

Data Clustering: Theory, Algorithms, and Applications

Guojun Gan, Chaoqun Ma, and Jianhong Wu



siam



Data Clustering

ASA-SIAM Series on Statistics and Applied Probability



The ASA-SIAM Series on Statistics and Applied Probability is published jointly by the American Statistical Association and the Society for Industrial and Applied Mathematics. The series consists of a broad spectrum of books on topics in statistics and applied probability. The purpose of the series is to provide inexpensive, quality publications of interest to the intersecting membership of the two societies.

Editorial Board

Martin T. Wells

Cornell University, Editor-in-Chief

H. T. Banks

North Carolina State University

Douglas M. Hawkins

University of Minnesota

Susan Holmes

Stanford University

Lisa LaVange

University of North Carolina

David Madigan

Rutgers University

Mark van der Laan

University of California, Berkeley

Gan, G., Ma, C., and Wu, J., *Data Clustering: Theory, Algorithms, and Applications*

Hubert, L., Arabie, P., and Meulman, J., *The Structural Representation of Proximity Matrices with MATLAB*

Nelson, P. R., Wludyka, P. S., and Copeland, K. A. F., *The Analysis of Means: A Graphical Method for Comparing Means, Rates, and Proportions*

Burdick, R. K., Borror, C. M., and Montgomery, D. C., *Design and Analysis of Gauge R&R Studies: Making Decisions with Confidence Intervals in Random and Mixed ANOVA Models*

Albert, J., Bennett, J., and Cochran, J. J., eds., *Anthology of Statistics in Sports*

Smith, W. F., *Experimental Design for Formulation*

Baglivo, J. A., *Mathematica Laboratories for Mathematical Statistics: Emphasizing Simulation and Computer Intensive Methods*

Lee, H. K. H., *Bayesian Nonparametrics via Neural Networks*

O'Gorman, T. W., *Applied Adaptive Statistical Methods: Tests of Significance and Confidence Intervals*

Ross, T. J., Booker, J. M., and Parkinson, W. J., eds., *Fuzzy Logic and Probability Applications: Bridging the Gap*

Nelson, W. B., *Recurrent Events Data Analysis for Product Repairs, Disease Recurrences, and Other Applications*

Mason, R. L. and Young, J. C., *Multivariate Statistical Process Control with Industrial Applications*

Smith, P. L., *A Primer for Sampling Solids, Liquids, and Gases: Based on the Seven Sampling Errors of Pierre Gy*

Meyer, M. A. and Booker, J. M., *Eliciting and Analyzing Expert Judgment: A Practical Guide*

Latouche, G. and Ramaswami, V., *Introduction to Matrix Analytic Methods in Stochastic Modeling*

Peck, R., Haugh, L., and Goodman, A., *Statistical Case Studies: A Collaboration Between Academe and Industry, Student Edition*

Peck, R., Haugh, L., and Goodman, A., *Statistical Case Studies: A Collaboration Between Academe and Industry*

Barlow, R., *Engineering Reliability*

Czitrom, V. and Spagon, P. D., *Statistical Case Studies for Industrial Process Improvement*

Data Clustering Theory, Algorithms, and Applications

Guojun Gan

York University
Toronto, Ontario, Canada

Chaoqun Ma

Hunan University
Changsha, Hunan, People's Republic of China

Jianhong Wu

York University
Toronto, Ontario, Canada



Society for Industrial and Applied Mathematics
Philadelphia, Pennsylvania



American Statistical Association
Alexandria, Virginia

The correct bibliographic citation for this book is as follows: Gan, Guojun, Chaoqun Ma, and Jianhong Wu, *Data Clustering: Theory, Algorithms, and Applications*, ASA-SIAM Series on Statistics and Applied Probability, SIAM, Philadelphia, ASA, Alexandria, VA, 2007.

Copyright © 2007 by the American Statistical Association and the Society for Industrial and Applied Mathematics.

10 9 8 7 6 5 4 3 2 1

All rights reserved. Printed in the United States of America. No part of this book may be reproduced, stored, or transmitted in any manner without the written permission of the publisher. For information, write to the Society for Industrial and Applied Mathematics, 3600 University City Science Center, Philadelphia, PA 19104-2688.

Trademarked names may be used in this book without the inclusion of a trademark symbol. These names are intended in an editorial context only; no infringement of trademark is intended.

Library of Congress Cataloging-in-Publication Data

Gan, Guojun, 1979-

 Data clustering : theory, algorithms, and applications / Guojun Gan, Chaoqun Ma,
 Jianhong Wu.

 p. cm. – (ASA-SIAM series on statistics and applied probability ; 20)

 Includes bibliographical references and index.

 ISBN: 978-0-898716-23-8 (alk. paper)

1. Cluster analysis. 2. Cluster analysis—Data processing. I. Ma, Chaoqun, Ph.D. II.
Wu, Jianhong. III. Title.

QA278.G355 2007

519.5'3—dc22

2007061713

Contents

List of Figures	xiii
List of Tables	xv
List of Algorithms	xvii
Preface	xix
I Clustering, Data, and Similarity Measures	1
1 Data Clustering	3
1.1 Definition of Data Clustering	3
1.2 The Vocabulary of Clustering	5
1.2.1 Records and Attributes	5
1.2.2 Distances and Similarities	5
1.2.3 Clusters, Centers, and Modes	6
1.2.4 Hard Clustering and Fuzzy Clustering	7
1.2.5 Validity Indices	8
1.3 Clustering Processes	8
1.4 Dealing with Missing Values	10
1.5 Resources for Clustering	12
1.5.1 Surveys and Reviews on Clustering	12
1.5.2 Books on Clustering	12
1.5.3 Journals	13
1.5.4 Conference Proceedings	15
1.5.5 Data Sets	17
1.6 Summary	17
2 Data Types	19
2.1 Categorical Data	19
2.2 Binary Data	21
2.3 Transaction Data	23
2.4 Symbolic Data	23
2.5 Time Series	24
2.6 Summary	24

3	Scale Conversion	25
3.1	Introduction	25
3.1.1	Interval to Ordinal	25
3.1.2	Interval to Nominal	27
3.1.3	Ordinal to Nominal	28
3.1.4	Nominal to Ordinal	28
3.1.5	Ordinal to Interval	29
3.1.6	Other Conversions	29
3.2	Categorization of Numerical Data	30
3.2.1	Direct Categorization	30
3.2.2	Cluster-based Categorization	31
3.2.3	Automatic Categorization	37
3.3	Summary	41
4	Data Standardization and Transformation	43
4.1	Data Standardization	43
4.2	Data Transformation	46
4.2.1	Principal Component Analysis	46
4.2.2	SVD	48
4.2.3	The Karhunen-Loëve Transformation	49
4.3	Summary	51
5	Data Visualization	53
5.1	Sammon's Mapping	53
5.2	MDS	54
5.3	SOM	56
5.4	Class-preserving Projections	59
5.5	Parallel Coordinates	60
5.6	Tree Maps	61
5.7	Categorical Data Visualization	62
5.8	Other Visualization Techniques	65
5.9	Summary	65
6	Similarity and Dissimilarity Measures	67
6.1	Preliminaries	67
6.1.1	Proximity Matrix	68
6.1.2	Proximity Graph	69
6.1.3	Scatter Matrix	69
6.1.4	Covariance Matrix	70
6.2	Measures for Numerical Data	71
6.2.1	Euclidean Distance	71
6.2.2	Manhattan Distance	71
6.2.3	Maximum Distance	72
6.2.4	Minkowski Distance	72
6.2.5	Mahalanobis Distance	72

6.2.6	Average Distance	73
6.2.7	Other Distances	74
6.3	Measures for Categorical Data	74
6.3.1	The Simple Matching Distance	76
6.3.2	Other Matching Coefficients	76
6.4	Measures for Binary Data	77
6.5	Measures for Mixed-type Data	79
6.5.1	A General Similarity Coefficient	79
6.5.2	A General Distance Coefficient	80
6.5.3	A Generalized Minkowski Distance	81
6.6	Measures for Time Series Data	83
6.6.1	The Minkowski Distance	84
6.6.2	Time Series Preprocessing	85
6.6.3	Dynamic Time Warping	87
6.6.4	Measures Based on Longest Common Subsequences	88
6.6.5	Measures Based on Probabilistic Models	90
6.6.6	Measures Based on Landmark Models	91
6.6.7	Evaluation	92
6.7	Other Measures	92
6.7.1	The Cosine Similarity Measure	93
6.7.2	A Link-based Similarity Measure	93
6.7.3	Support	94
6.8	Similarity and Dissimilarity Measures between Clusters	94
6.8.1	The Mean-based Distance	94
6.8.2	The Nearest Neighbor Distance	95
6.8.3	The Farthest Neighbor Distance	95
6.8.4	The Average Neighbor Distance	96
6.8.5	Lance-Williams Formula	96
6.9	Similarity and Dissimilarity between Variables	98
6.9.1	Pearson's Correlation Coefficients	98
6.9.2	Measures Based on the Chi-square Statistic	101
6.9.3	Measures Based on Optimal Class Prediction	103
6.9.4	Group-based Distance	105
6.10	Summary	106
II	Clustering Algorithms	107
7	Hierarchical Clustering Techniques	109
7.1	Representations of Hierarchical Clusterings	109
7.1.1	<i>n</i> -tree	110
7.1.2	Dendrogram	110
7.1.3	Banner	112
7.1.4	Pointer Representation	112
7.1.5	Packed Representation	114
7.1.6	Icicle Plot	115
7.1.7	Other Representations	115

7.2	Agglomerative Hierarchical Methods	116
7.2.1	The Single-link Method	118
7.2.2	The Complete Link Method	120
7.2.3	The Group Average Method	122
7.2.4	The Weighted Group Average Method	125
7.2.5	The Centroid Method	126
7.2.6	The Median Method	130
7.2.7	Ward's Method	132
7.2.8	Other Agglomerative Methods	137
7.3	Divisive Hierarchical Methods	137
7.4	Several Hierarchical Algorithms	138
7.4.1	SLINK	138
7.4.2	Single-link Algorithms Based on Minimum Spanning Trees	140
7.4.3	CLINK	141
7.4.4	BIRCH	144
7.4.5	CURE	144
7.4.6	DIANA	145
7.4.7	DISMEA	147
7.4.8	Edwards and Cavalli-Sforza Method	147
7.5	Summary	149
8	Fuzzy Clustering Algorithms	151
8.1	Fuzzy Sets	151
8.2	Fuzzy Relations	153
8.3	Fuzzy k -means	154
8.4	Fuzzy k -modes	156
8.5	The c -means Method	158
8.6	Summary	159
9	Center-based Clustering Algorithms	161
9.1	The k -means Algorithm	161
9.2	Variations of the k -means Algorithm	164
9.2.1	The Continuous k -means Algorithm	165
9.2.2	The Compare-means Algorithm	165
9.2.3	The Sort-means Algorithm	166
9.2.4	Acceleration of the k -means Algorithm with the kd -tree	167
9.2.5	Other Acceleration Methods	168
9.3	The Trimmed k -means Algorithm	169
9.4	The x -means Algorithm	170
9.5	The k -harmonic Means Algorithm	171
9.6	The Mean Shift Algorithm	173
9.7	MEC	175
9.8	The k -modes Algorithm (Huang)	176
9.8.1	Initial Modes Selection	178
9.9	The k -modes Algorithm (Chaturvedi et al.)	178

9.10	The k -probabilities Algorithm	179
9.11	The k -prototypes Algorithm	181
9.12	Summary	182
10	Search-based Clustering Algorithms	183
10.1	Genetic Algorithms	184
10.2	The Tabu Search Method	185
10.3	Variable Neighborhood Search for Clustering	186
10.4	AI-Sultan's Method	187
10.5	Tabu Search–based Categorical Clustering Algorithm	189
10.6	J -means	190
10.7	GKA	192
10.8	The Global k -means Algorithm	195
10.9	The Genetic k -modes Algorithm	195
10.9.1	The Selection Operator	196
10.9.2	The Mutation Operator	196
10.9.3	The k -modes Operator	197
10.10	The Genetic Fuzzy k -modes Algorithm	197
10.10.1	String Representation	198
10.10.2	Initialization Process	198
10.10.3	Selection Process	199
10.10.4	Crossover Process	199
10.10.5	Mutation Process	200
10.10.6	Termination Criterion	200
10.11	SARS	200
10.12	Summary	202
11	Graph-based Clustering Algorithms	203
11.1	Chameleon	203
11.2	CACTUS	204
11.3	A Dynamic System–based Approach	205
11.4	ROCK	207
11.5	Summary	208
12	Grid-based Clustering Algorithms	209
12.1	STING	209
12.2	OptiGrid	210
12.3	GRIDCLUS	212
12.4	GDILC	214
12.5	WaveCluster	216
12.6	Summary	217
13	Density-based Clustering Algorithms	219
13.1	DBSCAN	219
13.2	BRIDGE	221
13.3	DBCLASD	222

