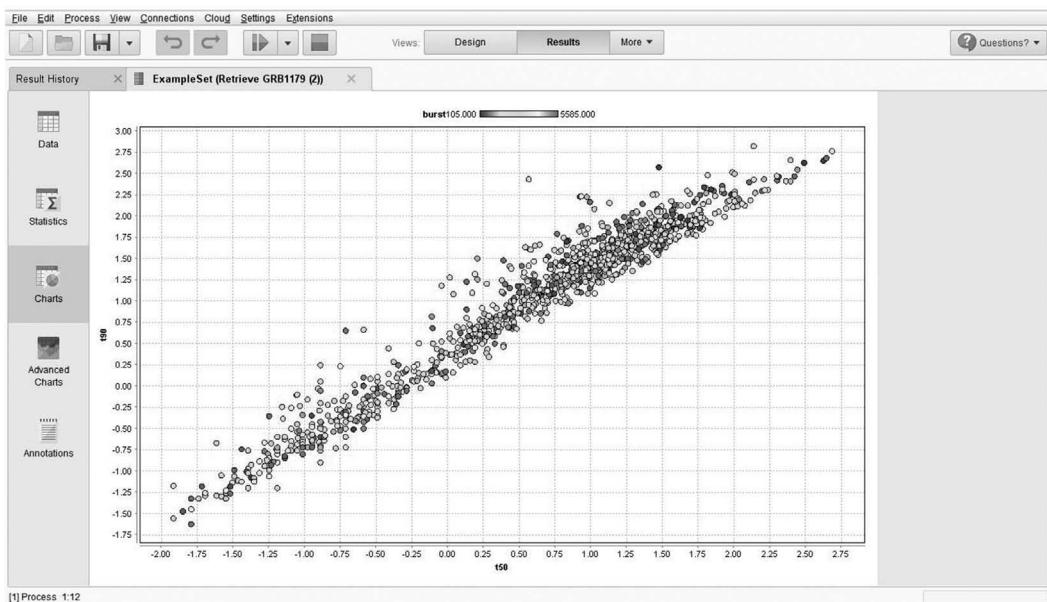


FIGURE 11.25 Scatterplot diagram showing the relationship between t90 and t50.



FIGURE 11.26 Performance vector resulting from the application of linear regression to the gamma-ray burst data set.

11.4 REGRESSION TREES

Regression trees represent an alternative to statistical regression. Regression trees get their name from the fact that most statisticians refer to any model that predicts numeric output as a regression model. Essentially, regression trees take the form of decision trees where the leaf nodes of the tree are numeric rather than categorical values. The value at an individual leaf node is computed by taking the numeric average of the output attribute for all instances passing through the tree to the leaf node position.

Regression trees are more accurate than linear regression equations when the data to be modeled are nonlinear. However, regression trees can become quite cumbersome and difficult to interpret. For this reason, regression trees are sometimes combined with linear regression to form what are known as *model trees*. With model trees, each leaf node of a partial regression tree represents a linear regression equation rather than an average of attribute values. By combining linear regression with regression trees, the regression tree structure can be simplified in that fewer tree levels are necessary to achieve accurate results.

Figure 11.27 presents a generic model tree structure containing four tests on real-valued attributes and five leaf nodes each representing a linear regression equation. The complexity of a model tree depends on the degree of linearity seen between the dependent and independent variables.

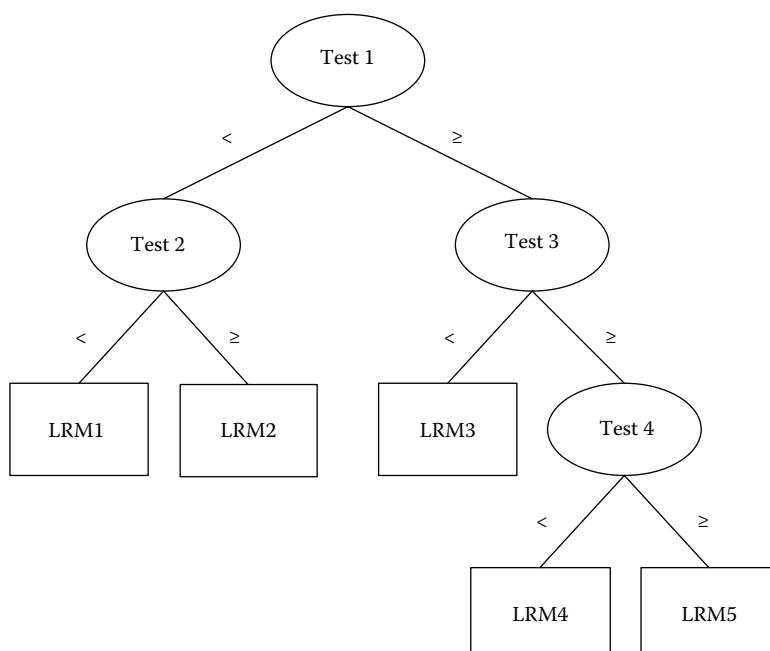


FIGURE 11.27 A generic model tree.

11.5 LOGISTIC REGRESSION

In Chapter 2, we used multiple linear regression with the credit card promotion database to develop an equation for determining values for output attribute *life insurance promotion*. As linear regression requires numeric attribute values, we transformed all categorical data to numeric equivalents. *Yes* and *no* values for attributes *magazine promotion*, *watch promotion*, and *life insurance promotion* were replaced with 1 and 0. For attribute *gender*, values *male* and *female* were also replaced with 1 and 0, respectively. Finally, values for categorical attribute *income range* were transformed to the lower end of each range score.

Our example was appropriate for illustrative purposes. However, the general methodology of using linear regression to model problems with observed outcome restricted to two values is seriously flawed. The problem lies in the fact that the value restriction placed on the dependent variable is not observed by the regression equation. That is because linear regression produces a straight-line function; values of the dependent variable are unbounded in both the positive and negative directions. Therefore, for the right-hand side of Equation 11.15 to be consistent with a binary outcome, we must transform the linear regression model. By transforming the linear model so as to restrict values for the output attribute to the [0, 1] interval range, the regression equation can be thought of as producing a probability of the occurrence or nonoccurrence of a measured event.

Although several options for transforming the linear regression model exist, we restrict our discussion to the logistic model. The logistic model applies a logarithmic transformation that makes the right-hand side of Equation 11.15 an exponential term in the transformed equation.

11.5.1 Transforming the Linear Regression Model

Logistic regression is a nonlinear regression technique that associates a conditional probability score with each data instance. To understand the transformation performed by the logistic model, we begin by thinking of Equation 11.15 as computing a probability. A probability value of 1 denotes the observation of one class (e.g., *life insurance promotion = yes*). Likewise, a probability of 0 indicates observance of the second class (e.g., *life insurance promotion = no*). Equation 11.18 is a modified form of Equation 11.15 where the left-hand side of the equation is written as a conditional probability.

$$p(y = 1|\mathbf{x}) = a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n + c \quad (11.18)$$

Equation 11.18 shows $p(y = 1|\mathbf{x})$ as an unbounded value denoting the conditional probability of seeing the class associated with $y = 1$ given the values contained in feature (attribute) vector \mathbf{x} . To eliminate the boundary problem seen in the equation, the probability is transformed into an odds ratio. Specifically,

$$\left(\frac{p(y = 1|\mathbf{x})}{1 - p(y = 1|\mathbf{x})} \right) \quad (11.19)$$

For any feature vector \mathbf{x} , the odds indicate how often the class associated with $y = 1$ is seen relative to the frequency in which the class associated with $y = 0$ is observed. The natural log of this odds ratio (known as the *logit*) is then assigned to the right-hand side of Equation 11.15. That is

$$\ln\left(\frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})}\right) = \mathbf{ax} + c \quad (11.20)$$

where,

$$\mathbf{x} = (x_1, x_2, x_3, x_4, \dots, x_n);$$

$$\mathbf{ax} + c = a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n + c;$$

Finally, we solve Equation 11.20 for $p(y = 1|\mathbf{x})$ to obtain a bounded representation for the probability, which is shown in Equation 11.21:

$$p(y = 1|\mathbf{x}) = \frac{e^{\mathbf{ax}+c}}{1+e^{\mathbf{ax}+c}} \quad (11.21)$$

where

e is the base of natural logarithms often denoted as \exp

11.5.2 The Logistic Regression Model

Equation 11.21 defines the logistic regression model. Figure 11.28 shows that the graph of the equation is an s-shaped curve bounded by the $[0, 1]$ interval range. As the exponent term approaches negative infinity, the right-hand side of Equation 11.21 approaches 0. Likewise, as the exponent becomes infinitely large in the positive direction, the right side of the equation approaches 1.

The method used to determine the coefficient values for the exponent term $\mathbf{ax} + c$ in Equation 11.21 is iterative. The purpose of the method is to minimize the sum of logarithms of predicted probabilities. Convergence occurs when the logarithmic summation is close to 0 or when the value does not change from one iteration to the next. Details of the

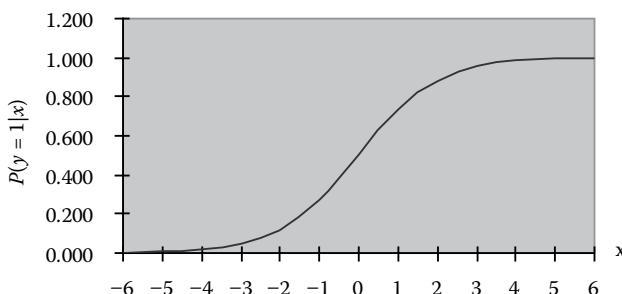


FIGURE 11.28 The logistic regression equation.

technique are beyond the scope of this book. For the interested reader, a description of the process is available from several sources (Hosmer and Lemeshow, 1989; Long, 1989).

11.6 CHAPTER SUMMARY

The naïve Bayes classifier offers a simple yet powerful supervised classification technique. The model assumes all input attributes to be of equal importance and independent of one another. Even though these assumptions are likely to be false, the Bayes classifier still works quite well in practice. Naïve Bayes can be applied to data sets containing both categorical and numeric data. Also, unlike many statistical classifiers, the Bayes classifier can be applied to data sets containing a wealth of missing items.

SVMs offer a unique approach for classifying both linearly separable and nonlinear data. SVMs are able to provide globally optimal problem solutions. Also, because SVMs use the most difficult-to-classify instances as the basis for their predictions, they are less likely to overfit the data.

A favorite statistical technique for estimation and prediction problems is linear regression. Linear regression attempts to model the variation in a numeric dependent variable as a linear combination of one or more numeric independent variables. Linear regression is an appropriate data mining strategy when the relationship between the dependent and independent variables is nearly linear. Linear regression is a poor choice when the outcome is binary. The problem lies in the fact that the value restriction placed on the dependent variable is not observed by the regression equation. That is because linear regression produces a straight-line function, and values of the dependent variable are unbounded in both the positive and negative directions. For the two-outcome case, logistic regression is a better choice. Logistic regression is a nonlinear regression technique that associates a conditional probability value with each data instance. With possible outcomes 0 or 1, the probability $p(y = 1|x)$ represents the conditional probability that the data instance with attribute vector x is part of the class associated with outcome 1. Likewise, $1 - p(y = 1|x)$ is the probability that the instance is part of the class associated with outcome 0. Regression trees and model trees provide additional options for building models having numeric output.

11.7 KEY TERMS

- *A priori probability.* The probability that a hypothesis is true lacking evidence to support or reject the hypothesis.
- *Naïve Bayes classifier.* A supervised learning approach that classifies new instances by using the Bayes theorem.
- *Bayes theorem.* The probability of a hypothesis given some evidence is equal to the probability of the evidence given the hypothesis, times the probability of the hypothesis, divided by the probability of the evidence.

- *Conditional probability.* The conditional probability of evidence E given hypothesis H , denoted by $P(E|H)$, is the probability that E is true given H is true.
- *Dot product.* Algebraically, the dot product of two vectors is the sum of the products of their corresponding entries.
- *Hyperplane.* A subspace one dimension less than its ambient space.
- *Laplace correction.* A correction parameter where one is added to each attribute value count in order to avoid zero probabilities.
- *Least-squares criterion.* The least-squares criterion minimizes the sum of squared differences between actual and predicted output values.
- *Linear regression.* A statistical technique that models the variation in a numeric dependent variable as a linear combination of one or several independent variables.
- *Logistic regression.* A nonlinear regression technique for problems having a binary outcome. A regression equation limits the values of the output attribute to values between 0 and 1. This allows output values to represent a probability of class membership.
- *Logit.* The natural logarithm of the odds ratio $p(y = 1|\mathbf{x})/[1 - p(y = 1|\mathbf{x})] \cdot p(y = 1|\mathbf{x})$ is the conditional probability that the value of the linear regression equation determined by feature vector \mathbf{x} is 1.
- *Maximum margin hyperplane.* The hyperplane showing the greatest separation between two classes.
- *Model tree.* A decision tree where each leaf node contains a linear regression equation.
- *One-versus-all method.* This method is used with models limited to two-class problems. When more than two classes are present, this method builds a model for each pair of classes.
- *One-versus-one method.* This method is used with models limited to two-class problems. When more than two classes are present, this method builds a model for each class.
- *Regression.* The process of developing an expression that predicts a numeric output value.
- *Regression tree.* A decision tree where leaf nodes contain averaged numeric values.
- *Simple linear regression.* A regression equation with a single independent variable.
- *Slope-intercept form.* A linear equation of the form $y = ax + b$, where a is the slope of the line and b is the y -intercept.
- *Support vector.* The boundary instances associated with the maximum margin hyperplane.

EXERCISES

Review Questions

1. Differentiate between the following:
 - a. Simple and multiple linear regression
 - b. Linear and logistic regression
 - c. One-versus-one and one-versus-all
 - d. Linear and nonlinear support vector machines
 - e. Regression tree and model tree
 - f. A priori and conditional probability

Data Mining Questions—RapidMiner

1. Modify the process model in Figures 11.1 and 11.2 by replacing the training/test set scenario with a 10-fold cross-validation.
 - a. Compare your results to the performance vector in Figure 11.4.
 - b. Modify your process model by using all known instances to build a final model. Apply the model built from all known data to find the likely churners within the instances of unknown outcome. How many likely churners did your model find?
2. For this exercise, you are to use RapidMiner's *Naïve Bayes* operator with the spam data set. Here is the procedure:
 - a. Mine the data using a 10-fold cross validation. Summarize the overall model test set accuracy as well as the ability of the model to detect spam.
 - b. Use one or more *attribute selection techniques* listed in the *Blending* folder in an attempt to improve your model.
 - c. Use the *t-test* and *ANOVA* operators to compare the model accuracy seen in part a with the best-performing model from part b. Is there a statistically significant difference between the two models?
3. Modify the data mining example whose main process is given in Figure 11.15 as follows:
 - a. Replace the training/test set scenario with a 10-fold cross-validation. Report the results displayed in the performance vector.
 - b. Run your model with the kernel degree set at 2, 3, and finally, 4. Report model accuracy for each kernel value. Summarize your results.
 - c. Replace mySVM with each of the following SVM operators: Fast Large Margin, Support Vector Machine (Linear), and Hyper Hyper. Write a summary comparing the performance of mySVM with each model.

Data Mining Questions—Weka

1. Compare Weka's supervised data mining algorithms for the numeric output listed below using a 10-fold cross-validation with the Gamma Ray burst data set.
 - a. Set burst length (T90) as the output attribute.
 - b. While in preprocess mode, click on visualize to see the high positive correlation between T90 and T50, Fl and P256, and HR32 and HR321.
 - c. Remove the burst number, T50, P256, and HR321 from the data.
 - d. For each technique, use the default parameter settings. Make a table to report the mean absolute and root mean squared error for each model.
 - e. Here is a list of models to use:
 - SMOReg—Support Vector Machine Regression
 - LinearRegression—Linear Regression
 - M5P—Model Tree
 - MultiLayerPerceptron—Backpropagation Neural Network
 - f. Lastly, choose two of the techniques and attempt to achieve a better result by modifying one or several default parameters. Summarize your findings.
2. For this exercise, you are to compare Weka's SimpleLogistic function with Weka's naïve Bayes function using the spam data set. Here is the procedure:
 - a. Determine the accuracy of each technique using a 10-fold cross-validation.
 - b. Use the method in Chapter 7 for comparing models to determine if your results in part a show that one model performs significantly better than the other. Be specific about how the models differ.
 - c. Return to preprocess mode and use *InfoGainAttributeEval* together with the *ranker* method to eliminate all but the 10 best input attributes. Repeat a and b.
3. For this exercise, you are to apply SMO to *SensorData.arff*, which has 2212 instances representing three classes. For problems with more than two classes, SMO uses the *one-versus-one* technique described previously. The instances contain sensor data collected by a company specializing in aeronautical products.
 - a. Open the *SensorData.arff* with a text editor or word processor to read documentation included with the file.
 - b. Use SMO with 10-fold cross-validation in an attempt to build a model representing the three classes. Record your results.
 - c. Use one or more attribute selection techniques to determine a best subset of input attributes for these data. Determine if the difference in classification correctness between the two models is statistically significant.

Computational Questions

1. Use algebra to verify that Equation 11.13 satisfies the condition stated in Equation 11.12.
2. Suppose the star class contains instances $(0, 0)$, $(0, 3)$, and $(1, 0)$. Also suppose the circle class shows instances $(4, 2)$, $(4, -1)$, $(5, 1)$, and $(4.5, 0)$. Refer to Section 11.2 to help answer the following:
 - a. Use simple Euclidean distance to list and verify the support vectors for each class.
 - b. Use the method described in Section 11.2 to determine the MMH. Show the system of equations you used to obtain your result.
 - c. Use Weka's SMO function with data normalization disabled or RapidMiner's Fast Large Margin support vector operator to verify your result.
3. Use the data contained in Table 11.1 to fill in the counts and probabilities in the following table. The output attribute is *life insurance promotion*.

	Magazine Promotion		Watch Promotion		Credit Insurance		Gender	
Life Insurance Promotion	Yes	No	Yes	No	Yes	No	Yes	No
Yes							Male	
No							Female	
Ratio: yes/total							Ratio: male/total	
Ratio: no/total							Ratio: female/ total	

- a. Use the completed table together with the naïve Bayes classifier to determine the value of *life insurance promotion* for the following instance:

Magazine Promotion = Yes

Watch Promotion = Yes

Credit Card Insurance = No

Gender = Female

Life Insurance Promotion = ?

- b. Repeat part a, but assume that the *gender* of the customer is unknown.
- c. Repeat part a, but use Equation 11.4 with $k = 1$ and $p = 0.5$ to determine the value of *life insurance promotion*.

Unsupervised Clustering Techniques

CHAPTER OBJECTIVES

- *Know how agglomerative clustering is applied to partition data instances into disjoint clusters*
- *Understand that conceptual clustering is an unsupervised data mining technique that builds a concept hierarchy to partition data instances*
- *Recognize how attribute-value predictability and predictiveness scores help determine categorical attribute significance*
- *Know how the expectation-maximization (EM) algorithm uses a statistical parameter adjustment technique to cluster data instances*
- *Understand how genetic learning can be used to perform unsupervised clustering*

In Chapter 3, we demonstrated how the *K*-means algorithm clusters numeric data. In Chapters 9 and 10, we showed you how to perform unsupervised clustering using a neural network approach. In this chapter, we provide a detailed overview of four additional unsupervised clustering techniques. As with the supervised methods presented in Chapter 11, each section of this chapter is self-contained. Although these techniques are generally considered to be statistical, only the EM algorithm actually makes limiting assumptions about the nature of the data.

In Section 12.1, we illustrate how the agglomerative clustering algorithm partitions instances into disjoint clusters. Cobweb's incremental hierarchical clustering technique is the topic of Section 12.2. In Section 12.3, we show how the EM algorithm uses classical statistics to perform unsupervised clustering. Section 12.4 offers an evolutionary approach to unsupervised clustering.

RapidMiner and Weka support several unsupervised clustering algorithms. Weka's unsupervised algorithms can be found in the Weka Clusterers folder. Most of RapidMiner's clustering algorithms are housed under models in the segmentation subfolder. Let's get started!

12.1 AGGLOMERATIVE CLUSTERING

Agglomerative clustering is a favorite unsupervised clustering technique. Unlike the K-means algorithm, which requires the user to specify the number of clusters to be formed, agglomerative clustering begins by assuming that each data instance represents its own cluster. The steps of the algorithm are as follows:

-
1. Begin by placing each data instance into a separate partition.
 2. Until all instances are part of a single cluster,
 - a. Determine the two most similar clusters.
 - b. Merge the clusters chosen in part a into a single cluster.
 3. Choose a clustering formed by one of the step 2 iterations as a final result.
-

Let's see how agglomerative clustering can be applied to the credit card promotion database!

12.1.1 Agglomerative Clustering: An Example

Table 12.1 shows five instances from the credit card promotion database described in Chapter 2. Table 12.2 offers the instance similarity scores computed for the first iteration of algorithm step 2a. For categorical attributes, an instance-to-instance similarity score

TABLE 12.1 Five Instances from the Credit Card Promotion Database

Instance	Income Range	Magazine Promotion	Watch Promotion	Life Insurance Promotion	Gender
I_1	40–50K	Yes	No	No	Male
I_2	25–35K	Yes	Yes	Yes	Female
I_3	40–50K	No	No	No	Male
I_4	25–35K	Yes	Yes	Yes	Male
I_5	50–60K	Yes	No	Yes	Female

TABLE 12.2 Agglomerative Clustering: First Iteration

	I_1	I_2	I_3	I_4	I_5
I_1	1.00				
I_2	0.20	1.00			
I_3	0.80	0.00	1.00		
I_4	0.40	0.80	0.20	1.00	
I_5	0.40	0.60	0.20	0.40	1.00

is computed by first counting the total number of attribute-value matches between two instances. The total number of attribute comparisons then divides this total, giving the similarity measure. For example, comparing I_1 with I_3 , we have four matches and five compares. This gives us the value of 0.80 seen in row I_3 , column I_1 of Table 12.2.

Step 2b requires a merger of the two most similar clusters formed in the first iteration. As table combinations $I_1 - I_3$ and $I_2 - I_4$ each show the highest similarity score, we can choose to merge I_1 with I_3 or I_2 with I_4 . We choose to merge I_1 with I_3 . Therefore, after the first iteration of step 2(b), we have three single-instance clusters (I_2, I_4, I_5) and one cluster having two instances (I_1 and I_3).

Next, we need a method to compute cluster-to-cluster similarity scores. Several possibilities exist. For our example, we use the average similarity of all instances involved in a single table computation. That is, to compute a score for merging the cluster containing I_1 and I_3 with the cluster having I_4 , we divide 7 attribute-value matches by 15 compares, giving a similarity score of 0.47. This similarity score is seen in row I_4 column I_1I_3 of Table 12.3. Note that the table gives all similarity scores for the second iteration of the algorithm.

In the preceding sequence of actions, we had the choice of merging I_1 with I_3 or I_2 with I_4 . When arbitrary choices like this occur, different choices can result in different clusterings. For this reason, it is wise to run several clustering iterations and then choose the best clustering using one of the approaches outlined below.

Table 12.3 tells us that the next iteration of the algorithm will merge I_4 with I_2 . Therefore, after the third iteration of step 2, we have three clusters: one cluster containing I_4 and I_2 , a second cluster having I_1 and I_3 , and the final cluster with I_5 as its only member. The merging of individual clusters continues until all instances are part of a single cluster.

The final step, requiring the choice of a final clustering, is the most difficult. Several statistical as well as heuristic measures can be applied. The following are three simple heuristic techniques that work well in a majority of situations. Specifically,

1. Invoke the similarity measure used to form the clusters and compare the average within-cluster similarity to the overall similarity of all instances in the data set. The overall or domain similarity is simply the score seen with the final iteration of the algorithm. In general, if the average of the individual cluster similarity scores shows a higher value than the domain similarity, we have positive evidence that the clustering is useful. As several clusterings of the algorithm may show the desired quality, this technique is best used to eliminate clusterings rather than to choose a final result.

TABLE 12.3 Agglomerative Clustering: Second Iteration

	I_1I_3	I_2	I_4	I_5
I_1I_3	0.80			
I_2	0.33	1.00		
I_4	0.47	0.80	1.00	
I_5	0.47	0.60	0.40	1.00

2. As a second approach, we compare the similarity within each cluster to the similarity between each cluster. For example, given three clusters, A, B, and C, we analyze cluster A by computing three scores. One score is the within-class similarity of the instances in cluster A. The second score is obtained by computing the similarity score seen when all the instances of cluster A are combined with the instances of B. The third score is the similarity value obtained by grouping the instances of cluster C with those of cluster A. We expect to see the highest scores for within-class similarity computations. As with the first technique, several clusterings of the algorithm may show the desired quality. Therefore, the technique is also best used to eliminate clusterings rather than to choose a final result.
3. One useful measure to be used in conjunction with the previous techniques is to examine the rule sets generated by each saved clustering. For example, let's assume 10 iterations of the algorithm are initially performed. Next, suppose 5 of the 10 clusterings are eliminated by using one of the previous techniques. To apply the current method, we present each clustering to a rule generator and examine the rules created from the individual clusters. The clustering showing a best set of rules, based on precision and coverage, is chosen as the final result.

Provided that certain assumptions can be made about the data, a statistical analysis may also be applied to help determine which of the clusterings is a best result. One common statistical test used to select a best partitioning is the *Bayesian Information Criterion (BIC)*. The BIC requires that the clusters be normally distributed. The BIC gives the odds for one model against another model assuming that neither model is initially favored (Dasgupta and Raftery, 1998).

12.1.2 General Considerations

Agglomerative clustering creates a hierarchy of clusterings by iteratively merging pairs of clusters. Although we have limited our discussion here, several procedures for computing cluster similarity scores and merging clusters exist. Also, when data are real-valued, defining a measure of instance similarity can be a challenge. One common approach is to use simple Euclidean distance.

A widespread application of agglomerative clustering is its use as a prelude to other clustering techniques. For example, the first iteration of the *K*-means algorithm requires a choice for initial cluster means. In the usual case, the choice is made in a random or arbitrary manner. However, the initial selection can have a marked effect on goodness of the final clustering. Therefore, to increase our chance of obtaining a best final clustering, we first apply agglomerative clustering to create the same number of clusters as that chosen for the *K*-means algorithm. Next, we compute the cluster means resulting from the agglomerative technique and use the mean scores as the initial choice for the first *K*-means clustering (Mukherjee et al., 1998).

12.2 CONCEPTUAL CLUSTERING

Conceptual clustering is an unsupervised clustering technique that incorporates incremental learning to form a hierarchy of concepts. The concept hierarchy takes the form of

a tree structure where the root node represents the highest level of concept generalization. Therefore, the root node contains summary information for all domain instances. Of particular interest are the basic-level nodes of the tree. The basic-level nodes are interesting in terms of human appeal and understanding. An appropriate measure of cluster quality forms these basic-level nodes at the first or second level of the concept tree. The following is a standard conceptual clustering algorithm:

-
1. Create a cluster with the first instance as its only member.
 2. For each remaining instance, take one of two actions at each level of the tree:
 - a. Place the new instance into an existing cluster.
 - b. Create a new concept cluster having the presented instance as its only member.
-

The algorithm clearly shows the incremental nature of the clustering process. That is, each instance is presented to the existing concept hierarchy in a sequential manner. Next, at each level of the hierarchy, an evaluation function is used to make a decision about whether to include the instance in an existing cluster or to create a new cluster with the new instance as its only member. In the next section, we describe the evaluation function used by a well-known probability-based conceptual clustering system.

12.2.1 Measuring Category Utility

Cobweb (Fisher, 1987) is a conceptual clustering model that stores knowledge in a concept hierarchy. Cobweb accepts instances in attribute-value format, where attribute values must be categorical. Cobweb's evaluation function has been shown to consistently determine psychologically preferred (basic) levels in human classification hierarchies. The evaluation function is a generalization of a measure known as *category utility*. The category utility function measures the gain in the "expected number" of correct attribute-value predictions for a specific object if it were placed within a given category.

The formula for category utility includes three probabilities. One measure is the conditional probability of attribute A_i having value V_{ij} given membership in class C_k , denoted as $P(A_i = V_{ij} | C_k)$. This is the formal definition of attribute-value *predictability*. If the value of $P(A_i = V_{ij} | C_k) = 1$, we can be certain that each instance of class C_k will always have V_{ij} as the value for attribute A_i . Attribute A_i having value V_{ij} is said to be a *necessary* condition for defining class C_k . The second probability, $P(C_k | A_i = V_{ij})$, is the conditional probability that an instance is in class C_k given that attribute A_i has value V_{ij} . This is the definition of attribute-value *predictiveness*. If the value of $P(C_k | A_i = V_{ij}) = 1$, we know that if A_i has value V_{ij} , the class containing this attribute value pair must be C_k . Attribute A_i having value V_{ij} is said to be a *sufficient* condition for defining class C_k . Given these definitions, we see that attribute-value predictability is a within-class measure and attribute-value predictiveness is a between-class measure.

These three probability measures are combined and summed across all values of i , j , and k to describe a heuristic measure of partition quality. Specifically,

$$\sum_k \sum_i \sum_j P(A_i = V_{ij}) P(C_k | A_i = V_{ij}) P(A_i = V_{ij} | C_k) \quad (12.1)$$

The probability $P(A_i = V_{ij})$ allows attribute values that are seen frequently to play a more important part in measuring partition quality. Using the expression for partition quality, category utility is defined by the formula

$$\frac{\sum_{k=1}^k P(C_k) \sum_i \sum_j P(A_i = V_{ij} | C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{k} \quad (12.2)$$

The first numerator term is the previously described partition-quality expression stated in an alternative form through the application of Bayes rule. The second term represents the probability of correctly guessing attribute values without any category knowledge. The division by k (total number of classes) allows Cobweb to consider variations in the total number of formed clusters.

12.2.2 Conceptual Clustering: An Example

To illustrate the process used by Cobweb to build a concept hierarchy, consider the hierarchy shown in Figure 12.1 created by Cobweb when presented with the instances in Table 12.4. Let's suppose we have a new instance to be placed in the hierarchy. The instance enters the hierarchy at root node N , and N 's statistics are updated to reflect the addition of the new instance. As the instance enters the second level of the hierarchy, Cobweb's evaluation function chooses one of four actions. If the new instance is similar enough to one of N_1 , N_2 , or N_4 , the instance is incorporated into the preferred node, and the instance proceeds to the second level of the hierarchy through the chosen path. As a second choice, the evaluation function can decide that the new instance is unique enough to merit the creation of a new first-level concept node. This being the case, the instance becomes a first-level concept node, and the classification process terminates.

Cobweb allows two other choices. In one case, the system considers merging the two best-scoring nodes into a single node. The last possibility actually removes the best-scoring node from the hierarchy. These final two choices are to help modify nonoptimal hierarchies that can result from skewed instance presentation—several highly similar instances are presented in sequence. If either choice is made, the merge or delete operation is processed, and the new instance is once again presented for classification to the modified hierarchy. This procedure continues at each level of the concept tree until the new instance becomes a terminal node.

As a final point, notice that all predictiveness and predictability scores are computed with respect to the parent node. For example, the predictiveness score of 1.0 for *nuclei* = *two*

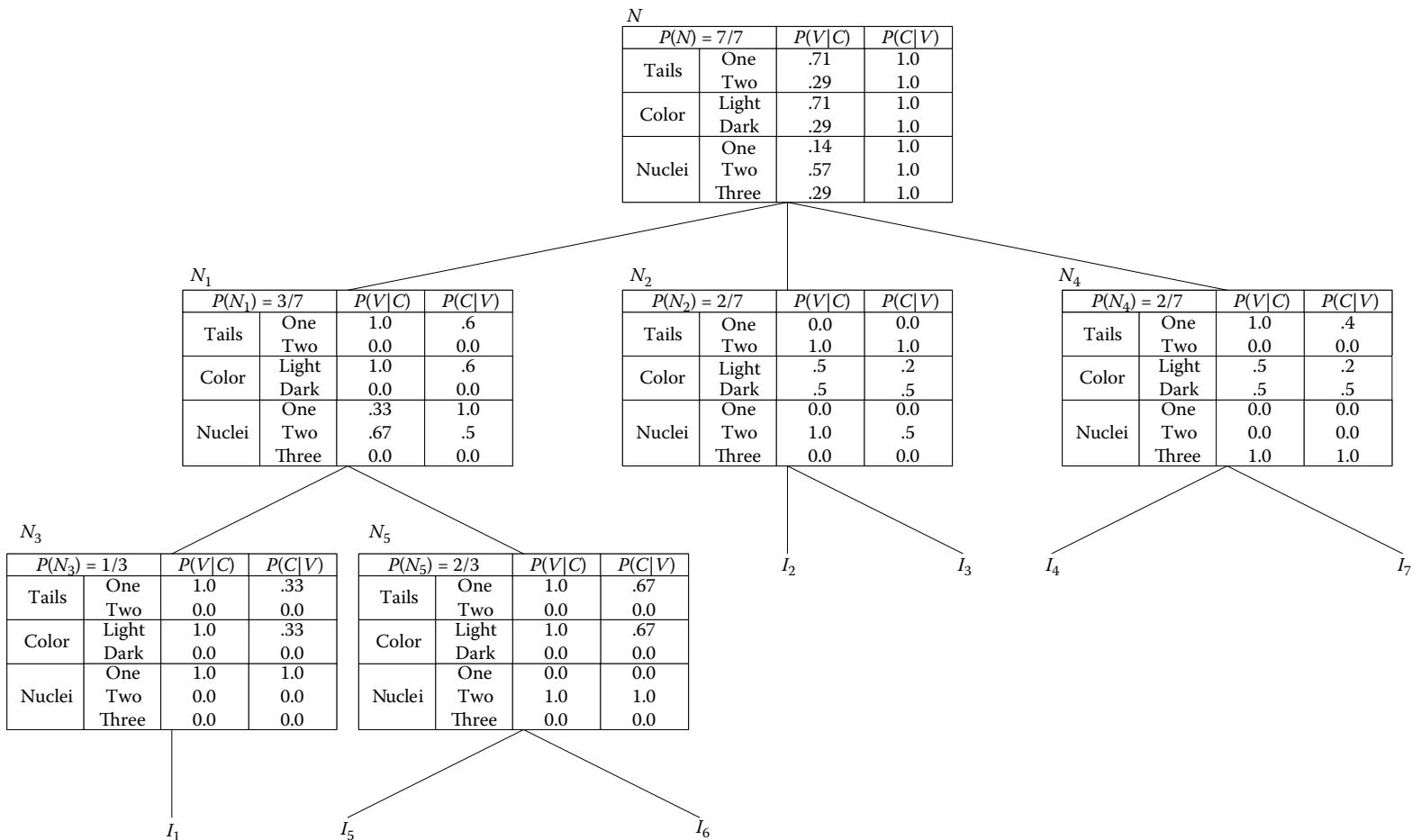


FIGURE 12.1 A Cobweb-created hierarchy.

TABLE 12.4 Data for Conceptual Clustering

	Tails	Color	Nuclei
I_1	One	Light	One
I_2	Two	Light	Two
I_3	Two	Dark	Two
I_4	One	Dark	Three
I_5	One	Light	Two
I_6	One	Light	Two
I_7	One	Light	Three

found in N_3 reflects the fact that all instances incorporated into N_2 with $nuclei = two$ have followed the path from N_2 to N_3 .

12.2.3 General Considerations

Although we have limited our discussion to Cobweb, several other conceptual clustering systems have been developed. Classit (Gennari, Langley, and Fisher, 1989) is a conceptual clustering system that uses an extension of Cobweb's basic algorithm to build a concept hierarchy and classify instances. The two models are very similar, with the exception that Classit's evaluation function is an equivalent transformation of Cobweb's category utility for real-valued attributes. Therefore, individual concept nodes store attribute mean and standard deviation scores rather than attribute value probabilities.

Like Cobweb, Classit is particularly appealing as its evaluation function has been shown to consistently determine psychologically preferred levels in human classification hierarchies. In addition, both Cobweb and Classit lend themselves well to explaining their behavior because each tree node contains a complete concept description at some level of abstraction.

Conceptual clustering systems also have several shortcomings. A major problem with conceptual clustering systems is that instance ordering can have a marked impact on the results of the clustering. A nonrepresentative ordering of the instances can lead to a less-than-optimal clustering. Clustering systems such as Cobweb and Classit use special operations to merge and split clusters in an attempt to overcome this problem. However, the results of these techniques have not been shown to be successful in all cases.

12.3 EXPECTATION MAXIMIZATION

The EM algorithm (Dempster, Laird, and Rubin, 1977) is a statistical technique that makes use of the finite Gaussian mixtures model. A *mixture* is a set of n probability distributions where each distribution represents a cluster. The mixtures model assigns each individual data instance a probability (rather than a specific cluster) that it would have a certain set of attribute values given that it was a member of a specified cluster. The mixtures model assumes all attributes to be independent random variables.

The EM algorithm is similar to the K -means procedure in that a set of parameters are recomputed until a desired convergence value is achieved. In the simplest case, $n = 2$, the probability distributions are assumed to be normal, and data instances consist of a single

real-valued attribute. Although the algorithm can be applied to a data set having any number of real-valued attributes, we limit our discussion here to this simplest case and provide a general example in the next section. Using the two-class, one-attribute scenario, the job of the algorithm is to determine the value of five parameters. Specifically,

- The mean and standard deviation for cluster 1.
- The mean and standard deviation for cluster 2.
- The sampling probability P for cluster 1—the number of instances in cluster 1 divided by the total number of domain instances. The probability for cluster 2 is therefore $(1 - P)$.

The general procedure used by EM is as follows:

1. Guess initial values for the five parameters given above.
2. Until a specified termination criterion is achieved,
 - a. Use the probability density function for normal distributions (Equation 11.5) to compute the cluster probability for each instance. In the two-cluster case, we have two probability distribution formulas, each with differing mean and standard deviation values.
 - b. Use the probability scores assigned to each instance in step 2a to reestimate the five parameters.

The algorithm terminates when a formula that measures cluster quality no longer shows significant increases. One measure of cluster quality is the likelihood that the data came from the data set determined by the clustering. The likelihood computation is straightforward. For each data instance, first sum the conditional probabilities that the instance belongs in each of the two clusters. In this way, each instance has an associated summed probability score. Next, multiply the summed probability scores to compute the likelihood value. Higher likelihood scores represent more optimal clusterings.

12.3.1 Implementations of the EM Algorithm

EM's solid statistical foundation as well as its similarity to the K -means algorithm makes it one of the most referenced clustering algorithms. We provide two tutorials to help you obtain a better understanding about how the EM algorithm clusters numeric data. The tutorial titled RapidMiner's EM Operator applies RapidMiner's Expectation Maximization operator to the gamma-ray burst data set. The tutorial, Weka's EM Clustering Model, uses Weka's EM algorithm to cluster the sensor data set first referenced in Chapter 11 exercises.

12.3.2 General Considerations

EM implements a statistical model that is guaranteed to converge to a maximum likelihood score. However, the maximum may not be global. For this reason, several applications of the algorithm may be necessary to achieve a best result. As initial mean and standard deviation scores selected by the algorithm affect the final result, an alternative technique

RAPIDMINER TUTORIAL

RapidMiner's EM Operator

In Chapter 5, we showed you how to use supervised learning to interpret the results of a K-means clustering of the gamma-ray burst data set—GRB4U.xlsx. Here we utilize the same procedure to illustrate RapidMiner's EM operator. Our goal is to determine if EM, when presented with the burst data, creates three well-defined clusters.

To follow our example, open the K-means process from Chapter 5 as shown in Figure 12.2 and replace K-means with the EM operator. Next, we examine four of EM's parameters. The topmost parameter, *k*, specifies the number of clusters. This parameter defaults to two. If *add cluster attribute* is checked, a new attribute named *cluster* is added to the data. For each instance, the value of the cluster attribute specifies the name of the cluster holding the instance. Recall that adding the cluster attribute allows us to use supervised learning for unsupervised clustering evaluation. *Max runs* gives the maximum number of randomly initialized operator runs. *Max optimization steps* specifies the maximum number of iterations within each run. Both parameters are optional and default to one. EM supports several other parameters that you can read about in the documentation. Let's continue with our example.

- Set the parameter values as shown in Figure 12.2. Set the maximal depth parameter for the *Decision Tree* operator at 4.
- Use the *Select Attributes* operator to remove the *burst* attribute.
- Set the correlational value for removal at 0.5.
- Place a *Breakpoint After* within *Remove Correlated Attributes* and *Clustering*.
- Specify an output file within *Write Excel*, and use *Set Role* to change the role of the *cluster* attribute to *label*. Click run.

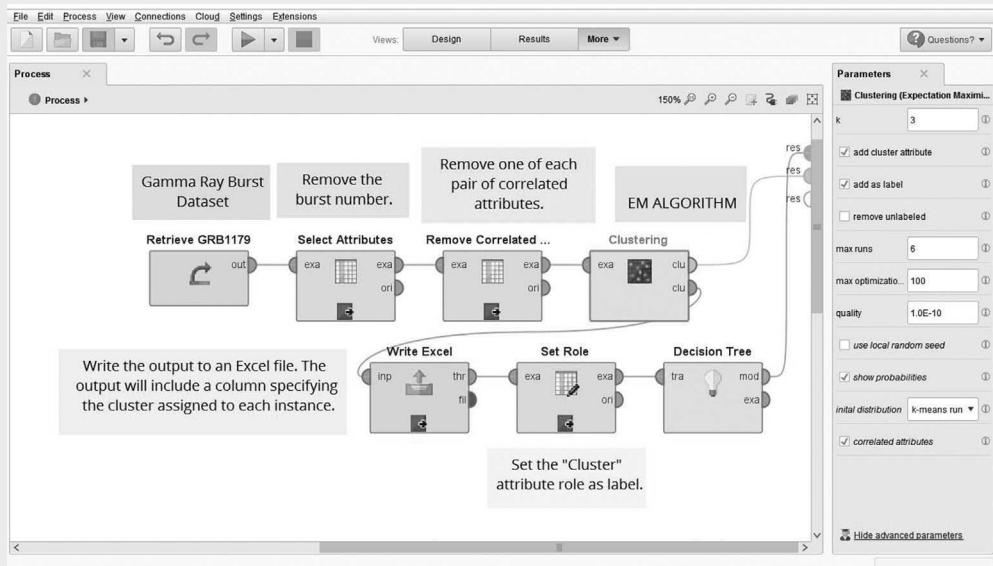


FIGURE 12.2 Applying EM to the gamma-ray burst data set.

Figure 12.3 shows the result of the *Remove Correlated Attributes* operation. Recall that the gamma-ray burst data set contains one pair of attributes for each of burst length, brightness, and hardness. The operator has removed one attribute from each pair. You may have noticed that the EM operator has a parameter that, when set, accounts for correlated attributes. However, as we are familiar with the nature of these data, it is best for us to directly deal with attribute correlations.

Figure 12.4 shows the first few data instances together with associated cluster probability values. For each instance, the value of the cluster attribute is the name of the cluster having the highest associated instance probability.

Data View:

Row No.	p256	hr32	t50
1	0.006	0.514	0.730
2	0.119	0.469	0.850
3	0.022	0.431	0.933
4	-0.027	0.450	0.867
5	0.043	0.484	1.093
6	-0.000	0.556	0.937
7	0.114	0.411	0.878
8	0.035	0.492	1.064
9	0.116	0.380	0.878
10	0.035	0.515	0.681
11	-0.025	0.426	0.741
12	0.035	0.396	0.736
13	-0.032	0.516	0.698
14	-0.068	0.444	0.704
15	0.180	0.525	0.802
16	0.007	0.530	0.728
17	-0.036	0.542	0.827

Repository View:

- Samples
- DB
- FromRepositoryRMS (local)
- Local Repository (local)
- Template (local)
- Windows7Repository (local)
- Cloud Repository (disconnected)

FIGURE 12.3 Removing correlated attributes from the gamma-ray burst data set.

Data View:

Row No.	id	cluster	cluster_0_p...	cluster_1_p...	cluster_2_p...	p256	hr32	t50
1	1	cluster_1	0.262	0.737	0.001	0.006	0.514	0.730
2	2	cluster_1	0.472	0.528	0.000	0.119	0.469	0.850
3	3	cluster_1	0.348	0.652	0.000	0.022	0.431	0.933
4	4	cluster_1	0.280	0.720	0.000	-0.027	0.450	0.867
5	5	cluster_1	0.376	0.624	0.000	0.043	0.484	1.093
6	6	cluster_1	0.254	0.746	0.000	-0.000	0.556	0.937
7	7	cluster_1	0.477	0.523	0.000	0.114	0.411	0.878
8	8	cluster_1	0.357	0.643	0.000	0.035	0.492	1.064
9	9	cluster_1	0.478	0.522	0.000	0.116	0.380	0.878
10	10	cluster_1	0.289	0.709	0.002	0.035	0.515	0.681
11	11	cluster_1	0.270	0.729	0.000	-0.025	0.426	0.741
12	12	cluster_1	0.341	0.659	0.000	0.035	0.396	0.736
13	13	cluster_1	0.218	0.781	0.001	-0.032	0.516	0.698
14	14	cluster_1	0.223	0.777	0.001	-0.068	0.444	0.704
15	15	cluster_0	0.559	0.441	0.001	0.180	0.525	0.802
16	16	cluster_1	0.252	0.747	0.001	0.007	0.530	0.728

Repository View:

- Samples
- DB
- FromRepositoryRMS (local)
- Local Repository (local)
- Template (local)
- Windows7Repository (local)
- Cloud Repository (disconnected)

FIGURE 12.4 An EM clustering of the gamma-ray burst data set.

Figure 12.5 displays resultant summary statistics. We see the number of instances assigned to each cluster, cluster probabilities, cluster means and the associated covariance matrix. Each covariance matrix offers a generalization of variance relative to the corresponding cluster.

Figure 12.6 gives us the graphical form of the decision tree created from the clustering. As expected, the tree shows burst length as the root node. Finally, Figure 12.7 displays a detailed description of the distribution of instances relative to each cluster. This description clearly shows one cluster of longer, brighter, softer bursts; a second cluster of longer, duller, softer bursts; and a third cluster of shorter, harder bursts.

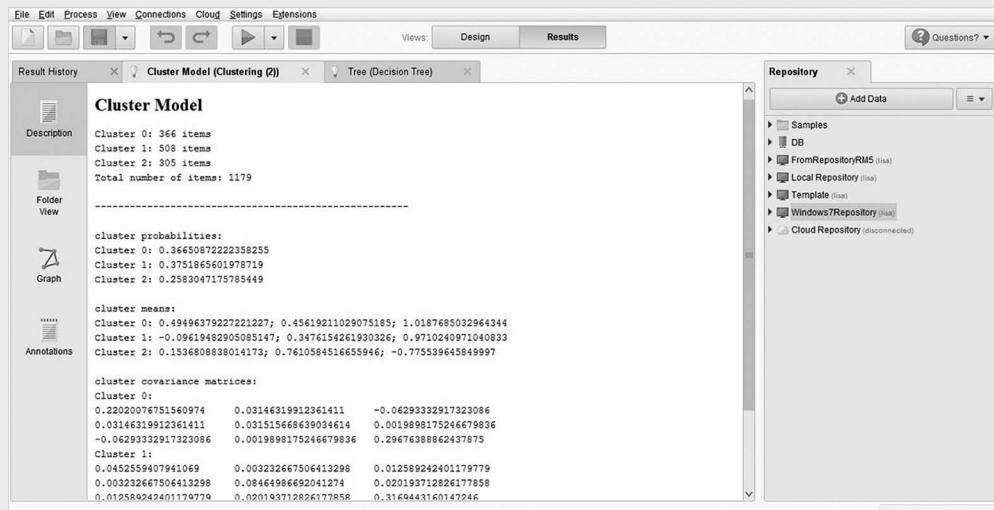


FIGURE 12.5 Summary statistics for an EM clustering of the gamma-ray burst data set.

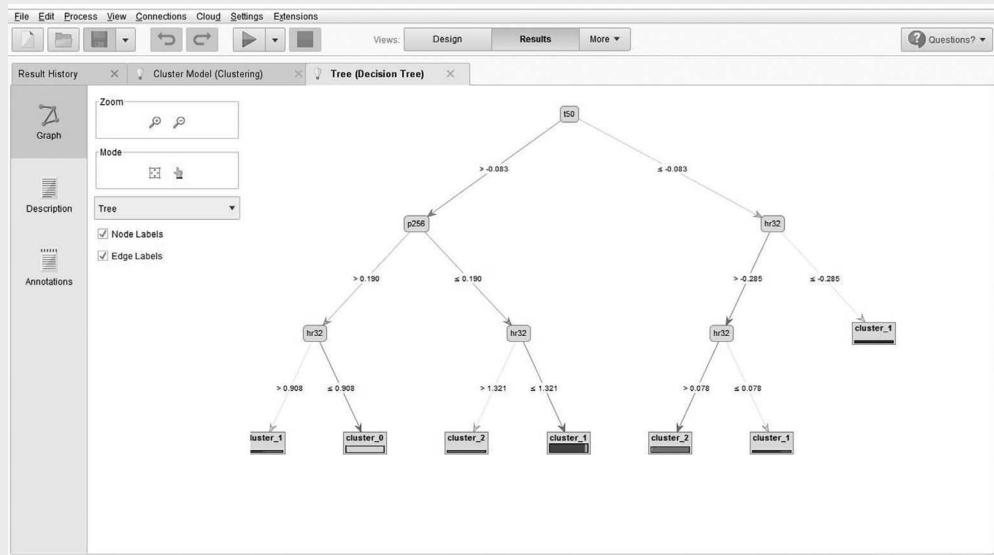


FIGURE 12.6 Decision tree representing a clustering of the gamma-ray burst data set.

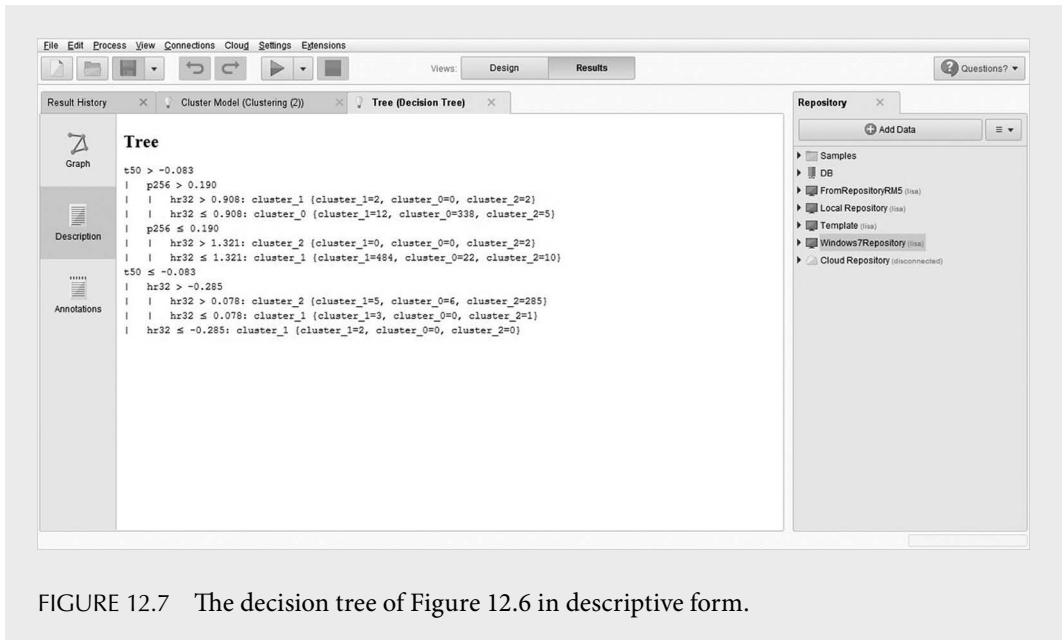


FIGURE 12.7 The decision tree of Figure 12.6 in descriptive form.

WEKA TUTORIAL

Weka’s EM Clustering Model

For our example illustrating Weka’s implementation of EM, we use the 2212-instance sensor data set found in SensorData.arff. The source and nature of the data are confidential. Each instance contains 12 numeric input attributes each representing a sensor designed to detect the presence of one of two groups of substances. As an alternative, the sensor readings specify a third group given as *false alarms*. Our goal is to determine how well the formed clusters represent the three classes defined within the data. Let’s begin!

- Load SensorData.arff into the Explorer and click on class. Your screen will appear as in Figure 12.8.

The figure shows one class of 947 instances, a second class of 799 instances, and a 466-instance class representing false alarms.

- On the Cluster tab, choose EM clustering and open Weka’s generic object editor as shown in Figure 12.9 to change numClusters from –1 to 3. Click OK.

Here we change the value as we are interested in how well the three clusters match the defined classes. When numClusters is set at its default of –1, the algorithm determines an optimal number of clusters based on the likelihood scores seen with each alternative clustering.

- Highlight the classes to clusters radio button and click start.

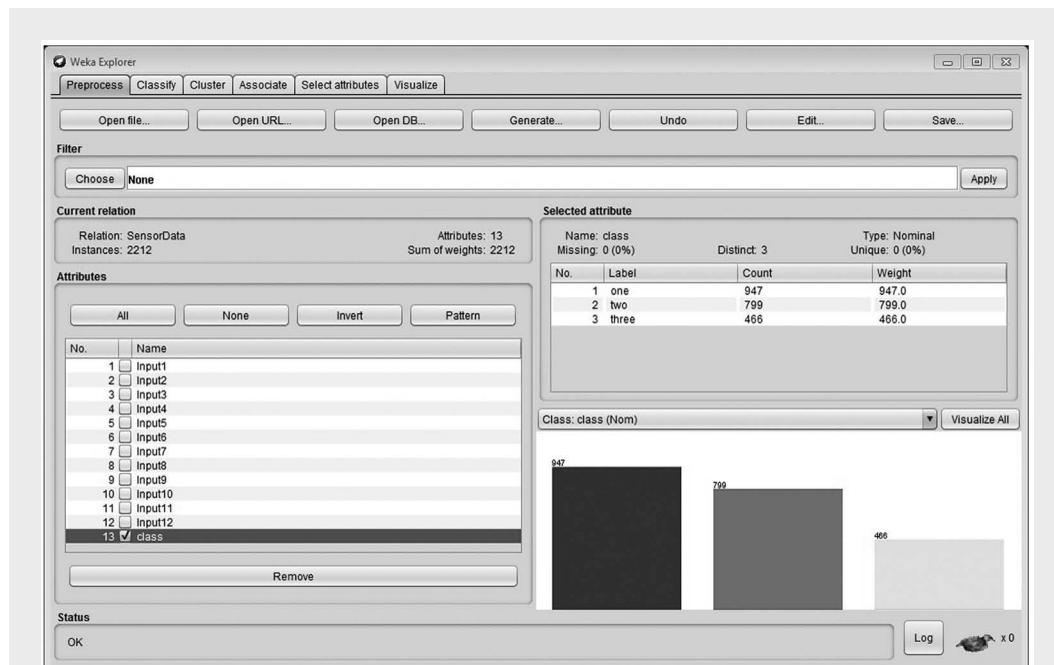


FIGURE 12.8 Classes of the sensor data set.

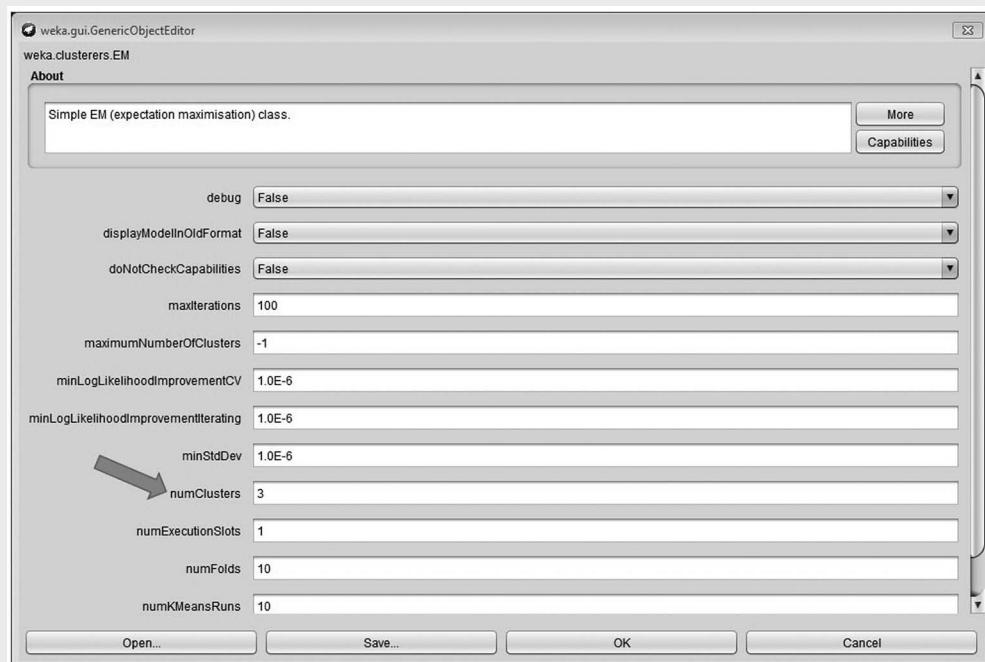


FIGURE 12.9 Generic object editor allows us to specify the number of clusters.

```

Clustered Instances
0    954 ( 43%)
1    496 ( 22%)
2    762 ( 34%)

Class attribute: class
Classes to Clusters:

0 1 2 <-- assigned to cluster

177 217 553 | one
590 0 209 | two
187 279 0 | three

Cluster 0 <-- two
Cluster 1 <-- three
Cluster 2 <-- one
Incorrectly clustered instances :      790.0  35.7143 %

```

FIGURE 12.10 Classes to clusters summary statistics.

The top portion of the output screen provides summary statistics in the form of mean and standard deviation scores for the input attributes.

- Scroll the output until your screen appears as in Figure 12.10.

The output tells us that cluster 0 contains 954 instances, 590 of which belong to class two. Cluster 1 shows 496 instances, 279 of which are housed in class 3. Cluster 2 contains 762 instances, 553 of which belong to class 1. To summarize, approximately 64% of the instances clustered with members of the same class. This is a positive indication that the input instances differentiate the classes. The greatest amount of variation is within cluster 1 as it contains 279 class 3, 217 class 1, and no class 2 instances. This last observation indicates the need for further investigation.

such as agglomerative clustering is often initially applied. EM then uses the mean and standard deviation values for the clusters determined by the preliminary technique as initial parameter settings.

A lack of explanation about what has been discovered is a problem with EM as it is with many clustering systems. For this reason, our suggested methodology of using a supervised model to analyze the results of an unsupervised clustering is often appropriate.

12.4 GENETIC ALGORITHMS AND UNSUPERVISED CLUSTERING

In Chapter 3, we saw how genetic algorithms can be used for supervised learning. Genetic learning is also a powerful unsupervised clustering technique. As a simple example, we

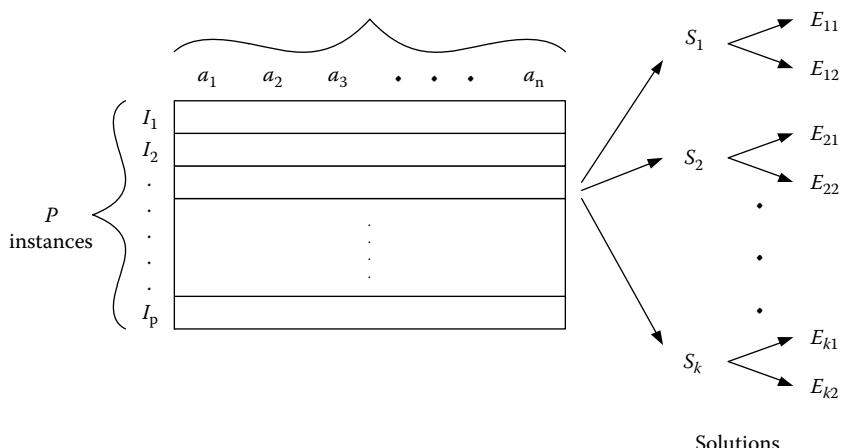


FIGURE 12.11 Unsupervised genetic clustering.

can use unsupervised genetic learning to develop a set of clusters for numerical data in an n -dimensional space. Suppose there are P data instances within the space, where each data instance consists of n attribute values. To formulate the problem, the algorithm is supplied with the number of clusters to be created. Suppose m clusters are desired. The model then generates k possible solutions. A specific solution contains m n -dimensional points, where each point is a best current representative element for one of the m clusters. Figure 12.11 illustrates the idea with $m = 2$. For example, S_2 represents one of the k possible solutions and contains two elements, E_{21} and E_{22} .

A crossover operation is accomplished by moving elements (n -dimensional points) from solution S_i to solution S_j . There are several possibilities for implementing mutation operations. One way to mutate solution S_i is to swap one or more point coordinates of the elements within S_i .

An applicable fitness function for solution S_j is the average Euclidean distance of the P instances in the n -dimensional space from their closest element within S_j . That is, to compute the fitness value for solution S_j , we take each instance I in P and compute the Euclidean distance from I to each of the m elements in S_j . The closest distance is saved and added to the sum total of closest values. The average of the summed distances is the fitness score for S_j . Clearly, lower values represent better fitness scores. Once genetic learning terminates, the best of the k possible solutions is selected as the final solution. Each instance in the n -dimensional space is then assigned to the cluster associated with its closest element in the final solution.

Let's see how the technique is applied to the six instances in Table 12.5. These same data were used to illustrate the K -means algorithm in Chapter 3. Assume we suspect that the data contain two clusters and we instruct the algorithm to start with a solution set consisting of three plausible solutions ($k = 3$). With $m = 2$, $P = 6$, and $k = 3$, the algorithm generates the initial set of solutions, shown in Table 12.6. An element in the solution space contains

TABLE 12.5 Instances for Unsupervised Genetic Learning

Instance	X	Y
1	1.0	1.5
2	1.0	4.5
3	2.0	1.5
4	2.0	3.5
5	3.0	2.5
6	5.0	6.0

TABLE 12.6 A First-Generation Population for Unsupervised Clustering

	S_1	S_2	S_3
Solution elements (initial population)	(1.0,1.0) (5.0,5.0)	(3.0,2.0) (3.0,5.0)	(4.0,3.0) (5.0,1.0)
Fitness score	11.31	9.78	15.55
Solution elements (second generation)	(5.0,1.0) (5.0,5.0)	(3.0,2.0) (3.0,5.0)	(4.0,3.0) (1.0,1.0)
Fitness score	17.96	9.78	11.34
Solution elements (third generation)	(5.0,5.0) (1.0,5.0)	(3.0,2.0) (3.0,5.0)	(4.0,3.0) (1.0,1.0)
Fitness score	13.64	9.78	11.34

a single representative data point for each cluster. For example, the data points for solution S_1 are (1.0,1.0) and (5.0,5.0).

To compute the fitness score of 11.31 for solution S_1 , the Euclidean distance between each instance in Table 12.5 and its closest data point in S_1 is summed. To illustrate this, consider instance 1 in Table 12.5. The Euclidean distance between (1.0,1.0) and (1.0,1.5) is computed as 0.50. The distance between (5.0,5.0) and (1.0,1.5) is 5.32. The smaller value of 0.50 is therefore represented in the overall fitness score for solution S_1 . Table 12.6 shows S_2 as the best first-generation solution.

The second generation is obtained by performing a crossover between solutions S_1 and S_3 with solution element (1.0,1.0) in S_1 exchanging places with solution element (5.0,1.0) in S_3 . As the table shows, the result of the crossover operation improves (decreases) the fitness score for S_3 , while the score for S_1 increases. The third generation is acquired by mutating S_1 . The mutation interchanges the y -coordinate of the first element in S_1 with the x -coordinate of the second element. The mutation results in an improved fitness score for S_1 . Mutation and crossover continue until a termination condition is satisfied. The terminating condition can be a maximum number of iterations of the algorithm or a minimum fitness score for one or several solutions. If the third generation is seen as terminal, the final solution will be S_2 . Computing the distances between the points in Table 12.5 and their closest element in S_2 results in instances 1, 3, and 5 forming one cluster and instances 2 and 6 forming the second cluster. As instance 4 is equidistant from both elements in S_2 , it can be placed in either cluster.

As is the case with supervised genetic learning, the fitness function determines the computational complexity of the algorithm. A fitness function having several calculations can be computationally complex.

12.5 CHAPTER SUMMARY

Agglomerative clustering is a favorite unsupervised technique. Agglomerative clustering begins by assuming each data instance represents its own cluster. Each iteration of the algorithm merges the most similar pair of clusters. The final iteration sees all data set items contained in a single cluster. Several options for computing instance and cluster similarity scores and cluster merging procedures exist. Also, when the data to be clustered are real-valued, defining a measure of instance similarity can be a challenge. One common approach is to use simple Euclidean distance. A widespread application of agglomerative clustering is its use as a prelude to other clustering techniques.

Conceptual clustering is an unsupervised technique that incorporates incremental learning to form a hierarchy of concepts. The concept hierarchy takes the form of a tree structure where the root node represents the highest level of concept generalization. Conceptual clustering systems are particularly appealing because the trees they form have been shown to consistently determine psychologically preferred levels in human classification hierarchies. Also, conceptual clustering systems lend themselves well to explaining their behavior. A major problem with conceptual clustering systems is that instance ordering can have a marked impact on the results of the clustering. A nonrepresentative ordering of data instances can lead to a less-than-optimal clustering.

The EM algorithm is a statistical technique that makes use of the finite Gaussian mixtures model. The mixtures model assigns each individual data instance a probability that it would have a certain set of attribute values given that it was a member of a specified cluster. The model assumes all attributes to be independent random variables. The EM algorithm is similar to the K -means procedure in that a set of parameters are recomputed until a desired convergence value is achieved. A lack of explanation about what has been discovered is a problem with EM as it is with many clustering systems. Our methodology of using a supervised model to analyze the results of an unsupervised clustering is one technique to help explain the results of an EM clustering.

Genetic learning can be used for both supervised and unsupervised learning. Genetic algorithms can also be applied in preprocess mode to help with attribute selection. Although genetic algorithms are designed to find globally optimized solutions, there is no guarantee that any given solution is not the result of a local rather than a global optimization.

12.6 KEY TERMS

- *Agglomerative clustering.* An unsupervised technique where each data instance initially represents its own cluster. Successive iterations of the algorithm merge pairs of highly similar clusters until all instances become members of a single cluster. In the last step, a decision is made about which clustering is a best final result.

- *Basic-level nodes.* The nodes in a concept hierarchy that represent concepts easily identified by humans.
- *Bayesian Information Criterion (BIC).* The BIC gives the posterior odds for one data mining model against another model assuming neither model is favored initially.
- *Category utility.* An unsupervised evaluation function that measures the gain in the “expected number” of correct attribute-value predictions for a specific object if it were placed within a given category or cluster.
- *Concept hierarchy.* A tree structure where each node of the tree represents a concept at some level of abstraction. Nodes toward the top of the tree are the most general. Leaf nodes represent individual data instances.
- *Conceptual clustering.* An incremental unsupervised clustering method that creates a concept hierarchy from a set of input instances.
- *Incremental learning.* A form of learning that is supported in an unsupervised environment where instances are presented sequentially. As each new instance is seen, the learning model is modified to reflect the addition of the new instance.
- *Mixture.* A set of n probability distributions where each distribution represents a cluster.

EXERCISES

Review Questions

1. Compare and contrast conceptual and agglomerative clustering. Make a list of similarities and differences between the two approaches.
2. Compare and contrast the EM algorithm with the K -means approach described in Chapter 3. Make a list of similarities and differences between the two approaches.

Data Mining Questions

1. Use Weka’s EM algorithm to mine the gamma-ray burst data set several times with the help of the numClusters parameter. Initially use default setting of -1, and then set the value at 2, followed by 3, then 4, and finally 5. Record the likelihood value for each mining session. Summarize your results in a table. Make a value judgment as to which clustering best represents the data. Justify your answer.
2. Use RapidMiner’s EM operator to mine the sensor data (SensorData.xlsx). Summarize your results.
3. Perform an unsupervised clustering using Weka’s agglomerative clustering algorithm using a data set of your choice. Summarize your results.

Computational Questions

1. Create tables to complete the agglomerative clustering example described in Section 12.1. Choose one of the techniques presented in Section 12.1 to pick a best clustering. As an alternative, develop your own method to make the choice. Explain why your choice represents a best clustering.
2. Consider the new instance I_8 with the attribute values listed below to be added to the concept hierarchy in Figure 12.1.

Tails = Two

Color = Dark

Nuclei = Two

- a. Show the updated probability values for the root node once the instance has been entered into the hierarchy.
- b. Add the new instance to node n_2 and show the updated probability values for all affected nodes.
- c. Rather than having the instance become part of node n_2 , assume that the instance creates a new first-level node. Add the new node to the hierarchy and show the updated probability values for all affected nodes.

Specialized Techniques

CHAPTER OBJECTIVES

- *Know how to perform a time-series analysis*
- *Know how data mining can be used to discover hidden patterns in data recorded about the clickstream activity of website visitors*
- *Know how data mining is used to automate website evaluation and adaptation*
- *Understand how data mining is employed to present Web users with information that interests them without requiring them to ask for it directly*
- *Understand that textual data mining concerns itself with extracting useful patterns from unstructured text*
- *Understand techniques for dealing with large-sized, imbalanced, and streaming data*
- *Know how and when bagging and boosting can be used to improve the performance of supervised classifiers*

In this chapter, we overview several specialized data mining techniques. In Section 13.1, we introduce time-series analysis and show how neural networks are used to solve time-series problems. In Section 13.2, it is shown how data mining is used for website evaluation, personalization, and adaptation. We also provide an overview of how the PageRank algorithm is used to determine the prestige of a page or website. Section 13.3 offers an overview of textual data mining. In Section 13.4, we discuss methods for dealing with large-sized data, imbalanced data, and streaming data. Section 13.5 presents two methods that, in some cases, can improve the classification correctness of supervised learner models.

13.1 TIME-SERIES ANALYSIS

Oftentimes, the data we wish to analyze contain a time dimension. Prediction applications with one or more time-dependent attributes are called *time-series problems*. Time-series

analysis usually involves predicting numeric outcomes, such as the future price of an individual stock or the closing price of a stock index. Three additional applications suitable for time-series analysis include the following:

- Tracking individual customers over time to determine if they are likely to terminate the use of their credit card
- Predicting the likelihood of automotive engine failure
- Predicting the weekly rushing yards of an NFL running back

Much of the work with time-dependent data analysis has been statistical and limited to predicting the future value of a single variable. However, we can use both statistical and nonstatistical data mining tools for time-series analysis on one or several variables. Fortunately, we are able to apply the same techniques we use for other data mining applications to solve time-series problems. Our ability to succeed is determined to a large extent by the availability of relevant attributes and instances, as well as by the difficulty of the problem at hand. Let's look at an interesting application of time-series analysis.

13.1.1 Stock Market Analytics

Human experts exist in fields such as medicine, automotive repair, computer programming, and judicial law to name a few. Although a solid education is part of the process of becoming an expert, experience most often plays the larger role. Unfortunately, all of the experience in the world does not create an “expert stock market analyst.” The problem is not only in the large number of variables affecting stock market trends but also in the fact that a change in the value of a variable might be interpreted in several ways. In one case, a rise in interest rates causes stocks to tumble as it now costs companies more to borrow money. In another case, that same rise in rates sends stocks soaring as it means that the economy is improving. One thing is certain: there is no lack of “experts” willing to give detailed ex post facto analyses of any market moving situation! Knowing this, many market analysts work very hard at their trade and tend to perform well most of the time. These individuals fall into one of four general groups.

One set concentrates solely on analyzing company fundamentals. They examine company financial statements and talk with company representatives in an attempt to better understand the company's future. A second group makes their trading choices based solely on recent market trends and trading movements. These individuals are known as technical analysts. These market technicians use stock charts, moving averages, technical support levels, and standard deviations to determine what the future holds for a particular stock or index fund. The third set of individuals combines both techniques, relying at times more on fundamentals and at other times more on technical analysis. Then there's the fourth group, known as contrarians, who base their decisions on market sentiment. These individuals rely on the fact that the market tends to be either too optimistic or too pessimistic at any given time. The contrarian bets against market sentiment, buying when most are selling or selling when the majority of folks are buying. Our interest lies with the group of

technical analysts, many of whom incorporate time-series analysis through data mining as part of their technical tool kit. Let's see how they do it!

13.1.2 Time-Series Analysis—An Example

For our example, we use an exchange traded fund (ETF) with symbol XIV. An ETF is an index fund that can be bought or sold just like a stock. For example, a commodity index fund such as GLD (gold) or USO (United States oil) trades in unison with the corresponding commodity. VGT is a technology index fund where Apple Computer, Microsoft Corporation, Facebook, and Google make up 35% of its holdings. XLF is a financial ETF with top holdings of Berkshire Hathaway, Wells Fargo, and JP Morgan Chase and Company. Owning an index fund is, in general, a much safer bet than owning individuals stocks as bad news for one stock doesn't necessarily mean disaster for the entire basket of stocks held by the fund. Because of this, the past few years have seen the birth of thousands of ETFs.

The index fund of interest to us is XIV, a fund that trades inversely with market volatility rather than any specific market sector. In a more technical sense, the inverse relationship is with the S&P 500 VIX short-term futures index. XIV is of special interest to market traders and especially market timers for at least two reasons. First, XIV's value tends to increase over time, meaning that if you hold the fund for long periods, you will likely make money or, in the worst case, you won't lose your initial investment. More importantly, XIV experiences large price swings to both the upside and downside, making it a frequent market trader's dream. It is not unusual to see XIV trade from three to five percent higher (or lower) in a single day! If we can build a model to identify when and in what direction XIV moves, the potential for significant profit is excellent!

To begin our experiment with XIV, we apply the KDD process model starting with a goal.

Our goal is to build a time-series model able to predict the next-day closing price for XIV 15 to 30 minutes before current-day market closure. For algorithms unable to provide numeric output, our model will determine if the next-day move in price will be to the upside (positive) or to the downside (negative).

The goal clearly states that the prediction comes right before the market closes. In this way, we can make and act on our decision prior to market closure. This is of primary importance as waiting to make the trade until the next-day market open may be too late. For example, if XIV is going to do well the next trading day, its next-day opening price will likely be higher than the current-day close. This thereby limits our opportunity to make a profit.

If our model proves to be satisfactory, we will use it as follows:

- If the model tells us XIV will show a next-trading-day gain, we purchase the fund.
- If the model tells us XIV is to close lower, we sell our shares.
- If we don't currently own XIV and our model indicates a lower next-day close, we consider purchasing shares of the volatility futures index fund VXX as its price movement shows an almost perfect negative correlation with XIV.

Lastly, if our model is able to provide numeric output, we have additional information and can base our decision on the size of the predicted move. If the predicted price move is minimal, we may consider simply holding any owned shares. Is it too early to start adding up our profits? Let's find out!

13.1.2.1 Creating the Target Data Set—Numeric Output

The data sets for our experiments with time-series data have been created for you. Here we briefly summarize the process used for creating the data.

We used the *Yahoo Historical Stock Data operator* available in the freely downloadable QuantX-1 Extension for RapidMiner to extract the end-of-day closing prices for XIV between January 1, 2016 and March 21, 2016. Figure 13.1 shows the process model for obtaining the prices. The process first attains the closing prices for XIV using the aforementioned operator. Figure 13.2 offers the output format for the obtained prices. Next, the *Transpose* operator is applied to the pricing data. The *transpose* operator works like a standard matrix transpose function in that columns become rows and rows become columns. That is, instead of having 50 rows and 2 columns of data, we now have 50 columns and 2 data rows. Finally, we write the transposed data to an Excel spreadsheet file. Once in this format, we can easily create the time-series structure defined below.

13.1.2.2 Data Preprocessing and Transformation

There are several methods for representing time-series data. Our method builds a time lag directly into the data. That is, any given row of data contains the current day's fund price followed in order by the fund price for a specific number of previous days. To see this,

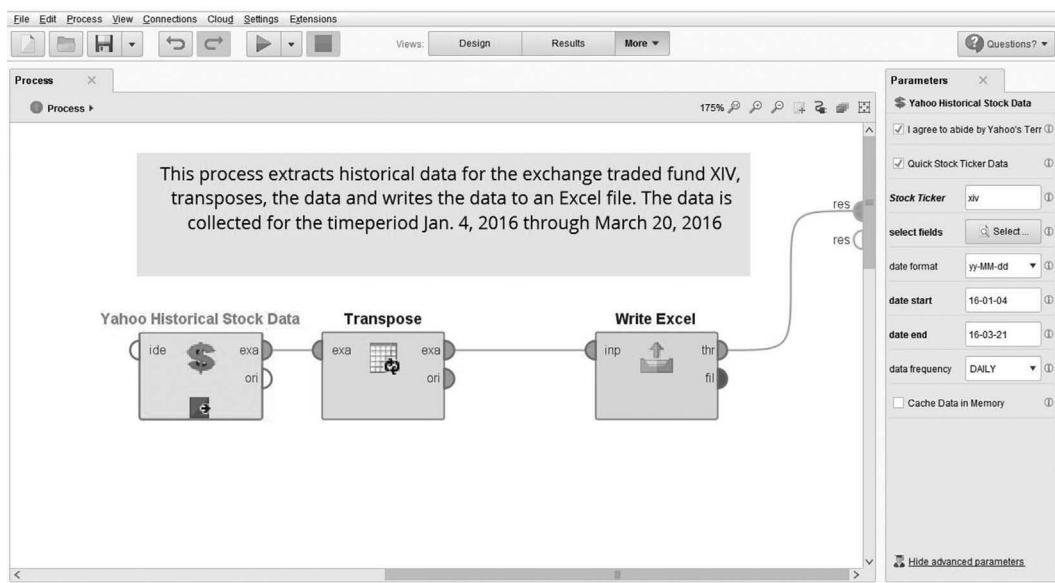


FIGURE 13.1 A process model for extracting historical market data.

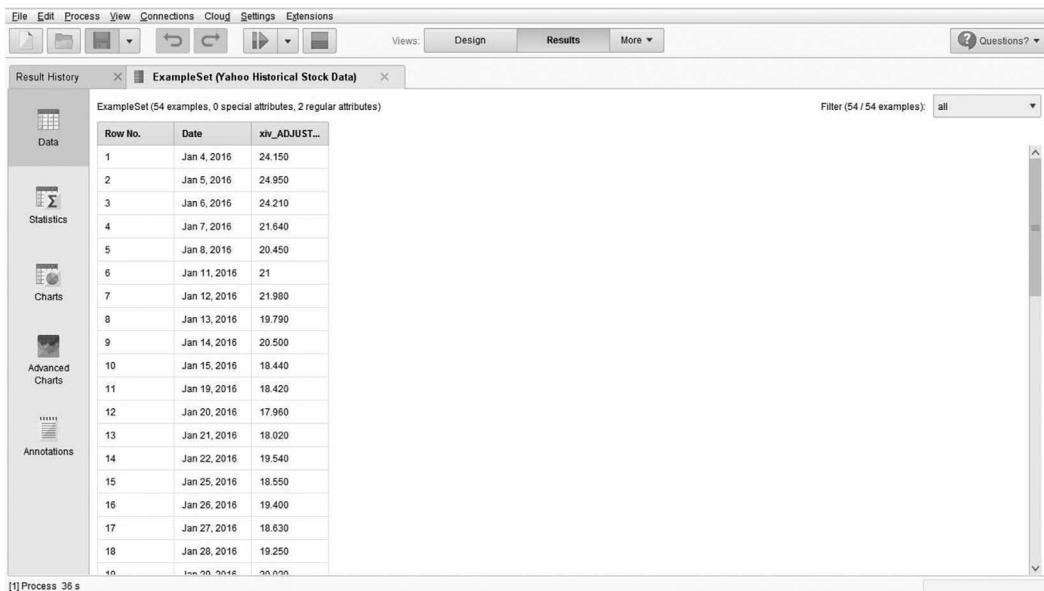


FIGURE 13.2 Historical data for XIV.

consider Table 13.1, which uses a time lag of 4 days. The table shows the closing price of XIV for the week of January 4 through January 8, 2016.

Data for the second day of trading is provided in Table 13.2. As January 8 is a Friday, the next trading day is January 11. The January 11 closing price is 21.00 and appears as the leftmost table entry. The next four table entries are taken from the first through fourth columns of Table 13.1. The data for January 4 is lost. This same procedure is repeated throughout to create 50 days of trading data.

Figure 13.3 gives the time-series data for the first 14 of our 50 days of trading. The data in rows 2 and 3 correspond respectively to the data in Tables 13.1 and 13.2. The entire data set is available in the file *13TimeSeriesNumeric.xlsx* located in *datasetsRapidMiner.zip*.

Column A in Figure 13.3 has been added for specific use with RapidMiner (see the next section). Upon opening the file, you will see that each row except the final one has *Input* in column A. This indicates that the data in the given row is used for building and testing

TABLE 13.1 Daily Closing Price for XIV, January 4, 2016 to January 8, 2016

Jan. 8	Jan. 7	Jan. 6	Jan. 5	Jan. 4
XIV	XIV-1	XIV-2	XIV-3	XIV-4
20.45	21.64	24.21	24.95	24.15

TABLE 13.2 Daily Closing Price for XIV January 5, 2016 to January 11, 2016

Jan. 11	Jan. 8	Jan. 7	Jan. 6	Jan. 5
XIV	XIV-1	XIV-2	XIV-3	XIV-4
21.00	20.45	21.64	24.21	24.95

	Status	Date	XIV	XIV-1	XIV-2	XIV-3	XIV-4										
1	Input	1/8/2016	20.45	21.64	24.21	24.95	24.15										
2	Input	1/11/2016	21.00	20.45	21.64	24.21	24.95										
3	Input	1/12/2016	21.98	21.00	20.45	21.64	24.21										
4	Input	1/13/2016	19.79	21.98	21.00	20.45	21.64										
5	Input	1/14/2016	20.50	19.79	21.98	21.00	20.45										
6	Input	1/15/2016	18.44	20.50	19.79	21.98	21.00										
7	Input	1/19/2016	18.42	18.44	20.50	19.79	21.98										
8	Input	1/20/2016	17.96	18.42	18.44	20.50	19.79										
9	Input	1/21/2016	18.02	17.96	18.42	18.44	20.50										
10	Input	1/22/2016	19.54	18.02	17.96	18.42	18.44										
11	Input	1/25/2016	18.55	19.54	18.02	17.96	18.42										
12	Input	1/26/2016	19.40	18.55	19.54	18.02	17.96										
13	Input	1/27/2016	18.63	19.40	18.55	19.54	18.02										
14	Input	1/28/2016	19.25	18.63	19.40	18.55	19.54										
15	Input																

FIGURE 13.3 Time-series data with numeric output.

time-series models. The final row shows *OUTPUT* in column A. This indicates that this row contains the unknown value to be predicted. Specifically, we use the first 49 instances for model training and testing. The data in row 51 (50th instance) is then given to the model to predict the next-day closing price. Scroll the data set to see a closing price of \$24.44 for the 50th trading day. In general, this value will not be known, and the cell will contain a “?”.

13.1.2.3 Creating the Target Data Set—Categorical Output

We can modify our data to perform time-series analysis using data mining tools limited to categorical output. Figure 13.4 shows the transformation we use for our example. Specifically, Figure 13.4 indicates that we have added an attribute identified as *class* that has value *pos* if the closing price difference (XIV minus XIV-1) is positive. Likewise, *class* has value *neg* if the computation is negative. Given this transformation, we must avoid the mistake of using XIV as an input attribute. Doing so would require us to know the closing price of XIV for the day our model is to give an up or down move in price!

The categorical form of the XIV data set is available in the file *13TimeSeriesCategorical.xlsx*. This form of the data is used for our RapidMiner experiment given in the next section.

13.1.2.4 Mining the Data—RapidMiner

For the data mining phase of our process, we first use RapidMiner’s *Neural Network* model together with the categorical form of the time-series data. In the next section, we apply Weka’s *MultiLayerPerceptron* function to both the categorical and numeric form of the data. Neural networks are one of the best choices for building time-series models. Several end-of-chapter exercises ask you to verify this by comparing neural network models with alternative techniques.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Status	Date	XIV	XIV-1	XIV-2	XIV-3	XIV-4	Class										
2	Input	1/8/2016	20.45	21.64	24.21	24.95	24.15	neg										
3	Input	1/11/2016	21.00	20.45	21.64	24.21	24.95	pos										
4	Input	1/12/2016	21.98	21.00	20.45	21.64	24.21	pos										
5	Input	1/13/2016	19.79	21.98	21.00	20.45	21.64	neg										
6	Input	1/14/2016	20.50	19.79	21.98	21.00	20.45	pos										
7	Input	1/15/2016	18.44	20.50	19.79	21.98	21.00	neg										
8	Input	1/19/2016	18.42	18.44	20.50	19.79	21.98	neg										
9	Input	1/20/2016	17.96	18.42	18.44	20.50	19.79	neg										
10	Input	1/21/2016	18.02	17.96	18.42	18.44	20.50	pos										
11	Input	1/22/2016	19.54	18.02	17.96	18.42	18.44	pos										
12	Input	1/25/2016	18.55	19.54	18.02	17.96	18.42	neg										
13	Input	1/26/2016	19.40	18.55	19.54	18.02	17.96	pos										
14	Input	1/27/2016	18.63	19.40	18.55	19.54	18.02	neg										
15	Input	1/28/2016	19.25	18.63	19.40	18.55	19.54	pos										

FIGURE 13.4 Time-series data with categorical output.

Figure 13.5 is a partial listing of the data as they appear in RapidMiner. When the data are added to the local repository, *class* is given the role of *label*. *XIV*'s role is designated as *ID*, *Status* as *ID1*, and *Date* as *ID2*.

XIV must play the role of an ID as it has been replaced by the categorical attribute *class*. Here's why. The output attribute for the *Neural Net* operator must be categorical. If we are to have *XIV* play a part in building the model, it has to be an input attribute. As we know all of

Row No.	Class	Status	Date	XIV	XIV-1	XIV-2	XIV-3	XIV-4
32	pos	Input	Feb 24, 2016 ...	18.770	18.700	19.690	18.650	18.190
33	pos	Input	Feb 25, 2016 ...	19.440	18.770	18.700	19.690	18.650
34	neg	Input	Feb 26, 2016 ...	19.210	19.440	18.770	18.700	19.690
35	neg	Input	Feb 29, 2016 ...	18.840	19.210	19.440	18.770	18.700
36	pos	Input	Mar 1, 2016 ...	20.550	18.840	19.210	19.440	18.770
37	pos	Input	Mar 2, 2016 ...	20.850	20.550	18.840	19.210	19.440
38	pos	Input	Mar 3, 2016 ...	21.520	20.850	20.550	18.840	19.210
39	neg	Input	Mar 4, 2016 ...	21.260	21.520	20.850	20.550	18.840
40	neg	Input	Mar 7, 2016 ...	21.180	21.260	21.520	20.850	20.550
41	neg	Input	Mar 8, 2016 ...	20.350	21.180	21.260	21.520	20.850
42	pos	Input	Mar 9, 2016 ...	20.620	20.350	21.180	21.260	21.520
43	pos	Input	Mar 10, 2016 ...	20.980	20.620	20.350	21.180	21.260
44	pos	Input	Mar 11, 2016 ...	22.090	20.980	20.620	20.350	21.180
45	pos	Input	Mar 14, 2016 ...	22.430	22.090	20.980	20.620	20.350
46	neg	Input	Mar 15, 2016 ...	22.090	22.430	22.090	20.980	20.620
47	pos	Input	Mar 16, 2016 ...	22.900	22.090	22.430	22.090	20.980
48	pos	Input	Mar 17, 2016 ...	23.640	22.900	22.090	22.430	22.090
49	pos	Input	Mar 18, 2016 ...	23.710	23.640	22.900	22.090	22.430
50	pos	OUTPUT	Mar 21, 2016 ...	24.440	23.710	23.640	22.900	22.090

FIGURE 13.5 Time-series data for processing with RapidMiner.

the values for XIV up until the day of prediction, we can actually build our model with XIV as an input attribute. The problem comes when it's time to apply the model. Recall that the job of the model is to predict whether the next-day closing price will be higher or lower. The value for the next-day closing price for output attribute *Class* will be ?. But what about XIV? The next-day closing price of XIV must also be ?. As XIV is an input attribute, upon seeing the unknown value, our model will tell us it can't handle missing data and terminate in error.

Figure 13.6 shows a 3-month price chart for XIV obtained from the data using a *series* chart. Notice that XIV performed poorly during the first 40 days of the 3-month time period but did very well during the final 6 weeks.

Before we look at our example, it is important to note that the sequential nature of time-series data makes model testing difficult. With these data, even a leave-one-out cross-validation will build models to predict supposed future fund prices that have long since occurred. A reasonable alternative is to iterate over the model building process by using the data from $n-1$ sequential trading days and applying the model to predict the outcome for the n th day of trading. That is, each new trading day builds a model that is used to predict the next-day outcome. The level of predictive success will be apparent after several trading days. As our example is for illustrative purposes, we show this idea for a single day of trading.

Figure 13.7 displays the main process model for our example. The categorical form of the time-series data is loaded into RapidMiner and retrieved. The *Filter Examples* operator extracts the first 49 instances for neural network for training. The 50th instance is passed to *Apply Model* (2) to test the created network. The neural network is built with one hidden layer of nine nodes. The shuffle parameter is turned off and *training cycles* is set at 2000. Higher values for this parameter create models that correctly classify all of the training data. However, using the smaller value reduces the chance that the model will work well when applied to the training data but fail when instances of unknown outcome are presented.

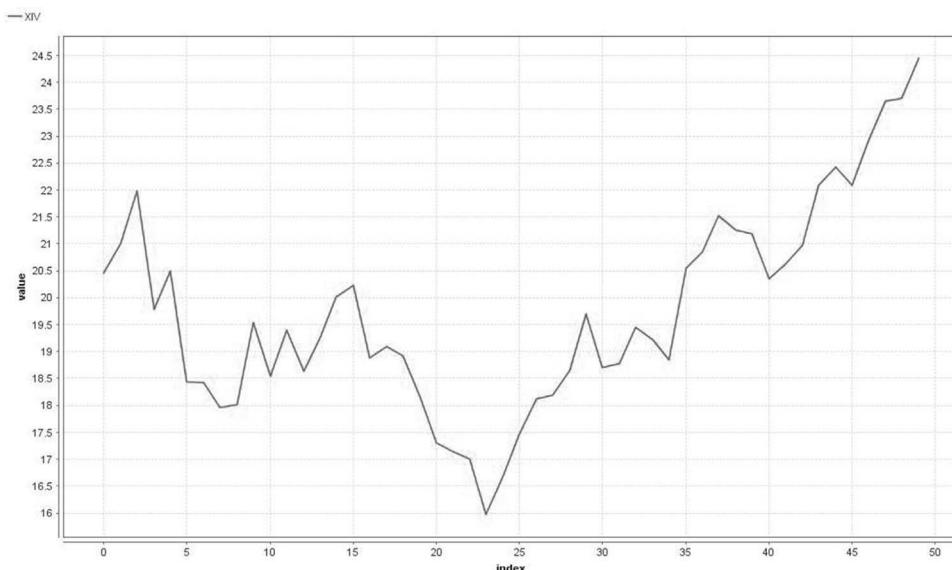


FIGURE 13.6 A 3-month price chart for XIV.