13.4	DENCLUE	223
13.5	CUBN	225
13.6	Summary	226
14	Model-based Clustering Algorithms	227
14.1	Introduction	227
14.2	Gaussian Clustering Models	230
14.3	Model-based Agglomerative Hierarchical Clustering	232
14.4	The EM Algorithm	235
14.5	Model-based Clustering	237
14.6	COOLCAT	240
14.7	STUCCO	241
14.8	Summary	242
15	Subspace Clustering	243
15.1	CLIQUE	244
15.2	PROCLUS	246
15.3	ORCLUS	249
15.4	ENCLUS	253
15.5	FINDIT	255
15.6	MAFIA	258
15.7	DOC	259
15.8	CLTree	261
15.9	PART	262
15.10	SUBCAD	264
15.11	Fuzzy Subspace Clustering	270
15.12	Mean Shift for Subspace Clustering	275
15.13	Summary	285
16	Miscellaneous Algorithms	287
16.1	Time Series Clustering Algorithms	287
16.2	Streaming Algorithms	289
16.2.1	LSEARCH	290
16.2.2	Other Streaming Algorithms	293
16.3	Transaction Data Clustering Algorithms	293
16.3.1	LargeItem	294
16.3.2	CLOPE	295
16.3.3	OAK	296
16.4	Summary	297
17	Evaluation of Clustering Algorithms	299
17.1	Introduction	299
17.1.1	Hypothesis Testing	301
17.1.2	External Criteria	302
17.1.3	Internal Criteria	303
17.1.4	Relative Criteria	304

17.2	Evaluation of Partitional Clustering	305
17.2.1	Modified Hubert's Γ Statistic	305
17.2.2	The Davies-Bouldin Index	305
17.2.3	Dunn's Index	307
17.2.4	The SD Validity Index	307
17.2.5	The S_Dbw Validity Index	308
17.2.6	The RMSSTD Index	309
17.2.7	The RS Index	310
17.2.8	The Calinski-Harabasz Index	310
17.2.9	Rand's Index	311
17.2.10	Average of Compactness	312
17.2.11	Distances between Partitions	312
17.3	Evaluation of Hierarchical Clustering	314
17.3.1	Testing Absence of Structure	314
17.3.2	Testing Hierarchical Structures	315
17.4	Validity Indices for Fuzzy Clustering	315
17.4.1	The Partition Coefficient Index	315
17.4.2	The Partition Entropy Index	316
17.4.3	The Fukuyama-Sugeno Index	316
17.4.4	Validity Based on Fuzzy Similarity	317
17.4.5	A Compact and Separate Fuzzy Validity Criterion	318
17.4.6	A Partition Separation Index	319
17.4.7	An Index Based on the Mini-max Filter Concept and Fuzzy Theory	319
17.5	Summary	320
III	Applications of Clustering	321
18	Clustering Gene Expression Data	323
18.1	Background	323
18.2	Applications of Gene Expression Data Clustering	324
18.3	Types of Gene Expression Data Clustering	325
18.4	Some Guidelines for Gene Expression Clustering	325
18.5	Similarity Measures for Gene Expression Data	326
18.5.1	Euclidean Distance	326
18.5.2	Pearson's Correlation Coefficient	326
18.6	A Case Study	328
18.6.1	C++ Code	328
18.6.2	Results	334
18.7	Summary	334
IV	MATLAB and C++ for Clustering	341
19	Data Clustering in MATLAB	343
19.1	Read and Write Data Files	343
19.2	Handle Categorical Data	347

19.3	M-files, MEX-files, and MAT-files	349
19.3.1	M-files	349
19.3.2	MEX-files	351
19.3.3	MAT-files	354
19.4	Speed up MATLAB	354
19.5	Some Clustering Functions	355
19.5.1	Hierarchical Clustering	355
19.5.2	k -means Clustering	359
19.6	Summary	362
20	Clustering in C/C++	363
20.1	The STL	363
20.1.1	The <i>vector</i> Class	363
20.1.2	The <i>list</i> Class	364
20.2	C/C++ Program Compilation	366
20.3	Data Structure and Implementation	367
20.3.1	Data Matrices and Centers	367
20.3.2	Clustering Results	368
20.3.3	The Quick Sort Algorithm	369
20.4	Summary	369
A	Some Clustering Algorithms	371
B	The <i>kd</i>-tree Data Structure	375
C	MATLAB Codes	377
C.1	The MATLAB Code for Generating Subspace Clusters	377
C.2	The MATLAB Code for the k -modes Algorithm	379
C.3	The MATLAB Code for the MSSC Algorithm	381
D	C++ Codes	385
D.1	The C++ Code for Converting Categorical Values to Integers	385
D.2	The C++ Code for the FSC Algorithm	388
Bibliography		397
Subject Index		443
Author Index		455

List of Figures

1.1	Data-mining tasks	4
1.2	Three well-separated center-based clusters in a two-dimensional space	7
1.3	Two chained clusters in a two-dimensional space	7
1.4	Processes of data clustering	9
1.5	Diagram of clustering algorithms	10
2.1	Diagram of data types	19
2.2	Diagram of data scales	20
3.1	An example two-dimensional data set with 60 points	31
3.2	Examples of direct categorization when $N = 5$	32
3.3	Examples of direct categorization when $N = 2$	32
3.4	Examples of k -means-based categorization when $N = 5$	33
3.5	Examples of k -means-based categorization when $N = 2$	34
3.6	Examples of cluster-based categorization based on the least squares partition when $N = 5$	36
3.7	Examples of cluster-based categorization based on the least squares partition when $N = 2$	36
3.8	Examples of automatic categorization using the k -means algorithm and the compactness-separation criterion	38
3.9	Examples of automatic categorization using the k -means algorithm and the compactness-separation criterion	39
3.10	Examples of automatic categorization based on the least squares partition and the SSC	40
3.11	Examples of automatic categorization based on the least squares partition and the SSC	40
5.1	The architecture of the SOM	57
5.2	The axes of the parallel coordinates system	60
5.3	A two-dimensional data set containing five points	60
5.4	The parallel coordinates plot of the five points in Figure 5.3	61
5.5	The dendrogram of the single-linkage hierarchical clustering of the five points in Figure 5.3	62
5.6	The tree maps of the dendrogram in Figure 5.5	62

5.7	Plot of the two clusters in Table 5.1	64
6.1	Nearest neighbor distance between two clusters.	95
6.2	Farthest neighbor distance between two clusters.	95
7.1	Agglomerative hierarchical clustering and divisive hierarchical clustering.	110
7.2	A 5-tree.	111
7.3	A dendrogram of five data points.	112
7.4	A banner constructed from the dendrogram given in Figure 7.3.	113
7.5	The dendrogram determined by the packed representation given in Table 7.3. . .	115
7.6	An icicle plot corresponding to the dendrogram given in Figure 7.3.	115
7.7	A loop plot corresponding to the dendrogram given in Figure 7.3.	116
7.8	Some commonly used hierarchical methods.	116
7.9	A two-dimensional data set with five data points.	119
7.10	The dendrogram produced by applying the single-link method to the data set given in Figure 7.9.	120
7.11	The dendrogram produced by applying the complete link method to the data set given in Figure 7.9.	122
7.12	The dendrogram produced by applying the group average method to the data set given in Figure 7.9.	125
7.13	The dendrogram produced by applying the weighted group average method to the data set given in Figure 7.9.	126
7.14	The dendrogram produced by applying the centroid method to the data set given in Figure 7.9.	131
7.15	The dendrogram produced by applying the median method to the data set given in Figure 7.9.	132
7.16	The dendrogram produced by applying Ward's method to the data set given in Figure 7.9.	137
14.1	The flowchart of the model-based clustering procedure.	229
15.1	The relationship between the mean shift algorithm and its derivatives.	276
17.1	Diagram of the cluster validity indices.	300
18.1	Cluster 1 and cluster 2.	336
18.2	Cluster 3 and cluster 4.	337
18.3	Cluster 5 and cluster 6.	338
18.4	Cluster 7 and cluster 8.	339
18.5	Cluster 9 and cluster 10.	340
19.1	A dendrogram created by the function <code>dendrogram</code>	359

List of Tables

1.1	A list of methods for dealing with missing values.	11
2.1	A sample categorical data set.	20
2.2	One of the symbol tables of the data set in Table 2.1.	21
2.3	Another symbol table of the data set in Table 2.1.	21
2.4	The frequency table computed from the symbol table in Table 2.2.	22
2.5	The frequency table computed from the symbol table in Table 2.3.	22
4.1	Some data standardization methods, where \bar{x}_j^* , R_j^* , and σ_j^* are defined in equation (4.3).	45
5.1	The coordinate system for the two clusters of the data set in Table 2.1.	63
5.2	Coordinates of the attribute values of the two clusters in Table 5.1.	64
6.1	Some other dissimilarity measures for numerical data.	75
6.2	Some matching coefficients for nominal data.	77
6.3	Similarity measures for binary vectors.	78
6.4	Some symmetrical coefficients for binary feature vectors.	78
6.5	Some asymmetrical coefficients for binary feature vectors.	79
6.6	Some commonly used values for the parameters in the Lance-Williams's formula, where $n_i = C_i $ is the number of data points in C_i , and $\Sigma_{ijk} = n_i + n_j + n_k$	97
6.7	Some common parameters for the general recurrence formula proposed by Jambu (1978).	99
6.8	The contingency table of variables u and v	101
6.9	Measures of association based on the chi-square statistic.	102
7.1	The pointer representation corresponding to the dendrogram given in Figure 7.3.113	
7.2	The packed representation corresponding to the pointer representation given in Table 7.1.	114
7.3	A packed representation of six objects.	114
7.4	The cluster centers agglomerated from two clusters and the dissimilarities between two cluster centers for geometric hierarchical methods, where $\mu(C)$ denotes the center of cluster C	117
7.5	The dissimilarity matrix of the data set given in Figure 7.9. The entry (i, j) in the matrix is the Euclidean distance between \mathbf{x}_i and \mathbf{x}_j	119

7.6	The dissimilarity matrix of the data set given in Figure 7.9.	135
11.1	Description of the chameleon algorithm, where n is the number of data in the database and m is the number of initial subclusters.	204
11.2	The properties of the ROCK algorithm, where n is the number of data points in the data set, m_m is the maximum number of neighbors for a point, and m_a is the average number of neighbors.	208
14.1	Description of Gaussian mixture models in the general family.	231
14.2	Description of Gaussian mixture models in the diagonal family. \mathbf{B} is a diagonal matrix.	232
14.3	Description of Gaussian mixture models in the diagonal family. \mathbf{I} is an identity matrix.	232
14.4	Four parameterizations of the covariance matrix in the Gaussian model and their corresponding criteria to be minimized.	234
15.1	List of some subspace clustering algorithms.	244
15.2	Description of the MAFIA algorithm.	259
17.1	Some indices that measure the degree of similarity between C and P based on the external criteria.	303
19.1	Some MATLAB commands related to reading and writing files.	344
19.2	Permission codes for opening a file in MATLAB.	345
19.3	Some values of precision for the <code>fwrite</code> function in MATLAB.	346
19.4	MEX-file extensions for various platforms.	352
19.5	Some MATLAB clustering functions.	355
19.6	Options of the function <code>pdist</code> .	357
19.7	Options of the function <code>linkage</code> .	358
19.8	Values of the parameter <code>distance</code> in the function <code>kmeans</code> .	360
19.9	Values of the parameter <code>start</code> in the function <code>kmeans</code> .	360
19.10	Values of the parameter <code>emptyaction</code> in the function <code>kmeans</code> .	361
19.11	Values of the parameter <code>display</code> in the function <code>kmeans</code> .	361
20.1	Some members of the <code>vector</code> class.	365
20.2	Some members of the <code>list</code> class.	366

List of Algorithms

Algorithm 5.1	Nonmetric MDS	55
Algorithm 5.2	The pseudocode of the SOM algorithm	58
Algorithm 7.1	The SLINK algorithm	139
Algorithm 7.2	The pseudocode of the CLINK algorithm	142
Algorithm 8.1	The fuzzy k -means algorithm	154
Algorithm 8.2	Fuzzy k -modes algorithm	157
Algorithm 9.1	The conventional k -means algorithm	162
Algorithm 9.2	The k -means algorithm treated as an optimization problem	163
Algorithm 9.3	The compare-means algorithm	165
Algorithm 9.4	An iteration of the sort-means algorithm	166
Algorithm 9.5	The k -modes algorithm	177
Algorithm 9.6	The k -probabilities algorithm	180
Algorithm 9.7	The k -prototypes algorithm	182
Algorithm 10.1	The VNS heuristic	187
Algorithm 10.2	Al-Sultan’s tabu search-based clustering algorithm	188
Algorithm 10.3	The J -means algorithm	191
Algorithm 10.4	Mutation (s_w)	193
Algorithm 10.5	The pseudocode of GKA	194
Algorithm 10.6	Mutation (s_w) in GKMODE	197
Algorithm 10.7	The SARS algorithm	201
Algorithm 11.1	The procedure of the chameleon algorithm	204
Algorithm 11.2	The CACTUS algorithm	205
Algorithm 11.3	The dynamic system-based clustering algorithm	206
Algorithm 11.4	The ROCK algorithm	207
Algorithm 12.1	The STING algorithm	210
Algorithm 12.2	The OptiGrid algorithm	211
Algorithm 12.3	The GRIDCLUS algorithm	213
Algorithm 12.4	Procedure $NEIGHBOR_SEARCH(B,C)$	213
Algorithm 12.5	The GDILC algorithm	215
Algorithm 13.1	The BRIDGE algorithm	221
Algorithm 14.1	Model-based clustering procedure	238
Algorithm 14.2	The COOLCAT clustering algorithm	240
Algorithm 14.3	The STUCCO clustering algorithm procedure	241
Algorithm 15.1	The PROCLUS algorithm	247