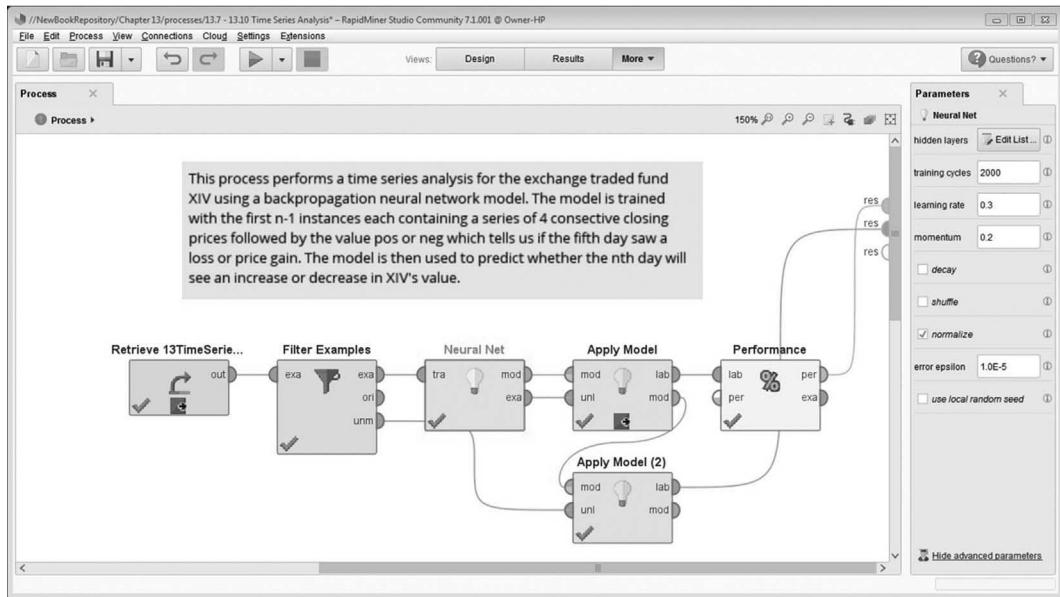


FIGURE 13.7 A process model for time-series analysis with categorical output.

Figure 13.8 displays a partial listing of actual and predicted class values as well as confidence scores for the training data. Figure 13.9 gives the performance vector for our model when applied to the training data. We see an average accuracy close to 96% and an absolute error (not shown) of 0.115. Figure 13.10 gives the prediction for the 50th day. Following the model's recommendation turns out to be the correct choice.

The screenshot shows the "Result History" view for the "ImprovedNeuralNet (Neural Net)" process. The "ExampleSet (Filter Examples)" tab is selected, displaying a table of 49 examples. The columns include Row No., Class, prediction(class), confidence(class), confidence(unlabeled), Status, Date, XIV, XIV-1, XIV-2, XIV-3, and XIV-4. The table shows various predictions and confidence scores for different dates, with most entries being "pos" or "neg".

Row No.	Class	prediction(class)	confidence(class)	confidence(unlabeled)	Status	Date	XIV	XIV-1	XIV-2	XIV-3	XIV-4
1	neg	neg	1.000	0.000	Input	Jan 8, 2016 ...	20.450	21.640	24.210	24.950	24.150
2	pos	pos	0.000	1.000	Input	Jan 11, 2016 ...	21	20.450	21.640	24.210	24.950
3	pos	pos	0.021	0.979	Input	Jan 12, 2016 ...	21.980	21	20.450	21.640	24.210
4	neg	neg	0.638	0.362	Input	Jan 13, 2016 ...	19.790	21.980	21	20.450	21.640
5	pos	pos	0.020	0.980	Input	Jan 14, 2016 ...	20.500	19.790	21.980	21	20.450
6	neg	neg	0.966	0.034	Input	Jan 15, 2016 ...	19.440	20.500	19.790	21.980	21
7	neg	neg	0.971	0.029	Input	Jan 19, 2016 ...	18.420	18.440	20.500	19.790	21.980
8	neg	neg	0.989	0.011	Input	Jan 20, 2016 ...	17.960	18.420	18.440	20.500	19.790
9	pos	neg	0.986	0.014	Input	Jan 21, 2016 ...	18.020	17.960	18.420	18.440	20.500
10	pos	pos	0.215	0.785	Input	Jan 22, 2016 ...	19.540	18.020	17.960	18.420	18.440
11	neg	neg	0.969	0.031	Input	Jan 25, 2016 ...	18.550	19.540	18.020	17.960	18.420
12	pos	pos	0.038	0.962	Input	Jan 26, 2016 ...	19.400	18.550	19.540	18.020	17.960
13	neg	neg	0.699	0.301	Input	Jan 27, 2016 ...	18.630	19.400	18.550	19.540	18.020
14	pos	pos	0.183	0.817	Input	Jan 28, 2016 ...	19.250	18.630	19.400	18.550	19.540
15	pos	pos	0.302	0.698	Input	Jan 29, 2016 ...	20.020	19.250	18.630	19.400	18.550
16	pos	pos	0.106	0.894	Input	Feb 1, 2016 ...	20.230	20.020	19.250	18.630	19.400
17	neg	neg	0.952	0.048	Input	Feb 2, 2016 ...	18.880	20.230	20.020	19.250	18.630
18	pos	pos	0.156	0.844	Input	Feb 3, 2016 ...	19.090	18.880	20.230	20.020	19.250
19	neg	neg	0.956	0.044	Input	Feb 4, 2016 ...	18.910	19.090	18.880	20.230	20.020

FIGURE 13.8 Predictions and confidence scores for time-series analysis.

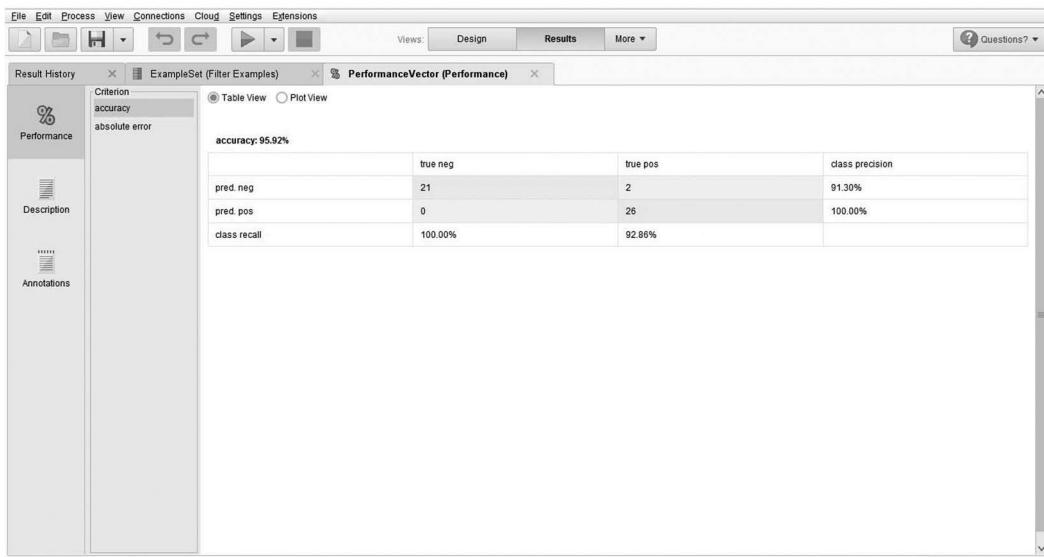


FIGURE 13.9 Performance vector—time-series analysis for XIV.

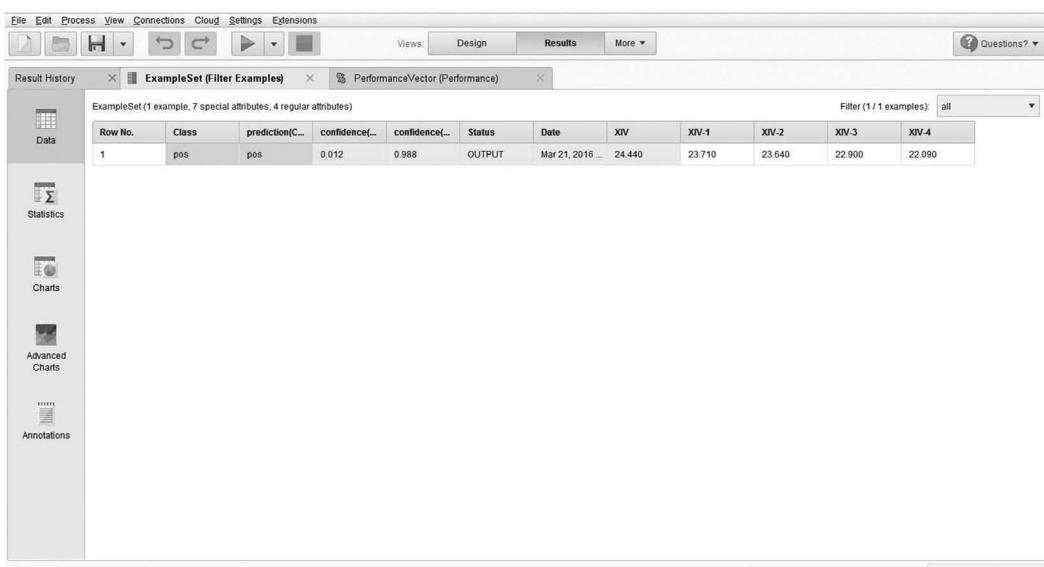


FIGURE 13.10 Predicting the next-day closing price of XIV.

13.1.2.5 Mining the Data—Weka

Weka's *MultiLayerPerceptron* function gives us the opportunity to experiment with both numeric and categorical output. The corresponding data sets, *13TimeSeriesNumeric.arff* and *13TimeSeriesCategorical.arff*, are found in the *TimeSeriesWeka.zip* file housed within *datasetsWeka.zip*. Figure 13.11 gives the first few instances of the categorical form of the data set. The numeric form of the data set (Figure 13.12) does not contain the *pos/neg* column as XIV is the numeric output attribute.

We initially built a model for the categorical form of the data set using the first 49 instances. The *date*, *status*, and XIV columns were removed, and all 49 instances were used for network training. After experimenting with several alternatives, a single hidden layer of nine nodes and 5000 training epochs was found to give an acceptable result. Figure 13.13 displays the mean absolute error (0.154) and classification correctness score (91.84%) for our model. We applied the model to predict the 50th-day price move of XIV. Figure 13.13 shows that the 50th-day move for XIV was correctly classified as positive.

To test the possibility of accurately predicting outcome over a period of several days, we built a model using the first 40 data set instances and applied the model to predict the outcome for day 41. We then added day 41, built a new model, and used it to predict the outcome for day 42. We repeated this process through day 49. Our results showed that 7 of the 10 models correctly predicted whether the next day move in XIV was positive or negative. The three errors were incorrect predictions of a negative price change. Although not conclusive, this result does provide motivation for additional experimentation.

```

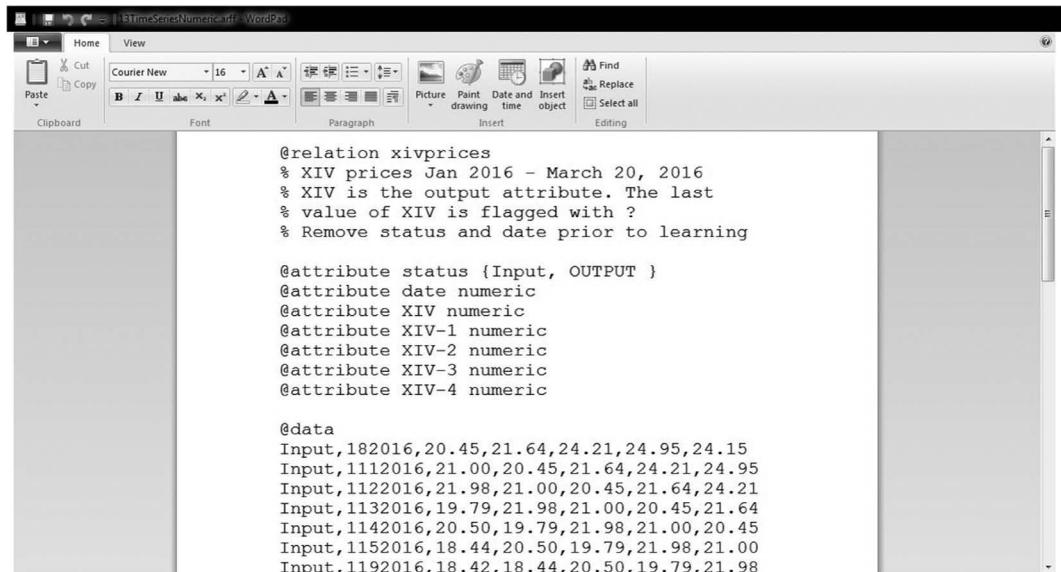
@relation xivprices
% XIV prices Jan 2016 through March 20, 2016
% Class is the output attribute
% !! Remove Status date and XIV prior to learning

@attribute status {Input, OUTPUT}
@attribute date numeric
@attribute XIV numeric
@attribute XIV-1 numeric
@attribute XIV-2 numeric
@attribute XIV-3 numeric
@attribute XIV-4 numeric
@attribute class {pos, neg}
@data

Input,182016,20.45,21.64,24.21,24.95,24.15,neg
Input,1112016,21.00,20.45,21.64,24.21,24.95, pos
Input,1122016,21.98,21.00,20.45,21.64,24.21, pos
Input,1132016,19.79,21.98,21.00,20.45,21.64, neg
Input,1142016,20.50,19.79,21.98,21.00,20.45, pos
Input,1152016,18.44,20.50,19.79,21.98,21.00, neg
Input,1162016,19.42,20.44,21.44,22.44,23.44, pos

```

FIGURE 13.11 Time-series data formatted for Weka—categorical output.



The screenshot shows a Microsoft WordPad window titled "Figure 13.12 - WordPad". The document contains the following text:

```

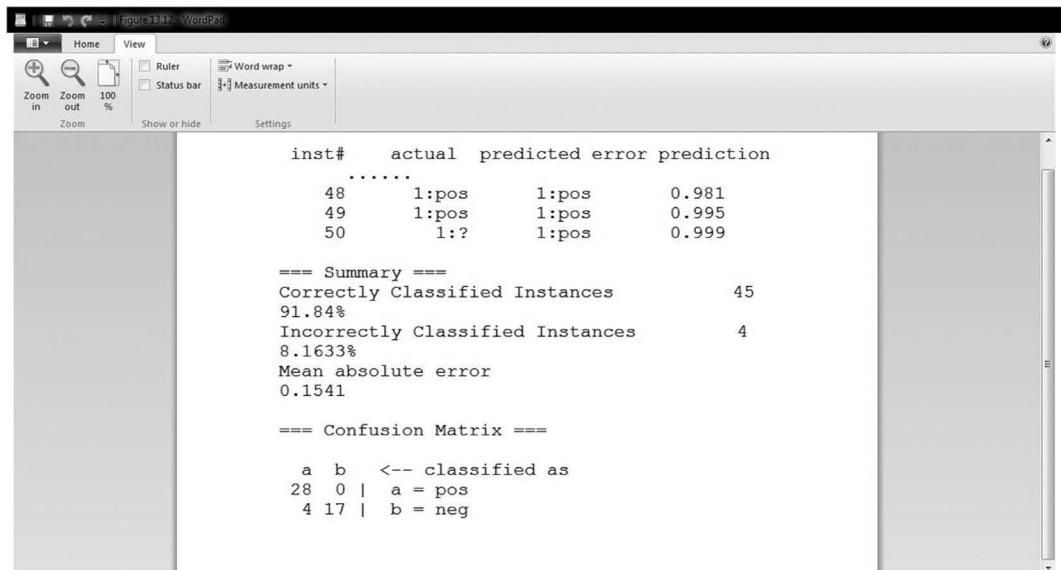
@relation xivprices
% XIV prices Jan 2016 - March 20, 2016
% XIV is the output attribute. The last
% value of XIV is flagged with ?
% Remove status and date prior to learning

@attribute status {Input, OUTPUT}
@attribute date numeric
@attribute XIV numeric
@attribute XIV-1 numeric
@attribute XIV-2 numeric
@attribute XIV-3 numeric
@attribute XIV-4 numeric

@data
Input,182016,20.45,21.64,24.21,24.95,24.15
Input,1112016,21.00,20.45,21.64,24.21,24.95
Input,1122016,21.98,21.00,20.45,21.64,24.21
Input,1132016,19.79,21.98,21.00,20.45,21.64
Input,1142016,20.50,19.79,21.98,21.00,20.45
Input,1152016,18.44,20.50,19.79,21.98,21.00
Input,1192016,18.42,18.44,20.50,19.79,21.98

```

FIGURE 13.12 Time-series data formatted for Weka—numeric output.



The screenshot shows a Microsoft WordPad window titled "Figure 13.12 - WordPad". The document contains the following text:

```

inst#    actual   predicted error prediction
.....
48      1:pos     1:pos      0.981
49      1:pos     1:pos      0.995
50      1:?       1:pos      0.999

==== Summary ====
Correctly Classified Instances          45
91.84%
Incorrectly Classified Instances        4
8.1633%
Mean absolute error                   0.1541

==== Confusion Matrix ====
a   b   <-- classified as
28  0   |   a = pos
  4  17  |   b = neg

```

FIGURE 13.13 Time-series analysis with categorical output.

Next, we turned our attention to the numeric form of the data set. Prior to conducting our experiment, we set XIV as the output attribute and removed attributes *status* and *date* from the data. After several tests, we finalized a model with two hidden layers each having 10 nodes. The model was trained with 50,000 epochs. This model gave us an excellent result when the training data were also used for model testing. Figure 13.14 shows this outcome. The mean absolute error of 0.0297 is impressive! However, when the model was applied to predict the 50th-day closing price of \$24.44, the \$22.42 prediction was disappointing.

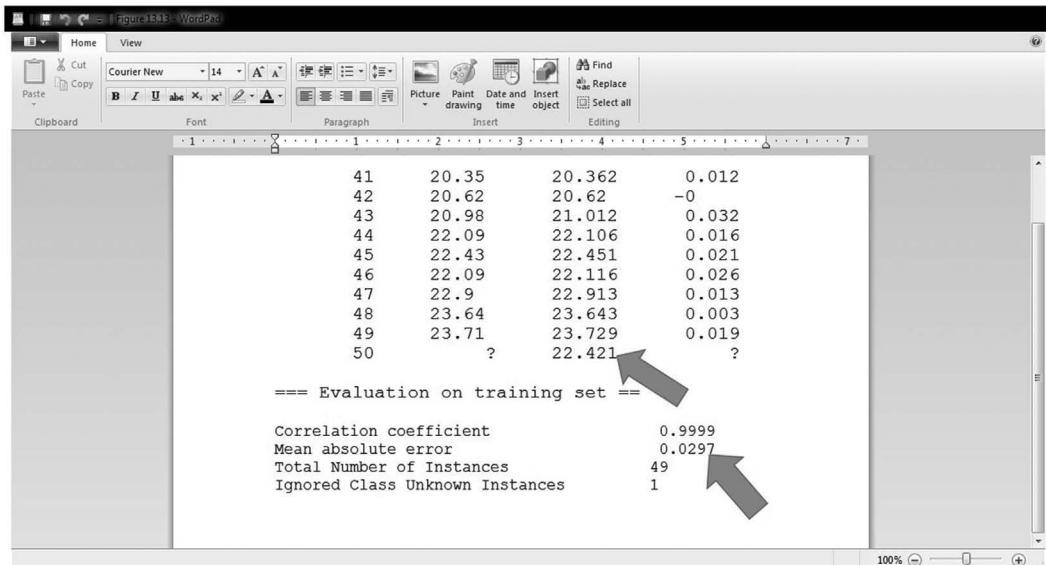


FIGURE 13.14 Time-series analysis with numeric output.

Although one test observation does not generalize, given the network's size, the possibility of an overtrained network must be considered. Performing the same experiments as described with the categorical form of the data is a logical choice for future tests.

For the final experiment, we applied Weka's SimpleKMeans clustering algorithm to the data set having categorical output to see how the instances cluster relative to the defined classes. Figure 13.15 gives the *Classes to Clusters* analysis, which shows a 48% classification

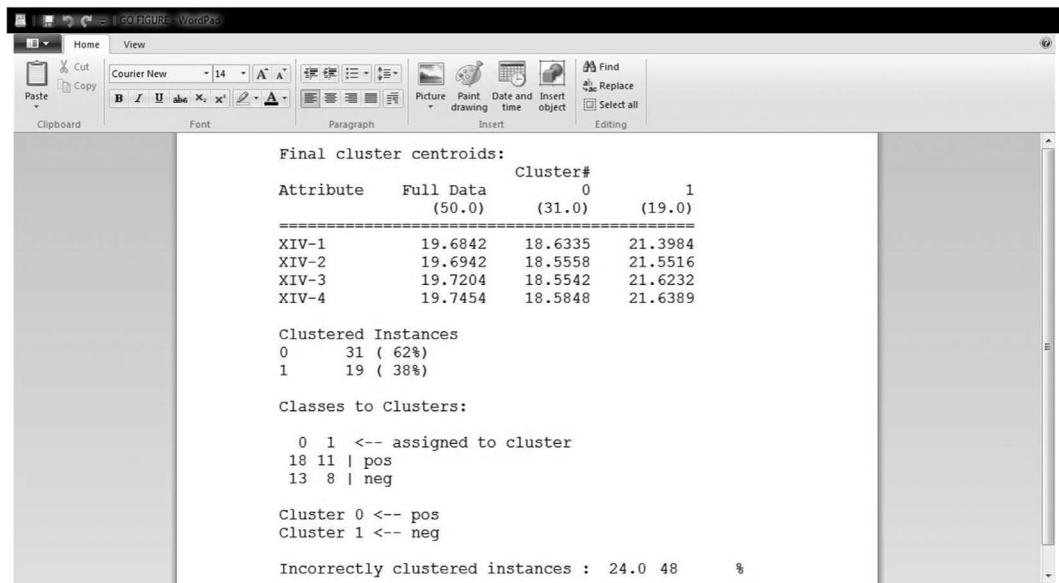


FIGURE 13.15 Cluster analysis using time-series data.

error. This error rate together with the values seen for the cluster centroids makes it clear that the instances clustered relative to their within-instance daily prices. One or several data transformations provide a starting point for future experiments.

13.1.2.6 Interpretation, Evaluation, and Action

The results of our experiments with RapidMiner and Weka indicate that at this point, predicting the direction of XIV's daily movement may be possible, but making an accurate estimate of the size of the move is not. Although the results seen when the output attribute is categorical are promising, further work by way of experimentation with additional data is necessary before any model can be put into practice.

13.1.3 General Considerations

Time plays a key role in many real-world problems. The following is a partial list of general considerations for building models for time-series applications:

- Once a model has been created, continue to test and modify the model as new data become available.
- If models built with the data in their current form are not acceptable, try one or more data transformations.

Common transformations include using time-series differences or percent change values. To illustrate, suppose we wish to represent the data in Table 13.1 as a set of difference values. To compute the leftmost entry for the first row of data, we subtract 21.64 from 20.45 giving, -1.19. Making similar computations, the modified row takes the following form:

$$\begin{array}{ccccc} -1.19 & -2.57 & -0.74 & 0.80 & 0.00 \end{array}$$

- Exercise caution when predicting future outcome with training data containing several fields with predicted rather than actual results.
- Use unsupervised clustering to determine if values of the input attributes allow the output attribute to cluster into meaningful categories. If this is not the case, the input attributes should be transformed or replaced as necessary.

Lastly, when the attribute to be predicted is categorical, we lose most of the precision found with numeric output. To help overcome this problem, we may consider further subdividing values for the output attribute. That is, rather than limiting the data in Figure 13.4 to two output attribute values (pos/neg), we transform the original numeric data into several categorical intervals. For example, we could have three subclasses for each gain or loss possibility: one subclass for gains less than 5%, a second subclass for gains between 5% and 10%, and a third subclass for gains greater than 10%. Subclasses for losses can be created in a similar manner. The end result is a six-class rather than a two-class structure.

13.2 MINING THE WEB

Companies relying on e-commerce for all or part of their business share a common goal: optimizing their website design so as to maximize sales. For example, a website that sells marketable products profits most by simultaneously displaying products customers are likely to buy together. On the other hand, a website requiring potential customers to spend an unwarranted amount of time to find their choice of products is likely to fail.

The success of a website ultimately depends on how it is viewed by the user community. Three major factors help determine how users perceive a website: the products or services offered by the site, individual Web page design, and overall site design (Spiliopoulou, 2000). Web page design and site design are related; however, the latter refers more directly to the intuitive nature of the indexing structure of a website.

Data mining can be deployed to help with at least three tasks that improve how users perceive and interact with a website. Data mining can be used to

- Evaluate a website to determine if the intent of the Web designer matches the desires of the user
- Personalize the products and pages displayed to a particular user or set of users
- Automatically adapt the indexing structure of a website to better meet the changing needs of its users

Before examining each of these possibilities in more detail, we first take a look at some of the issues seen with Web-based data mining.

13.2.1 Web-Based Mining: General Issues

Web-based data mining presents a whole new set of unique challenges not seen with other data mining applications. Let's examine a few of these issues through the eyes of the KDD process model.

13.2.1.1 *Identifying the Goal*

Recall that the first step of the KDD process requires us to establish one or more goals. Here is a partial list of plausible goals for a Web-based mining project.

- Decrease the average number of pages visited by a customer before a purchase transaction
- Increase the average number of pages viewed per user session
- Increase Web server efficiency
- Increase average visitor retention rates
- Decrease the total number of returns on purchased items
- Personalize Web pages for customers
- Determine those products for sale at a website that tend to be purchased or viewed together

Regardless of our goals, over 80% of the time spent on a Web-based mining project involves steps 2, 3, and 4 of the KDD process model. We detail these steps under the general category of data preparation.

13.2.2 Preparing the Data

The data available to us as a result of one or more Web-based user sessions are stored in Web server log files. A typical server log file houses information describing the *clickstream* sequences followed by users as they investigate Web pages and follow page links.

Server log files most often provide information in what is known as *extended common log file format*. The fields associated with the extended common log format are, in order: host address, date/time, request, status, bytes, referring page, and browser type.

For data mining, we are interested in those common log file fields that allow us to determine the sequence of clickstreams followed by each user as they navigate our website. Therefore, the job of the data preparation process is to extract relevant data from the Web sever logs and create a file suitable for data mining.

Figure 13.16 shows that the file created by the data preparation process is known as a *session file*. A session file contains from a few to several thousand records, each representing an instance of a user session. A user *session* is simply a set of pageviews requested by a single user from a single Web server. A single *pageview* is made up of one or more page files, each forming a single display window in a Web browser. Each pageview is tagged with a unique uniform resource identifier (URI) for purposes of identification during the data mining process.

Creating the session file is a difficult task for several reasons. First, to create the individual server sessions, we must be able to identify each user among the several users listed in a log file. Host addresses are of limited help because multiple users may be accessing a site from the same host. When the host address is combined with the referring page, we can more easily distinguish one user session from another. However, we are best able to differentiate between users if sites are allowed to use cookies. A *cookie* is a data file placed

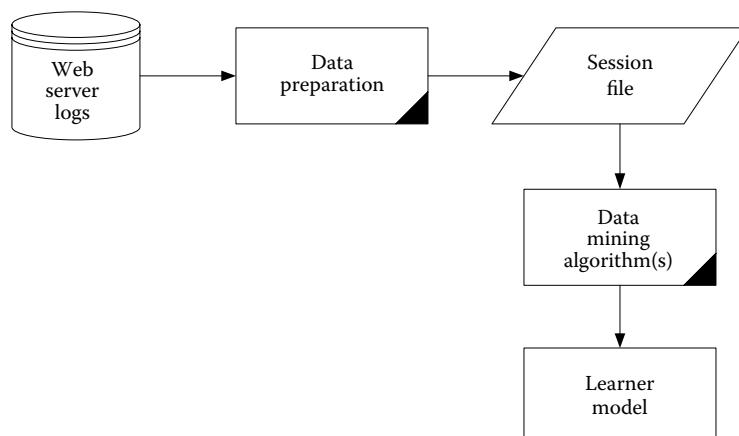


FIGURE 13.16 A generic Web usage data mining model.

on a user's computer that contains session information. Unfortunately, we cannot rely on the potential information available in cookies as many users are reluctant to give websites the authority to place cookies on their machines.

Problems with creating the session file do not end with difficulties in distinguishing users. A second problem is the fact that a single user page request oftentimes generates multiple log file entries from several types of servers. Many of these servers are image or ad servers. As we are not interested in entries created by application, image, and ad servers, we must have a technique to identify unwanted log entries so they do not become part of the session file. Several other potential problems with creating session files are summarized by Edelstein (2001).

Finally, one or more data transformations in the form of adding new variables to session records may be necessary. If repeat-customer records are available, information about the amount of money previously spent, average purchase amounts, as well as time of most recent transaction may prove useful.

13.2.2.1 Mining the Data

Once the session file is created, the session data are presented for data mining. The algorithms applied to the data may be traditional techniques such as association rule generators or clustering methods. However, depending on the goals of a data mining project, algorithms specifically designed for determining sequences of Web page visits may be a better choice.

13.2.2.2 Interpreting and Evaluating Results

The instances of a session file represent the pageview behavior of a single user during one session. Here we offer a simple example that shows how to interpret the results of a hypothetical Web-based data mining session using an association rule technique.

Consider the four hypothetical session instances where each P_i represents a pageview:

$$I_1: P_5 \rightarrow P_4 \rightarrow P_{10} \rightarrow P_3 \rightarrow P_{15} \rightarrow P_2 \rightarrow P_1$$

$$I_2: P_2 \rightarrow P_4 \rightarrow P_{10} \rightarrow P_8 \rightarrow P_{15} \rightarrow P_4 \rightarrow P_{15} \rightarrow P_1$$

$$I_3: P_4 \rightarrow P_3 \rightarrow P_7 \rightarrow P_{11} \rightarrow P_{14} \rightarrow P_8 \rightarrow P_2 \rightarrow P_{10}$$

$$I_4: P_1 \rightarrow P_3 \rightarrow P_{10} \rightarrow P_{11} \rightarrow P_4 \rightarrow P_{15} \rightarrow P_9$$

Suppose an association rule generator outputs the following rule from our hypothetical session data:

$$\begin{aligned} \text{IF } & \rightarrow P_4 \text{ and } P_{10} \\ \text{THEN } & P_{15} \{3/4\} \end{aligned}$$

The rule tells us that there are three instances where P_4 , P_{10} , and P_{15} appear together in a single session record. Also, a total of four session instances have P_4 and P_{10} appearing in

the same session. Therefore, we are 75% confident that each time a user references P_4 and P_{10} , the user also references P_{15} . Provided that direct links do not exist between the three pageviews, this result may warrant modifying the website indexing structure by placing one or more direct links between the pages. As a second possibility, if the precondition of the rule matches the activity of a current user of the system, P_{15} can be added to a list of recommended pageviews to be automatically presented to the user. In this way, the pages viewed by the user can be personalized to his/her likely interests.

Unsupervised clustering can also be employed to form clusters of similar session file instances. A variety of clustering algorithms and similarity measures can be used. One plausible strategy is to use agglomerative clustering, where instance similarity is computed by dividing the total number of pageviews each pair of instances have in common by the total number of pageviews contained within the instances. To illustrate this method, consider the following session instances:

$$I_1: P_5 \rightarrow P_4 \rightarrow P_{10} \rightarrow P_3 \rightarrow P_{15} \rightarrow P_2 \rightarrow P_1$$

$$I_2: P_2 \rightarrow P_4 \rightarrow P_{10} \rightarrow P_8 \rightarrow P_{15} \rightarrow P_4 \rightarrow P_{15} \rightarrow P_1$$

As instances I_1 and I_2 share five of eight total pageviews, the computed $I_1 I_2$ similarity is 0.625. In Section 13.2.4, we show one way to use the clusters formed by similar sessions to personalize the pages viewed by website users.

In addition to discovering patterns in Web data, it is often desirable to obtain summary statistics about the activities taking place at a website. Several commercial and free downloadable Web server log analyzers are available that offer log file summaries of website activity. The output of most log analyzers is an aggregation of gathered log file data displayed in a graphical format. A typical log analyzer produces statistics such as how often a website is visited, how many individuals fill a market basket but fail to complete a transaction, and which website products are the best and worst sellers.

13.2.2.3 Taking Action

Several possibilities exist for taking action based on the results of a Web-based data mining project. Here is a short list of candidate actions:

- Implement a strategy based on created user profiles to personalize the Web pages viewed by site visitors
- Adapt the indexing structure of a website to better reflect the paths followed by typical users
- Set up online advertising promotions for registered website customers
- Send e-mail to promote products of likely interest to a select group of registered customers

- Modify the content of a website by grouping products likely to be purchased together, removing products of little interest, and expanding the offerings of high-demand products
- Lastly, as the interests of users visiting a website are in a constant state of change, the effectiveness of the actions taken must be closely monitored and modified as necessary.

13.2.3 Data Mining for Website Evaluation

Website evaluation is concerned with determining whether the actual use of a site matches the intentions of its designer. If the paths traveled by a majority of site visitors are not those expected by the designer, the site is likely to be seen as difficult to navigate. If this is the case, the Web designer must consider changing the architecture of the website to more clearly fit the needs of its users.

Data mining can help with site evaluation by determining the frequent patterns and routes traveled by the user population. However, unlike many website data mining applications, website evaluation is concerned not only about clusterings of pageviews but also with the sequential ordering of pageviews. In this way, the Web designer can determine if the path structure of the website is indeed a best representation of the navigational habits of its users.

Because the data mining problem becomes one of sequence identification, a special class of data mining algorithms known as *sequence miners* are often used for website evaluation. Sequence miners are able to discover frequently accessed pages that occur in the same order.

13.2.4 Data Mining for Personalization

The goal of *personalization* is to present Web users with what interests them without requiring them to ask for it directly. Personalization can be manually implemented or performed automatically with the help of data mining methods. Manual techniques force users to register at a website as well as answer questions or fill out check boxes. As user bias is a factor, manual techniques are not likely to provide websites with a true picture of expected user actions. By using data mining to automate personalization, the subjectivity of user response is replaced by actual user behavior.

Figure 13.17 shows the general steps for automated personalization. The process involves creating usage profiles from stored session data. The most difficult part of the procedure is moving from the clusters formed by the data mining process to the usage profiles. Profiling is easier if users are required to register in order to use the services provided by a website. In this case, actual user browsing behavior can be supplemented with compiled demographic data.

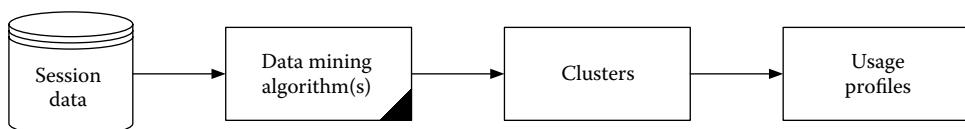


FIGURE 13.17 Creating usage profiles from session data.



FIGURE 13.18 Hypertext link recommendations from usage profiles.

Lastly, Figure 13.18 shows a recommendation engine that takes as input the set of created usage profiles and matches current user navigational activity with the stored profiles. The recommendation engine outputs recommended hypertext links to be displayed to the user. For e-commerce applications, the majority of recommended pageviews will be links to products and advertising.

13.2.5 Data Mining for Website Adaptation

Web masters make initial decisions about how a new website is to be organized. Their decisions are strongly influenced by how they believe the new site will be accessed by its users. However, the chances of a Web designer anticipating a majority of actual user needs is quite slim. Even if an initial design is adequate for most users, over time, the needs of users change. Once-useful index links become unused, and new patterns of Web usage emerge. Therefore, for a website to remain competitive, it becomes necessary for the indexing structure of the site to change over time. New page links must be added and little-used links deleted. The necessary changes can be manually applied; however, a better approach is to use data mining to automate the process. Websites that are able to semiautomatically improve their internal structure as well as their methods of presentation by learning from visitor access patterns are known as *adaptive websites*.

The internal structure of a website has at its core a set of index pages. An *index page* is a Web page having links to a set of pages detailing a particular topic. Therefore, the problem of automatically improving the internal structure of a website becomes one of index page synthesis. Perkowitz and Etzioni (2000) provide a formal definition of the *index synthesis problem*:

Given a website and a vistor access log, create new index pages containing collections of links to related but currently unlinked pages (p. 155).

Perkowitz and Etzioni (2000) describe IndexFinder, an automated page synthesis system that uses a clustering algorithm to generate candidate index pages for an existing website. IndexFinder also creates a rule-based description of each candidate page. The Web master must decide whether to accept or reject the candidate pages. If a new index page is accepted, IndexFinder creates a final page form and automatically adds the page to the website. The Web master gives the page a title and determines where in the site the page will reside.

13.2.6 PageRank and Link Analysis

One of the great Web mining successes is the PageRank algorithm developed in 1996 by Larry Page and Sergey Brin, the cofounders of Google. PageRank is classified as a *link*

mining algorithm and plays an important part in the scheme used by Google and other search engines to determine the links displayed as the result of a user query. Link mining focuses on the relationships seen between the nodes of a network. With the Internet as the network structure, the task of link analysis is to make deductions about the nature or importance of websites based on the incoming and outgoing hyperlinks.

PageRank gives a numerical value of importance or prestige to webpage P using the following factors:

- The set of pages that link to P
- The number of outbound links from P
- The total number of pages under consideration

For any page P , the greater the number of incoming links to P , the higher its prestige. This makes intuitive sense as it implies that a large community is interested in what P has to offer. Further, the amount of prestige added to P by incoming link Q depends on Q 's own prestige value. Lastly, the fewer the number of outbound links from P , the higher its prestige value. This also makes intuitive sense as it implies that P contains most of the information of interest and hence need not send outbound links to other pages. To visualize these ideas, consider the page link structure shown in Figure 13.19. Page P_2 has five incoming links and one outbound link. P_0, P_1, P_4, P_7 , and P_3 all contribute to the prestige of P_2 . However, as P_1 has three incoming links and the single outbound link, it contributes more to the prestige value of P_2 than any of the other links. Likewise, P_9 contributes more to the prestige of P_1 than either P_5 or P_6 .

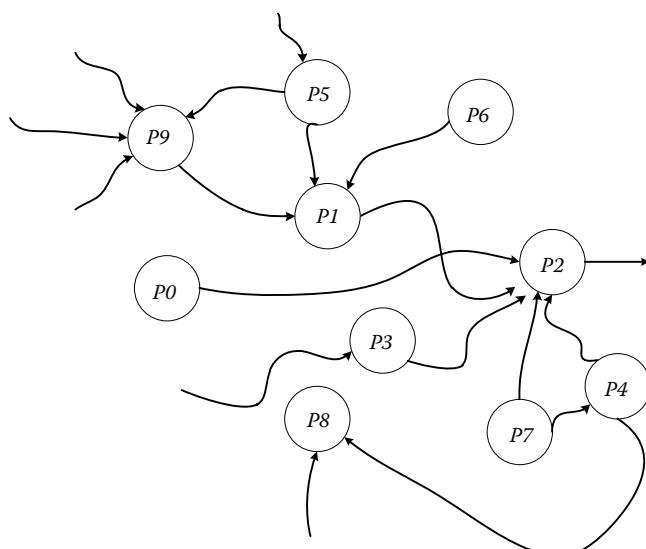


FIGURE 13.19 A page link structure.

Although we do not give the exact equation for the algorithm, the PageRank of P is given as follows:

1. For each incoming page P_i , divide the PageRank of P_i by the total number n_i of its outbound pages.
2. Sum all values computed in (1) to obtain the PageRank of P .

Lastly, it is important to note that the PageRank algorithm is applied only after the initial set of relevant pages have been determined by several other factors.

13.2.7 Operators for Web-Based Mining

A freely downloadable Web extension is available with RapidMiner studio. To install the extension, open RapidMiner, click extensions and then marketplace, and search for the Web mining extension package, which contains several operators, including operators to retrieve single or multiple Web pages. Of particular interest is the *Crawl Web* operator, which allows you to crawl the Web and store the collected pages as an example set. Particular caution must be taken with this operator as the number of collected pages can be unmanageable. We encourage you to start your experiments with the *Crawl Web* operator with the URL of your personal webpage.

13.3 MINING TEXTUAL DATA

Textual data mining involves extracting useful patterns from free text. As textual data are unstructured, an initial investigation of such a task seems to be nearly impossible. However, textual data mining is possible because we are only trying to categorize textual data, not completely understand its contents.

A common problem that can be solved using textual data mining is to determine whether or not a document is about a specific topic. This problem can be considered a binary classification problem with the output of a classifier limited to a *yes* or *no* response. For example, we may wish to examine daily articles in several newspapers that deal in one way or another with the stock market. Here is a textual mining algorithm that could be used for the problem:

1. *Train.* Create an attribute dictionary where the attributes represent words from articles dealing with stock market topics. Choose only those words that occur a minimum number of times.
2. *Filter.* Remove common words known to be useless in differentiating articles of one type from another.
3. *Classify.* Check each new document to be classified for the presence and frequency of the chosen attributes. If a certain document contains a predetermined minimum number of references to the chosen attributes, classify the document as a member of the class of articles dealing with stock market topics.

More difficult textual mining problems involve the analysis of free-form text as it is found in e-mail documents, recorded telephone transcripts, and the like. The nature of such messages requires that the textual mining system deal with ambiguities as well as spelling and grammar errors. In the case of transcript analysis, it is easy to understand the desire to be able to analyze transcript documents to better understand customers likely to make purchases or to continue with a provided service. Some success has been achieved in this area by performing a quantitative as well as a qualitative data analysis. The quantitative analysis looks at directly measurable items such as the length of calls and call time of day frequency distributions. The qualitative analysis concerns itself with document content for the purpose of classifying individual transcripts. To better understand this idea, let's consider a simple example that combines both quantitative and qualitative analysis.

13.3.1 Analyzing Customer Reviews

Suppose an online discount shoe store asks customers to rate their purchases relative to shoe comfort, price, durability, and fit. Customers also provide an overall purchase *satisfaction score*. All attributes are rated on a scale from 1 to 10, with 10 being the best possible score. In addition to these measures, customers are provided with a text box where they are able to give a written account of their purchases. The rating scale scores as well as their written comments are viewed by a member of the staff prior to having them appear on the store's website. Both positive and negative reviews are posted, with the exception of reviews containing inappropriate language.

The company desires to use data mining in order to automate the posting process. By automating the process, they free a part-time staff position. More importantly, based on comments from the staff who currently handle the reviews, the firm believes that an automated analytics procedure will help them better understand the needs of their customers. The company is particularly interested in identifying discrepancies between overall satisfaction scores and written evaluations.

To automate the process, customer satisfaction scores are grouped into one of three classes. Satisfaction scores between 8 and 10 are considered positive. Scores between 5 and 7 are tagged as neutral, and scores below 5 are given a negative label. An instance of data consists of the written critique together with its associated positive, neutral, or negative class label. As the written evaluation is free text, each evaluation must be converted into a *feature vector*, where selected words become features (attributes) and frequency counts make up their values.

The construction of the feature vector involves preprocessing and dimension reduction. Preprocessing consists of several steps, including an initial conversion that removes irrelevant sentences, takes care of spelling errors, and transforms variations of a given word into a single lexical equivalent. Reviews are also scanned for vulgar terminology. Any such review is discarded.

Additional preprocessing consists of *tokenization* (which breaks sentences into smaller units called tokens), stopword removal, case conversion, and word *stemming*. Word

stemming is a procedure designed to convert words to their root form in order to avoid having a single word represented by two or more features.

Once the feature vector is preprocessed, some form of dimensionality reduction must take place. Except in the most trivial of cases, without a reduction in the number of features, the feature space of words and word counts is not manageable. One popular reduction technique is singular value decomposition (SVD), which determines a weight matrix that projects the features into a smaller dimension while still maintaining most of the information in the original data. Once the feature vectors have been preprocessed, supervised learning will be used to build a model representing the data.

The model will be trained with instances tagged by staff reviewers prior to preprocessing that represent each of the three classes. As new reviews are written, they are preprocessed and presented to the supervised model. The model classifies the review as positive, negative, or neutral. If the classification of the written review matches the satisfaction score category, the review is posted. If the model misclassifies the review, the review is transferred to a staff member for analysis. An example of a misclassification might be a positive product satisfaction score but a negative review because the customer had a poor experience in a phone conversation about a billing issue. In this case, the staff member handling the discrepancy can call the customer and offer an apology for his/her negative experience.

To be sure, our illustration is by no means complete as our example set is small and our process model is missing an evaluation component. We hope the example provides enough information to motivate you to further investigate this rapidly expanding area of data mining. As written text has become our most common form of information exchange, textual data mining will continue to play an increasingly important role in our society. In addition, the expansion of the World Wide Web makes the prospects for textual data mining unlimited.

RAPIDMINER TUTORIAL

Text Mining Customer Reviews with RapidMiner

Figures 13.20 through 13.27 offer a partial implementation of the customer review model described above. For our example, we used RapidMiner's freely downloadable text processing extension together with the data found in the zip file *CustomerReviews* housed within *datasetsRapidMiner.zip*.

- To follow our illustration, create three folders respectively named positive, negative, and neutral. Place the four text files in the zip file *PositiveFolder* into the positive folder. Place the two text files in *NegativeFolder* into the negative folder. Finally, place the two text files in *NeutralFolder* into the neutral folder.
- Create the main process model as in Figure 13.20. *Process Documents From Files* is a nested operator and serves two purposes. First, it is used to bring the individual text documents into the processing environment. Secondly, it is used to preprocess the data, thereby creating the feature vectors (examples) to be mined. Click the operator then mouse to and click edit list. Your screen will display as in Figure 13.21.
- Enter appropriate class names and specify the pathnames to the three folders containing the individual text documents (Figure 13.22). Click *Apply*.

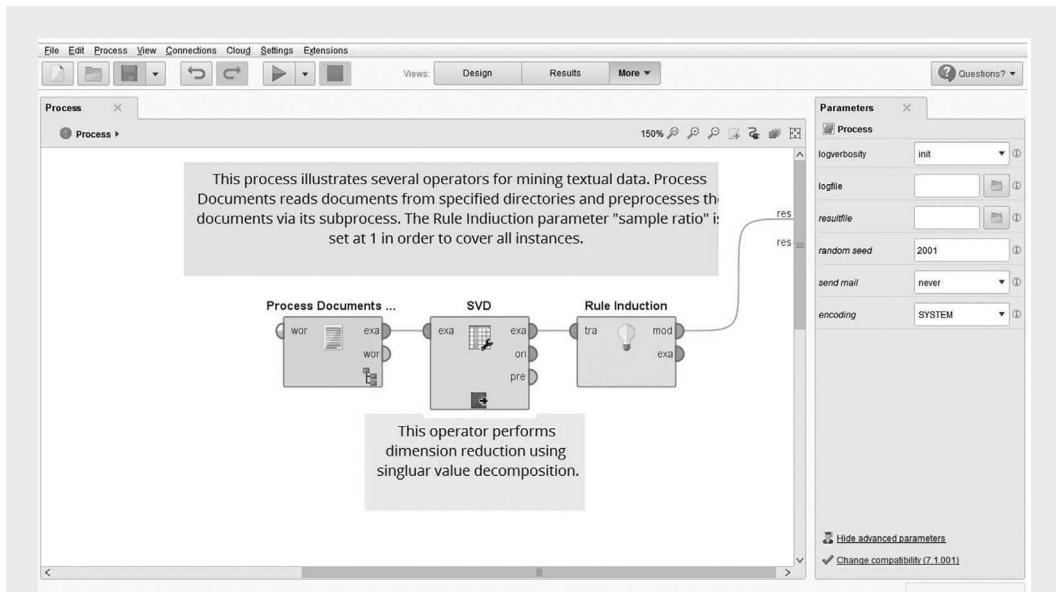


FIGURE 13.20 A main process model for mining textual data.

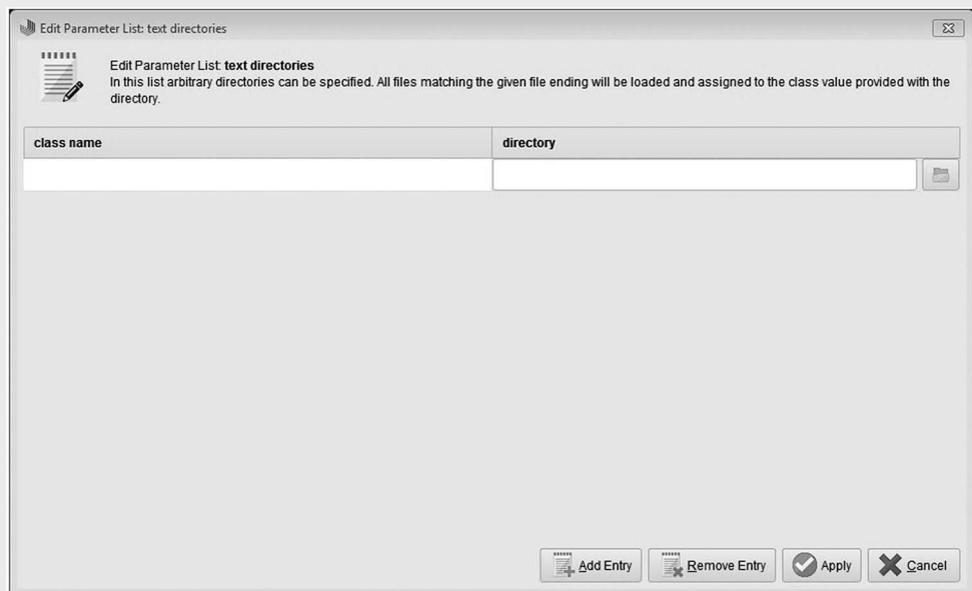


FIGURE 13.21 A template to enter folder names used for textual data.

- Double click on *Process Documents from Files* and create the nested process displayed in Figure 13.23. Place a breakpoint after each operator in Figures 13.20 and 13.23 so you can follow the output of each preprocessing operator.
- Return to the main process. Click on *SVD* operator and set the dimensions parameter at 2. Execute your process model and watch how the individual preprocessing operators modify the text.

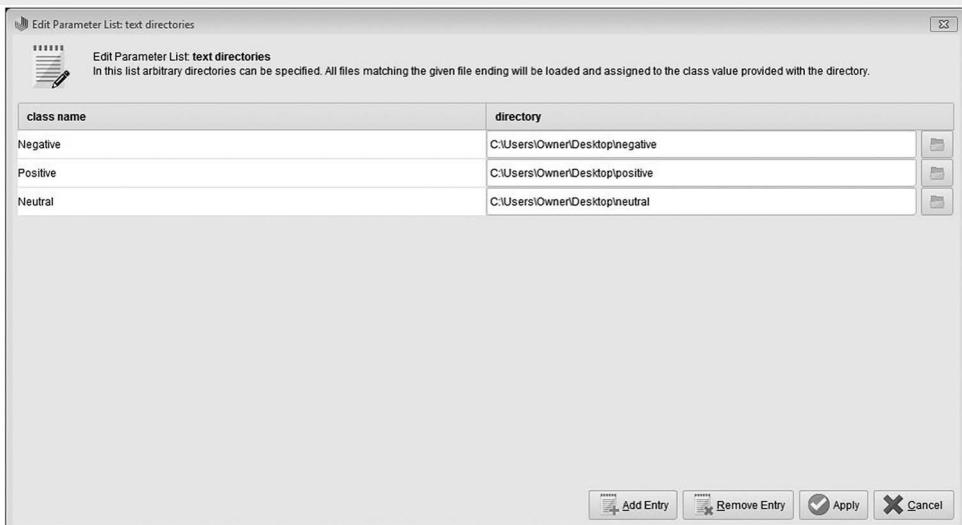


FIGURE 13.22 Class and folder names containing textual data.

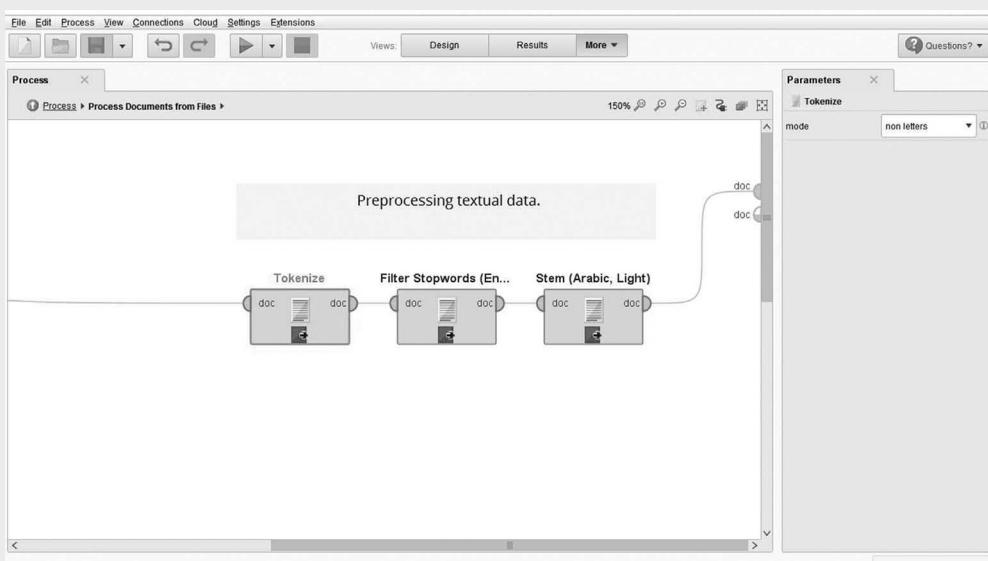


FIGURE 13.23 Subprocess for tokenizing and stemming textual data.

Step through the breakpoints until you see Figure 13.24, which shows a split screen with the original text for pos3.txt in the lower half of the screen and the resultant tokenization in the upper screen portion. Keep stepping through breakpoints until the top half of your screen shows the tokenized and stemmed evaluation displayed in Figure 13.25. Figure 13.26 shows the SVD dimension reduction whereby each original text document is now represented by

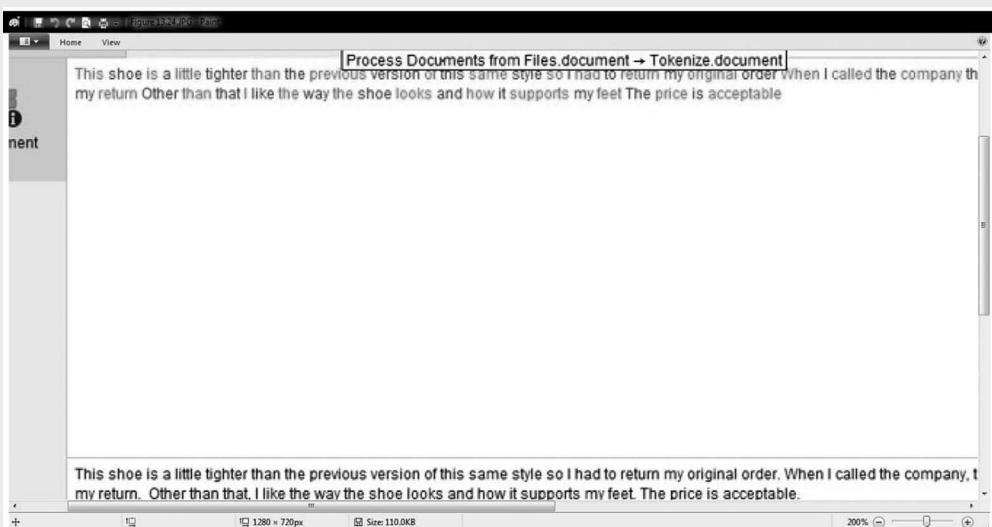


FIGURE 13.24 A tokenized positive evaluation.

shoe tighter previous version style i return original order
called company helpful return i shoe looks supports feet
price acceptable

FIGURE 13.25 A tokenized and stemmed positive evaluation.

label	metadata_file	metadata_p...	metadata_d...	svd_1	svd_2
Negative	neg1.txt	C:\Users\Ow...	Apr 1, 2016 7:...	-0.360	-0.569
Negative	neg2.txt	C:\Users\Ow...	Apr 1, 2016 7:...	-0.086	0.019
Positive	pos1.txt	C:\Users\Ow...	Apr 1, 2016 7:...	-0.578	0.334
Positive	pos2.txt	C:\Users\Ow...	Apr 1, 2016 7:...	-0.185	0.154
Positive	pos3.txt	C:\Users\Ow...	Apr 1, 2016 9:...	-0.225	-0.181
Positive	pos4.txt	C:\Users\Ow...	Apr 1, 2016 7:...	-0.366	0.223
Neutral	nut1.txt	C:\Users\Ow...	Apr 1, 2016 7:...	-0.452	0.304
Neutral	nut2.txt	C:\Users\Ow...	Apr 1, 2016 7:...	-0.325	-0.605

FIGURE 13.26 Textual data reduced to two dimensions.

RuleModel

```
if svd_2 > -0.375 and svd_1 ≤ -0.135 and svd_1 > -0.409 then
Positive (0 / 3 / 0)
```

```
if svd_1 > -0.406 and svd_2 > -0.587 then Negative (2 / 0 / 0)
```

```
if svd_1 > -0.515 then Neutral (0 / 0 / 2)
```

```
else Positive (0 / 1 / 0)
```

```
correct: 8 out of 8 training examples.
```

FIGURE 13.27 Rules defining the three product evaluation classes.

two numeric attributes. Lastly, Figure 13.27 gives the rules created by the rule induction operator when presented with the transformed instances. We now have an automated technique that accepts written customer comments and determines whether the comments are positive, negative, or neutral. Most importantly, discrepancies between overall satisfaction scores and written evaluations can be identified and acted upon as deemed necessary.

13.4 TECHNIQUES FOR LARGE-SIZED, IMBALANCED, AND STREAMING DATA

Previously, you learned that acquiring and preprocessing data is the most time-consuming part of the data mining process. Here are three circumstances that often require special attention:

- The acquired data are too large to fit into memory.
- The classes we are trying to model are not equally represented in the data.
- Data are streaming and cannot be preprocessed.

Here we offer suggestions about how to deal with each situation.

13.4.1 Large-Sized Data

While traditional data mining algorithms assume that the entire data set resides in memory, current data sets are often too large to satisfy this requirement. One possible way to deal with this problem is to process as much of the data as possible while repeatedly retrieving the remaining data from a secondary storage device. Clearly, this solution is not reasonable given the inefficiency of secondary storage data retrieval. Imagine attempting to train a backpropagation neural network with this approach! Another possibility is to employ a distributed environment where data can be divided among several processors. When this is not feasible, we must limit our list of plausible algorithm choices to those that exhibit the property of scalability.

An algorithm is said to be *scalable* if given a fixed amount of memory, its runtime increases linearly with the number of records in the data set. The simplest approach to scalability is sampling. With this technique, models are built and tested using a subset of the data that can be efficiently processed in memory. This method works well with supervised learning and unsupervised clustering when the data contain a minimal number of outliers. However, it would be difficult to put our trust in this method as a general approach for handling large-sized data.

Some traditional algorithms such as Naïve Bayes classifier are scalable. Cobweb and Classit are two scalable unsupervised clustering techniques as data is processed and discarded incrementally. One way to approach scalability is to make a slight modification to a traditional algorithm. For supervised learning with decision trees, there are several scalable techniques including the *rainforest algorithm*, the *sprint algorithm*, and the *Bootstrapped Optimistic Algorithm for Tree Construction*. All three take advantage in some way of one or more special properties of decision trees. For example, the rainforest algorithm (Gehrke et al., 1998) uses a technique whereby the memory requirement is dependent on the number of attributes, the number of distinct values per attribute, and the number of classes rather than on the number of instances. As the total number of instances grows, the efficiency of the rainforest method increases relative to a traditional approach.

The standard Apriori association rule algorithm is not scalable, as the algorithm must scan the data several times. Optimizations of the Apriori algorithm focus on reducing the number of database scans and/or the number of candidate itemsets counted in each scan. FP-Growth is a scalable algorithm for generating association rules. Unlike the Apriori algorithm, FP-Growth scans the database twice. The first scan divides the database into disjoint partitions, each able to fit into memory. Locally frequent itemsets are computed for each partition. The second scan counts the support of all locally frequent itemsets relative to the entire database. Both RapidMiner and Weka have implementations of the FP-Growth algorithm. Lastly, scalable support vector machine algorithms take advantage of the observation that several irrelevant instances can be identified and therefore removed early in the learning process.

13.4.2 Dealing with Imbalanced Data

The *class imbalance* or *rare class problem* occurs when there are several more instances of some classes than others. With imbalanced data, it is most often the rare class or classes that we wish to identify. For example, fraudulent credit card transactions occur much less frequently than valid credit card purchases, yet these are the transactions we must be able to detect.

The *rare cases problem* represents within-class imbalance. To illustrate the difference between rare classes and rare cases, consider that approximately 1 in every 100 individuals has celiac disease. Celiac disease is characterized as an autoimmune reaction to the consumption of a product containing wheat, rye, or barley. With 1 million randomly chosen individuals, we expect approximately 10,000 to have celiac disease. Relative to the overall population, individuals with celiac disease represent a rare class. The disease most often expresses itself as some form of intestinal distress or some type of skin problem. However,

in rare cases, the only symptom of a person suffering from celiac disease is anxiety. The class of individuals with celiac disease having this single symptom is a rare-case subgroup within the class of celiac patients. Rare cases can be treated as a subconcept of the parent class or considered as a class separate from the larger group. Rare classes and rare cases are often thought of as distinct problems.

We can also make a distinction between *absolute rarity*, defined as a lack of data, and *relative rarity*, which is measured in terms of instance rarity relative to other items. For example, an association rule miner may not find an interesting relationship with *relatively* rare items as minimum support levels are not met.

Our focus here is on methods for addressing the class-imbalance or rare-class problem. Here are three reasons why imbalanced data are difficult to mine.

- Classification accuracy is a poor but often-used metric for rarity.
- Rare classes have fewer examples, so it takes fewer “noisy” instances to adversely affect the learning process.
- Rules covering few training examples tend to be highly error sensitive, but removing the rules decreases the degree of global coverage of all instances.

13.4.2.1 Methods for Addressing Rarity

One common supervised approach for addressing rarity is *one-class* learning. As an example, a modification of the nearest neighbor algorithm might maintain a list of instances for a single class. When a new instance is presented, the amount of similarity between the new instance and the class instances is measured. If the new instance does not meet a predefined threshold score, it is flagged as a rare item. A second method is to replace classification accuracy with more appropriate metrics that focus on rarity. One such method is the topic of the next section.

13.4.2.2 Receiver Operating Characteristics Curves

Receiver operating characteristics (ROC) curves are a graphical approach for depicting the tradeoff between true-positive rate and false-positive rate. ROC curves were first used during World War II for analyzing radar images. Today, ROC graphs are an especially useful visualization and analysis tool in two-class domains having imbalanced or cost-sensitive data.

ROC curves are two-dimensional graphs where the true-positive rate is plotted on the *y*-axis and the false-positive rate is plotted on the *x*-axis. Recall that for class *C*, the true positive rate TP is computed by dividing the number of instances correctly classified with *C* by the total number of instances actually belonging to *C*. The false positive rate FP for class *C* is the total number of instances wrongly classified with *C* divided by the total number of instances not in *C*.

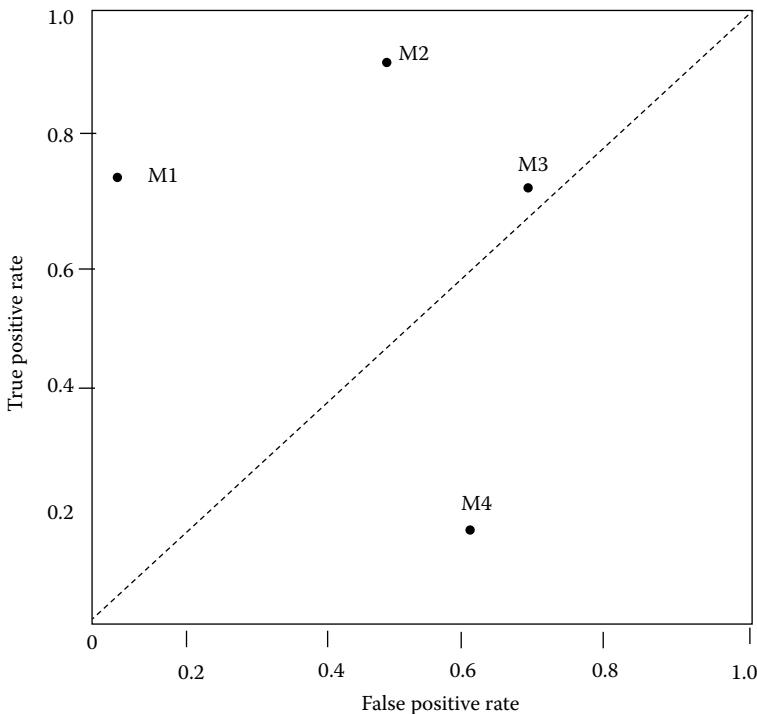


FIGURE 13.28 An ROC graph for four competing models.

Each point on an ROC curve represents one model induced by the classifier. With imbalanced data, the rare class is treated as the positive class as it is of most interest. ROC graphs are best illustrated by example.

The simplest case is given in Figure 13.28, which shows an ROC graph for four competing models. Each model is discrete as it outputs a single class label and is associated with exactly one point of the graph (TP rate, FP rate). Any point along the diagonal line represents random guessing. Let's assume each model is one alternative for detecting credit card fraud. The fraudulent transactions are the rare class and hence are represented by the true positive axis. The graph shows that M1 captures over 70% of the fraudulent transactions while accepting less than 10% of false-positive instances. M2 secures around 90% of the true positives but also accepts approximately 50% of the false positives. M3 performs a little better than random guessing, and M4 gives a false positive rate of more than 60%.

ROC curves are not able to tell us which model is the best choice, as we are unaware of the costs associated with incorrect classifications. However, the ROC graph can be a key component of the analytics process, especially in domains involving rarity. Let's take a look at a more complex example.

WEKA TUTORIAL

Creating and Analyzing ROC Graphs with Weka

For our example, we return to the spam data set and Weka's PART rule generator. Our goal is to build a model able to identify spam e-mail and at the same time minimize incorrect classification of valid e-mail messages.

Figure 13.29 shows the result of applying PART together with a 10-fold cross-validation to the data. We see a classification accuracy over 93% with TP = 0.915 (1659/1813) and FP = 0.049 (137/2788). This tells us that the point (0.915, 0.049) is the location on the ROC graph that represents the chosen model. To see this, let's create the ROC curve.

- Load the spam.arff data set.
- Click on *Classify* in the top left portion of your window.
- Click on More options, and mouse to *Output predictions*. Click on Choose and select *Plain Text*; click *OK*.
- Click on the Choose button, mouse to rules, and locate and click on PART.
- Apply PART to obtain the result displayed in Figure 13.29.
- Right click on *rules.PART* given in the result window.
- Mouse over *cost/benefit analysis* and choose 1 to have the ROC curve displayed relative to the spam class.
- In the upper left of your screen, change the x-axis from *sample size* to *false positive rate*.

The ROC curve with false-positive rate on the x-axis and true-positive rate on the y-axis is displayed in Figure 13.30. Notice the arrow pointing to the X in the lower left part of the ROC graph. The confusion matrix displayed directly below the graph shows the situation corresponding to the positioning of X. With this scenario, there are no true- or false-positive instances, 1813 false-negative instances, and 2788 true negatives. The

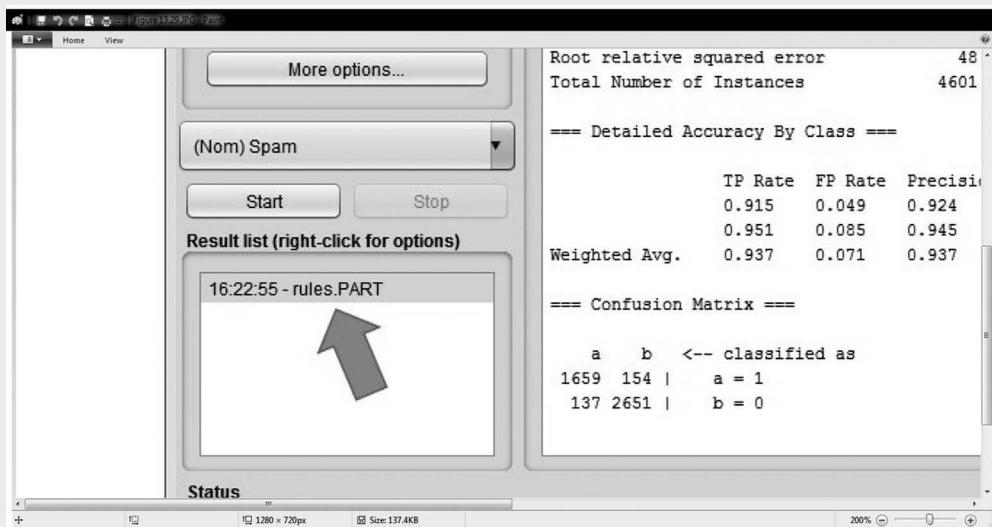


FIGURE 13.29 The PART algorithm applied to the spam data set.

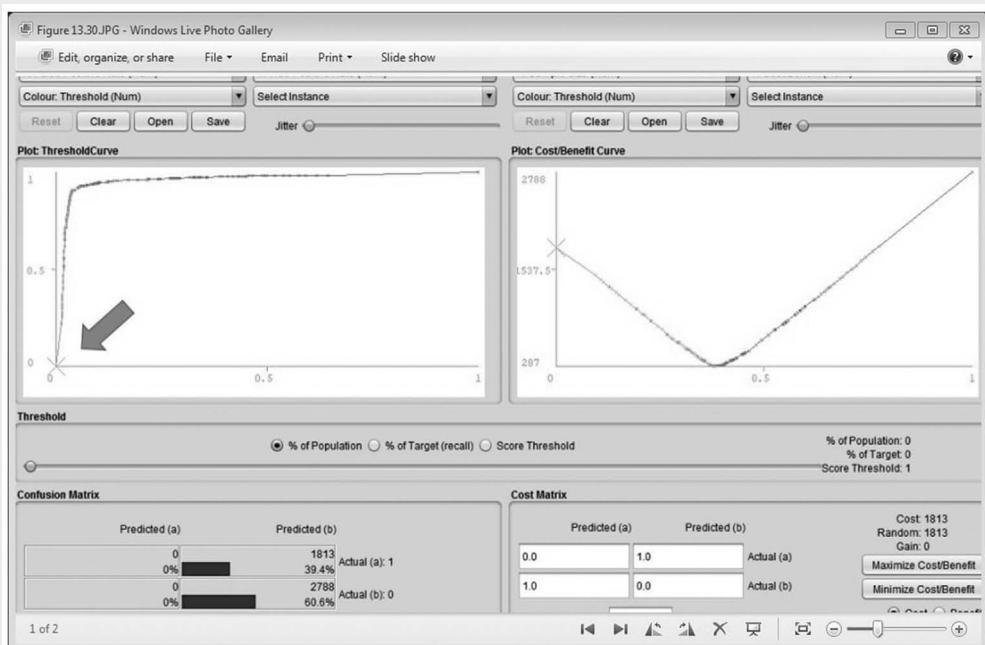


FIGURE 13.30 An ROC curve created by applying PART to the spam data set.

current location of X does not reflect the chosen model. To place X in the correct location, we do the following:

- Make sure the % population radio button is checked, and then move the slide bar corresponding to % population to the right. As you do so, watch the X move along the axis of the ROC curve.
- Position the slide bar to show as in Figure 13.31. This is the position corresponding to the (TP, FP) point seen with the model.

The *% of population* value tells us our model has identified 39.035% of the instances as spam (2.98% are misclassifications). The *% of target* is the true-positive rate and informs us that we have correctly identified 91.5058% of all spam instance—154 spam instances were classified as valid e-mails.

At this point, if you are not at least a bit confused, congratulations! Just in case, let's review. Recall that in order to create the ROC curve, the classifier must output a numeric confidence score for each classification. A list of confidence scores for the first few instances resulting from the model created with PART is shown in Figure 13.32. The ROC algorithm uses the numeric scores to sort all of the instances according to their likelihood of belonging to the *positive* (spam) class.

To see this, we saved the instances and their corresponding likelihood values in a text file. After deleting unnecessary information, the instance numbers together with their probability values were copied into an MS Excel spreadsheet. Next, we sorted the instances by the probability that they belong to the positive (spam) class. Figure 13.33 shows the first few sorted instances. Notice that the first four classifications are false positives.

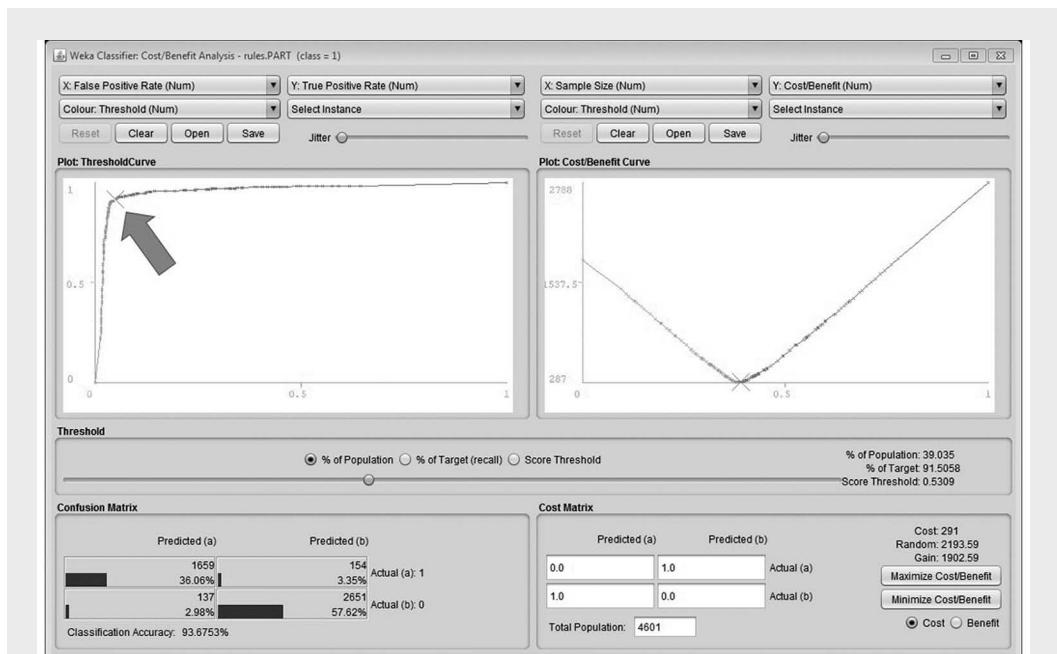


FIGURE 13.31 Locating the true- and false-positive rate position in the ROC curve.

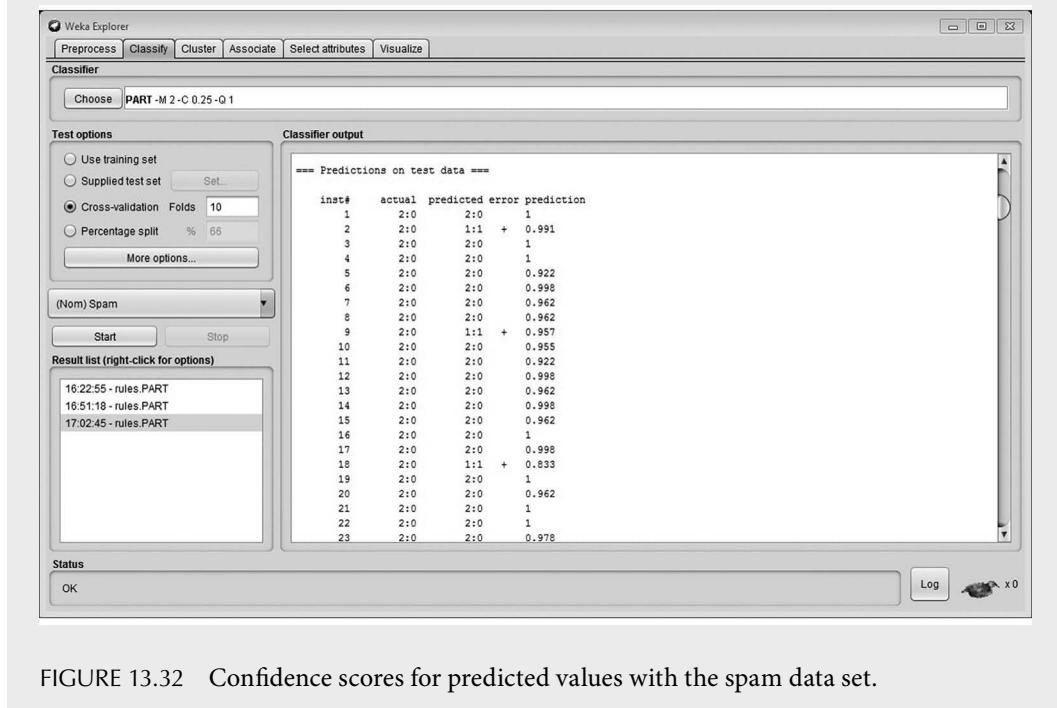
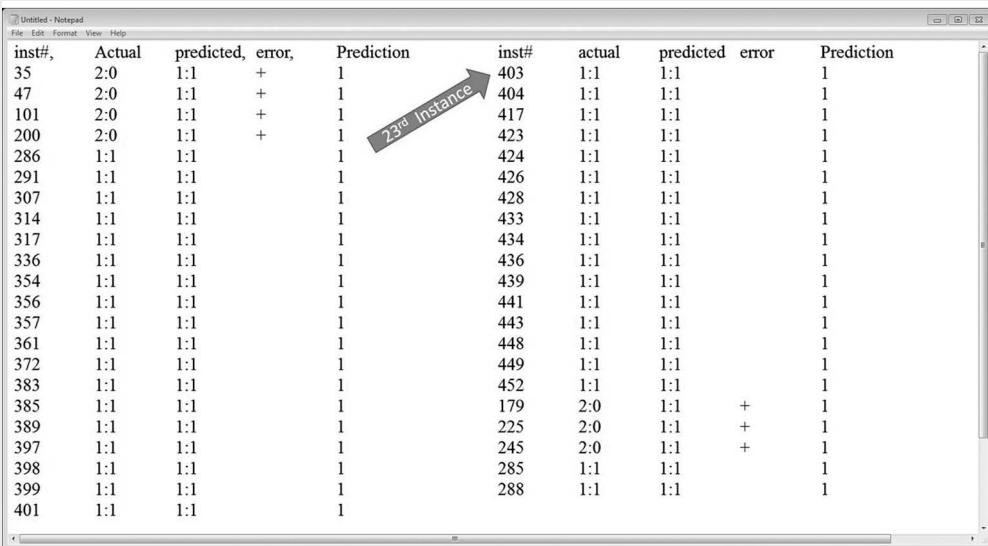


FIGURE 13.32 Confidence scores for predicted values with the spam data set.



inst#	Actual	predicted	error	Prediction	inst#	actual	predicted	error	Prediction
35	2:0	1:1	+	1	403	1:1	1:1		1
47	2:0	1:1	+	1	404	1:1	1:1		1
101	2:0	1:1	+	1	417	1:1	1:1		1
200	2:0	1:1	+	1	423	1:1	1:1		1
286	1:1	1:1		1	424	1:1	1:1		1
291	1:1	1:1		1	426	1:1	1:1		1
307	1:1	1:1		1	428	1:1	1:1		1
314	1:1	1:1		1	433	1:1	1:1		1
317	1:1	1:1		1	434	1:1	1:1		1
336	1:1	1:1		1	436	1:1	1:1		1
354	1:1	1:1		1	439	1:1	1:1		1
356	1:1	1:1		1	441	1:1	1:1		1
357	1:1	1:1		1	443	1:1	1:1		1
361	1:1	1:1		1	448	1:1	1:1		1
372	1:1	1:1		1	449	1:1	1:1		1
383	1:1	1:1		1	452	1:1	1:1		1
385	1:1	1:1		1	179	2:0	1:1	+	1
389	1:1	1:1		1	225	2:0	1:1	+	1
397	1:1	1:1		1	245	2:0	1:1	+	1
398	1:1	1:1		1	285	1:1	1:1		1
399	1:1	1:1		1	288	1:1	1:1		1
401	1:1	1:1		1					

FIGURE 13.33 Sorted confidence scores for the spam data set.

The 1796th instance (not shown) in the sort listed is the last instance where the confidence for the positive (spam) class is above 50%. This is the last instance to be classified by our model as spam (1659 correct, 137 incorrect).

The question remains as to exactly how the points are plotted to make the ROC curve. To see how the plotting is done, return to the ROC graph in Figure 13.31. Recall that we have 2788 valid e-mails and 1813 spam instances. As shown in Figure 13.33, the first four sorted instances are false-positive classifications. Starting with the first instance on the list, we plot the point $(1/2788, 0)$. That is, the first instance is an incorrect classification, so we move horizontally along the x-axis. Likewise, the second plotted point is $(2/2788, 0)$. The third and fourth points are plotted as $(3/2788, 0)$ and $(4/2788, 0)$. To see this horizontal movement, slide the jitter located above the ROC graph to the right.

With the fifth instance (inst# 286 in Figure 13.33), the graph moves in the vertical direction. The plot for the fifth point is $(4/2788, 1/1813)$. The graph continues to move directly north until instance number 179 (Figure 13.33), which begins another horizontal move given by three false-positive classifications. The FP seen with instance 179 plots the point $(5/2788, 35/1813)$. It is obvious that the ideal situation is a continual ascent up the y-axis until we run out of instances. Further, it is important to note that the ROC curve does not determine the classification of any instance. It simply reflects the class probability values associated with the instances as determined by the data mining algorithm. The benefit of the ROC plot is being able to visualize the price that must be paid in order to correctly identify the rare-class instances.

Lastly, as we move the % population bar in Figure 13.31 to the right of its current position, the false-positive rate begins to dramatically increase. Moving the bar to the furthest right position results in a model whose classification accuracy falls below 40% as all instances are classified as spam. In this case, both the TP rate and FP rate are 1.0. In this position, all spam e-mails have been correctly identified ($TP = 1.0$), and all valid e-mails are also identified as spam ($FP = 1.0$).

Here, our example focused on Weka's implementation of ROC graphs. RapidMiner's *Compare ROC* operator is straightforward and allows you to examine ROC curves created by one or several models. Selected end-of-chapter exercises ask you to build ROC graphs with the help of Weka and RapidMiner.

13.4.3 Methods for Streaming Data

A data stream is a sequence of data items where the learning model has but a single chance to read and process each instance. Once processed, the item is discarded. This is in sharp contrast to the traditional data mining methodology, where data instances can be read and manipulated several times. That is, a typical decision tree algorithm may explore the same training instance several times before it reaches a terminal node. Feed-forward neural networks trained with backpropagation learning pass the training data through the network thousands or possibly tens of thousands of times. The *K*-means and EM algorithms as well as agglomerative clustering all require several passes over the data. Unmodified, these techniques are useless in a streaming environment.

Applications of data stream mining include locating anomalies in computer network traffic data, assessing sentiment on a particular topic from newsfeed sources such as Twitter, processing video streams, and monitoring sensor networks, to name a few. Given these applications, a model for streaming data must be able to

- Continuously process single data items
- Process the data in a single pass
- Work efficiently in an environment of limited resources
- Be ready for application at point in any time
- Learn incrementally

One supervised technique able to meet these conditions without modification is the naïve Bayes classifier. Recall that the data structure used with naïve Bayes is a simple table. As new instances enter the data stream, the attribute-value table is updated. The algorithm can use the table for classification or predictive purposes at any point in time.

For unsupervised learning, the conceptual clustering algorithms discussed in Chapter 12 can be modified to fit the above criteria by setting a maximum tree depth and discarding instances after they have been processed. Meeting these criteria is not a problem, as the clusters formed by the first and second levels of the tree are of primary interest. One concern that remains with conceptual clustering is the issue of how to deal with skewed instance presentation.

Adaptations of standard data mining algorithms have also been applied to streaming data. Examples include association rule mining (Calders et al., 2007) and supervised classification (Domingos and Hulten, 2000).

The more difficult issue with data stream mining is data preprocessing. A generalized model might follow the workflow environment available with both RapidMiner and Weka, where individual operators each perform a specialized task prior to passing the modified instance to the next operator for further processing. This paradigm works well with textual conversation, where the first operator removes irrelevant sentences, converts lexical variations, and fixes spelling errors. The next operator tokenizes the data, and the third operator categorizes the tokens into one of several defined categories. Depending on the complexity of the domain, further operators address remaining text processing issues until a feature vector is created and given to the data mining algorithm.

13.5 ENSEMBLE TECHNIQUES FOR IMPROVING PERFORMANCE

When we make decisions, we usually rely on the help of others. In the same way, it makes intuitive sense to formulate classification decisions by combining the output of several data mining models. Unfortunately, attempts at employing a multiple-model approach to classification have been met with mixed results. This can be explained in part by the fact that most models misclassify the same instances. However, some success has been seen with a multiple-model approach where each model is built by applying the same data mining algorithm. In this section, we describe two techniques that make use of the same data mining algorithm to construct multiple-model (ensemble) classifiers.

Both approaches described here work best with unstable data mining algorithms. Recall that unstable algorithms show significant changes in model structure when slight changes are made to the training data. Many standard data mining algorithms, including decision trees, have been shown to display this characteristic.

13.5.1 Bagging

Bagging (bootstrap aggregation) is a supervised learning approach developed by Leo Breiman (1996) that allows several models to have an equal vote in the classification of new instances. The same mining tool is most often employed to create the multiple models to be used. The models vary in that different training instances are selected from the same pool to build each model. Here's how the approach works:

-
1. Randomly sample several training data sets of equal size from the domain of training instances. Instances are sampled with replacement. This allows each instance to be present in several training sets.
 2. Apply the data mining algorithm to build a classification model for each training data set. N sets of training data result in N classification models.
 3. To classify unknown instance I , present I to each classifier. Each classifier is allowed one vote. The instance is placed in the class achieving the most votes.
-

Bagging can also be applied to estimation problems where predictions are numeric values. To determine the value of an unknown instance, the estimated output is given as the average of all individual classifier estimations.

13.5.2 Boosting

Boosting, introduced by Freund and Schapire (1996), is more complex. *Boosting* is like bagging in that several models are used to vote in the classification of new instances. However, there are two main differences between bagging and boosting. Specifically,

- Each new model is built based upon the results of previous models. The latest model concentrates on correctly classifying those instances incorrectly classified by prior models. This is usually accomplished by assigning weights to individual instances. At the start of training, all instances are assigned the same weight. After the latest model is built, those instances classified correctly by the model have their weights decreased. Instances incorrectly classified have their weights increased.
- Once all models have been built, each model is assigned a weight based on its performance on the training data. Because of this, better-performing models are allowed to contribute more during the classification of unknown instances.

As you can see, boosting builds models that complement one another in their ability to classify the training data. An obvious problem exists if the data mining algorithm cannot classify weighted training data. The following is one way to overcome this problem:

-
1. Assign equal weights to all training data and build the first classification model.
 2. Increase the weights of incorrectly classified instances and have the weights of correctly classified instances decreased.
 3. Create a new training data set by sampling instances with replacement from the previous training data. Select instances having higher weight values more frequently than instances with lower weight values. In this way, even though the data mining algorithm cannot use instance weights *per se*, the most difficult-to-classify instances will receive increased exposure to subsequent models.
 4. Repeat the previous step for each new model.
-

As incorrectly classified instances are sampled more frequently, the weight values still play their part in the model building process. However, the data mining algorithm does not actually use the weight values during the model building process.

13.5.3 AdaBoost—An Example

For our example, we use the sonar data set located in the samples directory to illustrate RapidMiner's AdaBoost operator. Please note that this is not the sonar data set we employed in Chapters 9 and 10 to illustrate backpropagation learning. The data contains 13 numeric input attributes and a categorical output attribute representing two classes (Rock or Mine). Figure 13.34 shows the main process for our example. The topmost cross-validation (Figure 13.35) uses default settings to build and test a decision tree model for the data. The cross validation seen on the lower part of the display (Figure 13.36) implements

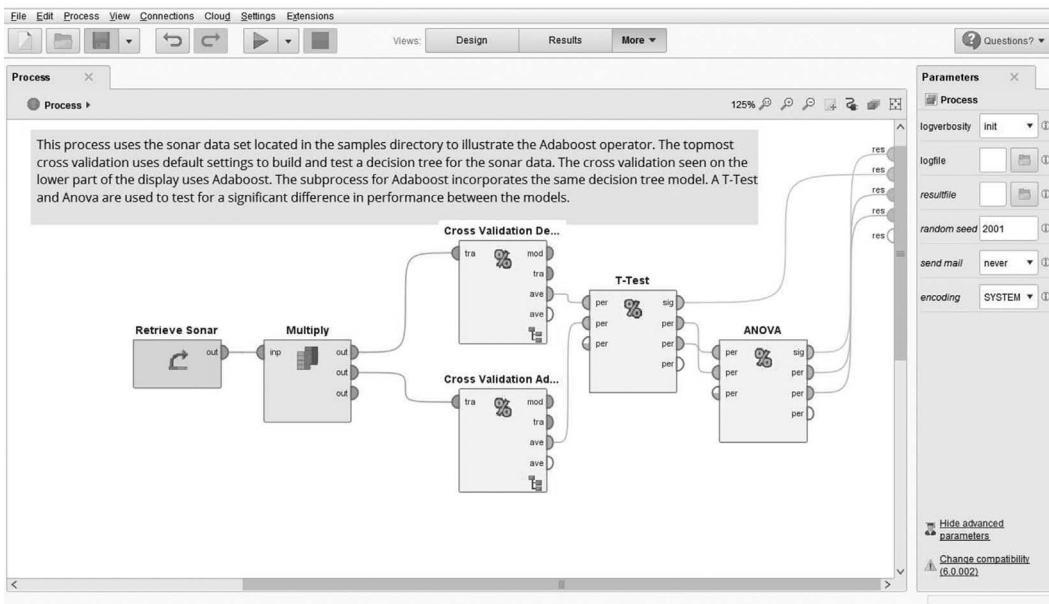


FIGURE 13.34 A main process for testing the AdaBoost operator.

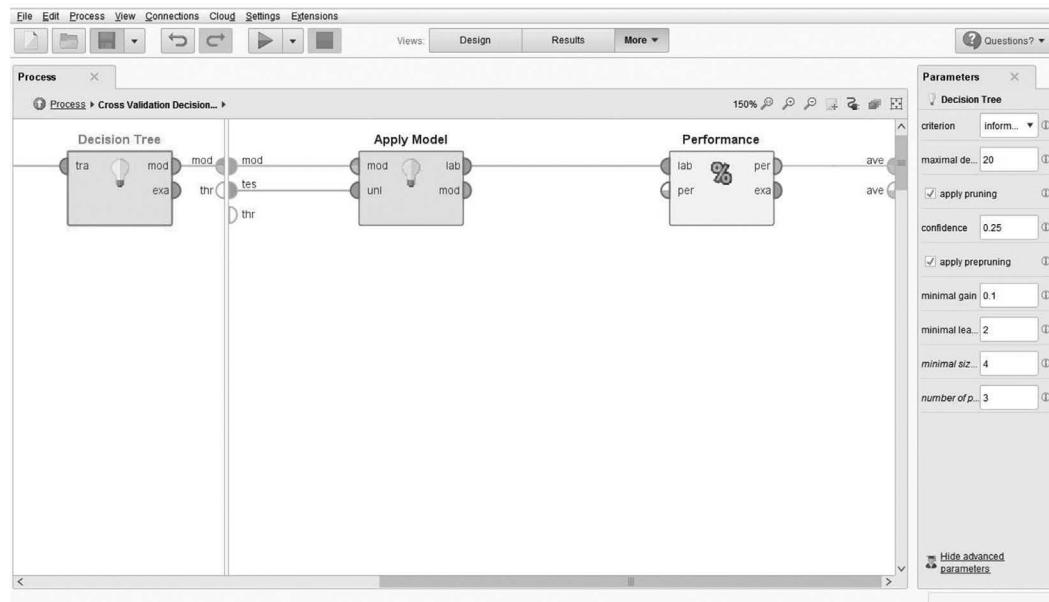


FIGURE 13.35 Subprocess using a decision tree without AdaBoost.

the AdaBoost operator. AdaBoost uses a subprocess to implement the data mining algorithm. For our illustration, the AdaBoost subprocess (not shown) simply contains the decision tree operator.