Algorithm 15.2	The pseudocode of the ORCLUS algorithm	249
Algorithm 15.3	$Assign(s_1, \dots, s_{k_c}, P_1, \dots, P_{k_c})$	250
Algorithm 15.4	$Merge(C_1, \dots, C_{k_c}, K_{new}, l_{new})$	251
Algorithm 15.5	$FindVectors(C, q)$	252
Algorithm 15.6	ENCLUS procedure for mining significant subspaces	254
Algorithm 15.7	ENCLUS procedure for mining interesting subspaces	255
Algorithm 15.8	The FINDIT algorithm	256
Algorithm 15.9	Procedure of adaptive grids computation in the MAFIA algorithm	258
Algorithm 15.10	The DOC algorithm for approximating an optimal projective cluster	259
Algorithm 15.11	The SUBCAD algorithm	266
Algorithm 15.12	The pseudocode of the FSC algorithm	274
Algorithm 15.13	The pseudocode of the MSSC algorithm	282
Algorithm 15.14	The postprocessing procedure to get the final subspace clusters .	282
Algorithm 16.1	The InitialSolution algorithm	291
Algorithm 16.2	The LSEARCH algorithm	291
Algorithm 16.3	The $FL(D, d(\cdot, \cdot), z, \epsilon, (I, a))$ function	292
Algorithm 16.4	The CLOPE algorithm	296
Algorithm 16.5	A sketch of the OAK algorithm	297
Algorithm 17.1	The Monte Carlo technique for computing the probability density function of the indices	301

Preface

Cluster analysis is an unsupervised process that divides a set of objects into homogeneous groups. There have been many clustering algorithms scattered in publications in very diversified areas such as pattern recognition, artificial intelligence, information technology, image processing, biology, psychology, and marketing. As such, readers and users often find it very difficult to identify an appropriate algorithm for their applications and/or to compare novel ideas with existing results.

In this monograph, we shall focus on a small number of popular clustering algorithms and group them according to some specific baseline methodologies, such as hierarchical, center-based, and search-based methods. We shall, of course, start with the common ground and knowledge for cluster analysis, including the classification of data and the corresponding similarity measures, and we shall also provide examples of clustering applications to illustrate the advantages and shortcomings of different clustering architectures and algorithms.

This monograph is intended not only for statistics, applied mathematics, and computer science senior undergraduates and graduates, but also for research scientists who need cluster analysis to deal with data. It may be used as a textbook for introductory courses in cluster analysis or as source material for an introductory course in data mining at the graduate level. We assume that the reader is familiar with elementary linear algebra, calculus, and basic statistical concepts and methods.

The book is divided into four parts: basic concepts (clustering, data, and similarity measures), algorithms, applications, and programming languages. We now briefly describe the content of each chapter.

Chapter 1. Data clustering. In this chapter, we introduce the basic concepts of clustering. Cluster analysis is defined as a way to create groups of objects, or clusters, in such a way that objects in one cluster are very similar and objects in different clusters are quite distinct. Some working definitions of clusters are discussed, and several popular books relevant to cluster analysis are introduced.

Chapter 2. Data types. The type of data is directly associated with data clustering, and it is a major factor to consider in choosing an appropriate clustering algorithm. Five data types are discussed in this chapter: categorical, binary, transaction, symbolic, and time series. They share a common feature that nonnumerical similarity measures must be used. There are many other data types, such as image data, that are not discussed here, though we believe that once readers get familiar with these basic types of data, they should be able to adjust the algorithms accordingly.

Chapter 3. Scale conversion. Scale conversion is concerned with the transformation between different types of variables. For example, one may convert a continuous measured variable to an interval variable. In this chapter, we first review several scale conversion techniques and then discuss several approaches for categorizing numerical data.

Chapter 4. Data standardization and transformation. In many situations, raw data should be normalized and/or transformed before a cluster analysis. One reason to do this is that objects in raw data may be described by variables measured with different scales; another reason is to reduce the size of the data to improve the effectiveness of clustering algorithms. Therefore, we present several data standardization and transformation techniques in this chapter.

Chapter 5. Data visualization. Data visualization is vital in the final step of data-mining applications. This chapter introduces various techniques of visualization with an emphasis on visualization of clustered data. Some dimension reduction techniques, such as multidimensional scaling (MDS) and self-organizing maps (SDMs), are discussed.

Chapter 6. Similarity and dissimilarity measures. In the literature of data clustering, a similarity measure or distance (dissimilarity measure) is used to quantitatively describe the similarity or dissimilarity of two data points or two clusters. Similarity and distance measures are basic elements of a clustering algorithm, without which no meaningful cluster analysis is possible. Due to the important role of similarity and distance measures in cluster analysis, we present a comprehensive discussion of different measures for various types of data in this chapter. We also introduce measures between points and measures between clusters.

Chapter 7. Hierarchical clustering techniques. Hierarchical clustering algorithms and partitioning algorithms are two major clustering algorithms. Unlike partitioning algorithms, which divide a data set into a single partition, hierarchical algorithms divide a data set into a sequence of nested partitions. There are two major hierarchical algorithms: agglomerative algorithms and divisive algorithms. Agglomerative algorithms start with every single object in a single cluster, while divisive ones start with all objects in one cluster and repeat splitting large clusters into small pieces. In this chapter, we present representations of hierarchical clustering and several popular hierarchical clustering algorithms.

Chapter 8. Fuzzy clustering algorithms. Clustering algorithms can be classified into two categories: hard clustering algorithms and fuzzy clustering algorithms. Unlike hard clustering algorithms, which require that each data point of the data set belong to one and only one cluster, fuzzy clustering algorithms allow a data point to belong to two or more clusters with different probabilities. There is also a huge number of published works related to fuzzy clustering. In this chapter, we review some basic concepts of fuzzy logic and present three well-known fuzzy clustering algorithms: fuzzy k -means, fuzzy k -modes, and c -means.

Chapter 9. Center-based clustering algorithms. Compared to other types of clustering algorithms, center-based clustering algorithms are more suitable for clustering large data sets and high-dimensional data sets. Several well-known center-based clustering algorithms (e.g., k -means, k -modes) are presented and discussed in this chapter.

Chapter 10. Search-based clustering algorithms. A well-known problem associated with most of the clustering algorithms is that they may not be able to find the globally optimal clustering that fits the data set, since these algorithms will stop if they find a local optimal partition of the data set. This problem led to the invention of search-based clus-

tering algorithms to search the solution space and find a globally optimal clustering that fits the data set. In this chapter, we present several clustering algorithms based on genetic algorithms, tabu search algorithms, and simulated annealing algorithms.

Chapter 11. Graph-based clustering algorithms. Graph-based clustering algorithms cluster a data set by clustering the graph or hypergraph constructed from the data set. The construction of a graph or hypergraph is usually based on the dissimilarity matrix of the data set under consideration. In this chapter, we present several graph-based clustering algorithms that do not use the spectral graph partition techniques, although we also list a few references related to spectral graph partition techniques.

Chapter 12. Grid-based clustering algorithms. In general, a grid-based clustering algorithm consists of the following five basic steps: partitioning the data space into a finite number of cells (or creating grid structure), estimating the cell density for each cell, sorting the cells according to their densities, identifying cluster centers, and traversal of neighbor cells. A major advantage of grid-based clustering is that it significantly reduces the computational complexity. Some recent works on grid-based clustering are presented in this chapter.

Chapter 13. Density-based clustering algorithms. The density-based clustering approach is capable of finding arbitrarily shaped clusters, where clusters are defined as dense regions separated by low-density regions. Usually, density-based clustering algorithms are not suitable for high-dimensional data sets, since data points are sparse in high-dimensional spaces. Five density-based clustering algorithms (DBSCAN, BRIDGE, DBCLASD, DENCLUE, and CUBN) are presented in this chapter.

Chapter 14. Model-based clustering algorithms. In the framework of model-based clustering algorithms, the data are assumed to come from a mixture of probability distributions, each of which represents a different cluster. There is a huge number of published works related to model-based clustering algorithms. In particular, there are more than 400 articles devoted to the development and discussion of the expectation-maximization (EM) algorithm. In this chapter, we introduce model-based clustering and present two model-based clustering algorithms: COOLCAT and STUCCO.

Chapter 15. Subspace clustering. Subspace clustering is a relatively new concept. After the first subspace clustering algorithm, CLIQUE, was published by the IBM group, many subspace clustering algorithms were developed and studied. One feature of the subspace clustering algorithms is that they are capable of identifying different clusters embedded in different subspaces of the high-dimensional data. Several subspace clustering algorithms are presented in this chapter, including the neural network-inspired algorithm PART.

Chapter 16. Miscellaneous algorithms. This chapter introduces some clustering algorithms for clustering time series, data streams, and transaction data. Proximity measures for these data and several related clustering algorithms are presented.

Chapter 17. Evaluation of clustering algorithms. Clustering is an unsupervised process and there are no predefined classes and no examples to show that the clusters found by the clustering algorithms are valid. Usually one or more validity criteria, presented in this chapter, are required to verify the clustering result of one algorithm or to compare the clustering results of different algorithms.

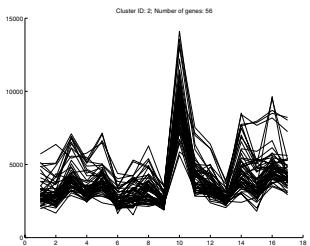
Chapter 18. Clustering gene expression data. As an application of cluster analysis, gene expression data clustering is introduced in this chapter. The background and similarity

measures for gene expression data are introduced. Clustering a real set of gene expression data with the fuzzy subspace clustering (FSC) algorithm is presented.

Chapter 19. Data clustering in MATLAB. In this chapter, we show how to perform clustering in MATLAB in the following three aspects. Firstly, we introduce some MATLAB commands related to file operations, since the first thing to do about clustering is to load data into MATLAB, and data are usually stored in a text file. Secondly, we introduce MATLAB M-files, MEX-files, and MAT-files in order to demonstrate how to code algorithms and save current work. Finally, we present several MATLAB codes, which can be found in Appendix C.

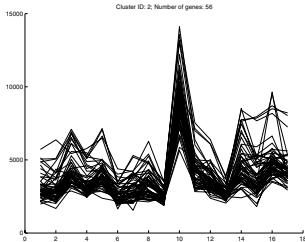
Chapter 20. Clustering in C/C++. C++ is an object-oriented programming language built on the C language. In this last chapter of the book, we introduce the Standard Template Library (STL) in C++ and C/C++ program compilation. C++ data structure for data clustering is introduced. This chapter assumes that readers have basic knowledge of the C/C++ language.

This monograph has grown and evolved from a few collaborative projects for industrial applications undertaken by the Laboratory for Industrial and Applied Mathematics at York University, some of which are in collaboration with Generation 5 Mathematical Technologies, Inc. We would like to thank the Canada Research Chairs Program, the Natural Sciences and Engineering Research Council of Canada's Discovery Grant Program and Collaborative Research Development Program, and Mathematics for Information Technology and Complex Systems for their support.



Part I

Clustering, Data, and Similarity Measures



Chapter 1

Data Clustering

This chapter introduces some basic concepts. First, we describe what data clustering is and give several examples from biology, health care, market research, image processing, and data mining. Then we introduce the notions of *records*, *attributes*, *distances*, *similarities*, *centers*, *clusters*, and *validity indices*. Finally, we discuss how cluster analysis is done and summarize the major phases involved in clustering a data set.

1.1 Definition of Data Clustering

Data clustering (or just clustering), also called cluster analysis, segmentation analysis, taxonomy analysis, or unsupervised classification, is a method of creating groups of objects, or clusters, in such a way that objects in one cluster are very similar and objects in different clusters are quite distinct. Data clustering is often confused with classification, in which objects are assigned to predefined classes. In data clustering, the classes are also to be defined. To elaborate the concept a little bit, we consider several examples.