The results of the T-test and ANOVA used to test for a significant difference in performance between the models are given in Figures 13.37 and 13.38. Given the probability

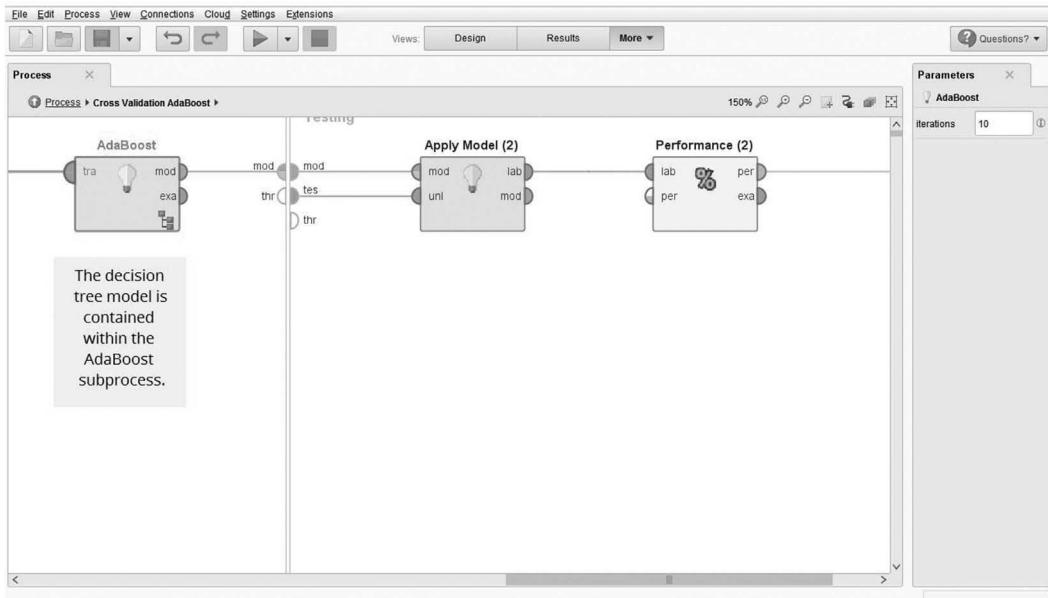


FIGURE 13.36 Subprocess using AdaBoost, which builds several decision trees.

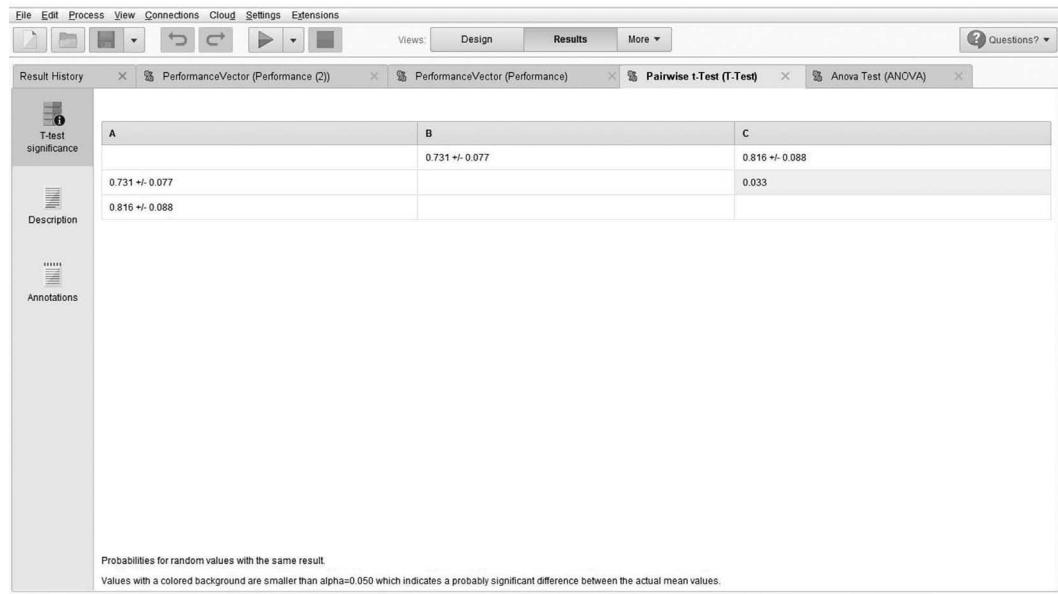


FIGURE 13.37 T-test results for testing AdaBoost.

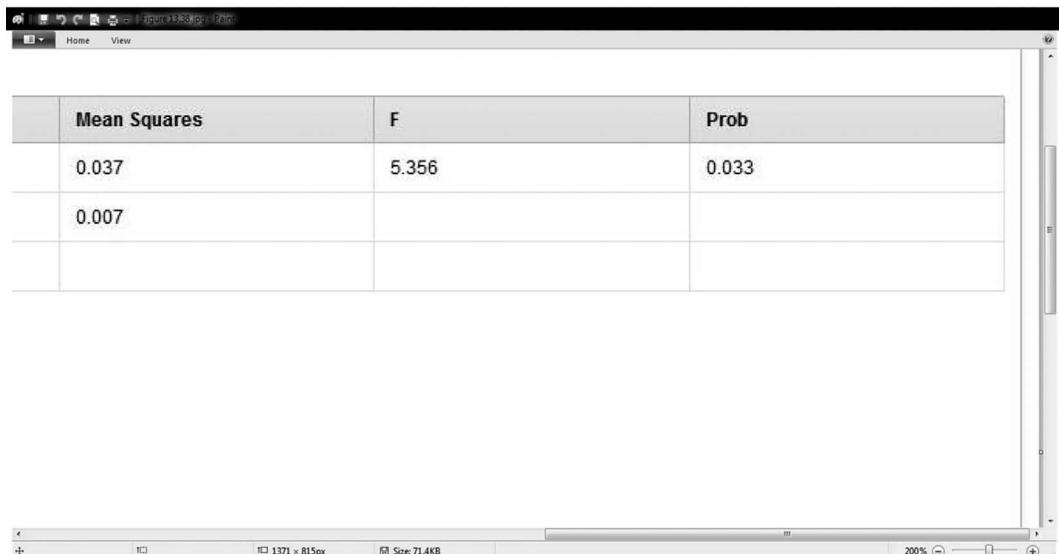


FIGURE 13.38 Results of the ANOVA with the AdaBoost operator.

threshold setting of 0.05, both the *T*-test and the ANOVA show the difference between the performance vectors (0.731 vs. 0.816) as significant.

Weka's implementations of the standard bagging (*Bagging*) and boosting (*AdaboostM1*) algorithms as well as their variations are found in the *meta* subfolder within the *classifiers* folder. Applying the algorithms is straightforward. A main consideration is our choice of the classification algorithm to be used in conjunction with the ensemble technique. The algorithm choice is made through the *GenericObjectEditor*.

13.6 CHAPTER SUMMARY

Time-series analysis concerns itself with data containing a time dimension and usually involves predicting numerical outcomes. Time-series analysis is useful because it allows us to use time as a variable for predicting the future. Caution must be exercised when predicting future outcome with training data containing several fields having predicted rather than actual results.

The success of a website ultimately depends on how it is perceived by the user community. Data mining can be applied to help determine perceptions users have about the websites they visit. However, Web-based data mining presents a new set of problems not seen with other data mining applications. The data available to us as a result of one or more Web-based user sessions are stored in Web server log files. The content of a typical log file houses information describing clickstream sequences followed by users as they investigate Web pages and follow page links. The greatest challenge of Web-based data mining is seen with taking server log files and extracting the information needed to create useful data mining models.

Textual data mining involves the extraction of patterns from free text. A basic algorithm for mining data in text format involves creating an attribute dictionary of frequently occurring words, filtering common words known to be of little value, and using the modified dictionary to classify new documents of unknown content. More complex textual mining problems require significant data preprocessing and dimensionality reduction. The expansion of the World Wide Web provides a bright future for textual data mining applications.

Scalable algorithms are useful when dealing with large-sized data. Given a fixed amount of memory, the runtime of scalable algorithms increases linearly with the number of records in the data set. Class imbalance occurs when there are several more instances of some classes than others. With imbalanced data, traditional model evaluation methods such as classification accuracy are of little use as it is most often the rare class or classes that we wish to identify. ROC curves are a good choice for evaluating two-class models built with imbalanced data. ROC curves offer a graphical approach for depicting the tradeoff between true-positive rate and false-positive rate. Data preprocessing is, at best, difficult with data stream learning as instances are read and processed but a single time. Some algorithms, including naïve Bayes and incremental hierarchical clustering schemes, are naturally able to build models from streaming data. Most traditional models require some modification in order to process streaming data.

Multiple-model methods such as bagging and boosting can sometimes improve model performance. Both approaches work best with unstable data mining algorithms.

13.7 KEY TERMS

- *Adaptive website*. A website having the ability to semiautomatically improve its internal structure as well as its methods of presentation by learning from visitor access patterns.
- *Bagging*. A supervised learning approach that allows several models to have an equal vote in the classification of new instances.
- *Boosting*. A supervised learning approach that allows several models to take part in the classification of new instances. Each model has an associated weight that is applied toward new-instance classification.
- *Class-imbalance* or *rare-class problem*. When there are several more instances of some classes than others.
- *Clickstream*. A sequence of links followed by a user as the user investigates Web pages and follows page links.
- *Cookie*. A data file placed on a user's computer that contains session information.
- *Extended common log file format*. A format frequently used to store Web server log file information.

- *Feature vector.* In text mining, a vector representing an instance of data made up of selected words (features) and frequency counts (feature values).
- *Index page.* A Web page having links to a set of pages detailing a particular topic.
- *Index synthesis problem.* Given a website and a visitor access log, the problem of creating new index pages containing collections of links to related but currently unlinked pages.
- *Link analysis.* A form of data mining that focuses on the relationships seen between the nodes of a network. With the Internet as the network structure, the task of link analysis is to make deductions about the nature or importance of websites based on the incoming and outgoing hyperlinks.
- *Pageview.* One or more page files that form a single display window in a Web browser.
- *Personalization.* The act of presenting Web users with information that interests them without requiring them to ask for it directly.
- *Receiver operating characteristics (ROC) curves.* A graphical approach for depicting the tradeoff between true-positive rate and false-positive rate.
- *Sequence miner.* A special data mining algorithm able to discover frequently accessed Web pages that occur in the same order.
- *Session.* A set of pageviews requested by a single user to a single Web server.
- *Session file.* A file that contains from a few to several thousand session instances.
- *Stemming.* Word stemming is a procedure designed to convert words to their root form.
- *Time-series problem.* A prediction problem with one or more time-dependent attributes.
- *Tokenization.* In textual mining, a procedure that breaks sentences into smaller units called tokens.

EXERCISES

Review Questions

1. Visit one or more websites that contain products or information that interests you. Answer the following questions for each site you visit.
 - a. How easy was it to find the information you were looking for?
 - b. How much information was provided to you that was of little or no interest?
 - c. How many advertising ads were you exposed to? Did the ads have anything to do with what you were looking for?

2. List the similarities and differences between bagging and boosting. State a scenario where a multiple model built with bagging is likely to outperform a model built by applying boosting. State a situation where boosting is likely to outperform bagging.
3. Are bagging and boosting appropriate for time-series problems? Why or why not?
4. Outline an index page structure for a website designed to answer user questions about automotive repair. State several typical paths that a novice user may follow in an attempt to find answers to his/her automotive repair questions.
5. Given that class C is the positive class, *Sensitivity* is defined as the total number of instances correctly classified with C divided by the total number of instances actually belonging to C . Also, *Specificity* is computed as the total number of instances correctly classified as not belonging to C divided by the total number of instances not in C . How are these terms related to TP and FP?

Data Mining Questions

1. Determine the confidence limits for the model whose performance vector is shown in Figure 13.9. Based on the confidence range, how much faith do you place in the model to correctly predict the next-day movement of XIV?
2. In Chapter 8, we introduced Kohonen self-organizing maps (SOMs) for unsupervised clustering. SOMs are often used for dimension reduction in text mining applications. Replace the SVD operator in the online shoe example of Section 13.3 with RapidMiner's SOM operator (*number of dimensions* = 2). Execute your process. Display the reduced dimension table and the resultant rules.
3. This question is about time-series analysis and has multiple parts:
 - a. Go to <http://finance.yahoo.com/> to obtain a short description of the ETF funds with symbols SPY, QQQ, and DIA.
 - b. Use the procedure specified in Section 13.1 to create a target dataset for one of the three funds. The dates in your target data set should match those used with XIV.
 - c. Repeat one or more of the experiments given in Section 13.1 using your data set.
 - d. Summarize your results.
4. Repeat one or more of the experiments given in Section 13.1 but use, in turn, a decision tree model, logistic regression, naïve Bayes classifier, and a support vector model. Make a table of your results. Are the differences in model performance significant?
5. Use linear regression and the appropriate *13TimeSeriesNumeric* file to perform a time-series analysis with XIV. Compare your results with the numeric output (Weka) example in Section 13.1.

6. Read about RapidMiner's *Compare ROC*'s operator. Create a process that uses the operator together with a data set of your choice and 10-fold cross-validation to compare the *Rule Induction* and *Naïve Bayes* operators. Display and comment on the output of the *Compare ROC*'s operator.
7. Try several experiments with the nearest neighbor classifier and the spam data set using both bagging and boosting. Are you able to create a significantly improved model?
8. Here is an exercise for a real sports fan. Develop a time-series model to determine how well a professional football running back will perform in any given week. The problem can be treated as a time-series application by having attributes that relate a given week's performance in terms of the performance of previous weeks. Select an appropriate time lag and choose those attributes you believe to be most predictive of future performance. You may wish to consider attributes such as total yards rushing year to date, average yards rushing per game, and average number of carries per game. You may also want to consider performance measures for the opposing team.
9. Perform a time-series analysis for your favorite stock or exchange traded fund using one of the time-series procedures illustrated in Section 13.1. Use and modify your model on a daily basis for 2 weeks. Write a short report detailing the performance of your model. Your report should contain at least one graph or chart.
10. Modify the model for the online shoe text mining example in Section 13.3 in one or more of the following ways:
 - a. Create and add several additional instances to all three classes.
 - b. Replace the rule induction operator with naïve Bayes and add a twofold cross-validation to test the accuracy of your model.
 - c. Compare the SVD models created by using alternative values for the vector creation parameter within the *Process Documents From Files* operator.
11. Apply RapidMiner's *Web Crawler* operator with default settings to your personal Web page. How many URLs does the operator find?



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

The Data Warehouse

CHAPTER OBJECTIVES

- *Understand why transactional databases do not support the redundancy required for data analytics*
- *Know that a data warehouse is implemented using a multidimensional or relational architecture*
- *Understand how a star schema is used for data warehouse modeling*
- *Know how online analytical processing can be applied to analyze the multidimensional data stored in a warehouse*
- *Understand how Microsoft Excel pivot tables can be used to model multidimensional data*

Data analytics has and will continue to take place in environments not supporting a data warehouse. This is currently seen with big data where relational database and warehouse systems have given way to more informal approaches that process streaming and often unstructured data. However, as volumes of data continue to be collected for the purpose of analysis, the need for organized, efficient data storage and retrieval architectures is obvious.

In this chapter, we provide an overview of the major concepts and issues that surround data warehouse design. Although several warehousing architectures exist, we focus on the most common design forms. We also introduce online analytical processing—a powerful multidimensional analytics tool. In Section 14.1, you will learn about the inherent problems in analyzing data stored in a transactional database. In Section 14.2, we define data warehousing and detail a popular model for data warehouse design. Section 14.3 shows how online analytical processing can be used to test human-generated hypotheses about data stored in a warehouse. We conclude this chapter with an introduction to MS Excel pivot tables.

14.1 OPERATIONAL DATABASES

We have seen that one possibility for gathering data for analysis is to extract data directly from one or more operational databases. Operational, or transactional, databases are designed to process individual transactions quickly and efficiently. This type of transaction-based interaction is known as *online transactional processing*, or simply *OLTP*.

In contrast, decision support systems are subject oriented. They incorporate facilities for reporting, analyzing, and mining data about a particular topic such as credit card promotions, automotive engine repairs, or heart disease. Because of the difference in intent between an operational and a decision support setting, models built for one methodology are not appropriate for the other. To demonstrate, let's take a quick look at the structures used for modeling an operational database environment.

14.1.1 Data Modeling and Normalization

The first step toward building a transactional database is data modeling. A *data model* documents the structure of the data to be placed into a system independent of how the data will be used. A common notation for data modeling is the *entity relationship diagram (ERD)*. An ERD shows the structure of the data in terms of entities and relationships between entities. An *entity* is like a concept in that it represents a class of persons, places, or things. An entity may contain one or several attributes. A key represented by a combination of one or several attributes uniquely identifies each instance of an entity.

Relationships between entities are either *one-to-one*, *one-to-many*, or *many-to-many*. In a monogamous society, the *husband–wife* relationship is one-to-one. The *father–child* relationship is one-to-many because a father may have one or several children, but a child has one and only one biological father. Finally, *student–teacher* is a many-to-many relationship because a teacher can have several students and a student is likely to have more than one teacher.

Figure 14.1 depicts a hypothetical ERD for two entities *vehicle-type* and *customer*. The key for each entity is marked with a dot. That is, *type ID* uniquely identifies a vehicle of a specific make and year, and *customer ID* is a unique identifier for the customer entity. The crows feet on *customer* indicate that the relationship between the two entities is one-to-many. That is, each customer owns exactly one vehicle, but several customers can own a vehicle having the same make and year.

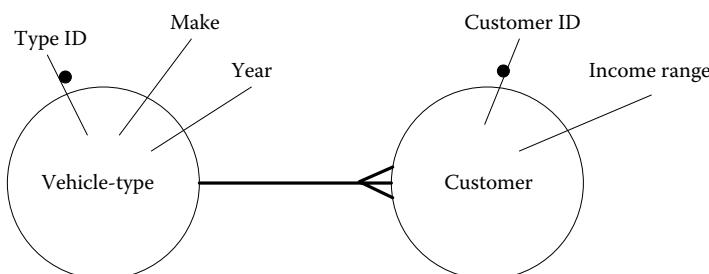


FIGURE 14.1 A simple entity-relationship diagram.

If crows feet also appear on the link to *vehicle-type*, the relationship is many-to-many, thus allowing one customer to own more than one vehicle. Many-to-many relationships are typically mapped as two one-to-many relationships by creating an *intersection entity* whose existence depends solely on the many-to-many relationship. Figure 6.2 in Chapter 6 depicts an ERD using the notation adopted by Microsoft Access. Attributes representing keys are highlighted, and crows feet are replaced by the symbol for infinity (∞). Figure 6.2 shows that *promotion-customer* is an intersection entity formed by a many-to-many relationship between *promotion* and *customer*. The relationship is many-to-many because several customers can take advantage of a single promotion and an individual customer may choose to purchase several promotional offerings.

Once an ERD has been completed, the model is analyzed for possible improvements. A main part of the analysis involves *normalization*. Normalization is a multistep process designed to remove any redundancies that may be present in the current model. Several normal forms have been defined. *First normal form (1NF)* requires all attributes to have a single value. An entity is in *second normal form (2NF)* if it is in first normal form and all nonkey attributes are dependent on the full primary key. The check for 2NF is relevant only when the key field is made up of more than one attribute. An entity is in *third normal form (3NF)* if it is in 2NF and the following condition is met:

Every nonprimary key attribute is dependent on the primary key, the whole primary key, and nothing but the primary key.

That is, 3NF requires that the values for all nonkey attributes are dependent solely on the complete primary key. Although higher normal forms have been defined, most data models are considered acceptable if all entities are in 3NF.

14.1.2 The Relational Model

ERDs map naturally to the relational model. Table 14.1 shows the relational table for *vehicle-type*, and Table 14.1 displays the relational table for *customer*. Notice that the one-to-many relationship between *vehicle-type* and *customer* is supported by placing the key field for *vehicle-type* as an attribute in *customer*. When an intersection entity is mapped to a relational table, it contains the key field from the two entities making up the original many-to-many relationship. This is seen in Figure 6.2, where the intersection entity *promotion-customer* holds both *customer-ID* and *promotion-ID*.

Relational databases are well suited for transactional processing because they can efficiently collect and manage data without loss of information. However, data mining and

TABLE 14.1 Relational Table for *Vehicle-Type*

Type ID	Make	Year
4371	Chevrolet	2005
6940	Cadillac	2010
4595	Chevrolet	2011
2390	Cadillac	2007

TABLE 14.2 Relational Table for *Customer*

Customer ID	Income Range (\$)	Type ID
0001	70–90K	2390
0002	30–50K	4371
0003	70–90K	6940
0004	30–50K	4595
0005	70–90K	2390

TABLE 14.3 Join of Tables 14.1 and 14.2

Customer ID	Income Range (\$)	Type ID	Make	Year
0001	70–90K	2390	Cadillac	2007
0002	30–50K	4371	Chevrolet	2005
0003	70–90K	6940	Cadillac	2010
0004	30–50K	4595	Chevrolet	2011
0005	70–90K	2390	Cadillac	2007

other forms of decision support are interested in analyzing rather than processing data. As the purpose of data analysis is to examine and uncover redundancies in data, the uniqueness constraints placed on the relational model by the normalization process are not desirable for a decision support environment. An example will make this point clear.

Let's consider the relations shown in Tables 14.1 and 14.2. The relational *join* operator is used to combine two relational tables by matching the values of an attribute common to both tables. We can join Tables 14.1 and 14.2 on the common attribute *type ID*. Table 14.3 displays the table resulting from the join operation. We can make at least two observations. First, Table 14.3 is denormalized because any combination of one or more attributes chosen as the key field will violate the functional dependency requirement. For example, creating a combined key field with attributes *customer ID* and *type ID* violates 2NF because *make*, *income range*, and *year* are each dependent on part of, but not the entire, key field.

A second observation is the relationship between an individual's salary and the type of car he or she owns. Although this relationship is not of interest in a transactional environment, it is of primary importance to decision support. Relationships showing this type of redundancy can only be observed by denormalizing the data. The long and short of this is that in a transactional environment supporting a real application, a significant amount of denormalization in the form of combining entities is necessary to prepare data for decision support. This in turn leads to new questions about which entities to combine, as well as how and where to store and maintain the combined entities. A better choice is to have one mechanism for storing, maintaining, and processing transactional data and a second to house data for decision support.

14.2 DATA WAREHOUSE DESIGN

When transactional data is no longer of value to the operational environment, it is removed from the database. If a business is without a decision support facility, the data are archived

and eventually destroyed. However, if there is a decision support environment, the data are transported to some type of interactive medium commonly referred to as a data warehouse.

In Chapter 1, we defined the data warehouse as a historical database designed for decision support. A more precise definition is given by W. H. Inmon (1996). Specifically,

A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision making process.

As the definition implies, significant variations in structure and intent exist between an OLTP database and a data warehouse. A few of the major differences are as follows:

- The data in a warehouse is subject oriented and based on one or more central themes. OLTP databases are process oriented and organized so as to maximize data update and retrieval.
- Warehouse data is denormalized and integrated; the data in an OLTP database is normalized and separated.
- An OLTP system supports data processing, collection, and management. A data warehouse stores data to be reported on, analyzed, and tested.
- OLTP deals with data necessary for the efficient day-to-day operation of the business or organization. Data records in a transactional database are subject to multiple access and constant update. In contrast, the data in a warehouse exist in part because the data are no longer of use to the OLTP environment. The majority of data in a warehouse are historical, time-stamped, and not subject to change (read-only).
- *Granularity* is a term used to describe the level of detail of stored information. Operational data represents the lowest level of granularity as each data item contains information about an individual transaction. The level of granularity for data stored in a warehouse is a design issue dependent on the desires of the user as well as on the amount of data being collected.

A data warehouse can also be viewed as a process for gathering, storing, managing, and analyzing data (Gardner, 1998). Figure 14.2 displays the key components of the warehousing process. Let's take a closer look at how data moves from the external environment into the data warehouse.

14.2.1 Entering Data into the Warehouse

Figure 14.2 shows data entering the data warehouse from three main sources. External data represent items such as economic indicators, weather information, and the like that are not specific to the internal organization. An *independent data mart* is a data store that is similar to a warehouse but limits its focus to a single subject. An independent data mart is structured using operational data as well as external data sources. The data stored in a data mart can be loaded into the central data warehouse for use by other facets of the

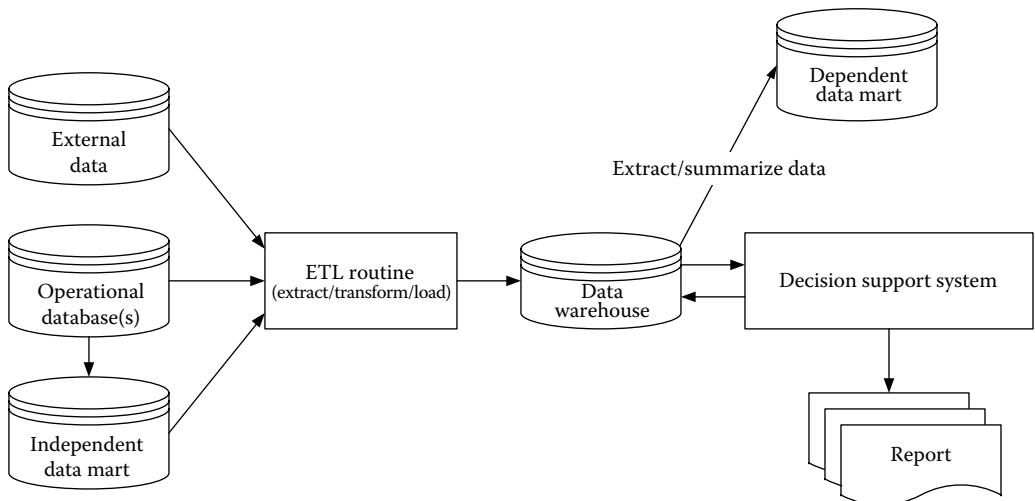


FIGURE 14.2 A data warehouse process model.

organization. Regardless of the external data source, the process of moving data into the warehouse will likely involve some low-level procedural programming.

Figure 14.2 illustrates that prior to entering the warehouse, data are processed by an *ETL* (extract, transform, and load) routine. The primary responsibilities of the *ETL* process include extracting data from one or more of the input sources shown in Figure 14.2, cleaning and transforming the extracted data as necessary, and loading the data into the warehouse. Data transformations are often used to resolve granularity issues, correct data inconsistencies between multiple operational databases, and time-stamp individual data records. Once transformed and cleaned, the data enter the warehouse, where they are stored in a relational or multidimensional format.

As a rule, once data enter the warehouse, they are not subject to change. An obvious exception to this rule is when errors are detected in the data. But what about special situations such as when an individual changes his/her address or marital status? Let's consider a data warehouse storing customer transactions for credit card purchases. Suppose a customer changes his/her marital status from single to married. A first thought is to simply change the marital status of all warehouse records referencing the customer. The problem with this solution is that any analysis that makes use of marital status information is processing corrupted data, as purchases made by the individual when he/she was single are considered as purchases made by a married customer. So how is such a change of information recorded? Several solutions have been proposed. One solution recommends creating record fields to hold previous as well as current values for each attribute. A second method suggests creating a new record each time an attribute value changes in an existing record. The existing record and the new record are then linked with a key field. A general solution for reflecting changing record status within the warehouse remains an unsolved problem.

The warehouse also stores another type of data known as *metadata*. Metadata are technically defined as data about data. The purpose of metadata is to allow for a better understanding of the nature of the data contained in the warehouse. Two general types of

metadata have been defined: structural and operational. Structural metadata emphasizes data descriptions, data types, representation rules, and relationships between data items. Operational metadata is primarily responsible for describing the quality and usage of data. A major difference between structural and operational data is that the latter are in a constant state of change, whereas the former are static.

14.2.2 Structuring the Data Warehouse: The Star Schema

Broadly speaking, two general techniques have been adopted for implementing a data warehouse. One method is to structure the warehouse model as a multidimensional array. In this case, the data are stored in a form similar to the format used for presentation to the user. In Section 14.3, you will learn more about the advantages and disadvantages of the multidimensional database model. A more common approach stores the warehouse data using the relational model and invokes a relational database engine to present the data to the user in a multidimensional format. Here we discuss a popular relational modeling technique known as the *star schema*.

Figure 14.3 outlines a star schema created from the Acme credit card database of Figure 6.2. The theme of the star schema is credit card purchases. At the center of the star schema

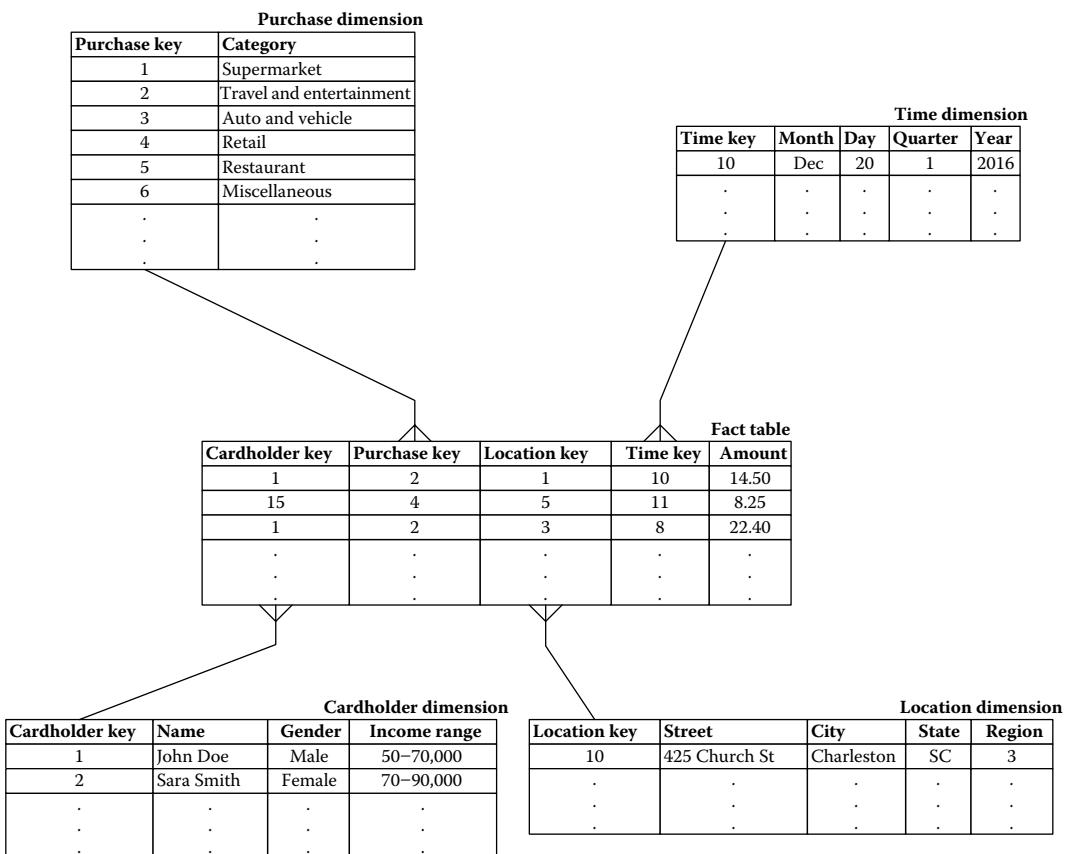


FIGURE 14.3 A star schema for credit card purchases.

is the *fact table*. The fact table defines the dimensions of the multidimensional space. The fact table seen in Figure 14.3 has four dimensions—*cardholder*, *purchase*, *location*, and *time*. Each record of the fact table contains two types of information—dimension keys and facts. The dimension keys are system-generated values that, taken together, uniquely define each record of the fact table. The dimension keys determine the coordinates of the multidimensional structure represented by the star schema.

Each dimension of the fact table may have one or more associated *dimension tables*. The dimension tables make up the points of a star whose center is the fact table—hence the name *star schema*. The dimension tables contain data specific to each dimension. The relationship between a dimension table and the fact table is one-to-many. Therefore, the dimension tables will be significantly smaller than the central fact table. Although the fact table is in 3NF, the dimension tables are not normalized. Instead, the choice of attributes comprising a dimension table is largely determined by the nature of the analytical questions to be answered by the star schema. Finally, the dimensions of the star schema are often referred to as *slowly changing dimensions*. This is because it is the dimension tables whose information is subject to the types of infrequent changes discussed in the previous section. Let's examine the dimension tables associated with the star schema for the Acme credit card database.

The star schema of Figure 14.3 shows four dimension tables. The *cardholder* dimension table is linked to the *customer* dimension and contains the name, gender, and income range of each customer stored in the database. The dimension table connected to the *purchase* dimension includes the possible categories for an individual credit card purchase. The dimension table associated with the *location* dimension stores information about the location of a purchased item. The *location* dimension holds a value for state as well as region. The model assumes four regions, with each state being part of one and only one region. Finally, the dimension table for *time* allows us to view time in terms of days, months, quarters, or years.

In addition to dimension key fields, the fact table may associate one or more facts with each record. The fact table displayed in Figure 14.3 has one fact representing the purchase amount for each credit card transaction. We can now read the first entry in the fact table. The credit card purchase was made by John Doe, a male cardholder who makes between \$50,000 and \$70,000 a year. The purchase was a travel and entertainment item and was made December 20, 2016. The amount of the purchase was \$14.50.

14.2.2.1 The Multidimensionality of the Star Schema

Let's take a closer look at the multidimensional nature of the star schema. The fact table in Figure 14.3 defines a four-dimensional space. Figure 14.4 maps three of the four dimensions—*purchase*, *location*, and *time*—to a three-dimensional coordinate system. A three-dimensional structure such as the one in Figure 14.4 exists for each cardholder (the fourth dimension) within the star schema. The figure indicates the general case where the *i*th cardholder is given as C_i .

Every record contained in the fact table is represented by exactly one point in this four-dimensional space. The point for the first fact table record ($C_1 = 1$) with coordinates

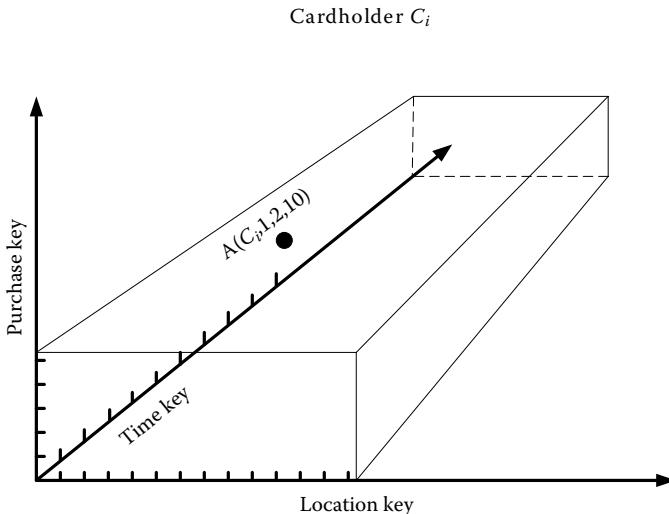


FIGURE 14.4 Dimensions of the fact table shown in Figure 14.3.

(C_i,1,2,10) is marked in Figure 14.4 by the letter A. The cell labeled A contains the value \$14.50. You will notice that the dimension tables of the star schema are not shown in Figure 14.4. Although the dimension tables are not explicitly shown in the figure, the contents of any dimension table are easily obtained by performing a join operation with the central fact table.

The granularity of the fact table is at the transaction level. That is, all credit card purchases made by each cardholder are individually recorded in the fact table. To increase the granularity, we could record a summary of all purchases of a single type by each customer for a single day. By doing this, the fact table in Figure 14.3 will contain one entry for the two purchases made by the customer with *card holder key* = 1, *purchase key* = 2, and *time key* = 10. The purchase amount for the combined entry will be recorded as \$36.90. An even higher level of granularity is achieved if we record total purchases by category for each customer on a monthly basis. If we choose this level of granularity, the attribute *day* would be removed from the *time* dimension table. To determine the level of granularity, we need a clear understanding about the level of detail required by the individuals who will be using the system. For any problem, a higher granularity will benefit system performance, as an increase in granularity allows for a decrease in the total number of fact table records.

14.2.2.2 Additional Relational Schemas

The *snowflake schema* is a variation of the star schema where some of the dimension tables linked directly to the fact table are further subdivided. This permits the dimension tables to be normalized, which in turn means less total storage. The advantage of this is increased storage efficiency because there is less redundancy. In addition, because the relational tables are smaller, join operations will show an improvement in performance. However, an increase in the total number of tables extends the complexity of the data queries required to extract the same information as is stored in tables that have not been normalized.

A second variation of the star schema that accounts for models requiring more than one central fact table is the *constellation schema*. A data warehouse supporting several subjects benefits most from the constellation schema. To illustrate, let's suppose we wish to include promotional information with the credit card purchase database defined in Figure 14.3. One possibility is to design a constellation schema having the current fact table for credit card purchases and a second fact table for information about promotional purchases. Figure 14.5 displays one configuration for a constellation schema that supports both credit card purchases and credit card promotions.

Figure 14.5 shows each record of the promotion fact table storing a cardholder key together with a *promotion key*, a *time key*, and a *response*. Notice that data redundancy is minimized by having the cardholder key dimension for both fact tables share the cardholder dimension table. The promotion dimension table describes each promotion in detail and links with the promotion key dimension. The time dimension key found within the promotion fact table links with the shared time dimension table. The time link signifies the start date for each promotional offering. As we have assumed a promotion duration of 1 month, a second time dimension indicating a promotion termination date is not necessary. However, if individual promotional offerings were to vary in duration, a second time dimension indicating promotion termination dates would be necessary. Finally, every record of the promotion fact table stores a single fact—denoted in Figure 14.5 as *response*—that indicates whether a cardholder did or did not take advantage of a specific promotion. As you can see, the credit card promotion database first described in Chapter 2 can be made readily available by querying the tables defined by the constellation schema of Figure 14.5.

14.2.3 Decision Support: Analyzing the Warehouse Data

The primary function of the data warehouse is to house data for decision support. Figure 14.2 shows that data are copied from the data warehouse for analysis by the decision support system. We also see data entering the data warehouse from the decision environment. Any data entering the warehouse from the decision support system will be in the form of metadata created from information gained through one or more decision support processes.

Three categories of decision support can be defined. Specifically,

1. *Reporting data.* Reporting is considered the lowest level of decision support. However, a reporting facility capable of generating informative reports for a variety of clientele is of utmost importance for the successful operation of any business.
2. *Online analytical processing.* Data analysis is accomplished with some form of multi-dimensional data analysis tool. Multidimensional data analysis is the topic of the next section.
3. *Data analytics.* Data analytics typically takes place through data mining. However, other techniques such as repeated query can sometimes uncover interesting patterns in data.

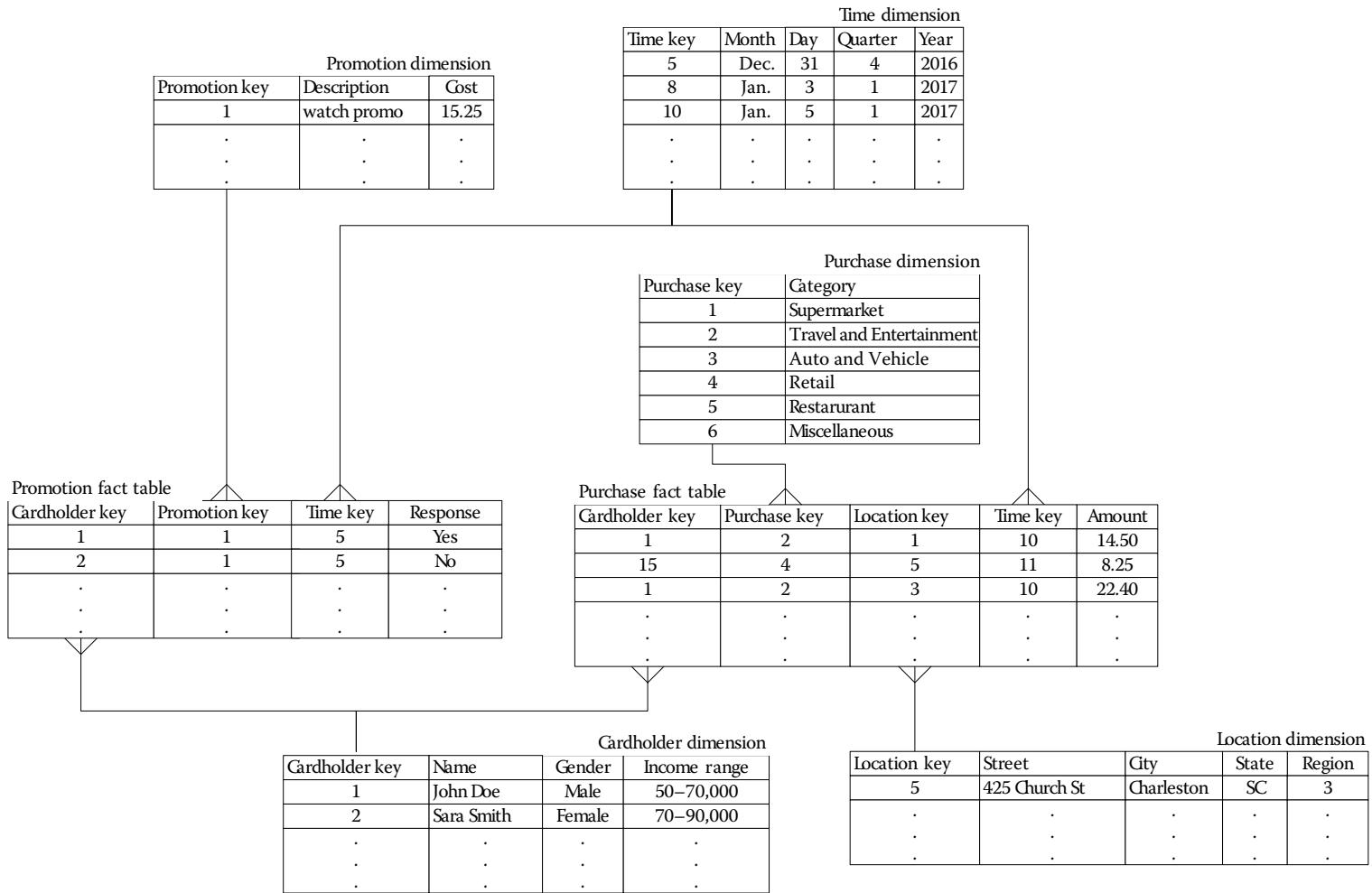


FIGURE 14.5 A constellation schema for credit card purchases and promotions.

Besides storing data for decision support, the warehouse is a data store for creating smaller departmental warehouses known as *dependent data marts*. A dependent data mart is typically about a single subject and designed for a specific purpose. In addition, the data mart is likely to contain summary information showing a higher level of granularity than that of the data warehouse.

14.3 ONLINE ANALYTICAL PROCESSING

Online analytical processing (OLAP) is a query-based methodology that supports data analysis in a multidimensional environment. OLAP is a valuable tool for verifying or refuting human-generated hypotheses and for performing manual data mining. A complete treatment of OLAP would take several chapters at best. Our discussion here offers a basic understanding of the types of problems that can be solved with a strategy based on OLAP technology.

An OLAP engine logically structures multidimensional data in the form of a cube like the one shown in Figure 14.6. The cube displays three dimensions—*purchase category*, *time in months*, and *region*. You will notice that the dimensions are a subset of attributes taken from the star schema shown in Figure 14.3. A three-dimensional cube such as the one in Figure 14.6 is easy to visualize. However, it is often difficult to picture a data cube having more than three dimensions. To help paint a clear image of a four-dimensional cube, we can visualize four dimensions by thinking of n three-dimensional cubes, where n is the total number of possible attribute values for the fourth dimension. To conceptualize higher-level dimensions, the process is repeated as necessary.

As an OLAP cube is designed for a specific purpose, it is not unusual to have several cubes structured from the data in a single warehouse. The design of a data cube includes decisions about which attributes to include in the cube as well as the granularity of each attribute. A well-designed cube is configured so as to avoid situations where data cells do not contain useful information. For example, a cube with two time dimensions, one for month and a second for fiscal quarter (Q1, Q2, Q3, Q4), is a poor choice because cell combinations such as (January, Q4) or (December, Q1) will always be empty.

One important feature of an OLAP technology is its underlying data store, which can be relational or multidimensional. If the data warehouse is relational, the internal representation of the OLAP cube is likely to be a star schema. With a multidimensional data store, the data are logically stored in arrays. Each storage method has its advantages. A relational data store allows the user to view the data at the detail level defined within the star schema. A second advantage of a relational data store is that the contents of the cells within a cube can be easily modified, even while the cube is in use. The primary advantage of a multidimensional data store is performance as measured by query speed. Whether the data store is relational or multidimensional, the user views the data as a multidimensional structure.

Another important feature of an OLAP system is the design of the user interface. Although several variations exist for the user interface, the usual structure is a modified spreadsheet format. A useful interface allows the user to display the data from different perspectives, perform statistical computations and tests, query the data into successively

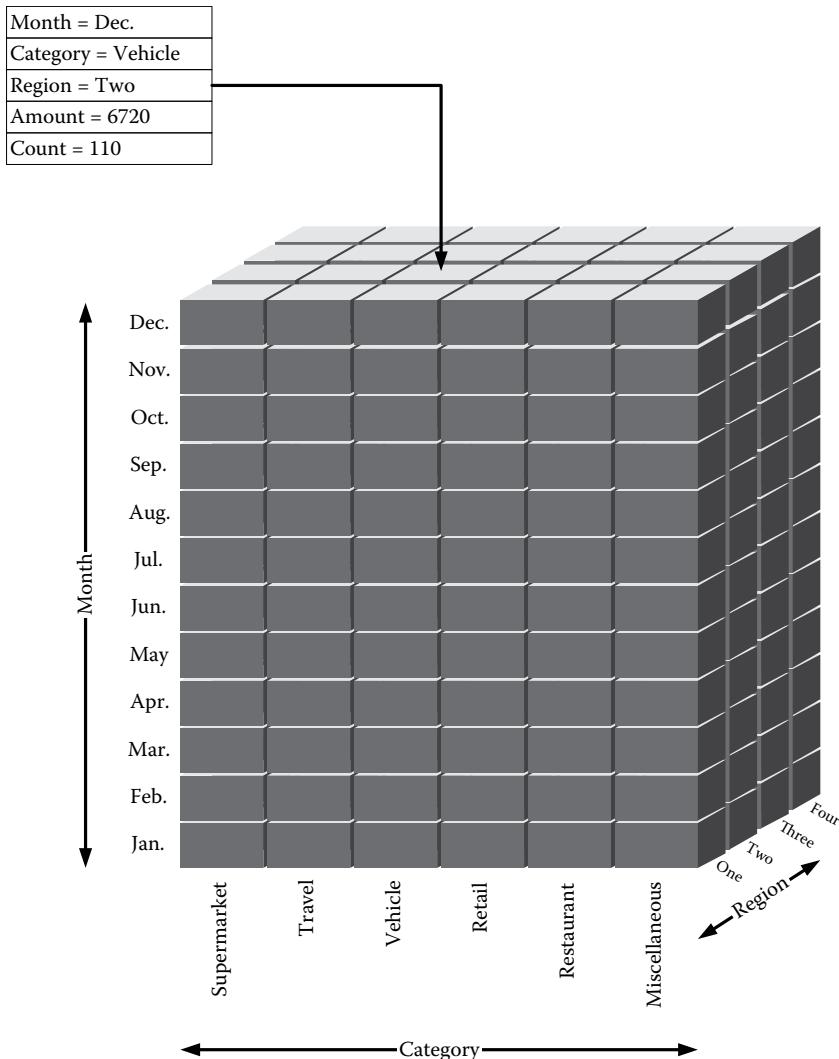


FIGURE 14.6 A multidimensional cube for credit card purchases.

higher and/or lower levels of detail, cross-tabulate the data, and view the data with graphs and charts. The capabilities of OLAP are best described with an example.

14.3.1 OLAP: An Example

Once again, we'll consider the credit card application described throughout the text. Thus far, our discussion has been limited to credit card promotions. By analyzing the data supplied in the star schema of Figure 14.3, we can instead address issues about when a promotional offering should be made available to credit card customers. Let's use OLAP to help us make decisions about times and locations for promotional offerings.

For our example, we return to Figure 14.6, which shows the three-dimensional data cube created from the star schema displayed in Figure 14.3. The cube contains $12 \times 6 \times 4 = 288$ cells.

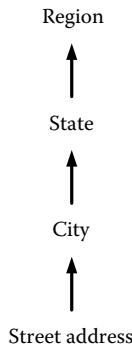


FIGURE 14.7 A concept hierarchy for location.

Stored within each cell is the total amount spent within a given category by all credit card customers for a specific month and region. If an average purchase amount is to be computed, the cube will also contain a count representing the total number of purchases for each month, category, and region. The arrow in Figure 14.6 points to a cube holding the total amount and the total number of auto and vehicle purchases in region 2 for the month of December.

Each attribute of an OLAP cube may have one or more associated *concept hierarchies*. A concept hierarchy defines a mapping that allows the attribute to be viewed from varying levels of detail. Figure 14.7 displays a concept hierarchy for the attribute *location*. As you can see, *region* holds the highest level of generality within the hierarchy. The second level of the hierarchy tells us that one or more states make up a region. The third and fourth levels show us that one or more cities are contained in a state and one or more addresses are found within a city. By definition, the hierarchy shows that each state is contained entirely within one and only one region and each city is part of exactly one state. Let's create a scenario where our OLAP cube, together with the concept hierarchy of Figure 14.7, will be of assistance in a decision-making process.

Suppose we wish to determine a best situation for offering a luggage and handbag promotion for travel. Our goal is to determine when and where the promotional offering will have its greatest impact on customer response. We can do this by determining those times and locations where relatively large amounts have been previously spent on travel. Once determined, we then designate the best regions and times for the promotional offering so as to take advantage of the likelihood of ensuing travel purchases. Here is a list of common OLAP operations together with a few examples for our travel promotion problem:

1. The *slice* operation selects data on a single dimension of an OLAP cube. For the cube in Figure 14.6, a slice operation leaves two of the three dimensions intact, while a selection on the remaining dimension creates a subcube from the original cube. Two queries for the slice operator are as follows:
 - a. Provide a spreadsheet of *month* and *region* information for all cells pertaining to *travel*.
 - b. Select all cells where *purchase category* = *restaurant* or *supermarket*.

2. The *dice* operation extracts a subcube from the original cube by performing a select operation on two or more dimensions. Here are three queries requiring one or more dice operations:
 - a. Identify the month of peak travel expenditure for each region.
 - b. Is there a significant variation in total dollars spent for travel and entertainment by customers in each region?
 - c. Which month shows the greatest amount of total dollars spent on travel and entertainment for all regions?
3. *Roll-up*, or aggregation, is a combining of cells for one of the dimensions defined within a cube. One form of roll-up uses the concept hierarchy associated with a dimension to achieve a higher level of generalization. For our example, this is illustrated in Figure 14.8, where the roll-up is on the *time* dimension. The cell pointed to in the figure contains the total of all data for the months of October, November, and December. A second type of roll-up operator actually eliminates an entire dimension. For our example, suppose we choose to eliminate the *location* dimension. The end result is a spreadsheet of total purchases delineated by month and category type.
4. *Drill-down* is the reverse of a roll-up and involves examining data at some level of greater detail. Drilling down on *region* in Figure 14.6 results in a new cube where each cell highlights a specific category, month, and state.

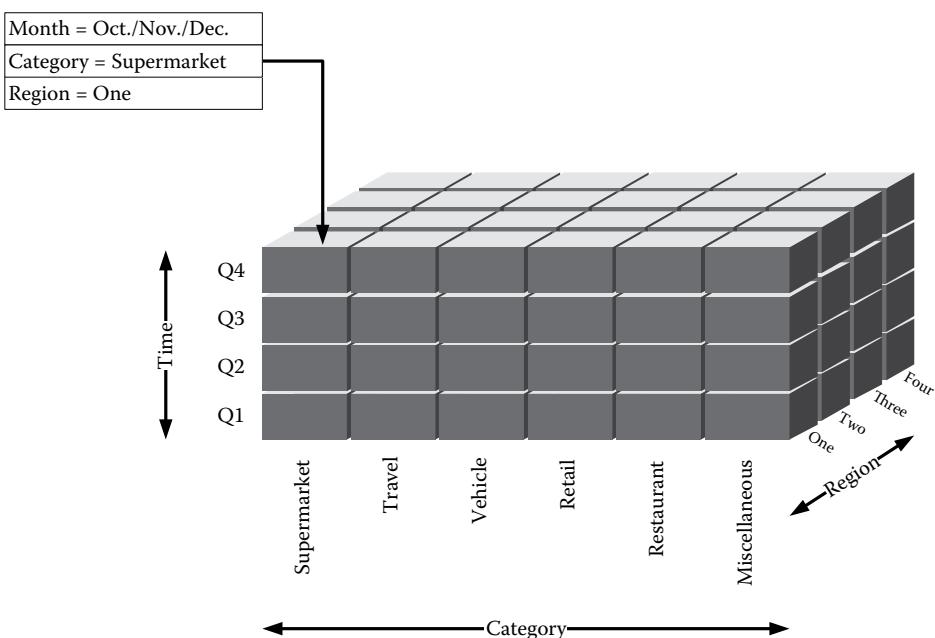


FIGURE 14.8 Rolling up from months to quarters.

5. *Rotation*, or pivoting, allows us to view the data from a new perspective. For our example, we may find it easier to view the OLAP cube in Figure 14.6 by having months displayed on the horizontal axis and purchase category on the vertical axis.

14.3.2 General Considerations

Most useful strategies for analyzing a cube require a sequence of two or more operations. For example, to view Q4 totals for travel and entertainment purchases in regions 1 and 2, we can roll up the original cube on the *time* dimension and then dice the resultant cube on *category* and *region*. Alternatively, the dice operator may be replaced by two slice operations, one on *region* and a second on *purchase category*. As a second example, as travel and entertainment are grouped into a single category, we can't be sure about the amount of money spent on items such as airline tickets as compared to tickets to the local ballpark. We can attempt to isolate overnight travel expenses by finding those times where large travel and entertainment purchases correlate with increased restaurant purchases and decreased supermarket purchases. To do this, we first slice the cube in Figure 14.6 to create a subcube where *purchase category* is limited to *supermarket*, *restaurant*, and *travel and entertainment*. After this, we rank *travel and entertainment* purchases and perform a statistical analysis to better determine those times within the various regions that see significant increases in travel and restaurant purchases together with decreases in supermarket sales. The level of sophistication for the user interface together with the flexibility of the OLAP engine will, to a large extent, determine the time needed to design solutions requiring multiple queries.

MS Excel provides an interface that allows us to build OLAP cubes from data stored in a relational database. The information contained in the cube can then be displayed and manipulated in Excel as a *pivot table*. Pivot tables are easy to use and offer many of the same features seen with more complex OLAP interface tools. Pivot tables also provide us with a powerful tool for analyzing spreadsheet data. Pivot table features include the ability to summarize data, group data in various ways, and display data using several alternative formats. Pivot tables are the topic of the next section.

14.4 EXCEL PIVOT TABLES FOR DATA ANALYTICS

The best way for you to understand how pivot tables are used is to work through an example or two. The following examples offer a good starting point for learning about pivot tables. We leave it up to you to further investigate the capabilities of this useful data analytics tool.

Our examples explain how pivot tables work in MS Excel 2013. Several marked differences exist with previous versions of Excel.

Here we briefly introduced pivot tables as a useful analytics tool. For more information about pivot tables, we encourage you to read the documents available through the MS Excel Help facility.

PIVOT TABLE TUTORIAL

Creating a Simple Pivot Table

For this example, we use the data in *CreditCardPromotion.xlsx*. Here is the process:

- Open *CreditCardPromotion.xlsx* and select all of the data including the column headings. Do not select additional blank cells. Click on the lowest right point of your data to bring up the charts and tables menu. Your screen will appear as in Figure 14.9.
- Click on TABLES then click on the left most *Pivot Table*.
- Uncheck any attributes in the pivot table fields menu shown on the right of your screen. Your screen will appear similar to Figure 14.10.

Let's create a simple two-dimensional table showing the relationship between *income range* and *credit card insurance*.

- Drag *credit card ins* into the columns field. Drag *income range* into the rows field. Finally, drag *income range* into the Σ values field. Your screen will appear as in Figure 14.11.

We now have a visualization of the relationship between *income range* and *credit card insurance*. Let's add a chart to our table.

- At the top middle of your Excel spreadsheet under *PivotTableTools*, click *Analyze*. This gives a new row of options seen at the top of your screen. Look to the right to find *PivotChart*. Click on *PivotChart* and then on *column*.

Income Range	Magazine Promo	Watch Promo	Life Ins Promo	Credit Card Ins.	Gender	Age
40-50,000	Yes	No	No	No	Male	45
30-40,000	Yes	Yes	Yes	No	Female	40
40-50,000	No	No	No	No	Male	42
30-40,000	Yes	Yes	Yes	Yes	Male	43
50-60,000	Yes	No	Yes	No	Female	38
20-30,000	No	No	No	No	Female	55
30-40,000	Yes	No	Yes	Yes	Male	35
20-30,000	No	Yes	No	No	Male	27
30-40,000	Yes	No	No	No	Male	43
30-40,000	Yes	Yes	Yes	No	Female	41
40-50,000	No	Yes	Yes	No	Female	43
20-30,000	No	Yes	Yes	No	Male	29
50-60,000	Yes	Yes	Yes	No	Female	39
40-50,000	No	Yes	No	No	Male	55
20-30,000	No	No	Yes	Yes	Female	19

FIGURE 14.9 Creating a pivot table.

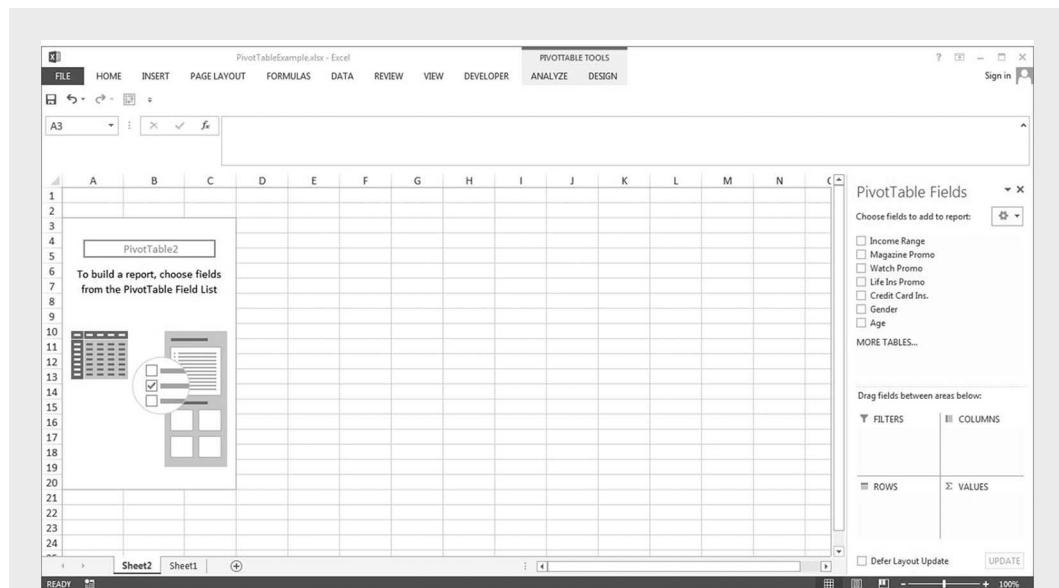
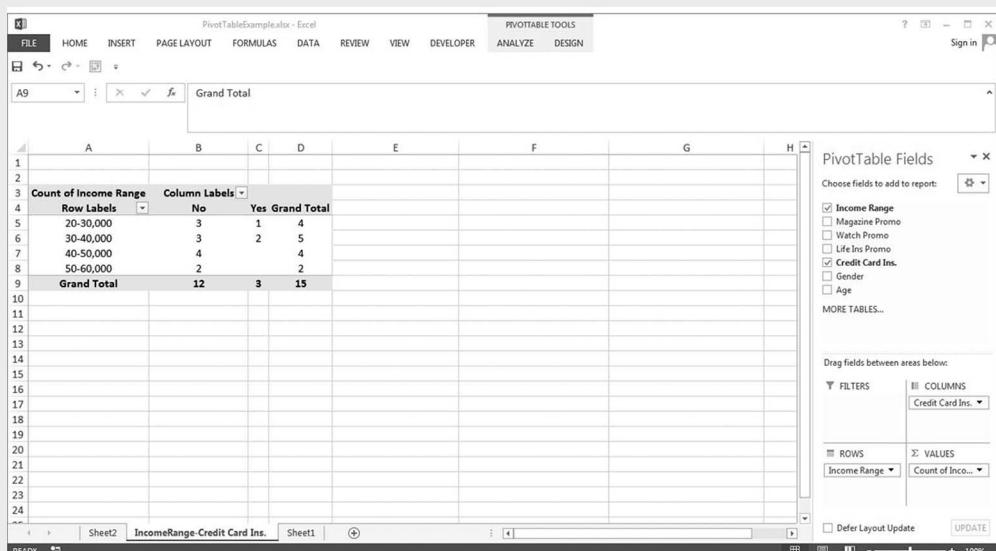
FIGURE 14.10 A blank *pivot table*.

FIGURE 14.11 A comparison of credit card insurance and income range.

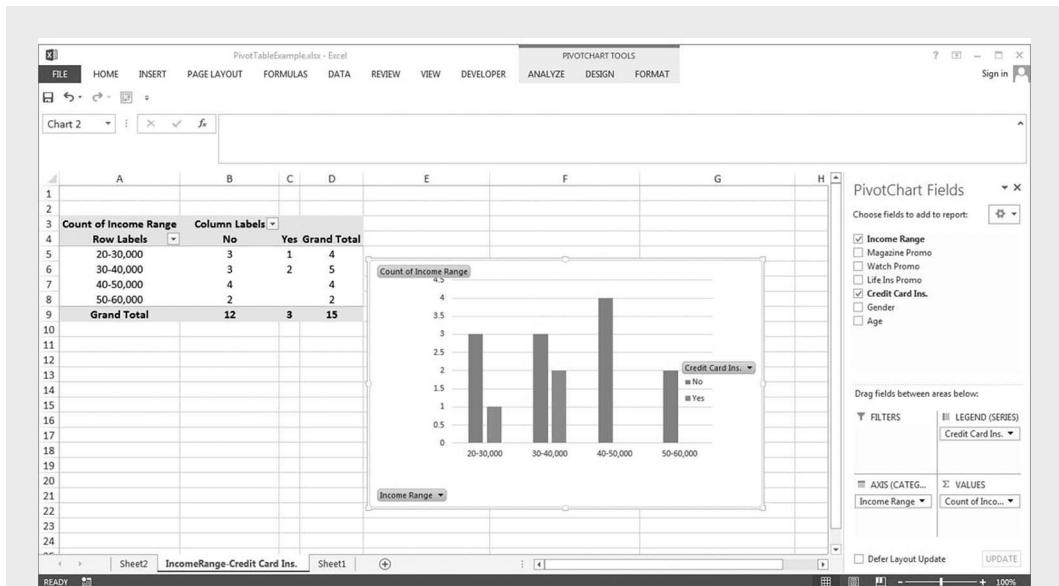


FIGURE 14.12 A chart comparing credit card insurance and income range.

Your screen will appear as in Figure 14.12. This simple illustration allows us to see the values in the table in chart format. Let's look at a more interesting example.

Creating a Multidimensional Pivot Table

For this example, we will use a pivot table to investigate relationships between the *magazine*, *watch*, and *life insurance* promotions relative to customer *gender* and *income range*. We will do this by creating a three-dimensional cube like the one shown in Figure 14.13. Each cell of the cube contains a count of the number of customers who either did or did not take part in the promotional offerings. The arrow in Figure 14.13 points to the cell holding the total number of customers who took advantage of the life insurance promotion and the magazine promotion, but who did not take advantage of the watch promotion. Here is the procedure to create the cube:

- Drag *Life Ins Promo* then *Magazine Promo* to the rows field.
- Drag *Watch Promo* to the columns field.
- Drag the attributes *Watch Promo*, *Life Ins Promo*, and *Magazine Promo*, in this order, from the PivotTable Fields area to the Σ values field.
- Highlight all columns and center your output.
- If your output does not appear as in Figure 14.14, rearrange the attribute positions within the Σ values field until they are ordered as in the *Pivot Table Fields* section of Figure 14.14.

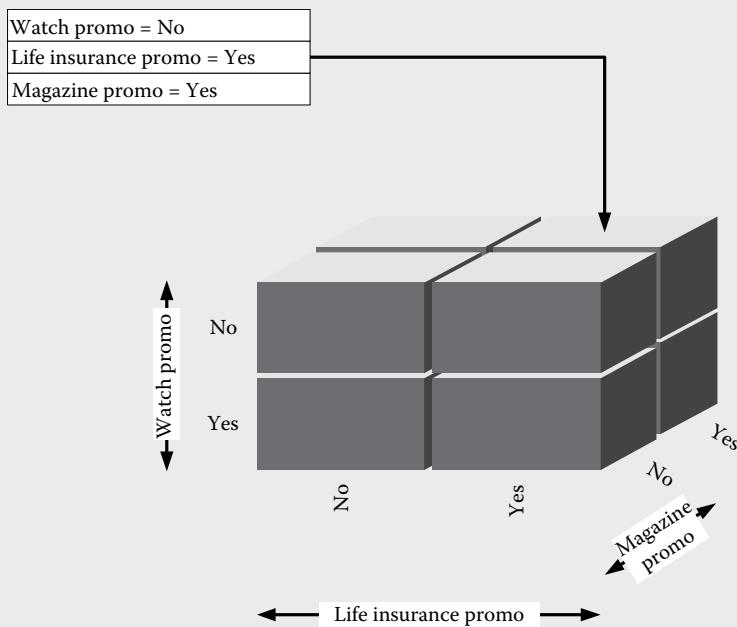


FIGURE 14.13 A credit card promotion cube.

A screenshot of Microsoft Excel showing a pivot table titled "CreditCardPromotion.xlsx". The pivot table has "Count of Watch Promo" in the Row Labels, "Count of Life Ins Promo" in the Column Labels, and "Count of Magazine Promo" in the Total Col. The data shows counts for combinations of Watch, Life Insurance, and Magazine promotions. The PivotTable Fields pane on the right lists fields like Income Range, Magazine Promo, Watch Promo, Life Ins Promo, Credit Card Ins., Gender, and Age.

	Count of Watch Promo	Count of Life Ins Promo	Count of Magazine Promo	Count of Watch Promo	Count of Life Ins Promo	Count of Magazine Promo	Total Col
No	4	4	4	2	2	2	2
No	2	2	2	2	2	2	2
Yes	2	2	2				
Yes	3	3	3	6	6	6	6
No	1	1	1	2	2	2	2
Yes	2	2	2	4	4	4	4
Grand Total	7	7	7	8	8	8	8

FIGURE 14.14 A pivot table for the cube shown in Figure 14.13.

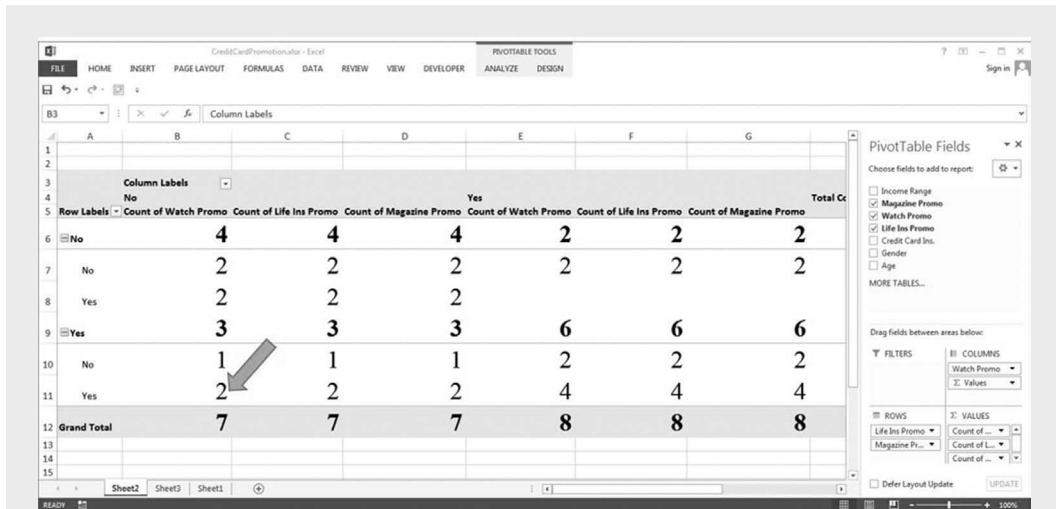


FIGURE 14.15 Pivot table position corresponding to the highlighted cell in Figure 14.13.

Reading the three-dimensional structure on a two-dimensional surface can be a challenge. However, holding your mouse over any position gives clearer insight into what is displayed. In Figure 14.15, the mouse position corresponds to the cell pointed to in Figure 14.13. That is, there are three individuals that said no to the *watch promotion* and yes to the *magazine* and *life insurance* promotions.

- We can drill down with a double click on any cell. A double click on the cell highlighted in Figure 14.15 gives us the result seen in Figure 14.16.

Figure 14.16 verifies that two individuals accepted the magazine and life insurance promotions but rejected the watch promotion. Finally, let's perform a slice operation!

- Locate Pivot Table Tools at the top of your screen and click on *Analyze*. Locate and click on *insert slicer*. Highlight *Gender* and click *OK*. Finally, highlight female. Your screen will appear as in Figure 14.17.

All values now appearing in the table represent instances of female gender.

We can also perform a slice by simply moving *Gender* to the *Filters* area within PivotTable Fields. A dropdown menu will appear as in Figure 14.18, where we highlight female and get the desired result.

The screenshot shows a Microsoft Excel window with the title bar "CreditCardPromotion.xlsx - Excel". The ribbon menu includes FILE, HOME, INSERT, PAGE LAYOUT, FORMULAS, DATA, REVIEW, VIEW, and DEVELOPER. The active sheet is Sheet3, which contains a pivot table with the following data:

	Income Range	Magazine	Watch Promo	Life Ins	Promo Credit Card	Gender	Age
2	30-40,000	Yes	No	Yes	Yes	Male	35
3	50-60,000	Yes	No	Yes	No	Female	38
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							

Below the table, the status bar shows "Sheet3 | Sheet2 | Sheet1 | +". The bottom right corner of the window shows "100%".

FIGURE 14.16 Drilling *down* into the cell highlighted in Figure 14.15.

The screenshot shows a Microsoft Excel window with the title bar "PivotTableExample.xlsx - Excel". The ribbon menu includes FILE, HOME, INSERT, PAGE LAYOUT, FORMULAS, DATA, REVIEW, VIEW, DEVELOPER, and SLICER TOOLS. The active sheet is Sheet3, which contains a pivot table with the following data:

	Column Labels	Count of Watch Promo	Count of Life Ins Promo	Count of Magazine Promo	Count of Watch Promo	Count of Life Ins Promo	Count of Magazine Promo	Total Count of W
4	No	1	1	1				1
5	Row Labels	Count of Watch Promo	Count of Life Ins Promo	Count of Magazine Promo	Count of Watch Promo	Count of Life Ins Promo	Count of Magazine Promo	Total Count of W
6	No	1	1	1				1
7	No	1	1	1				1
8	Yes	2	2	2	4	4	4	6
9	No	1	1	1	1	1	1	2
10	Yes	1	1	1	3	3	3	4
11	Grand Total	3	3	3	4	4	4	7

A slicer titled "Gender" is displayed on the right side of the pivot table, with "Female" selected. The status bar at the bottom shows "Sheet2 | IncomeRange-Credit Card Ins. | Sheet4 | Sheet5 | Sheet3 | Sheet1 | +". The bottom right corner of the window shows "100%".

FIGURE 14.17 Highlighting female customers with a slice operation.

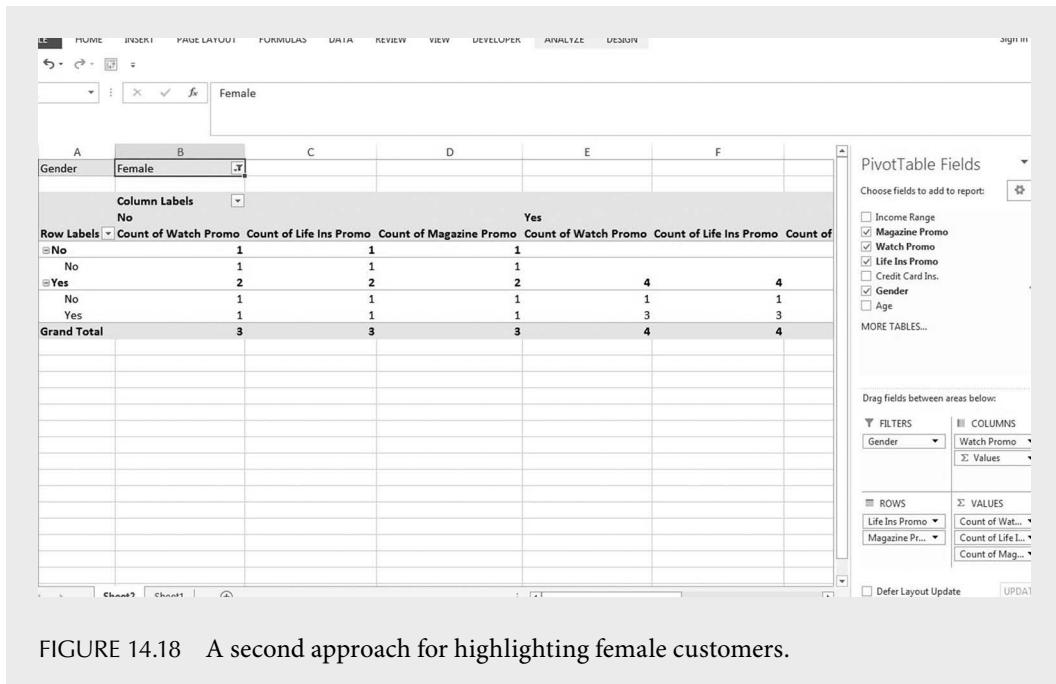


FIGURE 14.18 A second approach for highlighting female customers.

14.5 CHAPTER SUMMARY

Operational databases are designed to process individual transactions quickly and efficiently. To accomplish this, operational data are often stored as a set of normalized relational tables. The normalization minimizes data redundancy, thus allowing for effective data processing. Once transactional data are no longer useful, they are transferred from the operational environment to a secondary storage medium. If a decision support facility exists, the storage facility will likely be a data warehouse. W. H. Inmon (1992) describes the data warehouse as a “subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s decision making process.”

Two methods have been developed for organizing a data warehouse. One method structures the data as a multidimensional array. A second method uses the relational model for data storage and retrieval. In either case, the user views the data as a multidimensional data store. Although each method has advantages, the relational approach—implemented with a star schema—is the more popular technique. The star schema supports a central fact table and several dimension tables. The fact table defines the dimensions of the multidimensional space. Individual dimension tables store detailed information about their associated fact table dimensions. One variation of the star schema is the snowflake schema, which encourages the structuring of normalized dimension tables. A second variation is the constellation schema, which can be used to accommodate a data warehouse containing several fact tables.

The primary function of warehoused data is decision support. Three categories of decision support are *reporting data*, *online analytical processing*, and *data analytics*. A data reporting facility must be capable of generating detailed reports about the data

stored in the warehouse. Data analytics is often accomplished through data mining. Online analytical processing is a query-based methodology that supports data analysis in a multidimensional environment. OLAP tools contain a friendly user interface and are capable of displaying data from different perspectives, performing statistical analysis, and querying data at successively lower and/or higher levels of detail. MS Excel pivot tables offer many of the same features seen with more complex OLAP tools. Pivot table features include the ability to summarize, group, data and display data in several formats.

14.6 KEY TERMS

- *Concept hierarchy*. A mapping that allows attributes to be viewed from varying levels of detail.
- *Constellation schema*. A variation of the star schema that allows more than one central fact table.
- *Data model*. A notational language that documents the structure of data independent of how the data will be used.
- *Dependent data mart*. A departmental warehouse designed for a specific purpose and created from the data in a larger warehouse structure.
- *Dice*. An OLAP operation on two or more dimensions that extracts a subcube from the original cube.
- *Dimension table*. A relational table containing information about one of the dimensions of a star schema.
- *Drill-down*. An OLAP operation performed on a data cube that allows for examining data at some level of greater detail.
- *Entity*. A generic representation for a class of persons, places, or things.
- *Entity relationship diagram (ERD)*. A data modeling tool that shows the structure of the data in terms of entities and relationships between entities. Entities represent classes of persons, places, or things. Relationships between entities can be one-to-one, one-to-many, or many-to-many.
- *ETL routine*. An ETL (extract, transform, and load) routine is the name given to a process that cleans, transforms, and loads data into a warehouse.
- *Fact table*. A relational table that defines the dimensions of the multidimensional space within a star schema.
- *First normal form (1NF)*. A rule that requires all attributes within an entity to have a single value.
- *Granularity*. A term used to describe the level of detail of stored information.

- *Independent data mart.* A data store that is similar to a warehouse but limits its focus to a single subject. An independent data mart is structured using operational data as well as external data sources.
- *Intersection entity.* An entity created by mapping a many-to-many relationship to two one-to-many relationships.
- *Many-to-many relationship.* A relationship between two entities, A and B , where each instance of A is associated with one or several instances of B , and each instance of B is associated with one or several instances of A .
- *Metadata.* Data about data.
- *Normalization.* A multistep process designed to remove redundancies from a data model.
- *One-to-many relationship.* A relationship between two entities, A and B , where each instance of A is associated with one or several instances of B .
- *One-to-one relationship.* A relationship between two entities, A and B , where each instance of A is associated with exactly one instance of B .
- *Online transactional processing (OLTP).* Database procedures designed to process individual transactions quickly and efficiently.
- *Pivot table.* An MS Excel-based data analysis tool that can be used to summarize data, group data in various ways, and display data using several alternative formats.
- *Roll-up.* An OLAP operation that combines the cells of a data cube to form a higher-level view of the data.
- *Rotation.* An OLAP tool with this capability allows the user to examine data from several perspectives.
- *Second normal form (2NF).* An entity is in 2NF if it is in 1NF and all nonkey attributes are dependent on the full primary key.
- *Slice.* An OLAP operation that creates a subcube by performing a selection on a single dimension of a data cube.
- *Slowly changing dimension.* A dimension whose attributes change over time.
- *Snowflake schema.* A variation of the star schema where some of the dimension tables linked directly to the fact table are further subdivided. This permits the dimension tables to be normalized, which in turn means less total storage.
- *Star schema.* A multidimensional data warehouse model implemented within a relational database. The model consists of a fact table and one or more dimension tables.
- *Third normal form (3NF).* An entity is in 3NF if it is in 2NF and every nonkey attribute is dependent entirely on the primary key.

EXERCISES

Review Questions

1. Differentiate between the following terms:
 - a. Independent data mart and dependent data mart
 - b. Fact table and dimension table
 - c. Slice and dice
 - d. Drill-down and roll-up
 - e. OLTP and OLAP
2. Specify each relationship as one-to-one, one-to-many, or many-to-many. Justify each answer.
 - a. Employer–employee
 - b. Automobile–license plate
 - c. Pastor–parish
 - d. Course–instructor
 - e. Home–owner
3. Consider the star schema in Figure 14.3. Suppose an individual customer would like us to extract a list of the specific items he/she purchased during a given month and year. Will you be able to accommodate the customer? Explain your answer.
4. Think of ways to incorporate promotions into the star schema of Figure 14.3 without creating a second fact table. What are the positive and negative aspects of each approach?
5. Sketch the OLAP cubes created from the slice and dice operations described in Section 14.3.

Computational Questions

1. Sketch a three-dimensional OLAP cube from the star schema in Figure 14.3 where the dimensions are *purchase category*, *gender*, and *income range*. Describe several slice, dice, and rotation operations for extracting useful information from the data cube.
2. Construct a pivot table with *CardiologyMixed.xls*. Make *angina* and *thal* row attributes and *class* a column attribute. Place *class*, *angina*, and *thal* in the data area. Specify *slope*, *gender*, and *#colored vessels* as filters. Use the pivot table to answer the following questions:
 - a. How many healthy males are in the database?
 - b. How many healthy females have three or more colored vessels?

- c. Determine values for *#colored vessels* and *angina* that are sufficient for defining a sick individual.
 - d. Verify or refute the hypothesis: The majority of individuals with *#colored vessels* = 0 are healthy.
 - e. Verify or refute the hypothesis: A typical healthy individual will show no symptoms of angina and will have a value of *normal* for attribute *thal*.
3. Recreate the pivot table shown in Figure 14.14 to answer the following:
- a. How many cardholders did not purchase a single promotional offering?
 - b. How many cardholders took advantage of the magazine and watch promotions but did not purchase the life insurance promotion?
 - c. How many male cardholders make between \$50,000 and \$60,000?
 - d. Verify or refute the hypothesis: Individuals who purchased all three promotional offerings also purchased credit card insurance.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Appendix A—Software and Data Sets for Data Mining

The examples, illustrations, tutorials, and end-of-chapter exercises rely on two well-known data mining and analytics tools. The text is designed in a way that allows you to learn about data mining techniques with the help of one or both tools.

WEKA

The *Waikato Environment for Knowledge Analysis* (Weka) is a multipurpose, easy to use tool for data mining and analysis. Weka was developed at the University of Waikato in New Zealand. Weka is written in the Java programming language and is publicly available under the terms of the GNU General Public License. Weka contains a wealth of preprocessing and data mining tools, graphical features, and visualization capabilities. The latest version of Weka can be freely downloaded by clicking the download link at the website

<http://www.cs.waikato.ac.nz/ml/weka/>

Once you have installed Weka, a display similar to Figure A.1 will appear. The figure shows the Weka GUI Chooser together with a message about Weka's package manager. The package manager is used to install algorithms not part of Weka's initial installation package. Two packages are of particular interest to us. The *SelfOrganizingMap* package contains the Kohonen clustering algorithm illustrated in Chapter 8. The second package is an implementation of the CART algorithm briefly described in Chapter 3. Here's how to install these packages:

Close the package manager message window.

Mouse to tools (arrow 1 in Figure A.1) and click on package manager.

Scroll the packages until you see *SelfOrganizingMap* (arrow 1 in Figure A.2).

Click on install (arrow 2 in Figure A.2).

Once again, scroll the packages but this time locate and install *simpleCart*.

Once the *simpleCart* installation is complete, click on *Installed* to determine if both packages have been installed (Figure A.3).



FIGURE A.1 A successful installation.

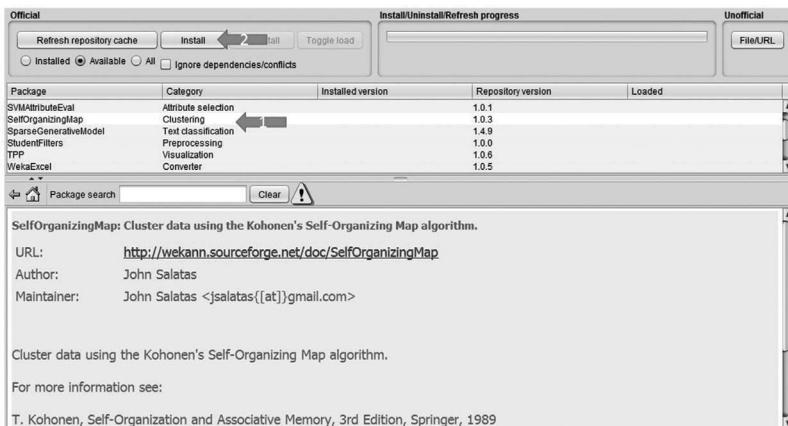


FIGURE A.2 Locating and installing a package.

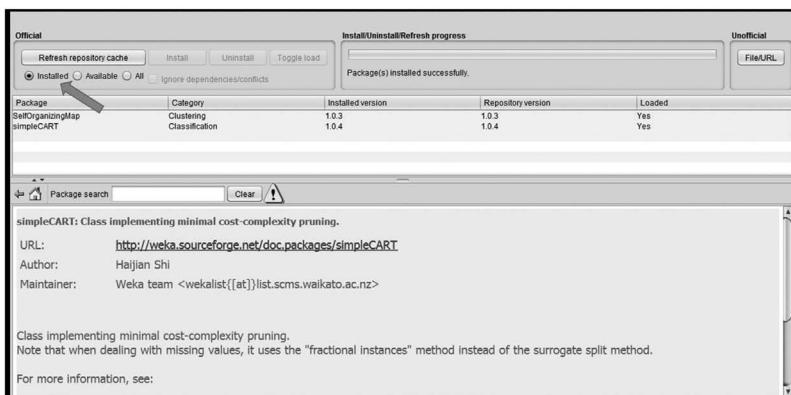


FIGURE A.3 List of installed packages.

This completes the process for installing the extra packages used in the text.

Lastly, arrow 2 in Figure A.1 points to the *ArffViewer*. ArffViewer displays Arff formatted files in an easy-to-read spreadsheet-like format. However, it is important to note that the viewer does not load files into the Weka Explorer.

RAPIDMINER

RapidMiner is a commercial data mining and predictive analytics tool. RapidMiner Studio is an easy-to-use, open-source, and code-free version of RapidMiner's commercial product. RapidMiner uses a workflow paradigm and offers several options for interacting with its interface.

Fundamental to RapidMiner's structure is the operator. Operators are used to preprocess, visualize, transform, create, evaluate, and optimize models. The text concentrates on a basic subset of RapidMiner's 1500-plus operators. Once familiar with this subset of operators, you will find it easy to add new operators to your list.

Go to the website <http://docs.rapidminer.com/studio/> and click on *installation* for instructions on downloading and installing RapidMiner Studio. This website also contains links to the RapidMiner Studio manual, operator reference guide, tutorials, and reference notes. During the installation process, you will have the option to create a free RapidMiner user account. By doing so, you join the RapidMiner community, which offers several benefits, including a community forum and support.

ATTRIBUTE AND CLUSTER ANALYZER (ACZ)

The Attribute and Cluster Analyzer (ACZ) was briefly introduced in Section 3.2. ACZ is designed to help determine a best set of attributes for building supervised learner models. ACZ is easy to use and offers several insights into the nature of any data set to be mined. ACZ accepts data sets given in Weka's standard ARFF format. ACZ is a work in progress. If you wish to experiment with ACZ, it can be freely downloaded at <http://krypton.mnsu.edu/~sa7379bt/>.

DATA SETS FOR DATA MINING

The data sets used for the tutorials as well as all illustrations are contained in a zip file available at two locations:

- The CRC website—LINK PENDING
- <http://krypton.mnsu.edu/~sa7379bt/>

Data Sets for Weka

Data sets for the Weka tutorials, illustrations, and exercises are given in *datasetsWeka.zip*. Descriptions of the data sets are provided in various chapters of the text. Data descriptions can also be found within the ARFF files. To see the description for a particular file, open the data file using a standard editor. Additional data sets formatted for Weka can be found at <http://www.cs.waikato.ac.nz/ml/weka/datasets.html>.

Data Sets for RapidMiner

The RapidMiner data sets are found in *datasetsRapidMiner.zip*. Several of the Excel-based files contain data descriptions in *Sheet2*. Additional data sets, processes, tutorials, and templates are in the *Samples* folder of your RapidMiner installation.

Community Data Sets—IBM Watson Analytics

IBM Watson Analytics is a commercial cloud-based predictive analytics and data visualization tool. Several interesting Excel-based data sets for data mining are available through the Watson Analytics community at <https://community.watsonanalytics.com/guide-to-sample-datasets/>.

These data sets can be used with RapidMiner without modification. You can also opt for a free trial of Watson Analytics by visiting <https://community.watsonanalytics.com/>.

KDnuggets

KDnuggets is the leading information repository for data mining and knowledge discovery. The site includes topics on data mining—companies, software, publications, courses, data sets, and more. Here is the home page of the KDnuggets website:

<http://www.kdnuggets.com>

Here is a link to popular data sets from several domains.

<http://www.kdnuggets.com/datasets/index.html>

Machine Learning Repository

The University of California–Irvine (UCI) Machine Learning Repository contains a wealth of data from several domains. Here is the UCI home page address.

<https://archive.ics.uci.edu/ml/datasets.html>

Here is the FTP archive site for downloading the data sets in the UCI library.

<ftp://ftp.ics.uci.edu/pub/machine-learning-databases/>

Appendix B—Statistics for Performance Evaluation

SINGLE-VALUED SUMMARY STATISTICS

The *mean* or average value is computed by summing the data and dividing the sum by the number of data items. Specifically,

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \quad (\text{B.1})$$

where

μ is the mean value

n is the number of data items

x_i is the i th data item

Here is a formula for computing variance.

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (\mu - x_i)^2 \quad (\text{B.2})$$

where

σ^2 is the variance

μ is the population mean

n is the number of data items

x_i is the i th data item

When calculating the variance for a sampling of data, a better result is obtained by dividing the sum of squares by $n - 1$ rather than by n . The proof of this is beyond the scope of this book; however, it suffices to say that when the division is by $n - 1$, the sample variance is an unbiased estimator of the population variance. An unbiased estimator has

the characteristic that the average value of the estimator taken over all possible samples is equal to the parameter being estimated.

THE NORMAL DISTRIBUTION

Here is the equation for the normal or bell-shaped curve.

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2} \quad (\text{B.3})$$

where

$f(x)$ is the height of the curve corresponding to values of x

e is the base of natural logarithms approximated by 2.718282

μ is the arithmetic mean for the data

σ is the standard deviation

COMPARING SUPERVISED LEARNER MODELS

In Chapter 7, we described a general technique for comparing two supervised learner models using the same test data set. Here we provide two additional techniques for comparing supervised models. In both cases, model test set error rate is treated as a sample mean.

Comparing Models with Independent Test Data

With two independent test sets, we simply compute the variance for each model and apply the classical hypothesis testing procedure. Here's an outline of the technique.

Given

- Two models M_1 and M_2 built with the same training data
- Two independent test sets, set A containing n_1 elements and set B with n_2 elements
- Error rate E_1 and variance v_1 for model M_1 on test set A
- Error rate E_2 and variance v_2 for model M_2 on test set B

Compute

$$T = \frac{|E_1 - E_2|}{\sqrt{(v_1/n_1 + v_2/n_2)}} \quad (\text{B.4})$$

Conclude

- If $T >= 2$, the difference in the test set performance of model M_1 and model M_2 is significant.

Let's look at an example. Suppose we wish to compare the test set performance of learner models, M_1 and M_2 . We test M_1 on test set A and M_2 on test set B. Each test set contains 100 instances. M_1 achieves an 80% classification accuracy with set A, and M_2 obtains a 70% accuracy with test set B. We wish to know if model M_1 has performed significantly better than model M_2 . Here are the computations.

- For model M_1

$$E_1 = 0.20$$

$$\nu_1 = 0.2 (1 - 0.2) = 0.16$$

- For model M_2 :

$$E_2 = 0.30$$

$$\nu_2 = 0.3 (1 - 0.3) = 0.21$$

- The computation for T is

$$T = \frac{|0.20 - 0.30|}{\sqrt{(0.16/100 + 0.21/100)}}$$

$$T \approx 1.4714$$

Since $T < 2$, the difference in model performance is not considered to be significant. We can increase our confidence in the result by switching the two test sets and repeating the experiment. This is especially important if a significant difference is seen with the initial test set selection. The average of the two values for T is then used for the significance test.