Example 1.1 (Cluster analysis for gene expression data). Clustering is one of the most frequently performed analyses on gene expression data (Yeung et al., 2003; Eisen et al., 1998). Gene expression data are a set of measurements collected via the cDNA microarray or the oligo-nucleotide chip experiment (Jiang et al., 2004). A gene expression data set can be represented by a real-valued expression matrix

$$D = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix},$$

where n is the number of genes, d is the number of experimental conditions or samples, and x_{ij} is the measured expression level of gene i in sample j . Since the original gene expression matrix contains noise, missing values, and systematic variations, preprocessing is normally required before cluster analysis can be performed.

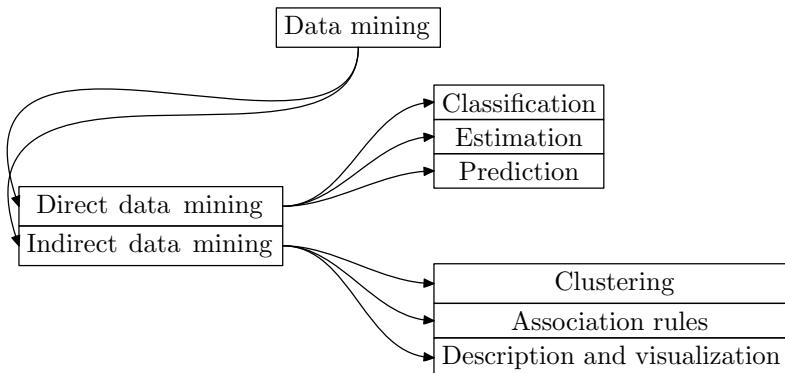


Figure 1.1. Data-mining tasks.

Gene expression data can be clustered in two ways. One way is to group genes with similar expression patterns, i.e., clustering the rows of the expression matrix D . Another way is to group different samples on the basis of corresponding expression profiles, i.e., clustering the columns of the expression matrix D . ■

Example 1.2 (Clustering in health psychology). Cluster analysis has been applied to many areas of health psychology, including the promotion and maintenance of health, improvement to the health care system, and prevention of illness and disability (Clatworthy et al., 2005). In health care development systems, cluster analysis is used to identify groups of people that may benefit from specific services (Hodges and Wotring, 2000). In health promotion, cluster analysis is used to select target groups that will most likely benefit from specific health promotion campaigns and to facilitate the development of promotional material. In addition, cluster analysis is used to identify groups of people at risk of developing medical conditions and those at risk of poor outcomes. ■

Example 1.3 (Clustering in market research). In market research, cluster analysis has been used to segment the market and determine target markets (Christopher, 1969; Saunders, 1980; Frank and Green, 1968). In market segmentation, cluster analysis is used to break down markets into meaningful segments, such as men aged 21–30 and men over 51 who tend not to buy new products. ■

Example 1.4 (Image segmentation). Image segmentation is the decomposition of a gray-level or color image into homogeneous tiles (Comaniciu and Meer, 2002). In image segmentation, cluster analysis is used to detect borders of objects in an image. ■

Clustering constitutes an essential component of so-called data mining, a process of exploring and analyzing large amounts of data in order to discover useful information (Berry and Linoff, 2000). Clustering is also a fundamental problem in the literature of pattern recognition. Figure 1.1 gives a schematic list of various data-mining tasks and indicates the role of clustering in data mining.

In general, useful information can be discovered from a large amount of data through automatic or semiautomatic means (Berry and Linoff, 2000). In indirect data mining, no

variable is singled out as a target, and the goal is to discover some relationships among all the variables, while in directed data mining, some variables are singled out as targets. Data clustering is indirect data mining, since in data clustering, we are not exactly sure what clusters we are looking for, what plays a role in forming these clusters, and how it does that.

The clustering problem has been addressed extensively, although there is no uniform definition for data clustering and there may never be one (Estivill-Castro, 2002; Dubes, 1987; Fraley and Raftery, 1998). Roughly speaking, by data clustering, we mean that for a given set of data points and a similarity measure, we regroup the data such that data points in the same group are similar and data points in different groups are dissimilar. Obviously, this type of problem is encountered in many applications, such as text mining, gene expressions, customer segmentations, and image processing, to name just a few.

1.2 The Vocabulary of Clustering

Now we introduce some concepts that will be encountered frequently in cluster analysis.

1.2.1 Records and Attributes

In the literature of data clustering, different words may be used to express the same thing. For instance, given a database that contains many records, the terms *data point*, *pattern case*, *observation*, *object*, *individual*, *item*, and *tuple* are all used to denote a single data item. In this book, we will use *record*, *object*, or *data point* to denote a single record. Also, for a data point in a high-dimensional space, we shall use *variable*, *attribute*, or *feature* to denote an individual scalar component (Jain et al., 1999). In this book, we almost always use the standard data structure in statistics, i.e., the cases-by-variables data structure (Hartigan, 1975).

Mathematically, a data set with n objects, each of which is described by d attributes, is denoted by $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})^T$ is a vector denoting the i th object and x_{ij} is a scalar denoting the j th component or attribute of \mathbf{x}_i . The number of attributes d is also called the dimensionality of the data set.

1.2.2 Distances and Similarities

Distances and similarities play an important role in cluster analysis (Jain and Dubes, 1988; Anderberg, 1973). In the literature of data clustering, similarity measures, similarity coefficients, dissimilarity measures, or distances are used to describe quantitatively the similarity or dissimilarity of two data points or two clusters.

In general, distance and similarity are reciprocal concepts. Often, similarity measures and similarity coefficients are used to describe quantitatively how similar two data points are or how similar two clusters are: the greater the similarity coefficient, the more similar are the two data points. Dissimilarity measure and distance are the other way around: the greater the dissimilarity measure or distance, the more dissimilar are the two data points or the two clusters. Consider the two data points $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ and $\mathbf{y} = (y_1, y_2, \dots, y_d)^T$, for

example. The Euclidean distance between \mathbf{x} and \mathbf{y} is calculated as

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^d (x_j - y_j)^2 \right)^{\frac{1}{2}}.$$

Every clustering algorithm is based on the index of similarity or dissimilarity between data points (Jain and Dubes, 1988). If there is no measure of similarity or dissimilarity between pairs of data points, then no meaningful cluster analysis is possible. Various similarity and dissimilarity measures have been discussed by Sokal and Sneath (1973), Legendre and Legendre (1983), Anderberg (1973), Gordon (1999), and Everitt et al. (2001). In this book, various distances and similarities are presented in Chapter 6.

1.2.3 Clusters, Centers, and Modes

In cluster analysis, the terms *cluster*, *group*, and *class* have been used in an essentially intuitive manner without a uniform definition (Everitt, 1993). Everitt (1993) suggested that if using a term such as *cluster* produces an answer of value to the investigators, then it is all that is required. Generally, the common sense of a cluster will combine various plausible criteria and require (Bock, 1989), for example, all objects in a cluster to

1. share the same or closely related properties;
2. show small mutual distances or dissimilarities;
3. have “contacts” or “relations” with at least one other object in the group; or
4. be clearly distinguishable from the complement, i.e., the rest of the objects in the data set.

Carmichael et al. (1968) also suggested that the set contain clusters of points if the distribution of the points meets the following conditions:

1. There are continuous and relative densely populated regions of the space.
2. These are surrounded by continuous and relatively empty regions of the space.

For numerical data, Lorr (1983) suggested that there appear to be two kinds of clusters: compact clusters and chained clusters. A compact cluster is a set of data points in which members have high mutual similarity. Usually, a compact cluster can be represented by a representative point or center. Figure 1.2, for example, gives three compact clusters in a two-dimensional space. The clusters shown in Figure 1.2 are well separated and each can be represented by its center. Further discussions can be found in Michaud (1997). For categorical data, a mode is used to represent a cluster (Huang, 1998).

A chained cluster is a set of data points in which every member is more like other members in the cluster than other data points not in the cluster. More intuitively, any two data points in a chained cluster are reachable through a path, i.e., there is a path that connects the two data points in the cluster. For example, Figure 1.3 gives two chained clusters—one looks like a rotated “T,” while the other looks like an “O.”

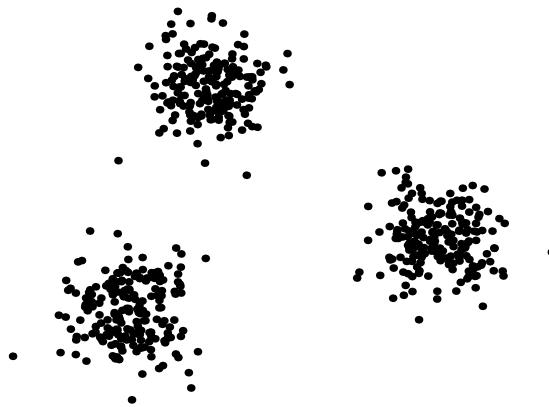


Figure 1.2. Three well-separated center-based clusters in a two-dimensional space.

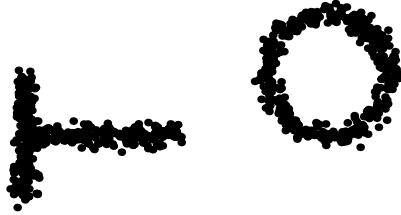


Figure 1.3. Two chained clusters in a two-dimensional space.

1.2.4 Hard Clustering and Fuzzy Clustering

In hard clustering, algorithms assign a class label $l_i \in \{1, 2, \dots, k\}$ to each object \mathbf{x}_i to identify its cluster class, where k is the number of clusters. In other words, in hard clustering, each object is assumed to belong to one and only one cluster.

Mathematically, the result of hard clustering algorithms can be represented by a $k \times n$ matrix

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{k1} & u_{n2} & \cdots & u_{kn} \end{pmatrix}, \quad (1.1)$$

where n denotes the number of records in the data set, k denotes the number of clusters, and u_{ji} satisfies

$$u_{ji} \in \{0, 1\}, \quad 1 \leq j \leq k, \quad 1 \leq i \leq n, \quad (1.2a)$$

$$\sum_{j=1}^k u_{ji} = 1, \quad 1 \leq i \leq n, \quad (1.2b)$$

$$\sum_{i=1}^n u_{ji} > 0, \quad 1 \leq j \leq k. \quad (1.2c)$$

Constraint (1.2a) implies that each object either belongs to a cluster or not. Constraint (1.2b) implies that each object belongs to only one cluster. Constraint (1.2c) implies that each cluster contains at least one object, i.e., no empty clusters are allowed. We call $U = (u_{ji})$ defined in equation (1.2) a hard k -partition of the data set D .

In fuzzy clustering, the assumption is relaxed so that an object can belong to one or more clusters with probabilities. The result of fuzzy clustering algorithms can also be represented by a $k \times n$ matrix U defined in equation (1.2) with the following relaxed constraints:

$$u_{ji} \in [0, 1], \quad 1 \leq j \leq k, \quad 1 \leq i \leq n, \quad (1.3a)$$

$$\sum_{j=1}^k u_{ji} = 1, \quad 1 \leq i \leq n, \quad (1.3b)$$

$$\sum_{i=1}^n u_{ji} > 0, \quad 1 \leq j \leq k. \quad (1.3c)$$

Similarly, we call $U = (u_{ji})$ defined in equation (1.3) a fuzzy k -partition.

1.2.5 Validity Indices

Since clustering is an unsupervised process and most of the clustering algorithms are very sensitive to their initial assumptions, some sort of evaluation is required to assess the clustering results in most of the applications. Validity indices are measures that are used to evaluate and assess the results of a clustering algorithm. In Chapter 17, we shall introduce some validity indices for cluster analysis.

1.3 Clustering Processes

As a fundamental pattern recognition problem, a well-designed clustering algorithm usually involves the following four design phases: data representation, modeling, optimization, and validation (Buhmann, 2003) (see Figure 1.4). The data representation phase predetermines what kind of cluster structures can be discovered in the data. On the basis of data representation, the modeling phase defines the notion of clusters and the criteria that separate desired group structures from unfavorable ones. For numerical data, for example, there are at least two aspects to the choice of a cluster structural model: compact (spherical or ellipsoidal) clusters and extended (serpentine) clusters (Lorr, 1983). In the modeling phase, a quality measure that can be either optimized or approximated during the search for hidden structures in the data is produced.

The goal of clustering is to assign data points with similar properties to the same groups and dissimilar data points to different groups. Generally, clustering problems can be divided into two categories (see Figure 1.5): hard clustering (or crisp clustering) and fuzzy clustering (or soft clustering). In hard clustering, a data point belongs to one and only one cluster, while in fuzzy clustering, a data point may belong to two or more clusters with some probabilities. Mathematically, a clustering of a given data set D can be represented

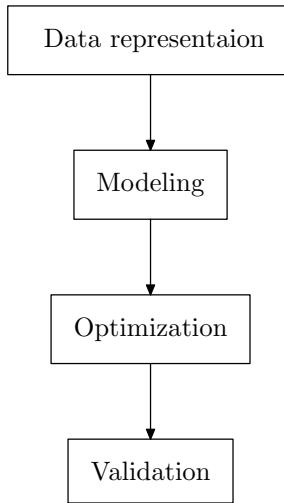


Figure 1.4. Processes of data clustering.

by an assignment function $f : D \rightarrow [0, 1]^k$, $\mathbf{x} \rightarrow f(\mathbf{x})$, defined as follows:

$$f(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_k(\mathbf{x}) \end{pmatrix}, \quad (1.4)$$

where $f_i(\mathbf{x}) \in [0, 1]$ for $i = 1, 2, \dots, k$ and $\mathbf{x} \in D$, and

$$\sum_{i=1}^k f_i(\mathbf{x}) = 1 \quad \forall \mathbf{x} \in D.$$

If for every $\mathbf{x} \in D$, $f_i(\mathbf{x}) \in \{0, 1\}$, then the clustering represented by f is a hard clustering; otherwise, it is a fuzzy clustering.

In general, conventional clustering algorithms can be classified into two categories: hierarchical algorithms and partitional algorithms. There are two types of hierarchical algorithms: divisive hierarchical algorithms and agglomerative hierarchical algorithms. In a divisive hierarchical algorithm, the algorithm proceeds from the top to the bottom, i.e., the algorithm starts with one large cluster containing all the data points in the data set and continues splitting clusters; in an agglomerative hierarchical algorithm, the algorithm proceeds from the bottom to the top, i.e., the algorithm starts with clusters each containing one data point and continues merging the clusters. Unlike hierarchical algorithms, partitioning algorithms create a one-level nonoverlapping partitioning of the data points.

For large data sets, hierarchical methods become impractical unless other techniques are incorporated, because usually hierarchical methods are $O(n^2)$ for memory space and $O(n^3)$ for CPU time (Zait and Messatfa, 1997; Hartigan, 1975; Murtagh, 1983), where n is the number of data points in the data set.

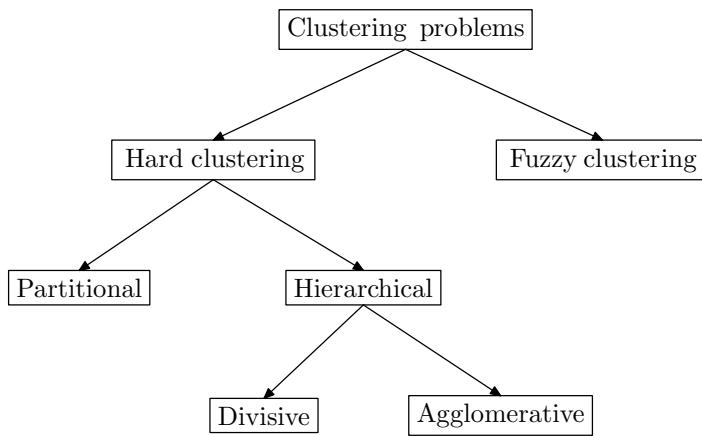


Figure 1.5. Diagram of clustering algorithms.

Although some theoretical investigations have been made for general clustering problems (Fisher, 1958; Friedman and Rubin, 1967; Jardine and Sibson, 1968), most clustering methods have been developed and studied for specific situations (Rand, 1971). Examples illustrating various aspects of cluster analysis can be found in Morgan (1981).

1.4 Dealing with Missing Values

In real-world data sets, we often encounter two problems: some important data are missing in the data sets, and there might be errors in the data sets. In this section, we discuss and present some existing methods for dealing with missing values.

In general, there are three cases according to how missing values can occur in data sets (Fujikawa and Ho, 2002):

1. Missing values occur in several variables.
2. Missing values occur in a number of records.
3. Missing values occur randomly in variables and records.

If there exists a record or a variable in the data set for which all measurements are missing, then there is really no information on this record or variable, so the record or variable has to be removed from the data set (Kaufman and Rousseeuw, 1990). If there are not many missing values on records or variables, the methods to deal with missing values can be classified into two groups (Fujikawa and Ho, 2002):

- (a) prereplacing methods, which replace missing values before the data-mining process;
- (b) embedded methods, which deal with missing values during the data-mining process.

A number of methods for dealing with missing values have been presented in (Fujikawa and Ho, 2002). Also, three cluster-based algorithms to deal with missing values have been proposed based on the mean-and-mode method in (Fujikawa and Ho, 2002):

Table 1.1. A list of methods for dealing with missing values.

<i>Method</i>	<i>Group</i>	<i>Attribute</i>	<i>Case</i>	<i>Cost</i>
Mean-and-mode method	(a)	Num & Cat	(2)	Low
Linear regression	(a)	Num	(2)	Low
Standard deviation method	(a)	Num	(2)	Low
Nearest neighbor estimator	(a)	Num & Cat	(1)	High
Decision tree imputation	(a)	Cat	(1)	Middle
Autoassociative neural network	(a)	Num & Cat	(1)	High
Casewise deletion	(b)	Num & Cat	(2)	Low
Lazy decision tree	(b)	Num & Cat	(1)	High
Dynamic path generation	(b)	Num & Cat	(1)	High
C4.5	(b)	Num & Cat	(1)	Middle
Surrogate split	(b)	Num & Cat	(1)	Middle

NCBMM (Natural Cluster Based Mean-and-Mode algorithm), RCBMM (attribute Rank Cluster Based Mean-and-Mode algorithm) and KMCMM (k -Means Cluster-Based Mean-and-Mode algorithm). NCBMM is a method of filling in missing values in case of supervised data. NCBMM uses the class attribute to divide objects into natural clusters and uses the mean or mode of each cluster to fill in the missing values of objects in that cluster depending on the type of attribute. Since most clustering applications are unsupervised, the NCBMM method cannot be applied directly. The last two methods, RCBMM and KMCMM, can be applied to both supervised and unsupervised data clustering.

RCBMM is a method of filling in missing values for categorical attributes and is independent of the class attribute. This method consists of three steps. Given a missing attribute a , at the first step, this method ranks all categorical attributes by their distance from the missing value attribute a . The attribute with the smallest distance is used for clustering. At the second step, all records are divided into clusters, each of which contains records with the same value of the selected attribute. Finally, the mode of each cluster is used to fill in the missing values. This process is applied to each missing attribute. The distance between two attributes can be computed using the method proposed in (Mántaras, 1991) (see Section 6.9).

KMCMM is a method of filling in missing values for numerical attributes and is independent of the class attribute. It also consists of three steps. Given a missing attribute a , firstly, the algorithm ranks all the numerical attributes in increasing order of absolute correlation coefficients between them and the missing attribute a . Secondly, the objects are divided into k clusters by the k -means algorithm based on the values of a . Thirdly, the missing value on attribute a is replaced by the mean of each cluster. This process is applied to each missing attribute.

Cluster-based methods to deal with missing values and errors in data have also been discussed in (Lee et al., 1976). Other discussions about missing values and errors have been presented in (Wu and Barbará, 2002) and (Wishart, 1978).

1.5 Resources for Clustering

In the past 50 years, there has been an explosion in the development and publication of cluster-analytic techniques published in a wide range of technical journals. Here we list some survey papers, books, journals, and conference proceedings on which our book is based.

1.5.1 Surveys and Reviews on Clustering

Several surveys and reviews related to cluster analysis have been published. The following list of survey papers may be interesting to readers.

1. *A review of hierarchical classification* by Gordon (1987)
2. *A review of classification* by Cormack (1971)
3. *A survey of fuzzy clustering* by Yang (1993)
4. *A survey of fuzzy clustering algorithms for pattern recognition. I* by Baraldi and Blonda (1999a)
5. *A survey of fuzzy clustering algorithms for pattern recognition. II* by Baraldi and Blonda (1999b)
6. *A survey of recent advances in hierarchical clustering algorithms* by Murtagh (1983)
7. *Cluster analysis for gene expression data: A survey* by Jiang et al. (2004)
8. *Counting dendrograms: A survey* by Murtagh (1984b)
9. *Data clustering: A review* by Jain et al. (1999)
10. *Mining data streams: A review* by Gaber et al. (2005)
11. *Statistical pattern recognition: A review* by Jain et al. (2000)
12. *Subspace clustering for high dimensional data: A review* by Parsons et al. (2004b)
13. *Survey of clustering algorithms* by Xu and Wunsch II (2005)

1.5.2 Books on Clustering

Several books on cluster analysis have been published. The following list of books may be helpful to readers.

1. *Principles of Numerical Taxonomy*, published by Sokal and Sneath (1963), reviews most of the applications of numerical taxonomy in the field of biology at that time. *Numerical Taxonomy: The Principles and Practice of Numerical Classification* by Sokal and Sneath (1973) is a new edition of *Principles of Numerical Taxonomy*. Although directed toward researchers in the field of biology, the two books review much of the literature of cluster analysis and present many clustering techniques available at that time.
2. *Cluster Analysis: Survey and Evaluation of Techniques* by Bijnen (1973) selected a number of clustering techniques related to sociological and psychological research.

3. *Cluster Analysis: A Survey* by Duran and Odell (1974) supplies an exposition of various works in the literature of cluster analysis at that time. Many references that played a role in developing the theory of cluster analysis are contained in the book.
4. *Cluster Analysis for Applications* by Anderberg (1973) collects many clustering techniques and provides many FORTRAN procedures for readers to perform analysis of real data.
5. *Clustering Algorithms* by Hartigan (1975) is a book presented from the statistician's point of view. A wide range of procedures, methods, and examples is presented. Also, some FORTRAN programs are provided.
6. *Cluster Analysis for Social Scientists* by Lorr (1983) is a book on cluster analysis written at an elementary level for researchers and graduate students in the social and behavioral sciences.
7. *Algorithms for Clustering Data* by Jain and Dubes (1988) is a book written for the scientific community that emphasizes informal algorithms for clustering data and interpreting results.
8. *Introduction to Statistical Pattern Recognition* by Fukunaga (1990) introduces fundamental mathematical tools for the supervised clustering classification. Although written for classification, this book presents clustering (unsupervised classification) based on statistics.
9. *Cluster Analysis* by Everitt (1993) introduces cluster analysis for works in a variety of areas. Many examples of clustering are provided in the book. Also, several software programs for clustering are described in the book.
10. *Clustering for Data Mining: A Data Recovery Approach* by Mirkin (2005) introduces data recovery models based on the k -means algorithm and hierarchical algorithms. Some clustering algorithms are reviewed in this book.

1.5.3 Journals

Articles on cluster analysis are published in a wide range of technical journals. The following is a list of journals in which articles on cluster analysis are usually published.

1. *ACM Computing Surveys*
2. *ACM SIGKDD Explorations Newsletter*
3. *The American Statistician*
4. *The Annals of Probability*
5. *The Annals of Statistics*
6. *Applied Statistics*
7. *Bioinformatics*
8. *Biometrics*
9. *Biometrika*
10. *BMC Bioinformatics*
11. *British Journal of Health Psychology*
12. *British Journal of Marketing*

13. *Computer*
14. *Computers & Mathematics with Applications*
15. *Computational Statistics and Data Analysis*
16. *Discrete and Computational Geometry*
17. *The Computer Journal*
18. *Data Mining and Knowledge Discovery*
19. *Engineering Applications of Artificial Intelligence*
20. *European Journal of Operational Research*
21. *Future Generation Computer Systems*
22. *Fuzzy Sets and Systems*
23. *Genome Biology*
24. *Knowledge and Information Systems*
25. *The Indian Journal of Statistics*
26. *IEEE Transactions on Evolutionary Computation*
27. *IEEE Transactions on Information Theory*
28. *IEEE Transactions on Image Processing*
29. *IEEE Transactions on Knowledge and Data Engineering*
30. *IEEE Transactions on Neural Networks*
31. *IEEE Transactions on Pattern Analysis and Machine Intelligence*
32. *IEEE Transactions on Systems, Man, and Cybernetics*
33. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*
34. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*
35. *Information Sciences*
36. *Journal of the ACM*
37. *Journal of the American Society for Information Science*
38. *Journal of the American Statistical Association*
39. *Journal of the Association for Computing Machinery*
40. *Journal of Behavioral Health Services and Research*
41. *Journal of Chemical Information and Computer Sciences*
42. *Journal of Classification*
43. *Journal of Complexity*
44. *Journal of Computational and Applied Mathematics*
45. *Journal of Computational and Graphical Statistics*
46. *Journal of Ecology*
47. *Journal of Global Optimization*
48. *Journal of Marketing Research*
49. *Journal of the Operational Research Society*
50. *Journal of the Royal Statistical Society. Series A (General)*
51. *Journal of the Royal Statistical Society. Series B (Methodological)*

52. *Journal of Software*
53. *Journal of Statistical Planning and Inference*
54. *Journal of Statistical Software*
55. *Lecture Notes in Computer Science*
56. *Los Alamos Science*
57. *Machine Learning*
58. *Management Science*
59. *Management Science (Series B, Managerial)*
60. *Mathematical and Computer Modelling*
61. *Mathematical Biosciences*
62. *Medical Science Monitor*
63. *NECTEC Technical Journal*
64. *Neural Networks*
65. *Operations Research*
66. *Pattern Recognition*
67. *Pattern Recognition Letters*
68. *Physical Review Letters*
69. *SIAM Journal on Scientific Computing*
70. *SIGKDD, Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining*
71. *SIGMOD Record*
72. *The Statistician*
73. *Statistics and Computing*
74. *Systematic Zoology*
75. *The VLDB Journal*
76. *World Archaeology*

1.5.4 Conference Proceedings

The following is a list of conferences related to data clustering. Many technical papers are published in their conference proceedings.