Pairwise Comparison with a Single Test Data Set

When the same test set is applied to the data, one option is to perform an instance-by-instance pairwise matching of the test set results. With an instance-based comparison, a single variance score based on pairwise differences is computed. Here is the formula for calculating the joint variance.

$$V_{12} = \frac{1}{n-1} \sum_{i=1}^n [(e_{1i} - e_{2i}) - (E_1 - E_2)]^2 \quad (\text{B.5})$$

where

V_{12} is the joint variance

e_{1i} is the classifier error on the i th instance for learner model 1

e_{2i} is the classifier error on the i th instance for learner model 2

$E_1 - E_2$ is the overall classifier error rate for model 1 minus the classifier error rate for model 2

n is the total number of test set instances

When test set error rate is the measure by which two models are compared, the output attribute is categorical. Therefore, for any instance i contained in class j , e_{ij} is 0 if the classification is correct and 1 if the classification is in error. When the output attribute is numerical, e_{ij} represents the absolute difference between the computed and actual output value. With the revised formula for computing joint variance, the equation to test for a significant difference in model performance becomes

$$T = \frac{|E_1 - E_2|}{\sqrt{V_{12}/n}} \quad (\text{B.6})$$

Once again, a 95% confidence level for a significant difference in model test set performance is seen if $T >= 2$. The aforementioned technique is appropriate only if an instance-based pairwise comparison of model performance is possible. In the next section, we address the case where an instance-based comparison is not possible.

CONFIDENCE INTERVALS FOR NUMERIC OUTPUT

Just as when the output is categorical, we are interested in computing confidence intervals for one or more numeric measures. For purposes of illustration, we use *mean absolute error*. As with classifier error rate, *mean absolute error* is treated as a sample mean. The sample variance is given by the formula

$$\text{variance}(\text{mae}) = \frac{1}{n-1} \sum_{i=1}^n (e_i - \text{mae})^2 \quad (\text{B.7})$$

where

e_i is the absolute error for the i th instance

n is the number of instances

Let's look at an example using the data in Table 7.2. To determine a confidence interval for the mean absolute error (mae) computed for the data in Table 7.2, we first calculate the variance. Specifically,

$$\begin{aligned} \text{variance}(0.0604) &= \frac{1}{14} \sum_{i=1}^{15} (e_i - 0.0604)^2 \\ &\approx (0.024 - 0.0604)^2 + (0.002 - 0.0604)^2 + \dots + (0.001 - 0.0604)^2 \\ &\approx 0.0092 \end{aligned}$$

Next, as with classifier error rate, we compute the standard error for the mae as the square root of the variance divided by the number of sample instances.

$$SE = \sqrt{(0.0092/15)} \approx 0.0248$$

Finally, we calculate the 95% confidence interval by respectively subtracting and adding two standard errors to the computed mae. This tells us we can be 95% confident that the actual mae falls somewhere between 0.0108 and 0.1100.

COMPARING MODELS WITH NUMERIC OUTPUT

The procedure for comparing models giving numeric output is identical to that for models with categorical output. In the case where two independent test sets are available and mae measures model performance, the classical hypothesis-testing model takes the form

$$T = \frac{|mae_1 - mae_2|}{\sqrt{(V_1/n_1 + V_2/n_2)}} \quad (\text{B.8})$$

where

mae_1 is the mean absolute error for model M_1

mae_2 is the mean absolute error for model M_2

V_1 and V_2 are variance scores associated with M_1 and M_2

n_1 and n_2 are the number of instances within each respective test set

When the models are tested on the same data and a pairwise comparison is possible, we use the formula

$$T = \frac{|mae_1 - mae_2|}{\sqrt{V_{12}/n}} \quad (\text{B.9})$$

where

mae_1 is the mean absolute error for model M_1

mae_2 is the mean absolute error for model M_2

V_{12} is the joint variance computed with the formula defined in Equation B.5

n is the number of test set instances

When the same test data are applied but a pairwise comparison is not possible, the most straightforward approach is to compute the variance associated with the mae for each model using the equation

$$\text{variance}(\text{mae}_j) = \frac{1}{n-1} \sum_{i=1}^n (e_i - \text{mae}_j)^2 \quad (\text{B.10})$$

where

mae_j is the mean absolute error for model j

e_i is the absolute value of the computed value minus the actual value for instance i

n is the number of test set instances

The hypothesis of no significant difference is then tested with the equation

$$T = \frac{|\text{mae}_1 - \text{mae}_2|}{\sqrt{\nu(2/n)}} \quad (\text{B.11})$$

where

ν is either the average or the larger of the variance scores for each model

n is the total number of test set instances

As is the case when the output attribute is categorical, using the larger of the two variance scores is the stronger test.

Bibliography

- Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining Association Rules Between Sets of Items in Large Databases. In P. Buneman and S. Jajodia, eds., *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York: ACM.
- Baltazar, H. (1997). Tracking Telephone Fraud Fast. *Computerworld*, 31, 11, 75.
- Baltazar, H. (2000). NBA Coaches' Latest Weapon: Data Mining. *PC Week*, March 6, 69.
- Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 5, 144–152.
- Brachman, R. J., Khabaza, T., Kloesgen, W., Piatetsky-Shapiro, G., and Simoudis, E. (1996). Mining Business Databases. *Communications of the ACM*, 39, 11, 42–48.
- Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24, 2, 123–140.
- Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Monterey, CA: Wadsworth International Group.
- Buchanan, B., Sutherland, G., and Feigenbaum, E. (1969). Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry. In B. Meltzer, D. Michie, and M. Swann, eds., *Machine Intelligence* (vol. 4). Edinburgh: Edinburgh University Press, 209–254.
- Burges, C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2, 121–167.
- Calders, T., Dexters, N., and Goethals, B. (2007). Mining Frequent Itemsets in a Stream. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining, ICDM '07*. Washington, DC: IEEE Computer Society, 83–92.
- Case, S., Azarmi, N., Thint, M., and Ohtani, T. (2001). Enhancing E-Communities with Agent-Based Systems. *Computer*, July, 64–69.
- Cendrowska, J. (1987). PRISM: An Algorithm for Inducing Modular Rules. *International Journal of Man-Machine Studies*, 27, 4, 349–370.
- Chester, M. (1993). *Neural Networks—A Tutorial*. Upper Saddle River, NJ: Prentice-Hall.
- Chou, W. Y. S., Hunt, Y. M., Beckjord, E. B., Moser, R. P., and Hesse, B. W. (2009). Social Media Use in the United States: Implications for Health Communication. *Journal of Medical Internet Research*, 11, 4, e48.
- Civco, D. L. (1991). Landsat TM Land Use and Land Cover Mapping Using an Artificial Neural Network. In *Proceedings of the 1991 Annual Meeting of the American Society for Photogrammetry and Remote Sensing*. Baltimore, MD, 3, 66–77.
- Cohen, W. (1995). Fast Effective Rule Induction. In *12th International Conference on Machine Learning*. San Francisco: Morgan Kaufmann, 115–123.
- Cox, E. (2000). Free-Form Text Data Mining Integrating Fuzzy Systems, Self-Organizing Neural Nets and Rule-Based Knowledge Bases. *PC AI*, September–October, 22–26.
- Culotta, A. (2010). Towards Detecting Influenza Epidemics by Analyzing Twitter Messages. In *Proceedings of the First Workshop on Social Media Analytics*, New York: ACM, 115–122.
- Dasgupta, A., and Raftery, A. E. (1998). Detecting Features in Spatial Point Processes with Clutter via Model-Based Clustering. *Journal of the American Statistical Association*, 93, 441, 294–302.

- Dawson, R. (1995). The “Unusual Episode” Data Revisited. *Journal of Statistics Education*, 3, 3.
- Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum-Likelihood from Incomplete Data via the EM Algorithm (with Discussion). *Journal of the Royal Statistical Society, Series B*, 39, 1, 1–38.
- Dixon, W. J. (1983). *Introduction to Statistical Analysis*, 4th ed. New York: McGraw-Hill.
- Domingos, P., and Hulten, G. (2000). Mining High Speed Data Streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*. New York: ACM, 71–80.
- Duda, R., Gaschnig, J., and Hart, P. (1979). Model Design in the PROSPECTOR Consultant System for Mineral Exploration. In D. Michie, ed., *Expert Systems in the Microelectronic Age*. Edinburgh: Edinburgh University Press, 153–167.
- Durfee, E. H. (2001). Scaling Up Agent Coordination Strategies. *Computer*, July, 39–46.
- Dwinnell, W. (1999). Text Mining Dealing with Unstructured Data. *PC AI*, May–June, 20–23.
- Edelstein, H. A. (2001). Pan for Gold in the Clickstream. *Information Week*, March 12.
- Fayyad, U., Haussler, D., and Stolorz, P. (1996). Mining Scientific Data. *Communications of the ACM*, 39, 11, 51–57.
- Fisher, D. (1987). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, 2, 2, 139–172.
- Freund, Y., and Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. In Saitta, L., ed., *Proc. Thirteenth International Conference on Machine Learning*. San Francisco: Morgan Kaufmann, 148–156.
- Ganti, V., Gehrke, J., and Ramakrishnan, R. (1999). Mining Very Large Databases. *Computer*, August, 38–45.
- Gardner, S. R. (1998). Building the Data Warehouse. *Communications of the ACM*, 41, 9, 52–60.
- Gehrke, J., Ramakrishnan, R., and Ganti, V. (1998). Rainforest—A Framework for Fast Decision Tree Construction of Large Datasets. *VLDB Conference*, 416–427.
- Gennari, J. H., Langley, P., and Fisher, D. (1989). Models of Incremental Concept Formation. *Artificial Intelligence*, 40, 1–3, 11–61.
- Giarratano, J., and Riley, G. (1989). *Expert Systems: Principles and Programming*. New York: PWS-Kent.
- Gill, H. S., and Rao, P. C. (1996). *The Official Guide to Data Warehousing*. Indianapolis, IN: Que Publishing.
- Giovinazzo, W. A. (2000). *Object-Oriented Data Warehouse Design (Building a Star Schema)*. Upper Saddle River, NJ: Prentice-Hall.
- Grossman, R. L., Hornick, M. F., and Meyer, G. (2002). Data Mining Standards Initiatives. *Communications of the ACM*, 45, 8.
- Haag, S., Cummings, M., and McCubrey, D. (2002). *Management Information Systems for the Information Age*, 3rd ed. Boston: McGraw-Hill.
- Haglin, D. J., and Roiger, R. J. (2005). A Tool for Public Analysis of Scientific Data. *Data Science Journal*, 4, 39–52.
- Hosmer, D. W., and Lemeshow, S. (1989). *Applied Logistic Regression*. New York: John Wiley & Sons.
- Huntsberger, D. V. (1967). *Elements of Statistical Inference*. Boston: Allyn and Bacon.
- Inmon, W. (1996). *Building the Data Warehouse*, 2nd ed. New York: John Wiley & Sons.
- Jain, A. K., Mao, J., and Mohiuddin, K. M. (1996). Artificial Neural Networks: A Tutorial. *Computer*, March, 31–44.
- Kass, G. V. (1980). An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, 29, 2, 119–127.
- Kaur, I., and Mann, D. (2014). Data Mining in Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4, 3, 1178–1183.
- Kimball, R., Reeves, L., Ross, M., and Thornthwaite, W. (1998). *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*. New York: John Wiley & Sons.

- Kohonen, T. (1982). Clustering, Taxonomy, and Topological Maps of Patterns. In M. Lang, ed., *Proceedings of the Sixth International Conference on Pattern Recognition*, Silver Spring, MD: IEEE Computer Society Press, 114–125.
- Kudyba, S. (2014). *Big Data, Mining and Analytics*. Boca Raton, FL: CRC Press.
- Larose, D. T., and Larose, C. D. (2015). *Data Mining and Predictive Analytics*, 2nd ed. New York: John Wiley & Sons.
- Lashkari, Y., Metral, M., and Maes, P. (1994). Collaborative Interface Agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Menlo Park, CA: American Association of Artificial Intelligence, 444–450.
- Lin, N. (2015). *Applied Business Analytics*. Upper Saddle River, NJ: Pearson.
- Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28, 2, 129–137.
- Long, S. L. (1989). *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, CA: Sage Publications.
- Maclin, R., and Opitz, D. (1997). An Empirical Evaluation of Bagging and Boosting. *Fourteenth National Conference on Artificial Intelligence*, Providence, RI: AAAI Press.
- Maiers, J., and Sherif, Y. S. (1985). Application of Fuzzy Set Theory. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15, 1, 41–48.
- Manganaris, S. (2000). Estimating Intrinsic Customer Value. *DB2 Magazine*, 5, 3, 44–50.
- Manning, A. (2015). *Databases for Small Business*. New York: Springer.
- McCulloch, W. S., and Pitts, W. (1943). A Logical Calculus of the Ideas Imminent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5, 115–137.
- Mena, J. (2000). Bringing Them Back. *Intelligent Enterprise*, 3, 11, 39–42.
- Merril, D. M., and Tennyson, R. D. (1977). *Teaching Concepts: An Instructional Design Guide*. Englewood Cliffs, NJ: Educational Technology Publications.
- Mitchell, T. M. (1997). Does Machine Learning Really Work? *AI Magazine*, 18, 3, 11–20.
- Mobasher, B., Cooley, R., and Srivastava, J. (2000). Automatic Personalization Based on Web Usage Mining. *Communications of the ACM*, 43, 8, 142–151.
- Mone, G. (2013). Beyond Hadoop. *Communications of the ACM*, 56, 1 22–24.
- Mukherjee, S., Feigelson, E. D., Babu, G. J., Murtagh, F., Fraley, C., and Rafter, A. (1998). Three Types of Gamma Ray Bursts. *Astrophysical Journal*, 508, 1, 314–327.
- Ortigosa, A., Carro, R. M., and Quiroga, J. I. (2014). Predicting User Personality by Mining Social Interactions in Facebook. *Journal of Computer and System Sciences*, 80, 1, 57–71.
- Peixoto, J. L. (1990). A Property of Well-Formulated Polynomial Regression Models. *American Statistician*, 44, 26–30.
- Perkowitz, M., and Etzioni, O. (2000). Adaptive Web Sites. *Communications of the ACM*, 43, 8, 152–158.
- Piegorsch, W. W. (2015). *Statistical Data Analytics*. New York: John Wiley & Sons.
- Platt, J. C. (1998). Fast Training of Support Vector Machines using Sequential Minimal Optimization. In Schoelkopf, B., Burges, C., and Smola, A. eds., *Advances in Kernel Methods—Support Vector Learning*. Cambridge, MA: MIT Press.
- Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1, 1, 81–106.
- Quinlan, J. R. (1993). *Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.
- Quinlan, J. R. (1994). Comparing Connectionist and Symbolic Learning Methods. In Hanson, S. J., G. A. Drastall, and R. L. Rivest, eds., *Computational Learning Theory and Natural Learning Systems*. Cambridge, MA: MIT Press, 445–456.
- Rich, E., and Knight, K. (1991). *Artificial Intelligence*, 2nd ed. New York: McGraw-Hill.
- Roger, R. J. (2005). Teaching an Introductory Course in Data Mining. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ACM Special Interest Group on Computer Science Education, Universidade Nova de Lisboa.

- Rowley, J. (2007). The Wisdom Hierarchy: Representations of the DIKW Hierarchy. *Journal of Information and Communication Science*, 33, 2, 163–180.
- Salkind, N. J. (2012). *Exploring Research*, 8th ed. Upper Saddle River, NJ: Pearson.
- Schmidt, C. W. (2012). Trending Now: Using Social Media to Predict and Track Disease Outbreaks. *Environmental Health Perspectives*, 120, 1, a30.
- Senator, T. E., Goldbert, H. G., Wooten, J., Cottini, M. A., Khan, A. F. U., Klinger, C. D., Llamas, W. M., Marrone, M. P., and Wong, R. W. H. (1995). The Financial Crimes Enforcement Network AI System (FAIS): Identifying Potential Money Laundering from Reports of Large Cash Transactions. *AI Magazine*, 16, 4, 21–39.
- Shafer, J., Agrawal, R., and Mehta, M. (1996). SPRINT: A Scalable Parallel Classifier for Data Mining. *VLDB Conference*, 544–555.
- Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, 41, 4, 256–275.
- Shavlik, J., Mooney, J., and Towell, G. (1990). Symbolic and Neural Learning Algorithms: An Experimental Comparison (Revised). Tech. Rept. No. 955, Computer Sciences Department, University of Wisconsin, Madison, WI.
- Shortliffe, E. H. (1976). *MYCIN: Computer-Based Medical Consultations*. New York: Elsevier.
- Signorini, A., Segre, A. M., and Polgreen, P. M. (2011). The Use of Twitter to Track Levels of Disease Activity and Public Concern in the US during the Influenza A H1N1 Pandemic. *PLoS One*, 6, 5, e19467.
- Spiliopoulou, M. (2000). Web Usage Mining for Web Site Evaluation. *Communications of the ACM*, 43, 8, 127–134.
- Sycara, K. P. (1998). The Many Faces of Agents. *AI Magazine*, 19, 2, 11–12.
- Thuraisingham, B. (2003). *Web Data Mining and Applications in Business Intelligence and Counter-Terrorism*. Boca Raton, FL: CRC Press.
- Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59, 433–460.
- Vafaie, H., and DeJong, K. (1992). Genetic Algorithms as a Tool for Feature Selection in Machine Learning. In *Proc. International Conference on Tools with Artificial Intelligence*. Arlington, VA: IEEE Computer Society Press, 200–205.
- Vapnik, V. (1998). *Statistical Learning Theory*. New York: John Wiley & Sons.
- Vapnik, V. (1999). *The Nature of Statistical Learning Theory*, 2nd edition. New York: Springer.
- Weiss, S. M., and Indurkhya, N. (1998). *Predictive Data Mining: A Practical Guide*. San Francisco: Morgan Kaufmann.
- Widrow, B., and Lehr, M. A. (1995). Perceptrons, Adalines, and Backpropagation. In M. A. Arbib, ed., *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press, 719–724.
- Widrow, B., Rumelhart, D. E., and Lehr, M. A. (1994). Neural Networks: Applications in Industry, Business and Science. *Communications of the ACM*, 37, 3, 93–105.
- Wilson, C., Boe, B., Sala, A., Puttaswamy, K. P., and Zhao, B. Y. (2009). User Interactions in Social Networks and Their Implications. In *Proceedings of the 4th ACM European Conference on Computer Systems*. New York: ACM, 205–218.
- Winston, P. H. (1992). *Artificial Intelligence*, 3rd ed. Reading, MA: Addison-Wesley.
- Witten, I. H., Frank, E., and Hall, M. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco: Morgan Kaufmann.
- Wu, X., and Kumar, V. (2009). *The Top 10 Algorithms in Data Mining*. Boca Raton, FL: Chapman & Hall/CRC.
- Zadeh, L. (1965). Fuzzy Sets. *Information and Control*, 8, 3, 338–353.
- Zikopoulos, P. C., Eaton, C., deRoos, D., Deutsch, T., and Lapis, G. (2012). *Understanding Big Data-Analytics for Enterprise Class Hadoop and Streaming Data*. New York: McGraw-Hill.

Index

Page numbers followed by f and t indicate figures and tables, respectively.

A

- Absolute rarity, 406
- Access to data
 - data warehouse, 20–21
 - distributed data access, 21
 - flat files, 21
 - relational databases, 21
- Accuracy
 - defined, 44
 - of model, performance evaluation and, 51
 - of model, technique selection and, 97
- Acme credit card database, 202, 203f, 429
- Actual customer value
 - vs. intrinsic value, 26–27, 26f
- ACZ. *see* Attribute and Cluster Analyzer (ACZ)
- AdaBoost operator, RapidMiner's, 414–415, 415f–417f
- Adaptive websites, 396
- Affinity analysis, 80
- Agglomerative clustering, 236, 358–360
 - applications, 360
 - credit card promotion database (example), 358–360, 358t, 359t
 - general considerations, 360
 - steps of, 358
- Aggregate operator, RapidMiner, 185
- ANOVA, one-way, 234
- ANOVA Matrix operator, 242
- ANOVA operator, RapidMiner, 222, 238–240, 239f–241f, 414, 417f
- Apple Computer, 379
- Apply Model operator, RapidMiner, 161–162, 164f, 167, 168, 172, 178, 244, 297, 345, 384
- Apriori association rule algorithm, 47, 82, 127, 181, 405
 - optimizations of, 405
 - parameters for, 129f
- ArffViewer, Weka, 453
- Artificial intelligence, 5
- Association rule(s)/association rule learning, 47–48, 96, 105, 393
- Apriori algorithm, 47, 82, 127, 181, 405
 - confidence and support, 80–81
 - credit card promotion database subset, 82–83, 82t–83t
 - example, 84–85
 - general considerations, 84–85
 - item sets, 82
 - limitations, 48
 - for market basket analysis, 47
 - mining (example), 82–84, 82t–83t
 - output attributes, 47, 48
 - in RapidMiner, 181–185
 - Aggregate* operator, 185
 - for credit card promotion database, 182–183, 182f–183f, 184f
 - Fp-Growth* operator, 181, 182, 185
 - interface for listing, 184f
 - market basket analysis template, 183, 184, 185f–186f
 - Pivot* operator, 185, 186f
 - single-item sets, 82, 83t
 - three-item sets, 84
 - two-item sets, 82–84, 83t
 - web-based data mining, 393–394
 - in Weka, 127–131
 - Apriori algorithm parameters for, 129f
 - for contact-lenses data set, 127–128, 128f
 - GenericObjectEditor, 128, 129f
 - LowerBoundMinSupport, 128, 131
 - supermarket data set, 128–129, 130f–131f
- Attribute and Cluster Analyzer (ACZ), 75, 79, 453
- Attribute filtering methods, 210–211
- Attribute filters, Weka, 111, 112f
- Attribute location, concept hierarchy for, 436, 436f
- Attribute-Relation File Format (ARFF), Weka, 106, 109, 117, 452f, 453
- Attribute(s); *see also specific attributes*

- creating new, 212–213
 defined, 202
 determination of, 96
 evaluation, 223
 for mixed data types, 241–243, 242f–243f
 gamma-ray burst data set, 187
 incorrect values, 204
 input, 8, 34
 missing value, 207
 output, 8, 34
Attribute selection, 209–213
 algorithm for, 209–213
 CORREL function (MS Excel), 211
 filtering methods, 210–211
 genetic learning for, 211–212, 212t
 and nearest neighbor classification, in
 RapidMiner, 191–194
 benchmark performance, 191, 192f
 forward selection subprocess for, 193f,
 210–211
 process design for, 192f
 subprocess for, 193f
 and nearest neighbor classification, in Weka,
 122–127
 attribute selection filter, 123, 124f
 IBk classifier, 122, 125
 InfoGainAttributeEval, 123, 124f
 numToSelect, 124–125, 126f
 output for spam data set, 123, 123f
 rankers, 123–124, 125f
principal component analysis, 211
Remove Correlated Attributes operator
 (RapidMiner), 211
scatterplot diagrams, 211
self-organizing map (SOM), 211
 wrapper techniques, 210–211
AttributeSelection filter, 210
@attribute symbol, Weka, 108
Attribute-value format, 8, 9t
Attribute-value predictability, 75, 76, 76f, 77, 361
Attribute-value predictiveness, 76–77, 77f, 361
@attribute XOR , 280
@attribute XOR real, 280
Average member technique, 261
-
- B**
- Backpropagation learning
 data sets for, 272–274
 exclusive OR (XOR) function, 272–273,
 272f–273f, 272t; *see also* Exclusive OR
 (XOR) function
 satellite image data set, 273–274, 274f
-
- C**
- CardiologyMixed*, 36
CardiologyNumerical, 36
Cardiology patient data set, 36, 37–38, 37t, 38t
 actual and predicted output for, 343f
 MMH for, 342f

- mySVM application, 341, 341f
- normalized, 342f
- performance vector for, 343f
- CART. *see* Classification and regression tree algorithm (CART)
- Categorical output
 - target data set, time-series analysis, 382, 383f
 - XOR function modeling, neural network building with Weka and, 280–282, 280f–282f
- Category utility measurement, conceptual clustering, 361–362
- C4.5 decision tree model, 64, 70–71, 73, 74
 - attribute selection, 65–67
- Chi-square automatic interaction detection (CHAID) algorithm, 74
 - vs.* classification and regression tree algorithm (CART), 74
- Central limit theorem, 227
- CHAID (Chi-square automatic interaction detection) algorithm, 74
- Chervonenkis, Alexey, 324
- Chi-square automatic interaction detection (CHAID) algorithm, 74
- Churning, 26–27
- Class attribute, 8
- Classical view, concepts, 6–7
- Classification accuracy, 97
- Classification and regression tree algorithm (CART), 73–74, 451
 - vs.* C4.5 decision tree model, 74
- Classification correctness, performance evaluation, 50–51, 53
- Classification model, 34–35
- Classifier error rate, 230, 236, 458
- Classifier(s); *see also specific entries*
 - Bayes classifier, 105
 - IBk nearest neighbor classifier, 122
 - instance-based, 213
 - K-nearest neighbor classifier, 19
 - lazy classifiers, Weka, 122
 - output options, in Weka, 115, 116f
- Class imbalance, 405, 406z
- Classit, 364, 405
- Cloud computing, 5, 24–25
- Clustering techniques, 48–49, 96, 105, 393
 - agglomerative clustering, 236
 - K-means algorithm, 85–90
 - applications, 88, 89t
 - clusters size and, 89
 - coordinate mapping of data, 86f
 - example, 86–88
 - general considerations, 89–90
 - input values, 86t
 - irrelevant attributes, 90
 - limitations, 89–90
 - number of clusters and, 89
 - optimal clustering for, 88
 - real-valued data and, 89
 - unsupervised, 11–13, 12t, 48–49; *see also* Unsupervised clustering techniques *vs.* supervised learning, 11–12
- Cobweb, 361, 362, 364, 405
- Cobweb-created hierarchy, 362, 363f
- Column vector, 332
- Commodity index fund, 379
- Community data sets, 454
- Compare ROC operator, RapidMiner, 407
- Computers, 6–13
 - learning. *see* Learning
- Computer science, 4
- Concept hierarchy(ies), 360–361, 362, 364
 - for attribute location, 436, 436f
 - defined, 436
 - OLAP system, 436, 436f
- Concepts, learning, 6
- Conceptual clustering, 360–364
 - category utility measurement, 361–362
 - Classit, 364
 - Cobweb, 361, 362, 364
 - example, 362–364, 363f, 364t
 - general considerations, 364
 - limitations, 364
- Concept views, 6–8
 - classical view, 6–7
 - exemplar view, 7–8
 - probabilistic view, 7
- Conditional probability, 318
- Confidence, association rules, 80–81
- Confidence intervals
 - for numeric output, 458–459
 - test set error rate computation, 230–231
- Confusion matrix, 50–51, 51t, 52, 52t
 - ideal model, 54–55, 54t, 55t
 - no model, 54, 54t, 55t
 - for null hypothesis, 229, 229t
 - for satellite image data set, 284f
 - supervised learner models evaluation, 50–51, 51t, 53
- three-class, 50–51, 51t
- two-class, 52, 52t
 - for XOR function, 281f, 282f
- Constellation schema, 432, 433f; *see also* Star schema
- Contact-lenses file/data set, Weka, 108, 109–110, 109f, 110f
- association rules for, 127–128, 128f
- decision trees for, 113–117, 113f–116f

- Contrarians, 378
 Control group, 228
 Conversion, data, 209
 Cookie, defined, 392–393
 Correlation coefficient, 22
Correlation Matrix operator, 304
CORREL function (MS Excel), 211
 Cost/benefit analysis
 in Weka, 131–136 132f–136f
 Covering rule algorithm, 42, 74–80
 Attribute and Cluster Analyzer (ACZ), 75, 79
 attribute-value predictability, 75, 76, 76f, 77
 attribute-value predictiveness, 76–77, 77f
 incremental reduced error pruning, 80
 Java Archive (JAR) file, 75
 predictability score, 75, 76, 77
 predictability values, 75–76, 76f
Crawl Web operator, 398
Create-Association-Rules operator, 182
Create Lift Chart operator, RapidMiner, 222,
 244–245, 244f, 247
CreditCardPromotion, 40
 Credit card promotion database, 40, 41–42, 41t
 agglomerative clustering application to, 358–360,
 358t, 359t
 association rules, 82–83, 82t–83t
 in RapidMiner, 181–185
 credit card insurance, 69–70, 72
 decision trees for, 70–73, 70f–71f, 72t
 in RapidMiner, 156–157, 157f–158f
 three-node decision tree, 71–72, 71f
 two-node decision tree, 71f, 72
 life insurance promotion, 42–43, 44, 45–46, 46t,
 67–68, 72
 OLAP application (example), 435–438, 436f,
 437f
 dice operation, 437
 drill-down, 437
 roll-up/aggregation, 437, 437f
 rotation/pivoting, 438
 slice operation, 436
 unsupervised clustering of, 48–49, 49f
 Credit card purchases, star schema for, 429–432,
 429f
 Credit card screening data set, 129
 cost/benefit output for, 132, 133f
 J48 classification of, 132, 132f
 K-means algorithm application, 137, 137f
 CRISP-DM process model, 215–216
 Cross Industry Standard Process for Data Mining
 (CRISP-DM), 199
 Crossover, genetic algorithms, 91, 94, 94f, 95
 Cross-validation, 231
 decision trees building in RapidMiner, 168–173,
 170f–171f
 in performance evaluation, 51–52
 Customer churn, 26–27
 prediction, *Neural Net* operator (RapidMiner),
 306–311
 cross-validation subprocess, 308f
 data preparation, 307, 307f
 goal identification, 307
 network architecture, 307, 308f
 network training, 307
 performance vector, 309f
 preprocessing, 308f
 results interpretation and reading, 308–311,
 309f–310f
 Customer churn data set
 in RapidMiner, 159–160, 160f, 163f
 decision tree for, 164f
 naïve Bayes operator, 325–326, 325f–327f
 partitioning, 162f
 performance vector for, 165f
 removing instances of unknown outcome, 161f
 Customer reviews analysis, textual data mining
 and, 399–400
 classification of written reviews, 400
 feature vector, 399, 400
 with RapidMiner, 400–404, 401f–404f
 rating scale scores, 399
 satisfaction scores, 399
 singular value decomposition (SVD), 400
 tokenization, 399
 word stemming, 399–400
 written evaluation, 399
 Customer satisfaction scores, 399
-
- D**
- Darwinian principle of natural selection, 90
 Data acquisition, 20–21
 Data analytics/data science, 432
 defined, 4
 Excel pivot tables for, 438–443, 439f–445f
 process, 4–5
 process model, 19–24, 20f
 data acquisition, 20–21
 data mining, 23
 data preprocessing, 21–22, 22f–23f
 result application, 24
 result interpretation, 23
 vs. KDD, 5–6
 Database management systems (DBMSs), 202
 Database query language, 21
 Data distribution, 96

- Data entering, into warehouse, 427–429, 428f
- Data marts
- dependent, 434
 - independent, 427–428
- Data mining, 4; *see also* Knowledge discovery in databases (KDD); Textual data mining algorithms, 5
- data sets, 453–454
 - defined, 5
 - ethics, 25
 - goal of, 215
 - in knowledge discovery, 201, 214
 - knowledge from, 5
 - knowledge types in, 14
 - learning. *see* Learning manual, 16
 - market analysis, 24
 - for personalization, 395–396, 395f–396f
 - as problem-solving strategy, decisions about, 14–16
 - process model, 19–24, 20f
 - data acquisition, 20–21
 - data preprocessing, 21–22, 22f–23f
 - result application, 24
 - result interpretation, 23 - software for, 451–454; *see also* RapidMiner; Weka techniques, 5
 - time-series analysis
 - RapidMiner, 382–387, 383f–386f
 - Weka, 387–390, 387f–389f - vs.* data query (example), 15–16
 - vs.* expert systems, 17–18
 - web-based. *see* Web-based data mining
 - for website adaptation, 396
 - for website evaluation, 395
- Data mining strategies, 34–40
- association rules, 47–48
 - classification, 34–35
 - clustering techniques, 48–49
 - estimation, 35–36
 - hierarchy of, 34f
 - market basket analysis, 40
 - performance evaluation, 49–56
 - prediction, 36–39, 37t, 38t
 - supervised learning strategies, 34
 - classification, 34–35
 - estimation, 35–36
 - prediction, 36–39, 37t, 38t - unsupervised clustering, 39–40
- Data mining techniques
- selection of, 95–97
 - supervised, 41
 - credit card promotion database, 41–42
 - neural networks, 44–46, 45f, 46t
- rule-based techniques, 42–44
- statistical regression, 46–47
- Data mining tools, 23
- Data model, 424–425
- Data normalization, 208–209
- operational databases, 424–425
- Data preprocessing, 21–22, 22f–23f, 404
- in knowledge discovery, 201
 - missing data, 207–208
 - noisy data, 203–207
 - time-series analysis, 380–382, 381t, 382f
- Data query, 14–15
- vs.* data mining (example), 15–16
- Data redundancy, 202, 432
- Data reporting, 432
- Data science. *see* Data analytics/data science
- Data sets, for data mining, 453–454
- community, 454
 - IBM Watson Analytics, 454
 - KDnuggets, 454
 - locations, 453
 - Machine Learning Repository, UCI, 454
 - for RapidMiner, 454
 - for Weka, 453
- DatasetsRapidMiner.zip.*, 454
- DatasetsWeka.zip.*, 453
- Data smoothing, 204–205
- external, 204
 - internal, 204
 - outliers and, 204–205
- Data stream, 412
- Data stream mining, 412–413
- @data symbol, Weka, 109
- Data transformation, 428
- in knowledge discovery, 201
 - attribute and instance selection, 209–213
 - data normalization, 208–209
 - data type conversion, 209
 - time-series analysis, 380–382, 381t, 382f
- Data warehouse, 20–21, 202–203, 423–445; *see also*
- Independent data mart
 - defined, 427
 - dependent data marts, 434
 - design, 426–434
 - data entering, 427–429, 428f
 - decision support, 432–434, 433f
 - structuring (star schema), 429–432, 429f
 - ETL process, 428
 - Excel pivot tables for data analytics, 438–443, 439f–445f
 - existing record, 428
 - metadata, 428–429
 - new record, 428

- online analytical processing (OLAP), 434–438
 example, 435–438, 435f–437f
 general considerations, 438
- operational databases, 424–426
 data modeling, 424–425
 data normalization, 424–425
 entity-relationship diagram, 424–425, 424f
 relational model, 425–426, 425t–426t
- overview, 423
- process model, 428f
- sources of data, 427
- vs. OLTP database, 427
- Data warehousing, 4
- DBMSs (database management systems), 202
- Decimal scaling, 208
- Decision support system
 categories, 432
 data analytics, 432
 data reporting, 432
 data warehouse for, 432, 434
 dependent data marts, 434
 online analytical processing, 432
- Decision Tree* operator, RapidMiner, 156, 157, 158, 159, 161, 172, 173, 188
- Decision tree(s), 18, 41, 96, 97, 105, 209, 210, 412
 advantages, 9–10, 74
 algorithm for building, 64–70
 C4.5 for attribute selection, 65–67
 credit card promotion database, 67t, 68
 gain ratio, 65–67
 steps, 64
 training data, 67t, 68
 attributes, 9–10
 Classification and regression tree algorithm (CART), 73–74
 C4.5 model, 64, 70–71, 73, 74
 attribute selection, 65–67
 for credit card promotion database, 70–73, 70f–71f, 72t
 three-node decision tree, 71–72, 71f
 two-node decision tree, 71f, 72
- defined, 9
- gamma-ray burst clustering, 190f
- issues-related, 74
- Iterative Dichotomiser 3 (ID3), 73
- other methods, 73–74
- overview, 64
- partial, 68–69, 68f, 69f
- production rules, 10–11
- in RapidMiner
 building, 159–173
 for credit card promotion database, 156–157, 157f–158f
- cross-validation scenario, 168–173, 170f–171f
 for customer churn data set, 164f
- descriptive form, 158f
- final model, creating and saving, 167–168, 168f, 169f
- Subprocess* operator, 165–166, 166f–167f
- training and test set scenario, 160–165, 160f–165f
- rules, 73
- supervised learning (example), 9–11, 10f, 10t
- training data, 11, 67t, 68, 72–73, 72t
 in Weka
 actual and predicted output, 115, 116f
 building, 109–117, 110f–116f
 classifier output options, 115, 116f
 for contact-lenses data set, 113–117, 113f–116f
 output, 114–115, 115f
 tree visualizer, 114f
- Deep knowledge, 14
- Delta rule, 264
- de Moivre, Abraham, 225
- Dependent data marts, 434
- Dependent variables, 34, 39; *see also* Output attribute(s)
- Detect Outlier (Distances)* operator, 205, 206
- Detrano, Robert, Dr., 36
- Dimension keys, fact table (star schema), 430
- Dimensions, fact table (star schema), 430
- Dimension tables, star schema, 429f, 430
- Discretize by Binning* operator, RapidMiner, 176, 177f
- Discretize by Frequency* operator, RapidMiner, 182
- Distributed data access, 21
- Distributed data mining, 5
- Distributed processing, 4
- Doe, John, 430
- Dot product, 332, 334
- Duplicate records, 204
-
- E
- E-commerce, 391, 396
- Economic indicators, 427
- Edit, transform, and load (ETL) process, 183
- Edit Parameter List*, 296, 296f–297f
- Education, 17
- EM algorithm. *see* Expectation maximization (EM) algorithm
- EM clustering model, Weka, 369–371, 370f–371f
- Emulate, defined, 17
- Entity(ies), 424
 intersection, 425
 relationships between, 424

- Entity-relationship diagram (ERD), 424
 operational databases, 424–425, 424f
- Epochs, 266
- ERD. *see* Entity-relationship diagram (ERD)
- Error rates, 52
 confidence intervals, computation, 230
 standard error and, 230
 test set, computation, 230–231
- Error(s)
 in data, 204
 defined, 264
 type 1, 229
 type 2, 229
- Estimation model, 35–36
 output attributes of, 35
- Ethics, data mining, 25
- ETL (extract, transform, and load) process, 428
- Euclidean distance, 19, 85, 86, 122, 332, 360, 372, 373
- Evaluation; *see also* Formal evaluation methods;
 Performance evaluation
 attributes, 223
 for mixed data types, 241–243, 242f–243f
 components, 222–223, 222f
K-means algorithm for, 235
 model builder, 223
 parameters, 223
 statistical findings and, 223–224
 supervised learner models, 222–223, 222f
 classifier error rate, 230
 comparison of, 233–234
 with numeric output, 236–238, 237t
 for unsupervised clustering, 235
 unsupervised clustering for, 235
 validation data, 230
 test set evaluation, 223
 tools for, 223–230
 hypothesis testing, classical model, 228–230,
 229t
 normal distribution, 225–226, 225f
 normal distributions and sample means,
 226–228, 226f
 single-valued summary statistics, 224–225
- training data, 223
- unsupervised clustering
 additional methods for, 236
 external, 236
 internal, 236
 supervised evaluation, 235
- website, data mining for, 395
- written, 399
- Evaluation and interpretation; *see also* Formal
 evaluation methods; Performance
 evaluation
 knowledge discovery
 experimental analysis, 214
 heuristic analysis, 214
 human analysis, 214
 in knowledge discovery, 201, 214–215
 purpose of, 214
 statistical analysis, 214
- Excel pivot tables, for data analytics, 438–443,
 439f–445f
- Excel spreadsheet, 4, 6
- Exchange traded fund (ETF), 379
- Exclusive OR (XOR) function, 272–273
 confusion matrix for, 281f, 282f
 defined, 272, 272t
 graphical interpretation, 272, 272f
 GUIs for, 275f
 modeling, *Neural Net* operator (RapidMiner),
 294–301, 295f–300f
 data preparation, 294
Edit Parameter List, 296, 296f–297f
 goal identification, 294
hidden layers parameter, 296, 296f
 network architecture, defining, 294–297,
 295f–297f, 298f
 network training, 297–298, 298f, 299f
Performance operator, 297, 298f
 results interpretation and reading, 299–301,
 299f–300f
shuffle box parameter, 297
training cycles parameter, 296
- modeling (categorical output), neural network
 building with Weka, 280–282, 280f–282f
- modeling (numeric output), neural network
 building with Weka, 272–279
 data preparation, 275, 275f
 goal identification, 274
 GUI parameter, 276
hiddenLayers parameter, 276–277
 network architecture, defining, 275–277, 276f,
 278f
 network training, 277, 278f
nominalToBinaryFilter, 277
 output attributes, 275
 results interpretation and reading, 277–279,
 278f–279f
seed parameter, 277
trainingTime parameter, 277
 statistics for, 295
 training data for, 273, 273f
- Exemplar view, concepts, 7–8
- Existing record, 428
- Expectation maximization (EM) algorithm,
 364–371, 412

- general considerations, 365, 371
 implementations, 365
 procedure, 365
 RapidMiner tutorial, 366–369, 366f–369f
 Weka tutorial, 369–371, 370f–371f
- Experience, 17
- Experimental analysis, 214
- Experimental group, 228
- Experimenter interface, Weka, 106
- Experts, 16–17
- Expert stock market analyst, lack of, 378
- Expert systems
 defined, 17
 vs. data mining, 17–18
- Explorer interface, Weka, 105–106, 107f
 navigating, 111f
- Extended common log file format, 392
- External data, 427–428
 smoothing, 204
- External evaluation; *see also* Evaluation
 unsupervised clustering, 236
- Extract, transform, and load (ETL) process, 428
-
- F
- Facebook, 24, 379
- Facts, learning, 6
- Fact table, star schema, 430–432, 431f
 dimension keys, 430, 432
 dimensions, 430, 431f
 dimension tables, 429f, 430
 granularity of, 431
- False accept, 52
- False-positive (FP) rate, ROC curve, 406, 407, 408, 409, 410f
- False reject, 52
- Feature vector, 399, 400
- Feed-forward neural networks, 44, 45f, 253, 254–258, 254f, 412; *see also* Neural network(s)
 bias node, 258
 building, with Weka, 271–288, 274
 backpropagation learning, data sets for, 272–274
 exclusive-OR function, 272–273, 272f–273f, 274–282; *see also* Exclusive OR (XOR) function
 MultiLayerPerceptron (MLP) function, 271, 274, 275, 276, 280
 overview, 271–272
 satellite image data set, 273–274, 274f, 282–287
 unsupervised neural net clustering, 287–288, 287f–288f
- fully connected, 254
- input format, 254–255
 output format, 255–256
 sigmoid evaluation function, 256–258, 257f
 supervised learning with, 258–259
 backpropagation learning, 258–259, 263–266
 genetic learning, 259, 259t
 threshold nodes, 258
- Filter Examples* operator, RapidMiner, 161, 162, 163f, 168, 307, 311, 384
- Filtering methods
 attribute selection, 210–211
- First normal form (1NF), 425
- First process model, RapidMiner, 149–156, 150f–156f
- Fitness function, supervised genetic learning, 91, 91f, 92–93, 93t, 95
- Fitness score, supervised genetic learning, 93–94
- Flat files, 21, 202
- Formal evaluation methods, 221–247; *see also*
 Evaluation; Evaluation and interpretation;
 Performance evaluation
 attribute evaluation for mixed data types, 241–243, 242f–243f
 components, 222–223, 222f
 overview, 221–222
- Pareto lift charts, 244–247, 244f–247f
- RapidMiner, comparing models with, 238–240, 239f–241f
- supervised learner models, comparing, 232–234
- supervised models with numeric output, 236–238, 237t
- test set confidence intervals, computing, 230–231
- tools, 223–230
 hypothesis testing, classical model, 228–230, 229t
 normal distribution, 225–226, 225f
 normal distributions and sample means, 226–228, 226f
 single-valued summary statistics, 224–225
- unsupervised clustering
 additional methods, 236
 for supervised evaluation, 235
 supervised evaluation for, 235
- Forward Selection* operator, RapidMiner, 191
- Four V's of big data, 24
- FP-Growth algorithm, 405
- Fp-Growth* operator, RapidMiner, 181, 182, 185
- Frequency, 224
-
- G
- Gain ratio, 65–67
- Gamma-ray burst data set, 187–191, 188f–190f
 attributes, 187

- decision tree, 190f
 linear regression, 346f–348f
 partial clustering of, 189f
 process design for, 188f
 RapidMiner's EM operator application, 366–369,
 366f–369f
 three clusters of, 189f
 Gaussian curve. *see* Normal distribution
 Gaussian mixtures model, 364
 Generic model tree, 349f
GenericObjectEditor, Weka, 112, 113f, 128, 129f, 137,
 276, 287, 415
 Genetic algorithms/genetic learning, 90–95, 97
 applications, 90, 95
 for attribute selection, 211–212, 212t
 background, 90
 crossover, 91
 general considerations, 95
 mutation, 91
 neural network training, 259, 259t
 problem-solving approach based on, 95
 selection operator, 91
 supervised learning, 91–95, 91f
 crossover operation, 94, 94f, 95
 fitness function, 91, 91f, 92–93, 93t
 fitness score, 93–94
 mutation, 94–95
 population elements, 91–93, 91f, 92t, 93t
 process, 91f
 question mark, 92
 second-generation population, 94, 94t
 selection operator, 92
 training data, 93t
 unsupervised clustering and, 371–374, 372f, 373t
- Gennari, John, Dr., 36
- GNU General Public License, 105, 451
- Goal identification
 customer churn prediction, RapidMiner, 307
 in knowledge discovery, 200, 201–202
 satellite image data set mining
 Neural Net operator (RapidMiner), 301
 Weka, 282
- Web-based data mining, 391–392
- XOR function modeling
 Neural Net operator (RapidMiner), 294
 numeric output, with Weka, 274
- Good credit risk
 classical-view definition, 6–7
 exemplar-view definition, 7–8
 probabilistic-view definition, 7
- Google, 379, 396, 397
- Granularity, 427
 of fact table (star schema), 431
- Graphical user interfaces (GUI's)
 Weka, 105–106
 Experimenter interface, 106
 Explorer interface, 105–106, 107f
 Knowledge Flow interface, 106
 for XOR training, 275f
 Grb4u, 187, 366
 Ground truth, of satellite image, 273
 Grouped ANOVA operator, 242
 GUI parameter, 276, 280
-
- H
- Hadoop, 24, 25
 Hadoop Distributed File System (HDFS), 24, 203
- Heuristic analysis, 214
- Hidden knowledge, 14
- HiddenLayers* parameter, 276–277, 280, 281, 296,
 296f
- Hierarchical clustering model, 48
- Holland, John, 90
- Human analysis, 214
- Human capital management (HCM), 25
- Hyperplane, SVMs, 327, 330–331, 330f, 331f, 333
 maximum margin hyperplane (MMH), 330, 331
 support vectors, 330–331, 331f
- Hypothesis, 15–16
- Hypothesis testing, classical model for, 228–230,
 229t
 supervised learner models comparison, 232–234
-
- I
- IBk (Weka) nearest neighbor classifier, 122
- IBM Watson Analytics, 454
- IDC (International Data Corporation), 4
- Ideal model, confusion matrices, 54–55, 54t, 55t
- Imbalanced data, handling, 405–407
 absolute rarity, 406
 rare cases, 406
 rare class problem, 405–406
 rarity, methods for, 406
 receiver operating characteristics (ROC) curves,
 406–407, 407f
 relative rarity, 406
- Incorrect attribute values, 204
- Incremental learning, 360
- Incremental reduced error pruning, covering rule
 algorithm, 80
- Independent data mart, 427–428; *see also* Data
 warehouse
- Independent test data, supervised learner models
 comparison with, 456–457