1. ACM Conference on Information and Knowledge Management
2. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)
3. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
4. ACM SIGMOD International Conference on Management of Data
5. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery
6. ACM Symposium on Applied Computing
7. Advances in Neural Information Processing Systems

8. Annual ACM Symposium on Theory of Computing
9. Annual ACM-SIAM Symposium on Discrete Algorithms
10. Annual European Symposium on Algorithms
11. Annual Symposium on Computational Geometry
12. Congress on Evolutionary Computation
13. IEEE Annual Northeast Bioengineering Conference
14. IEEE Computer Society Conference on Computer Vision and Pattern Recognition
15. IEEE International Conference on Acoustics, Speech, and Signal Processing
16. IEEE International Conference on Computer Vision
17. IEEE International Conference on Data Engineering
18. IEEE International Conference on Data Mining
19. IEEE International Conference on Fuzzy Systems
20. IEEE International Conference on Systems, Man, and Cybernetics
21. IEEE International Conference on Tools with Artificial Intelligence
22. IEEE International Symposium on Information Theory
23. IEEE Symposium on Bioinformatics and Bioengineering
24. International Conference on Advanced Data Mining and Applications
25. International Conference on Extending Database Technology
26. International Conference on Data Warehousing and Knowledge Discovery
27. International Conference on Database Systems for Advanced Applications
28. International Conference on Image Processing
29. International Conferences on Info-tech and Info-net
30. International Conference on Information and Knowledge Management
31. International Conference on Machine Learning
32. International Conference on Machine Learning and Cybernetics
33. International Conference on Neural Networks
34. International Conference on Parallel Computing in Electrical Engineering
35. International Conference on Pattern Recognition
36. International Conference on Signal Processing
37. International Conference on Software Engineering
38. International Conference on Very Large Data Bases
39. International Geoscience and Remote Sensing Symposium
40. International Joint Conference on Neural Networks
41. International Workshop on Algorithm Engineering and Experimentation
42. IPPS/SPDP Workshop on High Performance Data Mining
43. Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining
44. SIAM International Conference on Data Mining
45. World Congress on Intelligent Control and Automation

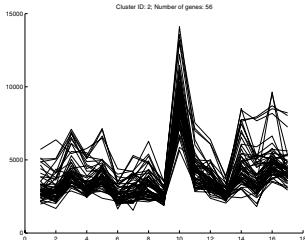
1.5.5 Data Sets

Once a clustering algorithm is developed, how it works should be tested by various data sets. In this sense, testing data sets plays an important role in the process of algorithm development. Here we give a list of websites on which real data sets can be found.

1. <http://kdd.ics.uci.edu/> The UCI Knowledge Discovery in Databases Archive (Hettich and Bay, 1999) is an online repository of large data sets that encompasses a wide variety of data types, analysis tasks, and application areas.
2. <http://lib.stat.cmu.edu/DASL/> The Data and Story Library (DASL) is an online library of data files and stories that illustrate the use of basic statistical methods. Several data sets are analyzed by cluster analysis methods.
3. <http://www.datasetgenerator.com/> This site hosts a computer program that produces data for the testing of data-mining classification programs.
4. <http://www.kdnuggets.com/datasets/index.html> This site maintains a list of data sets for data mining.

1.6 Summary

This chapter introduced some basic concepts of data clustering and the clustering process. In addition, this chapter presented some resources for cluster analysis, including some existing books, technical journals, conferences related to clustering, and data sets for testing clustering algorithms. Readers should now be familiar with the basic concepts of clustering. For more discussion of cluster analysis, readers are referred to Jain et al. (1999), Murtagh (1983), Cormack (1971), and Gordon (1987).



Chapter 2 Data Types

Data-clustering algorithms are very much associated with data types. Therefore, understanding scale, normalization, and proximity is very important in interpreting the results of clustering algorithms. Data type refers to the degree of quantization in the data (Jain and Dubes, 1988; Anderberg, 1973)—a single attribute can be typed as binary, discrete, or continuous. A binary attribute has exactly two values, such as true or false. A discrete attribute has a finite number of possible values, thus binary types are a special case of discrete types (see Figure 2.1).

Data scales, which indicate the relative significance of numbers, are also an important issue in clustering. Data scales can be divided into qualitative scales and quantitative scales. Qualitative scales include nominal scales and ordinal scales; quantitative scales include interval scales and ratio scales (see Figure 2.2). Details of data types will be considered in this chapter.

2.1 Categorical Data

Categorical attributes are also referred to as nominal attributes, which are simply used as names, such as the brands of cars and names of bank branches. Since we consider data sets

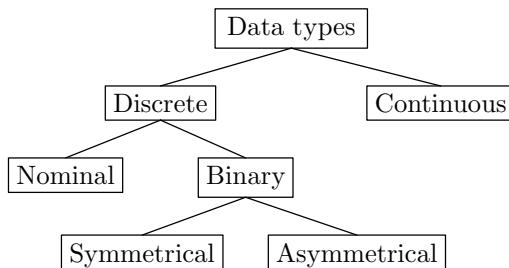


Figure 2.1. Diagram of data types.

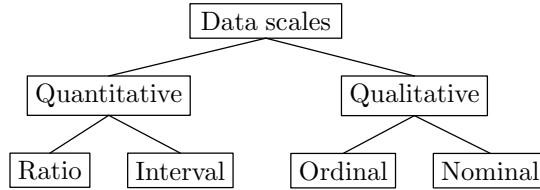


Figure 2.2. Diagram of data scales.

Table 2.1. A sample categorical data set.

	Records	Values
\mathbf{x}_1		(A, A, A, A, B, B)
\mathbf{x}_2		(A, A, A, A, C, D)
\mathbf{x}_3		(A, A, A, A, D, C)
\mathbf{x}_4		(B, B, C, C, D, C)
\mathbf{x}_5		(B, B, D, D, C, D)

with a finite number of data points, a nominal attribute of the data points in the data set can have only a finite number of values; thus the nominal type is also a special case of the discrete type.

In this section, we shall introduce symbol tables and frequency tables and some notation for categorical data sets, which will be used throughout this book.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a categorical data set with n instances, each of which is described by d categorical attributes v_1, v_2, \dots, v_d . Let $\text{DOM}(v_j)$ denote the domain of the attribute v_j . In the categorical data set given in Table 2.1, for example, the domains of v_1 and v_4 are $\text{DOM}(v_1) = \{A, B\}$ and $\text{DOM}(v_4) = \{A, C, D\}$, respectively.

For a given categorical data set D , we suppose that $\text{DOM}(v_j) = \{A_{j1}, A_{j2}, \dots, A_{jn_j}\}$ for $j = 1, 2, \dots, d$. We call A_{jl} ($1 \leq l \leq n_j$) a state of the categorical attribute v_j , and n_j is the number of states of v_j in the given data set D . Then a symbol table T_s of the data set is defined as

$$T_s = (\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_d), \quad (2.1)$$

where \mathbf{s}_j ($1 \leq j \leq d$) is a vector defined as $\mathbf{s}_j = (A_{j1}, A_{j2}, \dots, A_{jn_j})^T$.

Since there are possibly multiple states (or values) for a variable, a symbol table of a data set is usually not unique. For example, for the data set in Table 2.1, both Table 2.2 and Table 2.3 are its symbol tables.

The frequency table is computed according to a symbol table and it has exactly the same dimension as the symbol table. Let C be a cluster. Then the frequency table $T_f(C)$ of cluster C is defined as

$$T_f(C) = (\mathbf{f}_1(C), \mathbf{f}_2(C), \dots, \mathbf{f}_d(C)), \quad (2.2)$$

where $\mathbf{f}_j(C)$ is a vector defined as

$$\mathbf{f}_j(C) = (f_{j1}(C), f_{j2}(C), \dots, f_{jn_j}(C))^T, \quad (2.3)$$

Table 2.2. One of the symbol tables of the data set in Table 2.1.

$$\begin{pmatrix} A & A & A & A & B & B \\ B & B & C & C & C & C \\ & & D & D & D & D \end{pmatrix}$$

Table 2.3. Another symbol table of the data set in Table 2.1.

$$\begin{pmatrix} A & B & D & A & B & C \\ B & A & C & C & C & B \\ & & A & D & D & D \end{pmatrix}$$

where $f_{jr}(C)$ ($1 \leq j \leq d$, $1 \leq r \leq n_j$) is the number of data points in cluster C that take value A_{jr} at the j th dimension, i.e.,

$$f_{jr}(C) = |\{\mathbf{x} \in C : x_j = A_{jr}\}|, \quad (2.4)$$

where x_j is the j -component value of \mathbf{x} .

For a given symbol table of the data set, the frequency table of each cluster is unique up to that symbol table. For example, for the data set in Table 2.1, let C be a cluster, where $C = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$. Then if we use the symbol table presented in Table 2.2, the corresponding frequency table for the cluster C is given in Table 2.4. But if we use the symbol table presented in Table 2.3, then the frequency table for the cluster C is given in Table 2.5.

For a given categorical data set D , we see that $T_f(D)$ is a frequency table computed based on the whole data set. Suppose D is partitioned into k nonoverlapping clusters C_1, C_2, \dots, C_k . Then we have

$$f_{jr}(D) = \sum_{i=1}^k f_{jr}(C_i) \quad (2.5)$$

for all $r = 1, 2, \dots, n_j$ and $j = 1, 2, \dots, d$.

2.2 Binary Data

A binary attribute is an attribute that has exactly two possible values, such as “true” or “false.” Note that binary variables can be further divided into two types: symmetric binary variables and asymmetric binary variables. In a symmetric binary variable, the two values are equally important. An example is “male-female.” Symmetric binary variables are nominal variables. In an asymmetric variable, one of its values carries more importance

Table 2.4. The frequency table computed from the symbol table in Table 2.2.

$$\begin{pmatrix} 3 & 3 & 3 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ & & 0 & 0 & 1 & 1 \end{pmatrix}$$

Table 2.5. The frequency table computed from the symbol table in Table 2.3.

$$\begin{pmatrix} 3 & 0 & 0 & 3 & 1 & 1 \\ 0 & 3 & 0 & 0 & 1 & 1 \\ & & 3 & 0 & 1 & 1 \end{pmatrix}$$

than the other. For example, “yes” stands for the presence of a certain attribute and “no” stands for the absence of a certain attribute.

A binary vector \mathbf{x} with d dimensions is defined as (x_1, x_2, \dots, x_d) (Zhang and Srihari, 2003), where $x_i \in \{0, 1\}$ ($1 \leq i \leq d$) is the j -component value of \mathbf{x} . The unit binary vector \mathbf{I} of d dimensions is a binary vector with every entry equal to 1. The complement of a binary vector \mathbf{x} is defined to be $\bar{\mathbf{x}} = \mathbf{I} - \mathbf{x}$, where \mathbf{I} is the unit binary vector of the same dimensions as \mathbf{x} .

Consider two binary vectors \mathbf{x} and \mathbf{y} in a d -dimensional space, and let $S_{ij}(\mathbf{x}, \mathbf{y})$ ($i, j \in \{0, 1\}$) denote the number of occurrences of matches i in \mathbf{x} and j in \mathbf{y} at the corresponding entries, i.e.,

$$S_{ij}(\mathbf{x}, \mathbf{y}) = |\{k : x_k = i \text{ and } y_k = j, k = 1, 2, \dots, d\}|. \quad (2.6)$$

Then, obviously, we have the following equalities:

$$S_{11}(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^d x_i y_i, \quad (2.7a)$$

$$S_{00}(\mathbf{x}, \mathbf{y}) = \bar{\mathbf{x}} \cdot \bar{\mathbf{y}} = \sum_{i=1}^d (1 - x_i)(1 - y_i), \quad (2.7b)$$

$$S_{01}(\mathbf{x}, \mathbf{y}) = \bar{\mathbf{x}} \cdot \mathbf{y} = \sum_{i=1}^d (1 - x_i)y_i, \quad (2.7c)$$

$$S_{10}(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \bar{\mathbf{y}} = \sum_{i=1}^d x_i(1 - y_i). \quad (2.7d)$$

Also, we have

$$d = S_{00}(\mathbf{x}, \mathbf{y}) + S_{01}(\mathbf{x}, \mathbf{y}) + S_{10}(\mathbf{x}, \mathbf{y}) + S_{11}(\mathbf{x}, \mathbf{y}). \quad (2.8)$$

2.3 Transaction Data

Given a set of items $I = \{I_1, I_2, \dots, I_m\}$, a transaction is a subset of I (Yang et al., 2002b; Wang et al., 1999a; Xiao and Dunham, 2001). A transaction data set D is a set of transactions, i.e., $D = \{t_i : t_i \subseteq I, i = 1, 2, \dots, n\}$.

Transactions can be represented by binary vectors, in which each entry denotes the presence or absence of the corresponding item. For example, we can represent a transaction t_i by the binary vector $(b_{i1}, b_{i2}, \dots, b_{im})$, where $b_{ij} = 1$ if $I_j \in t_i$ and $b_{ij} = 0$ if $I_j \notin t_i$. From this point of view, transaction data are a special case of binary data.

The most common example of transaction data is market basket data. In a market basket data set, a transaction contains a subset of the total set of items that could be purchased. For example, the following are two transactions: {apple, cake}, {apple, dish, egg, fish}.

Generally, many transactions are made of sparsely distributed items. For example, a customer may only buy several items from a store with thousands of items. As pointed out by Wang et al. (1999a), for transactions that are made of sparsely distributed items, pairwise similarity is neither necessary nor sufficient for judging whether a cluster of transactions are similar.

2.4 Symbolic Data

Categorical data and binary data are classical data types, and symbolic data is an extension of the classical data type. In conventional data sets, the objects are treated as individuals (first-order objects) (Malerba et al., 2001), whereas in symbolic data sets, the objects are more “unified” by means of relationships. As such, the symbolic data are more or less homogeneous or groups of individuals (second-order objects) (Malerba et al., 2001).

Malerba et al. (2001) defined a symbolic data set to be a class or group of individuals described by a number of set-valued or modal variables. A variable is called set valued if it takes its values in the power set of its domain. A modal variable is a set-valued variable with a measure or a distribution (frequency, probability, or weight) associated with each object.

Gowda and Diday (1992) summarized the difference between symbolic data and conventional data as follows:

- All objects in a symbolic data set may not be defined on the same variables.
- Each variable may take more than one value or even an interval of values.
- The variables in a complex symbolic data set may take values including one or more elementary objects.
- The description of a symbolic object may depend on the relations existing between other objects.
- The values the variables take may indicate frequency of occurrence, relative likelihood, level of importance of the values, and so on.

Symbolic data may be aggregated from other conventional data due to reasons such as privacy. In census data, for example, the data are made available in aggregate form in

order to guarantee that the data analysts cannot identify an individual or a single business establishment. Symbolic clustering will not be discussed in this book. Readers who are interested in this subject are referred to (Dinesh et al., 1997), (Malerba et al., 2001), (Gowda and Diday, 1992), and the references therein.

2.5 Time Series

Time series are the simplest form of temporal data. Precisely, a time series is a sequence of real numbers that represent the measurements of a real variable at equal time intervals (Gunopulos and Das, 2000). For example, stock price movements, the temperature at a given place, and volume of sales over time are all time series. Some other examples of time series can be found in (Kendall and Ord, 1990).

A time series is discrete if the variable is defined on a finite set of time points. Most of the time series encountered in cluster analysis are discrete time series. When a variable is defined at all points in time, then the time series is continuous.

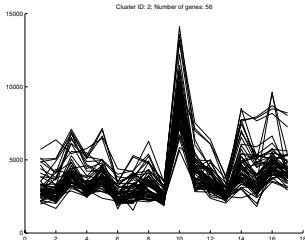
In general, a time series can be considered as a mixture of the following four components (Kendall and Ord, 1990):

1. a trend, i.e., the long-term movement;
2. fluctuations about the trend of greater or less regularity;
3. a seasonal component;
4. a residual or random effect.

Classical time series analysis has been presented and discussed in (Tsay, 2002) and (Kendall and Ord, 1990). Neural network-based analysis of financial time series has been discussed in (Shadbolt and Taylor, 2002) and (Azoff, 1994). To cluster a time series data set, the similarity measures between time series become important. Gunopulos and Das (2000) present a tutorial for time series similarity measures. Time series similarity problems have also been discussed in (Bollobás et al., 1997) and (Das et al., 1997).

2.6 Summary

Some basic types of data encountered in cluster analysis have been discussed in this chapter. In the real world, however, there exist various other data types, such as image data and spatial data. Also, a data set may consist of several types of data, such as a data set containing categorical data and numerical data. To conduct cluster analysis on data sets that contain unusual types of data, the similarity or dissimilarity measures should be defined in a meaningful way. The types of data are highly area specific, for example, DNA data in biology (Yeung et al., 2001), and financial time series in marketing research.



Chapter 3

Scale Conversion

In many applications, the variables describing the objects to be clustered will not be measured in the same scales. They may often be variables of completely different types, some interval, others categorical. There are three approaches to cluster objects described by variables of different types. One is to use a similarity coefficient, which can incorporate information from different types of variable, such as the general coefficients described in Section 6.5. The second is to carry out separate analyses of the same set of objects, each analysis involving variables of a single type only, and then to synthesize the results from different analyses. The third is to convert the variables of different types to variables of the same type, such as converting all variables describing the objects to be clustered into categorical variables.

3.1 Introduction

In Chapter 2, we saw that there are four types of scales: nominal scales, ordinal scales, interval scales, and ratio scales (see Figure 2.2). We shall see that any scale can be converted to any other scale. Several cases of scale conversion are described by Anderberg (1973), including interval to ordinal, interval to nominal, ordinal to nominal, nominal to ordinal, ordinal to interval, nominal to interval, and dichotomization (binarization). These scale conversions are reviewed briefly in this section.

3.1.1 Interval to Ordinal

The scale conversion from interval to ordinal creates contiguous categories over the interval scale such that objects within a category are equally ranked while the ordinal order relation among objects in different categories is maintained. During the conversion, the following two forms of information are lost: distinctions between objects within the same category and the magnitudes of the distinctions between objects in different categories.

Anderberg (1973) suggested 11 methods to convert a variable from interval to ordinal.

- (a) **No substantive change necessary.** For example, age and education are ratio variables when measured in terms of years; they naturally form ordered categories.

- (b) **Substitution.** The method of substitution is to delete a variable and replace it by another that is closely related but measured on the desired kind of scale. Consider again the education variable. For example, instead of being measured in terms of years of education or highest grade completed, education could be measured by the following ordered categories: no grammar school, some grammar school, eighth grade completed, high school graduate, bachelor's degree, etc.
- (c) **Equal length categories.** The problem of deciding the appropriate number of categories arises when there is no satisfactory substitute variable available. The method of equal length categories chooses some number of categories and then divides the range of the interval variable into equal length intervals. The method can also be extended to deal with the situation when the relevant number of categories is not obvious by considering two, three, four, etc., categories and choosing the most satisfactory grouping.
- (d) **Equal membership categories.** The method of equal length categories can result in a gross imbalance of membership among the categories. The method of equal membership categories is another simple alternative that partitions the data into classes with equal membership or membership as near to equal as possible.
- (e) **Assumed distribution for proportion in each class.** This is a more statistically sophisticated method. Let X be a random variable with cumulative distribution function (CDF) $F(X)$. Assume X is defined over the interval $[a, b]$ such that $F(a) = 0$ and $F(b) = 1$. Then the separation points that divide the interval $[a, b]$ into n equal intervals are

$$\begin{aligned}y_0 &= a, \\y_i &= y_{i-1} + \frac{b-a}{n}, \quad i = 1, 2, \dots, n-1, \\y_n &= b.\end{aligned}$$

The proportion of the population falling in the i th interval $(y_{i-1}, y_i]$ is $F(y_i) - F(y_{i-1})$. The method of equal length categories and the method of equal membership categories are special cases of this method if $F(X)$ is taken as the empirical distribution and the uniform distribution of the sample, respectively.

- (f) **Assumed distribution for cut points between classes.** The method of assumed distribution for cut points between classes uses a distributional assumption to divide the distribution into equal probability classes, that is,

$$F(y_i) - F(y_{i-1}) = \frac{1}{n}, \quad i = 1, 2, \dots, n.$$

This method is the same as the method of equal membership categories in that they both assign the first $\frac{m}{n}$ observations to the first class, the second $\frac{m}{n}$ to the second class, and so on, where m is the number of observations in the sample that approximate the population closely.

- (g) **One-dimensional hierarchical linkage methods.** One-dimensional hierarchical linkage methods convert a variable from interval to ordinal scale by employing certain

cluster analysis techniques to organize observations into groups. Single-linkage and complete linkage methods are used here. These clustering methods are discussed in Chapter 7 in detail.

- (h) **Ward's hierarchical clustering method.** Ward's method is presented in Section 7.2.7. Here Ward's method is used to cluster a one-dimensional data set.
- (i) **Iterative improvement of a partition.** The method of iterative improvement of a partition tries to find the partition of a one-dimensional data set that minimizes the sum of squares. Details of this approach can be found in Section 3.2.
- (j) **The linear discriminant function.** For one-dimensional data, the multiple group discriminant analysis can be described as follows. Suppose there are g normal populations present in the grand ensemble in proportions p_1, p_2, \dots, p_g . If an observation from population j is classified wrongly as being from population i , then a loss $L(i : j)$ is incurred. The optimal Bayes classification rule is to assign a new observation x to that class i such that

$$\frac{x(\bar{x}_i - \bar{x}_j) - \frac{1}{2}(\bar{x}_i + \bar{x}_j)(\bar{x}_i - \bar{x}_j)}{s_x^2} \geq \ln \left(\frac{p_j L(i : j)}{p_i L(j : i)} \right),$$

where \bar{x}_i and \bar{x}_j are the mean of the i th population and the j th population, respectively, and s_x^2 is the common variance of the g normal populations. In the one-dimensional case, comparing two populations i and j can be accomplished by calculating the cut point between classes as follows. Assume $\bar{x}_i > \bar{x}_j$. Then x is assigned to population i instead of population j if

$$x \geq C_{ij} = \frac{s_x^2}{\bar{x}_i - \bar{x}_j} \ln \left(\frac{p_j L(i : j)}{p_i L(j : i)} \right) + \frac{\bar{x}_i + \bar{x}_j}{2}.$$

- (k) **Cochran and Hopkins method.** Cochran and Hopkins (1961) suggested another method based on discriminant analysis. Assume an interval variable is normally distributed in each of two classes with means \bar{x}_1 and \bar{x}_2 and common variance s_x^2 . Then they consider how to categorize the interval variable into n groups to retain maximum discriminability under the assumption that costs and probabilities of misclassification are equal.

3.1.2 Interval to Nominal

Converting a variable from interval to nominal scales is very similar to converting a variable from interval to ordinal scales, and all the methods described in the previous subsection may be used to do this job. However, the resulting categories obtained by the methods of interval to ordinal conversion are constrained to satisfy some order restriction. To avoid this problem, one may use the central-extreme principle, which gives noncontiguous categories. Under the central-extreme principle, some observations are treated as central while others are extreme.

3.1.3 Ordinal to Nominal

A natural way to convert a set of ordered categories to nominal categories is to ignore the order properties and keep the same categories. One may also use the degree of extremes. If category C is the central category in the sequence $A > B > C > D > E$, for example, one might adopt the following revised classifications: C for normal, $B \cup D$ for moderately deviant, and $A \cup E$ for extreme.

3.1.4 Nominal to Ordinal

The essential problem of converting a variable from nominal to ordinal is to impose an ordering on nominal categories. Anderberg (1973) suggested two approaches.

1. **Correlation with an interval variable.** Let X be a nominal variable with g classes and Y be an interval variable. One well-known way to use X to predict Y most effectively is to find the b coefficients that maximize R^2 for the regression equation

$$Y = \sum_{i=1}^g b_i X_i,$$

where $X_i = 1$ if an observation falls into the i th class and $X_i = 0$ if otherwise. It can be shown that R^2 is maximized by choosing b_i as the mean Y value for all observations in the i th class, i.e.,

$$b_i = \sum_{j=1}^{n_i} \frac{y_{ij}}{n_i},$$

where n_i is the number of observations in the i th class and y_{ij} is the j th observation in the i th class.

Then the class means of Y can be assigned to nominal classes of X such that items within the same class receive the same score. This technique provides order information.

2. **Rank correlation and mean ranks.** Let X be a nominal variable of g groups, with n_i observations in the i th group, and let Y be an ordinal reference variable. Denote as x_i the common rank of all elements in the i th group of X and as y_{ij} the Y rank of the j th element in the i th class. Then the Spearman rank correlation between X and Y can be formulated as

$$r = 1 - 6 \frac{\sum_{i=1}^g \sum_{j=1}^{n_i} (x_i - y_{ij})^2}{N(N^2 - 1)},$$

where N is the number of data units measured on X and Y . The problem of ordering the nominal classes is to find class ranks such that the rank correlation is maximized. Obviously, maximizing the rank correlation r is equivalent to minimizing

$$\sum_{i=1}^g \sum_{j=1}^{n_i} (x_i - y_{ij})^2$$

or

$$\sum_{i=1}^g \left(n_i x_i^2 - 2n_i x_i \bar{y}_i + \sum_{j=1}^{n_i} y_{ij}^2 \right),$$

where \bar{y}_{ij} is the mean Y rank in the i th nominal class. Therefore, the problem is transformed to the problem of assigning the integers $1, 2, \dots, g$ to x_1, x_2, \dots, x_g such that

$$C = \sum_{i=1}^g n_i x_i (x_i - 2\bar{y}_{ij})$$

is minimized.