- Independent variables, 34; *see also* Input attribute(s)
- IndexFinder, 396
- Index page, 396
- Index synthesis problem, 396
- Induction-based learning, 5
- Inference problem, 25
- Information theory, 64
- Inmon, W. H., 427
- Input attribute(s), 8, 34
- Input format
- feed-forward neural networks, 254–255
- Input layer, Kohonen networks, 259
- Installation instructions
- RapidMiner, 146, 453
 - Weka, 106–109, 107f–109f, 451–453, 452f
- Install folder, Weka, 108f
- Instance, of data, 8, 9
- Instance-based classifiers, 213
- Instance selection, 213
- Interface
- RapidMiner, 146–149, 147f–148f
 - Weka
 - Experimenter interface, 106
 - Explorer interface, 105–106, 107f, 111f
 - GUI's, 105–106
 - Knowledge Flow interface, 106
- Internal data smoothing, 204
- Internal evaluation; *see also* Evaluation
- unsupervised clustering, 236
- International Data Corporation (IDC), 4
- Internet, 4, 24, 26, 397
- Interpretation and evaluation. *see* Evaluation and interpretation
- Intersection entity, 425
- Intrinsic modeling, 27
- Intrinsic value, 26–27
- vs.* actual customer value, 26–27, 26f
- Item sets, association rules, 82
- Iterative Dichotomiser 3 (ID3), 73
-
- J
- J48, Weka, 111–113, 112f–113f
- advantages, 122
 - classification of credit card screening data set, 132–136, 132f–136f
- Java Archive (JAR) file, 75
- Java programming language, 105, 451
-
- K
- K best rules parameter, 178, 180
- KDD. *see* Knowledge discovery in databases (KDD)
- KDnuggets, 454
- Kernel function, 336, 337
- K-means algorithm, 85–90, 261, 357, 360, 364, 372, 412
- applications, 88, 89t
 - clusters size and, 89
 - coordinate mapping of data, 86f
 - for evaluation, 235
 - example, 86–88
 - general considerations, 89–90
 - input values, 86t
 - irrelevant attributes, 90
 - limitations, 89–90
 - number of clusters and, 89
 - optimal clustering for, 88
 - real-valued data and, 89
 - SimpleKMeans, 137, 138f
 - unsupervised clustering with
 - gamma-ray burst data set, 187–191, 188f–190f
 - in RapidMiner, 187–191, 188f–190f
 - in Weka, 137–141, 137f–140f
- K-means operator, RapidMiner, 187
- K-nearest neighbor classifier, 19
- Knowledge, 254
- from data mining, 5
 - deep, 14
 - hidden, 14
 - multidimensional, 14
 - shallow, 14
 - types, in data mining, 14
- Knowledge discovery, 4, 5
- application, actions for, 201, 215
 - CRISP-DM process model, 215–216
 - data mining, 201, 214
 - data preprocessing, 201
 - missing data, 207–208
 - noisy data, 203–207
 - data transformation, 201
 - attribute and instance selection, 209–213
 - data normalization, 208–209
 - data type conversion, 209
 - goal identification, 200, 201–202
 - interpretation and evaluation, 201, 214–215
 - legal issues, 201
 - process model for, 199–201, 200f
 - process steps, 215
 - with RapidMiner. *see* RapidMiner
 - target data set, 200, 202–203, 203f
 - with Weka. *see* Weka
- Knowledge discovery in databases (KDD), 199;
- see also* Data mining
 - goal of, 5
 - process model, 199–201, 200f

time-series analysis, 379
 variations, 199
vs. data science, 5–6
 Web-based data mining, 391–392

Knowledge engineer, 16–18, 17f

Knowledge Flow interface, Weka, 106

Kohonen, Teuvo, 259

Kohonen clustering algorithm, 451

Kohonen networks, 259–260, 260f
 input layer, 259
 nodes, 259–260
 output layer, 259–260

Kohonen self-organizing maps (SOMs), 253, 259, 263
 example, 266–268, 266f

L

Laplace correction, 325, 329

Large-sized data, handling, 404–405

Lazy classifiers, Weka, 122

Learning
 concepts, 6
 concept views, 6–8
 facts, 6
 levels, 6
 principles, 6
 procedures, 6
 supervised, 8–11; *see also* Supervised learning
 unsupervised clustering, 11–13; *see also*
 Unsupervised clustering

Learning phase, neural networks, 45

Least-squares criterion, 344

Lift, measurement, models comparison by, 53–55, 54f, 54t–55t

Lift chart, 54, 54f

Linearly separable, 272, 273

Linearly separable classes
 SVM algorithm for, 332–336, 335f

Linear regression
 defined, 340, 350
 gamma-ray burst data set, 346f–348f
 multiple, 344–345
 RapidMiner, 345, 346f–348f
 simple, 344
 Weka, 344, 345f

Linear regression analysis/model, 46, 340–345
 transforming, 350–352

Link analysis
 web-based data mining, 396–398

Link mining algorithm, 396–397

Logarithmic normalization, 209

Logistic regression, 350–352

Logistic regression model, 351–352, 351f

Logit, 351

LowerBoundMinSupport, 128, 131

M

Machine learning, 4, 5

Machine Learning Repository, UCI, 454

Manual data mining, 16

Many-to-many relationship, 424, 425

Mapping function, 336–337

MapReduce, 24, 203

Market basket analysis, 40
 association rules for, 47
 in RapidMiner, 181, 183, 184, 185f–186f
 example, 84–85

Market Basket Analysis template, RapidMiner, 183

Marketing applications, 53

Market technicians, 378

Mass mailings, 53
vs. targeted mailing, 54f

Mathematical equations, 6

Mathematics, 4, 5

Maximum margin hyperplane (MMH), 330, 331
 for cardiology patient data set, 342f
 determining, 332–336, 335f

Mean, 224–225, 455

Mean absolute error (mae), 53, 236, 237, 237t, 281, 458
 computations for, 237t, 238

Mean squared error (mse), 53, 236–237, 237t
 computations for, 237t, 238

Mean value smoothing, 204

Metadata, 428–429
 operational, 429
 structural, 429

Meta-operator. *see Tree to Rules operator*

Microsoft Access, 425

Microsoft Corporation, 379

Microsoft (MS) Excel, 40

Microsoft (MS) Excel format
 RapidMiner files in, 146

Mining, data. *see Data mining*

Min–max normalization, 209

Missing data, 207–208
 naïve Bayes classifier, 321–322

Missing values, 207–208

Mixed data types, attributes evaluation for, 241–243, 242f–243f

Mixture, defined, 364

Mixtures model, 364

MLP function. *see MultiLayerPerceptron (MLP)* function

MMH. *see* Maximum margin hyperplane (MMH)
 Model accuracy, performance evaluation and, 51
 Model builder, evaluation, 223
 Model trees, 349, 349f; *see also* Regression trees
 Multidimensional data store, advantages, 434
 Multidimensionality, of star schema, 429f, 430–431, 431f
 Multidimensional knowledge, 14
 Multidimensional pivot table, 441–443, 442f–445f
 MultiLayerPerceptron (MLP) function, Weka, 271, 274, 275, 276, 280, 382, 387
 Multiple linear regression, 344–345
 RapidMiner, 345, 346f–348f
 Weka, 344, 345f
 Multiple-model approaches, 97, 413; *see also specific entries*
 Mutation, genetic algorithms, 91, 94–95
 MySVM, 341, 341f

N

Naïve Bayes classifier, 317–324, 405, 412
 advantages, 324
 Bayes' theorem, 318
 data for, 318t
 disadvantages, 324
 example, 318–321, 318t
 general considerations, 324
 implementations, 324
 missing data, 321–322
 numeric data, 322–324, 323t
 RapidMiner tutorial, 325–326, 325f–327f
 Weka tutorial, 328–329, 328f–329f
 zero-valued attribute counts, 321
 Naïve Bayes operator, RapidMiner's, 325–326, 325f–327f
 National Aeronautics and Space Administration's (NASA's) Compton Gamma-Ray Observatory, 187
 Natural selection, Darwinian principle of, 90
 Nearest neighbor classification method, 18–19, 96, 210
 and attribute selection, in RapidMiner, 191–194
 benchmark performance, 191, 192f
 forward selection subprocess for, 193f, 210–211
 process design for, 192f
 subprocess for, 193f
 and attribute selection, in Weka, 122–127
 attribute selection filter, 123, 124f
 IBk classifier, 122, 125
 InfoGainAttributeEval, 123, 124f
 numToSelect, 124–125, 126f
 output for spam data set, 123, 123f
 rankers, 123–124, 125f

modification of, 406
 problems related to, 122
 Negative correlation, 22, 23f
 Network architecture
 customer churn prediction, RapidMiner, 307, 307f
 satellite image data set
 RapidMiner, 301–303, 302f
 Weka, 283, 284f
 XOR function
 Neural Net parameters, RapidMiner, 294–297, 295f–297f, 298f
 Weka, 275–277, 276f, 278f
 Network convergence, 259
 Neural Net operator, RapidMiner, 293, 382, 383
 customer churn prediction, 306–311
 cross-validation subprocess, 308f
 data preparation, 307, 307f
 goal identification, 307
 network architecture, 307, 308f
 network training, 307
 performance vector, 309f
 preprocessing, 308f
 results interpretation and reading, 308–311, 309f–310f
 supervised neural net learning, satellite image
 data set mining, 301–306, 302f–306f
 data preparation, 301, 301f
 goal identification, 301
 network architecture, 301–303, 302f
 network training, 303, 304f
 results interpretation and reading, 303–306, 304f–306f
 supervised neural net learning, XOR modeling, 294–301, 295f–300f
 data preparation, 294
 goal identification, 294
 network architecture, defining, 294–297, 295f–297f, 298f
 network training, 297–298, 298f, 299f
 results interpretation and reading, 299–301, 299f–300f
 Neural network(s), 44–46, 45f, 46t, 96, 97, 204, 208, 210, 253–268
 actual and computed output values, 45–46, 46t
 advantages, 253, 254, 262
 backpropagation, 237
 building, with RapidMiner, 293–313
 customer churn data set, 306–311, 307f–310f
 overview, 293–294
 satellite image data mining, 301–306, 302f–306f
 SOM operator, 311–313, 312f
 XOR function modeling, 294–301, 295f–300f

- building, with Weka, 271–288
 backpropagation learning, data sets for, 272–274
 exclusive-OR function, 272–273, 272f–273f, 274–282; *see also* Exclusive OR (XOR) function
 MultiLayerPerceptron (MLP) function, 271, 274, 275, 276, 280
 overview, 271–272
 satellite image data set, 273–274, 274f, 282–287; *see also* Satellite image data set
SelfOrganizingMap algorithm, 271
 unsupervised neural net clustering, 287–288, 287f–288f
 disadvantage, 260–261, 262–263
 explanation, 260–261
 feed-forward, 44, 45f, 253, 254–258, 254f
 bias node, 258
 fully connected, 254
 input format, 254–255
 output format, 255–256
 sigmoid evaluation function, 256–258, 257f
 threshold nodes, 258
 general considerations, 262–263
 Kohonen networks, 259–260, 260f
 Kohonen self-organizing maps (SOMs), 253, 259, 263
 example, 266–268, 266f
 learning, 254
 learning phase, 45
 limitations, 46
 operation phase, 45
 overview, 253–254
 perceptron networks, 273
 performance, 262
 for supervised learning, 44
 training, 258
 backpropagation learning, 258–259, 263–266
 detailed view, 263–268
 genetic learning, 259, 259t
 Kohonen self-organizing maps, 266–268, 266f
 supervised learning with feed-forward networks, 258–259
 unsupervised clustering with self-organizing maps, 259–260, 260f
 for unsupervised clustering, 44
 Neurodes, 254
 New record, 428
 Nodes, Kohonen networks, 259–260
 Noise, defined, 203
 Noisy data, 97, 203–207
 data smoothing, 204–205
 duplicate records, 204
 incorrect attribute values, 204
 outliers detection, 205–207, 205f–207f
NominalToBinaryFilter, 277
NominalToBinary filter, Weka, 137
 No model, confusion matrices, 54, 54t, 55t
 Nonlinear data
 SVM algorithm for, 336–337
 Normal distribution, 225–226, 225f
 background, 225
 performance evaluation and, 456
 sample means and, 226–228, 226f
 Normalization, data, 208–209, 425
 Normalized cardiology patient data set, 342f
 Normal probability curve. *see* Normal distribution
 Null hypothesis, 228
 confusion matrix for, 229, 229t
 supervised learner models comparison, 232–234
 Numeric data
 naïve Bayes classifier, 322–324, 323t
 Numeric output
 confidence intervals for, 458–459
 evaluation, 53
 supervised learner models
 comparison, 459–460
 evaluation, 236–238, 237t
 target data set, time-series analysis, 380, 380f, 381f
 XOR function modeling, neural network
 building with Weka and, 272–279
 data preparation, 275, 275f
 goal identification, 274
 GUI parameter, 276
 hiddenLayers parameter, 276–277
 network architecture, defining, 275–277, 276f, 278f
 network training, 277, 278f
 nominalToBinaryFilter, 277
 output attributes, 275
 results interpretation and reading, 277–279, 278f–279f
 seed parameter, 277
 trainingTime parameter, 277
-
- O
- OLAP. *see* Online analytical processing (OLAP)
 OLTP. *see* Online transactional processing (OLTP)
 One-class learning, 406
 One-to-many relationship, 424
 One-to-one relationship, 424
 One-versus-all method, 337
 One-vs-one method, 337, 338
 One-way ANOVA, 234

- Online analytical processing (OLAP), 432, 434–438
 advantages, 434
 concept hierarchies, 436, 436f
 credit card application (example), 435–438, 436f, 437f
 dice operation, 437
 drill-down, 437
 roll-up/aggregation, 437, 437f
 rotation/pivoting, 438
 slice operation, 436
 features, 434–435
 general considerations, 438
 three-dimensional cube, 434, 435f
 user interface design, 434–435
- Online transactional processing (OLTP), 424
 database *vs.* data warehouse, 427
- Operational databases, 20–21, 424–426
 data modeling, 424–425
 data normalization, 424–425
 entity-relationship diagram, 424–425, 424f
 granularity, 427
 relational model, 425–426, 425t–426t
- Operational metadata, 429
- Operation phase, neural networks, 45
- Operators; *see also specific operators*
 for web-based data mining, 398
- Optimize Selection* operator, RapidMiner, 191
- Outliers, 40, 405
 data smoothing and, 204–205
 detection, 205–207, 205f–207f
Detect Outlier (Distances) operator, 205, 206
 RapidMiner tutorial, 205–207, 205f–207f
 unsupervised method, 205
- Output attribute(s), 8, 34, 35
 association rules, 47, 48
 estimation model, 35
 predictive model, 36
- Output format
 feed-forward neural networks, 255–256
- Output layer, Kohonen networks, 259–260
-
- P
- Page, Larry, 396
- PageRank algorithm, 377, 396–398, 397f
- Pageview, Web-based data mining, 392
- Parameters, model
 evaluation, 223
- Parameters* panel, RapidMiner, 160, 161
- Pareto lift charts, 244–247, 244f–247f
- PART, Weka, 117
 advantages of, 122
 production rules generation with, 117–122
- customer churn data, 117, 118f
 customer churn instances, loading, 120, 120f
 customer churn output, 119, 119f
 decision list, 117
 decision list for customer churn data, 117, 118f
 prediction probability, 120–121, 121f
 save result buffer, 121
- PART rule generator, Weka, 408
- Perceptron neural networks, 273
- Performance evaluation, 49–50; *see also Evaluation; Evaluation and interpretation; Formal evaluation methods*
 bootstrapping, 52
 confusion matrix, 50–51, 51t
 cross-validation, 51–52
 model accuracy, 51
 models comparison by lift measurement, 53–55, 54f, 54t–55t
 numeric output, 53
 questions, 49–50
 statistics for, 455–460
 confidence intervals for numeric output, 458–459
 normal distribution, 456
 numeric output, comparing models with, 459–460
 single-valued summary statistics, 455–456
 supervised learner models, comparing, 456–458
 supervised learner models, 50–52, 51t
 two-class error analysis, 52–53, 52t
 unsupervised model, 55–56
- Performance improvement, ensemble techniques for, 413–415
- AdaBoost operator, RapidMiner's, 414–415, 415f–417f
 bagging, 413
 boosting, 413–414
- Performance* operator, 297, 298f
 RapidMiner, 162f, 167, 172, 178
- Personalization, data mining for, 395–396, 395f–396f
- Pivot* operator, RapidMiner, 185, 186f
- Pivot tables, 438
 blank, 440f
 MS Excel, for data analytics, 438–443, 439f–445f
 multidimensional, 441–443, 442f–445f
- Platt, John, 340
- Polynomial kernel function, 337, 341
- Population elements, supervised genetic learning, 91–93, 91f, 92t, 93t

Population variance, 227
 Positive correlation, 22, 22f
 Predictability
 attribute-value, 75, 76, 76f, 77, 361
 defined, 75–76
 vs. predictiveness, 77
 Predictability score, 75, 76, 77, 211
 Prediction model(s), 35, 36–39
 cardiology patient data set, 36, 37–38, 37t, 38t
 numeric attributes, 36–37, 37t
 output attribute(s) of, 36, 37t
 Predictive analytics, 4
 Predictiveness
 attribute-value, 76–77, 77f, 361
 vs. predictability, 77
PreProcess operator, RapidMiner, 170, 244
 Price to earnings (P/E) ratio, 212
 Principal component analysis
 attribute selection, 211
 Principles, learning, 6
 Priori probability, 318
 PRISM algorithm, 75
 Probabilistic view, concepts, 7
 Probability distribution, 224
 Problem-solving strategy
 data mining as, decisions about, 14–16
 Procedures, learning, 6
Process Documents From Files operator, 400, 401, 401f
 Process model
 data mining, 19–24, 20f
 data acquisition, 20–21
 data mining, 23
 data preprocessing, 21–22, 22f–23f
 result application, 24
 result interpretation, 23
 for knowledge discovery, 199–201, 200f
 Production rules, 6, 10, 47, 97, 105, 117
 decision tree, 10–11
 generation with PART (Weka), 117–122
 customer churn data, 117, 118f
 customer churn instances, loading, 120, 120f
 customer churn output, 119, 119f
 decision list, 117
 decision list for customer churn data, 117, 118f
 prediction probability, 120–121, 121f
 save result buffer, 121

Q

Qualitative data analysis, 399
 Quantitative data analysis, 399

R

Radoop, 25
 Rainforest algorithm, 405
 Rankers, Weka, 123–124, 125f, 210
RapidMiner, 25, 36, 37, 41, 42, 106, 145–194, 201;
 see also specific operators
 AdaBoost operator, 414–415, 415f–417f
 ANOVA operator, 222, 238–240, 239f–241f
 Apply Model operator, 161–162, 164f, 167, 168, 172, 178, 244, 297, 345, 384
 association rule learning, 181–185
 Aggregate operator, 185
 for credit card promotion database, 182–183, 182f–183f, 184f
 Fp-Growth operator, 181, 182, 185
 interface for listing, 184f
 market basket analysis template, 183, 184, 185f–186f
 Pivot operator, 185, 186f
 attribute selection and nearest neighbor classification, 191–194, 192f–194f
 Backward Elimination operator, 191
 bar graph, 155f
 breakpoints, 158–159, 159f
 Compare ROC operator, 407
 comparison of models with, 238–240, 239f–241f
 Create Lift Chart operator, 244–245, 244f, 247
 customer churn data set, 159–160, 160f, 163f
 decision tree for, 164f
 partitioning, 162f
 performance vector for, 165f
 removing instances of unknown outcome, 161f
 data files, 146
 data mining, time-series analysis, 382–387, 383f–386f
 data sets for, 454
 Decision Tree operator, 156, 157, 158, 159, 161, 172, 188
 decision trees
 building, 159–173
 for credit card promotion database, 156–157, 157f–158f
 cross-validation scenario, 168–173, 170f–171f
 for customer churn data set, 164f
 final model, creating and saving, 167–168, 168f, 169f
 Subprocess operator, 165–166, 166f–167f, 170
 training and test set scenario, 160–165, 160f–165f
 Detect Outlier (Distances) operator, 205, 206
 Discretize by Binning operator, 176, 177f

- Discretize by Frequency* operator, 182
EM operator, 366–369, 366f–369f
Filter Examples operator, 161, 162, 163f, 168, 307, 311, 384
 first process model, 149–156, 150f–156f
Forward Selection operator, 191
 FP-Growth algorithm, 405
 installation, 146, 453
 interface, navigating, 146–149, 147f–148f
 introduction to, 147f
K-means operator, 187
 linear regression model, 47, 345, 346f–348f
 naïve Bayes operator, 325–326, 325f–327f
Neural Net operator, 293
 neural networks building with, 293–313
 customer churn data set, 306–311, 307f–310f
 overview, 293–294
 satellite image data mining, 301–306, 302f–306f
 SOM operator, 311–313, 312f
 XOR function modeling, 294–301, 295f–300f
 neural network software packages, 46
 new blank process, 145–147, 146f
 operators, 145, 453; *see also specific operators*
Optimize Selection operator, 191
 outliers detection, 205–207, 205f–207f
 overview, 145–146, 453
 Parameters panel, 160, 161
Performance operator, 162f, 167, 172, 178
PreProcess operator, 170, 244
Remove Correlated Attributes operator, 211, 304, 305, 367
Retrieve operator, 154, 156, 160
 rules generation, 173–181
 Rule Induction operator, 173, 176–177, 176f–178f
 subgroup discovery, 178–181, 179f–181f
 Tree to Rules operator, 173, 174f–175f
 scatterplot charts, 156f
Select Attributes operator, 187
Self-Organizing Map (SOM) operator, 293
Set Role operator, 160, 161, 188
Split Data operator, 161, 178
 subgroup discovery operator, 42, 44
 SVM operators, 341, 341f–343f
 text mining customer reviews with, 400–404, 401f–404f
T-test operator, 222, 238–240, 239f–241f
 unsupervised clustering with *K-means*
 gamma-ray burst data set, 187–191, 188f–190f
Validation operator, 170, 172
 Web mining extension package, 398
 website, 146, 453
 wrapper approach, 194
Write Excel operator, 168, 188
Write Model operator, 167, 309
X-Prediction operator, 306
X-Validation operator, 170, 301–303, 306
 RapidMiner Studio, 145, 146
 RapidMiner Studio 7, 145, 453
 Rare cases, 406
 Rare class problem, 405–406
 Rarity, methods for, 406
 Rating scale scores, 399
 Receiver operating characteristics (ROC) curves, 406–407, 407f
 false-positive (FP) rate, 406, 407, 408, 409, 410f
 RapidMiner’s *Compare ROC* operator, 407
 true-positive (TP) rate, 406, 407, 408, 409, 410f
 Weka
 creation and analysis, 408–411, 408f–411f
 implementation of, 407
 Redundancy, data, 202
 Regression
 defined, 340
 logistic, 350–352
 Regression analysis, 74, 105
 linear. *see Linear regression analysis*
 Regression trees, 96, 349, 349f; *see also Decision tree(s)*
 Relational databases, 21, 202, 425–426, 425t–426t
 advantages, 434
 Relational model
 operational databases, 425–426, 425t–426t
 @ relation symbol, Weka, 108
 Relative rarity, 406
Remove Correlated Attributes operator, RapidMiner, 211, 304, 305, 367
 Repeated incremental pruning to produce error reduction (RIPPER) algorithm, 80, 173
 Reporting data, 432
 Result application, 24
 Result interpretation, 23
Retrieve operator, RapidMiner, 154, 156, 160
 Reviews, classification of, 400
 Reviews analysis. *see Customer reviews analysis*
 RIPPER (repeated incremental pruning to produce error reduction) algorithm, 80
 ROC. *see Receiver operating characteristics (ROC) curves*
 Root mean squared error (rms), 53, 236, 237, 266, 281
 computations for, 237t, 238
 Row vector, 332
 Rueping, Stefan, 341
 Rule accuracy, 44

Rule-based techniques, 42–44
 covering approach, 42
 subgroup discovery method, 42–43

Rule coverage, 13, 37, 38, 42, 43

Rule extraction methods, 261

Rule Induction operator, RapidMiner, 176–177, 176f–178f
 process design for, 176f

Rule precision, 13, 37, 38, 42, 43

Rules, decision tree, 73

Rules generation
 in RapidMiner, 173–181
Rule Induction operator, 173, 176–177, 176f–178f
 subgroup discovery, 178–181, 179f–181f
Tree to Rules operator, 173, 174f–175f

S

Sample data, 226, 227

Sample mean, 224, 458
 normal distribution and, 226–228, 226f

Sample variance, 224, 458

Sampling technique, 405

Satellite image data set, 273–274, 274f
 categories, 273
 confusion matrix for, 284f
 ground truth, 273
 mining, neural networks building with
 RapidMiner, 301–306, 302f–306f
 data preparation, 301, 301f
 goal identification, 301
 network architecture, 301–303, 302f
 network training, 303, 304f
 results interpretation and reading, 303–306,
 304f–306f

mining, neural networks building with Weka,
 282–287
 data preparation, 282–283
 goal identification, 282
 network architecture, defining, 283, 284f
 network training, 283
 results interpretation and reading, 283–285,
 284f

Satisfaction scores, 399

Scalability, 405

Scalable techniques, 405

Scalable unsupervised clustering techniques, 405

Scatterplot diagram, 22, 22f–23f
 attribute selection, 211
 in RapidMiner, 156f

SE. *see* Standard error (SE)

Second-generation population, supervised genetic learning, 94, 94t

Second normal form (2NF), 425

Seed parameter, 277

Select Attributes operator, RapidMiner, 187

Selection operator, genetic algorithms, 91, 92

SelfOrganizingMap algorithm, Weka, 271, 287–288,
 287f–288f, 451

Self-Organizing Map (SOM) operator, RapidMiner,
 293, 311–313, 312f

Self-organizing maps (SOMs), 211
 attribute selection, 211
 Kohonen, 253, 259, 263
 example, 266–268, 266f
 unsupervised clustering with, 259–260, 260f

Sensitivity analysis, 261

Sequence miners, 395

Sequential minimum optimization algorithm
 (SMO), Weka, 340

SVMs implementations, 338–340, 338f–340f, 340

Session, Web-based data mining, 392

Session file, Web-based data mining, 392, 392f

Set Role operator, RapidMiner, 160, 161, 188

Shallow knowledge, 14

Shuffle box parameter, 297

Sigmoid evaluation function, 275, 294
 feed-forward neural networks, 256–258, 257f

Simple linear regression, 344

Single-item sets, association rules, 82, 83t

Single test data set, model comparison with,
 457–458

Single-valued summary statistics, 224–225
 for performance evaluation, 455–456

Singular value decomposition (SVD), 400

Site design, 391

Slope–intercept form, of regression equation, 344

Snowflake schema, 431; *see also* Star schema

Software. *see* RapidMiner; Weka

Software as a service (SaaS), 24

SOM (Self-Organizing Map) operator, RapidMiner,
 293, 311–313, 312f

SOMs. *see* Self-organizing maps (SOMs)

Speech recognition
 SVMs application to, 327

Split Data operator, RapidMiner, 161, 178

Sprint algorithm, 405

SQL, 21

Standard deviation, 224

Standard error (SE), 227, 459
 computation, 230–231
 error rates and, 230

Standard probability density function, 322

Star schema, data warehouse, 429–432, 429f
 additional relational schemas, 431–432
 for credit card purchases, 429–432, 429f

- dimension tables, 429f, 430
- fact table, 430, 431f
- multidimensional array, 429
- multidimensionality of, 429f, 430–431, 431f
- slowly changing dimensions, 430
- Statistical analysis, 214
 - evaluation and
 - findings from, 223–224
 - normal distribution, 225–226, 225f
 - normal distributions and sample means, 226–228, 226f
 - single-valued summary statistics, 224–225
- Statistical analysis system (SAS), 74
- Statistical package for the social sciences (SPSS), 74
- Statistical regression, 46–47
- Statistical significance, 229
- Statistics, 4, 5
- Statistics, for performance evaluation, 455–460
 - confidence intervals for numeric output, 458–459
 - normal distribution, 456
 - numeric output, comparing models with, 459–460
 - single-valued summary statistics, 455–456
 - supervised learner models, comparing, 456–458
- Stemming, word, 399–400, 402, 402f–403
- Stock market analytics, time-series analysis, 378–379
- Stratification, 223
- Streaming data, 412–413
- Structural metadata, 429
- Structuring (star schema), data warehouse, 429–432, 429f
 - additional relational schemas, 431–432
 - for credit card purchases, 429–432, 429f
 - dimension tables, 429f, 430
 - fact table, 430, 431f
 - multidimensional array, 429
 - multidimensionality of, 429f, 430–431, 431f
 - slowly changing dimensions, 430
- Subgroup discovery method, 42
- Subgroup Discovery* operator, RapidMiner, 178–181, 179f–181f
 - process design for, 179f
 - rules generated by, 180f
 - subprocess design for, 179f
- Subprocess* operator, RapidMiner, 165–166, 166f–167f, 170
- Supermarket data set
 - association rules in Weka, 128–129, 130f–131f
- Supervised attribute filter, Weka, 111
- Supervised genetic learning, 91–95, 91f
 - crossover operation, 94, 94f, 95
- fitness function, 91, 91f, 92–93, 93t
- fitness score, 93–94
- mutation, 94–95
- population elements, 91–93, 91f, 92t, 93t
- process, 91f
- question mark, 92
- second-generation population, 94, 94t
- selection operator, 92
- training data, 93t
- Supervised learner models, 50–52, 51t
 - comparison of, 232–234
 - independent test data, 456–457
 - with numeric output, 459–460
 - pairwise comparison with single test data set, 457–458
 - performance evaluation statistics, 456–458
 - performance of two models, 233
 - performance of two or more models, 234
 - with RapidMiner, 238–240, 239f–241f
 - components, 222–223, 222f
 - evaluation, 222–223
 - classifier error rate, 230
 - with numeric output, 236–238, 237t
 - for unsupervised clustering, 235
 - unsupervised clustering for, 235
 - validation data, 230
 - test set error rates two, 232
- Supervised learning, 8–11
 - bagging, 413
 - components, 222–223, 222f
 - data mining strategies, 34
 - classification, 34–35
 - estimation, 35–36
 - prediction, 36–39, 37t, 38t
 - data mining techniques, 41
 - credit card promotion database, 41–42
 - neural networks, 44–46, 45f, 46t
 - rule-based techniques, 42–44
 - statistical regression, 46–47
 - decision tree example, 9–11, 10f, 10t
 - decision trees. *see* Decision tree(s)
 - with feed-forward networks, 258–259
 - backpropagation learning, 258–259
 - genetic learning, 259, 259f
 - genetic algorithms and, 91–95, 91f, 92t–94t, 94f
 - neural networks for, 44
 - scalable techniques for, 405
 - statistical regression, 46–47
 - vs. unsupervised clustering, 11–12
- Supervised neural net learning
 - satellite image data set mining, *Neural Net* operator (RapidMiner), 301–306, 302f–306f

- data preparation, 301, 301f
 goal identification, 301
 network architecture, 301–303, 302f
 network training, 303, 304f
 results interpretation and reading, 303–306,
 304f–306f
- XOR modeling, *Neural Net* operator
 (RapidMiner), 294–301, 295f–300f
 data preparation, 294
 goal identification, 294
 network architecture, defining, 294–297,
 295f–297f, 298f
 network training, 297–298, 298f, 299f
 results interpretation and reading, 299–301,
 299f–300f
- Supervised statistical techniques, 317–352
 linear regression analysis, 340–348
 logistic regression, 350–352
 naïve Bayes classifier, 317–324
 example, 318–321, 318t
 general considerations, 324
 implementations, 324
 missing data, 321–322
 numeric data, 322–324, 323t
 zero-valued attribute counts, 321
- overview, 317
 regression trees, 349, 349f
 support vector machines (SVMs), 324–340
 applications, 327
 background, 324
 binary classification models, 327
 example-based approach, 327
 general considerations, 337
 hyperplane, 327, 330–331, 330f, 331f
 implementations of, 340
 kernel function, 336, 337
 limitations, 337
 linearly separable classes, 332–336, 335f
 mapping function, 336–337
 maximum margin hyperplane (MMH), 330,
 331
 nonlinear case, 336–337
 RapidMiner tutorial, 341, 341f–343f
 support vectors, 330–331
 Weka tutorial, 338–340, 338f–340f
- Support, association rules, 80–81
- Support vector machines (SVMs), 324–340
 applications, 327
 background, 324
 binary classification models, 327
 example-based approach, 327
 general considerations, 337
 hyperplane, 327, 330–331, 330f, 331f
 implementations of, 340
 kernel function, 336, 337
 limitations, 337
 linearly separable classes, 332–336, 335f
 mapping function, 336–337
 maximum margin hyperplane (MMH), 330,
 331
 nonlinear case, 336–337
 RapidMiner tutorial, 341, 341f–343f
 support vectors, 330–331
 Weka tutorial, 338–340, 338f–340f
- implementations of, 340
 kernel function, 336, 337
 limitations, 337
 linearly separable classes, 332–336, 335f
 mapping function, 336–337
 maximum margin hyperplane (MMH), 330, 331
 nonlinear case, 336–337
 RapidMiner tutorial, 341, 341f–343f
 support vectors, 330–331, 331, 333, 334–335
 Weka tutorial, 338–340, 338f–340f
- Support vector model for regression (SMOreg), 340
- Support vectors, 330–331, 331f, 333, 334–335
- SVD (singular value decomposition), 400
- SVD operator, 401
- SVM operators, RapidMiner's, 341, 341f–343f
- SVMs. *see* Support vector machines (SVMs)
-
- T
- Target data set, creating
 in knowledge discovery, 200, 202–203, 203f
 sources, 202
 time-series analysis
 categorical output, 382, 383f
 numeric output, 380, 380f, 381f
- Targeted vs. mass mailings, 54f
- Technical analysts, 378, 379
- Test set, 11
 Test set error rates, 458
 computation, 230–231
 supervised learner models, 232
- Test set evaluation, 223
- Textual data mining, 398–404
 applications, 399
 customer reviews analysis, 399–400
 customer reviews with RapidMiner, 400–404,
 401f–404f
 process, 398
 SVMs application to, 327
- Third normal form (3NF), 425
- Three-class confusion matrix, 50–51, 51t
- Three-item sets, association rules, 84
- Three-node decision tree, for credit card promotion
 database, 71–72, 71f
- Threshold nodes, 258, 297
- Time limit, technique selection and, 97
- Time-series analysis, 213, 377–390
 applications, 378
 data mining
 RapidMiner, 382–387, 383f–386f
 Weka, 387–390, 387f–389f
 data preprocessing and transformation, 380–382,
 381t, 382f

- example, 379–390
 general considerations, 390
 interpretation, evaluation, and action, 390
 KDD application, 379
 overview, 377–378
 stock market analytics, 378–379
 SVMs application to, 327
 target data sets
 - categorical output, 382, 383f
 - numeric output, 380, 380f, 381f
 tools for, 378
- Tokenization, 399, 402, 402f–403f
- Training
 - neural networks, 258, 277, 283, 297–298, 298f–299f, 303, 304f, 307
 - backpropagation learning, 258–259, 263–266
 - detailed view, 263–268
 - genetic learning, 259, 259t
 - Kohonen self-organizing maps, 266–268, 266f
 - supervised learning with feed-forward networks, 258–259
 - unsupervised clustering with self-organizing maps, 259–260, 260f
- Training and test set scenario
 - decision trees building in RapidMiner, 160–165, 160f–165f
- Training cycles* parameter, 296
- Training data
 - decision tree, 11, 67t, 68, 72–73, 72t
 - evaluation, 223
 - for XOR function, 273, 273f
- TrainingTime* parameter, 277, 280, 281
- Transactional database, 424; *see also* Operational databases
- Transpose* operator, 380
- Tree structures, 6
- Tree to Rules* operator, RapidMiner, 173
 - model building with, 174f
 - rules generated by, 175f
 - subprocess for, 174f
- True accept, 52
- True-positive (TP) rate, ROC curve, 406, 407, 408, 409, 410f
- True reject, 52
- T-test, 229, 230, 234, 414, 416f
- T-test* operator, RapidMiner, 222, 238–240, 239f–241f
- Tuples, 202
- Twitter, 5, 21, 24, 412
- Two-class confusion matrix, 52, 52t
- Two-class error analysis, 52–53, 52t
- Two-item sets, association rules, 82–84, 83t
- Two-node decision tree, for credit card promotion database, 71f, 72
- Type 1 error, 229
- Type 2 error, 229
-
- U
- Uniform resource identifier (URI), 392
- University of California–Irvine (UCI) Machine Learning Repository, 454
- Unstable algorithm, 74
- Unsupervised attribute filter, Weka, 111
- Unsupervised clustering, 11–13, 12t, 48–49
 - between-cluster attribute values, 236
 - of credit card promotion database, 48–49, 49f
 - data mining strategies, 39–40
 - evaluation
 - additional methods for, 236
 - external, 236
 - internal, 236
 - supervised, 235
 - instance selection and, 213
 - K-means algorithm, 85–90
 - applications, 88, 89t
 - clusters size and, 89
 - coordinate mapping of data, 86f
 - example, 86–88
 - gamma-ray burst data set, 187–191, 188f–190f
 - general considerations, 89–90
 - input values, 86t
 - irrelevant attributes, 90
 - limitations, 89–90
 - number of clusters and, 89
 - optimal clustering for, 88
 - in RapidMiner, 187–191, 188f–190f
 - real-valued data, 89
 - in Weka, 137–141, 137f–140f
 - model evaluation, 55–56
 - neural networks for, 44
 - outlier detection, 205
 - questions, 12
 - with self-organizing maps, 259–260, 260f
 - for supervised evaluation, 235
 - supervised evaluation for, 235
 - uses, 39
 - vs. supervised learning, 11–12
 - web-based data mining, 394
- Unsupervised clustering techniques, 48–49, 357–374
 - agglomerative clustering, 358–360
 - example, 358–360, 358t, 359t
 - general considerations, 360
 - conceptual clustering, 360–364
 - category utility measurement, 361–362

example, 362–364, 363f, 364t
 general considerations, 364
 expectation maximization (EM) algorithm,
 364–371
 general considerations, 365, 371
 implementations, 365
 RapidMiner tutorial, 366–369, 366f–369f
 Weka tutorial, 369–371, 370f–371f
 genetic algorithms and, 371–374, 372f, 373t
 overview, 357–358
 scalable, 405
 Unsupervised InterquartileRange filter, Weka, 111
 Unsupervised neural net clustering, 287–288,
 287f–288f
 User interface, OLAP system, 434–435
 User session, Web-based data mining, 392
 User-specified learning parameters, 223

V

Validation data, 230
Validation operator, RapidMiner, 170, 172
 Vapnik, Vladimir, 324, 341
 Variance, 224–225, 455
 of population, 227
 Variety, big data, 24
 Vector, defined, 332
 Velocity, big data, 24
 Veracity, big data, 24
 VGT, 379
 Volume, big data, 24

W

Waikato Environment for Knowledge Analysis.
see Weka
 Warehouse, data, 20–21, 202–203, 423–445; *see also*
 Independent data mart
 defined, 427
 dependent data marts, 434
 design, 426–434
 data entering, 427–429, 428f
 decision support, 432–434, 433f
 structuring (star schema), 429–432, 429f
 ETL process, 428
 Excel pivot tables for data analytics, 438–443,
 439f–445f
 existing record, 428
 metadata, 428–429
 new record, 428
 online analytical processing (OLAP), 434–438
 example, 435–438, 435f–437f
 general considerations, 438

operational databases, 424–426
 data modeling, 424–425
 data normalization, 424–425
 entity-relationship diagram, 424–425, 424f
 relational model, 425–426, 425t–426t
 overview, 423
 process model, 428f
 sources of data, 427
 vs. OLTP database, 427
 Watch promotion attribute, 322
 Weather information, 427
 Web-based data mining, 391
 actions/steps, 394–395
 association rule technique, 393–394
 clickstream sequences, 392
 data preparation, 392–395
 FP-Growth algorithm, 405
 general issues, 391
 goal identification, 391–392
 link analysis, 396–398
 operators for, 398
 PageRank algorithm, 396–398, 397f
 pageview, 392
 for personalization, 395–396, 395f–396f
 results interpretation and evaluation, 393–394
 session file, 392, 392f
 techniques, 393
 unsupervised clustering, 394
 user session, 392
 for website adaptation, 396
 for website evaluation, 395
 Web page design, 391
 Website evaluation, data mining for, 395
 Website(s)
 adaptation, data mining for, 396
 adaptive, 396
 factors for success, 391
 index page, 396
 index synthesis problem, 396
 RapidMiner, 146, 453
 sequence miners, 395
 site design, 391
 Web page design, 391
 Weka, 105, 106, 451
 Weka, 36, 40–41, 105–141, 201
 ArffViewer, 453
 association rules, 127–131
 Apriori algorithm parameters for, 129f
 for contact-lenses data set, 127–128, 128f
 GenericObjectEditor, 128, 129f
 LowerBoundMinSupport, 128, 131
 supermarket data set, 128–129, 130f–131f
 attribute filters, 111, 112f

- Attribute-Relation File Format (ARFF), 106, 117, 452f, 453
 @attribute symbol, 108
 bagging, implementation, 415
 boosting, implementation, 415
 CART algorithm, 451
 contact-lenses file/data set, 108, 109–110, 109f, 110f
 decision tree for, 113–117, 113f–116f
 cost/benefit analysis, 131–136 132f–136f
 credit card screening data set, 129
 cost/benefit output for, 132, 133f
 J48 classification of, 132, 132f
 data mining, time-series analysis, 387–390, 387f–389f
 data sets for, 453
 @data symbol, 109
 decision trees
 actual and predicted output, 115, 116f
 building, 109–117, 110f–116f
 classifier output options, 115, 116f
 for contact-lenses data set, 113–117, 113f–116f
 output, 114–115, 115f
 tree visualizer, 114f
 discretize filter, 127
 download link, 106, 451
 EM clustering model, 369–371, 370f–371f
 Experimenter interface, 106
 Explorer GUI's, 106, 107f
 Explorer interface, 105–106
 GenericObjectEditor, 112, 113f
 get started with, 106–109, 107f–109f, 451–453, 452f
 graphical user interfaces (GUI's), 105–106
 for XOR training, 275f
 IBk nearest neighbor classifier, 122
 installation, 106–109, 107f–109f, 451–453, 452f
 install folder, 108f
 J48, 111–113, 112f–113f
 advantages, 122
 classification of credit card screening data set, 132–136, 132f–136f
 Knowledge Flow interface, 106
 Kohonen clustering algorithm, 451
 latest stable versions of, 106
 lazy classifiers, 122
 linear regression models, 47, 344, 345f
 naïve Bayes classifier, 328–329, 328f–329f
 nearest neighbor classification and attribute selection, 122–127
 attribute selection filter, 123, 124f
 IBk classifier, 122, 125
InfoGainAttributeEval, 123, 124f
numToSelect, 124–125, 126f
 output for spam data set, 123, 123f
 rankers, 123–124, 125f
 neural networks building with, 271–288
 backpropagation learning, data sets for, 272–274
 exclusive-OR (XOR) function, 272–273, 272f–273f, 274–282; *see also* Exclusive OR (XOR) function
 MultiLayerPerceptron (MLP) function, 271, 274, 275, 276, 280
 overview, 271–272
 satellite image data set, 273–274, 274f, 282–287; *see also* Satellite image data set
SelfOrganizingMap algorithm, 271
 unsupervised neural net clustering, 287–288, 287f–288f
 neural network software packages, 46
 NominalToBinary filter, 137
 overview, 105, 451
 package manager, 451
 PART rule generator, 408
 production rules, generation with PART, 117–122
 advantages, 122
 customer churn data, 117, 118f
 customer churn instances, loading, 120, 120f
 customer churn output, 119, 119f
 decision list, 117
 decision list for customer churn data, 117, 118f
 prediction probability, 120–121, 121f
 save result buffer, 121
 @ relation symbol, 108
 ROC curves
 creation and analysis, 408–411, 408f–411f
 implementation, 407
 sample data sets, 108f
SelfOrganizingMap algorithm, 271, 287–288, 287f–288f, 451
 SMO algorithm, 338–340, 338f–340f
 supervised attribute filter, 111
 support vector machines, 338–340, 338f–340f
 unsupervised attribute filter, 111
 unsupervised clustering with K-means algorithm, 137–141, 137f–140f
 website, 105, 106
 WEKA GUI Chooser, 106, 107f, 451, 452f
 ZeroR, 111
 WEKA GUI Chooser, 106, 107f, 451, 452f
WekaManual.pdf, 106
 Weka's 3.7 knowledge-flow interface, 25
Weka Workbench, 106
 Word stemming, 399–400, 402, 402f–403

World Wide Web, 400

Wrapper approach, 194

Wrapper techniques

attribute selection, 210–211

Write Excel operator, RapidMiner, 168, 188

Write Model operator, RapidMiner, 167, 309

Written evaluation, 399

Written reviews, classification of, 400

X

XLF, 379

X-Prediction operator, RapidMiner, 306

χ^2 statistical test, 74

X-Validation operator, RapidMiner, 170, 301–303, 306

Y

Yahoo Historical Stock Data operator, 380

Z

ZeroR, Weka, 111

Zero-valued attribute counts, naïve Bayes classifier, 321

Z-scores

normalization using, 209