3.1.5 Ordinal to Interval

The problem here is to assign a score to each class that preserves the rank order and exhibits magnitude differences among classes. Anderberg (1973) suggested four approaches to solving this problem.

1. **Class ranks.** This is a simple and intuitive approach to assigning scores to ordered classes. It simply uses class ranks and assumes that the classes are spaced equally along the interval scale.
2. **Expected order statistics.** Here expected order statistics are used to assign scores to ordinal data. Suppose x_1, x_2, \dots, x_n is a random sample from a population with continuous cumulative distribution function $F(x)$. Then the i th largest of the n observations is the i th-order statistic for the sample.
3. **Assuming underlying distribution for the data units.** To convert a variable from ordinal to interval, it is natural to associate very large classes with long segments of the derived interval scale and to associate small classes with short segments. This can be done by assuming that the data units were drawn from an underlying distribution and that the observed categories are sections from the distribution. Class scores can be obtained by sectioning the assumed distribution using the proportions observed in the sample and calculating a summary statistic (e.g., mean or median) for each section.
4. **Correlation with a reference variable.** This approach chooses class scores by selecting values that maximize the correlation of the ordinal variable with some reference variable.

3.1.6 Other Conversions

Anderberg (1973) also discussed methods to convert a variable from nominal to interval, from ordinal to binary, and from nominal to binary. In the case of converting from nominal to interval, the classes are unordered, so any method adopted should induce an ordering and a spacing of the classes as well. One way to do this is to first convert from nominal to ordinal and then convert from ordinal to interval; another way is to use a reference variable.

In the case of converting from ordinal with g ordered classes to binary, the problem is to choose among the $g - 1$ possible splits that preserve order. In this problem, a reference variable can be used. In the case of converting from nominal to binary, the problem is to cluster the nominal classes into two groups.

3.2 Categorization of Numerical Data

By a numerical variable, we generally mean a variable measured by interval scales or ratio scales. By a categorical variable, we generally mean a nominal variable, i.e., a variable measured by nominal scales. In this section, we shall present some examples and methods of categorization, i.e., converting numerical data to categorical data.

Given a numerical data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ with n objects, each of which is described by d variables, to convert D into a categorical data set, we can convert variable by variable, i.e., each conversion involves one dimension only. That is, for each j , we convert $x_{1j}, x_{2j}, \dots, x_{nj}$ into categorical values $y_{1j}, y_{2j}, \dots, y_{nj}$; then we obtain a categorical data set $D' = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$. Hence, we only need to consider one-dimensional cases.

3.2.1 Direct Categorization

Direct categorization is the simplest way to convert numerical data into categorical data. Let x be a numerical variable that takes values $X = \{x_1, x_2, \dots, x_n\}$ in a data set of n records. To convert the numerical values x_i to categorical values y_i by the method of direct categorization, we first need to find the range of x in the data set. Let x_{min} and x_{max} be defined as

$$\begin{aligned} x_{min} &= \min_{1 \leq i \leq n} x_i, \\ x_{max} &= \max_{1 \leq i \leq n} x_i. \end{aligned}$$

Then the range of x in the data set is $[x_{min}, x_{max}]$. Without loss of generality, we can assume that $x_{min} \neq x_{max}$. Let N ($N \geq 2$) be the number of categories we want to make. Then we divide the interval $[x_{min}, x_{max}]$ into N ($N \geq 2$) small intervals and let the j th ($j = 1, 2, \dots, N$) small interval I_j be

$$I_j = \begin{cases} \left[x_{min} + \frac{j-1}{N}L, x_{min} + \frac{j}{N}L \right) & \text{for } j = 1, 2, \dots, N-1, \\ \left[x_{min} + \frac{N-1}{N}L, x_{max} \right] & \text{for } j = N, \end{cases}$$

where L is defined as

$$L = x_{max} - x_{min}.$$

Now we can define the categorical value y_i of x_i as

$$y_i = j_0 \text{ if } x_i \in I_{j_0}. \quad (3.1)$$

To determine a j_0 for x_i ($1 \leq i \leq n$), we assume that $x_i \in I_{j_0}$ with $j_0 \leq N - 1$. Then we have

$$x_{min} + \frac{j-1}{N}L \leq j_0 < x_{min} + \frac{j}{N}L,$$

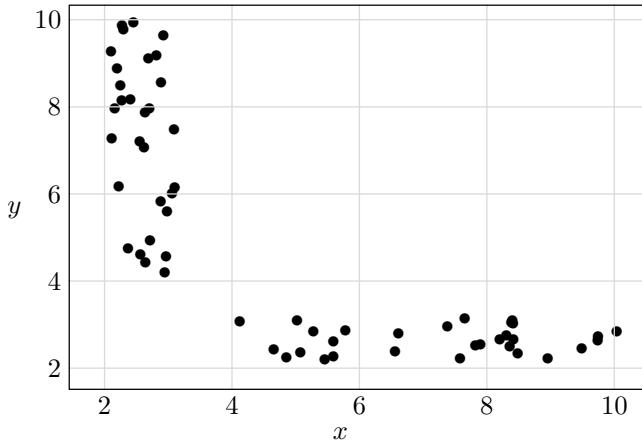


Figure 3.1. An example two-dimensional data set with 60 points.

which is equivalent to

$$j_0 - 1 \leq \frac{N(x_i - x_{min})}{L} < j_0.$$

Therefore, we can compute j_0 as

$$j_0 = \left\lceil \frac{N(x_i - x_{min})}{L} \right\rceil + 1, \quad (3.2)$$

where $[x]$ denotes the largest integer less than or equal to x .

Thus, if $x_i < x_{max}$, then the corresponding categorical value y_i is in $\{1, 2, \dots, N\}$; hence we can use equation (3.2) to compute the categorical value for x_i . Obviously, if $x_i = x_{max}$, from equation (3.2), we get $j_0 = N + 1$, and we should set $y_i = N$.

Example 3.1 (Direct categorization). Consider the data set depicted in Figure 3.1. If we convert the values in the x dimension into categorical values using the method of direct categorization with $N = 5$, we obtain the result given in Figure 3.2(a). Similarly, if we convert the values in the y dimension into categorical values with $N = 5$, we get the result given in Figure 3.2(b). Figure 3.3(a) and Figure 3.3(b) give further examples. ■

3.2.2 Cluster-based Categorization

Cluster-based categorization is more complicated than direct categorization, but it is very effective for converting numerical data into a categorical data. In this section, we will discuss how to convert one-dimensional data into categorical data using the method of cluster-based categorization.

Most of the numerical clustering algorithms can be used to convert numerical data into categorical data, such as k -means (Macqueen, 1967; Hartigan and Wong, 1979; Faber, 1994), CURE (Guha et al., 1998), CLARANS (Ng and Han, 1994), BIRCH (Zhang et al.,

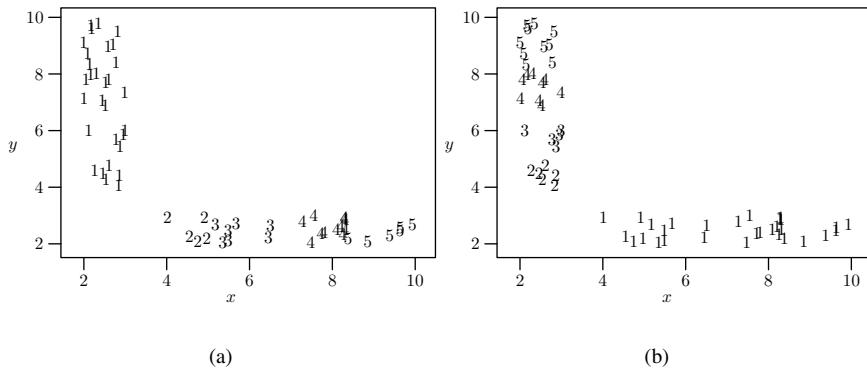


Figure 3.2. Examples of direct categorization when $N = 5$. In (a), the values in the x dimension are categorized. In (b), the values in the y dimension are categorized. The label of each data point is plotted at that point.

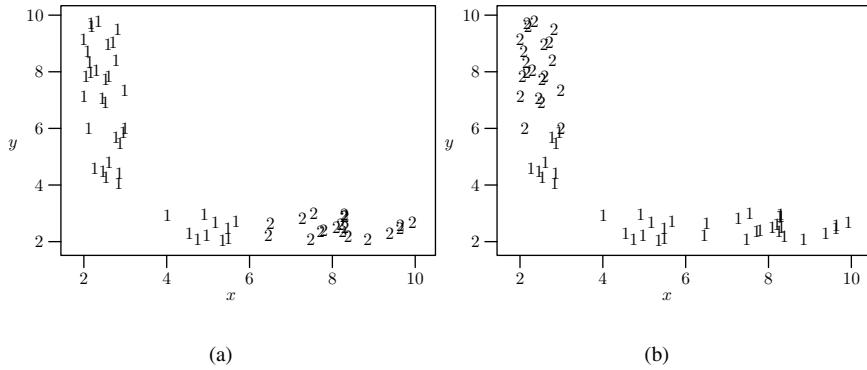


Figure 3.3. Examples of direct categorization when $N = 2$. In (a), the values in the x dimension are categorized. In (b), the values in the y dimension are categorized. The label of each data point is plotted at that point.

1996), and DENCLUE (Hinneburg and Keim, 1998). In this section, we shall illustrate how to use the k -means algorithm, and least squares (Fisher, 1958) to convert one-dimensional numerical data into categorical data.

Example 3.2 (k -means-based categorization). Let the one-dimensional data set be $X = \{x_1, x_2, \dots, x_n\}$. If X is not a large data set, the Quick Sort algorithm (Hoare, 1961; Sedgewick, 1978) can be used to sort x_1, x_2, \dots, x_n such that

$$x_{i_1} \leq x_{i_2} \leq \dots \leq x_{i_n}, \quad (3.3)$$

where i_1, i_2, \dots, i_n is a combination of $1, 2, \dots, n$.

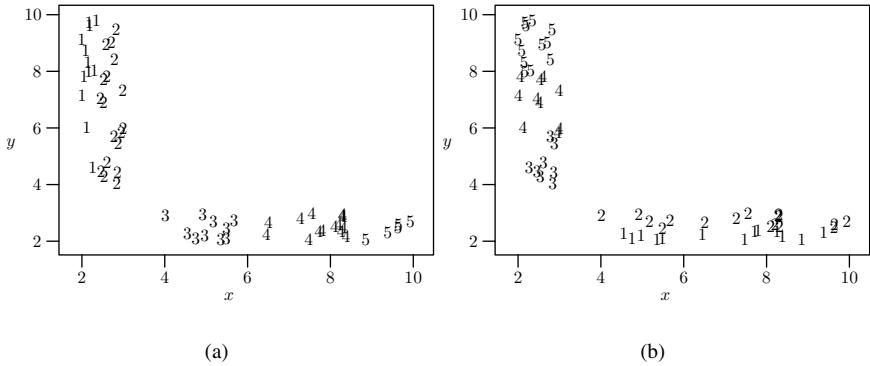


Figure 3.4. Examples of k -means-based categorization when $N = 5$. In (a), the values in the x dimension are categorized. In (b), the values in the y dimension are categorized. The label of each data point is plotted at that point.

Let N ($N > 1$) be the number of initial centers. Then we can choose x_{i_t} ($t \in S$) as the initial centers, where S is the set of subscripts defined as

$$S = \left\{ t : t = 1 + \left[\frac{(n-1)(j-1)}{N-1} \right], j = 1, 2, \dots, N \right\}, \quad (3.4)$$

where $[x]$ denotes the largest integer less than or equal to x .

With this method of choosing initial centers, the N most dissimilar variable values are chosen as initial centers. After choosing initial centers, we can get an initial partition by assigning the rest of the data to the nearest center. Then we can use the conventional k -means algorithm (see Section 9.1) to cluster the data. Finally, we use the cluster index of each data point as its categorical value. Figures 3.4 and 3.5 provide illustrative examples for the data set given in Figure 3.1. ■

Example 3.3 (Least squares-based categorization). Given a set of points in a straight line, how can we group the points into N groups so that the sum of squared distances (SSD) of the individual points to their group centers (or means) is minimized? Fisher (1958) describes a procedure to find such a partition. In this subsection, we will briefly introduce Fisher's method and use it for categorization.

Let $D = \{x_1, x_2, \dots, x_n\}$ be a group of data points in a straight line, i.e., D is a one-dimensional data set, and let c_j ($j = 1, 2, \dots, N$) be the center of the j th group C_j . Then the SSD W for the whole data set is defined as (Fisher, 1958)

$$W = \sum_{j=1}^N W_j, \quad (3.5)$$

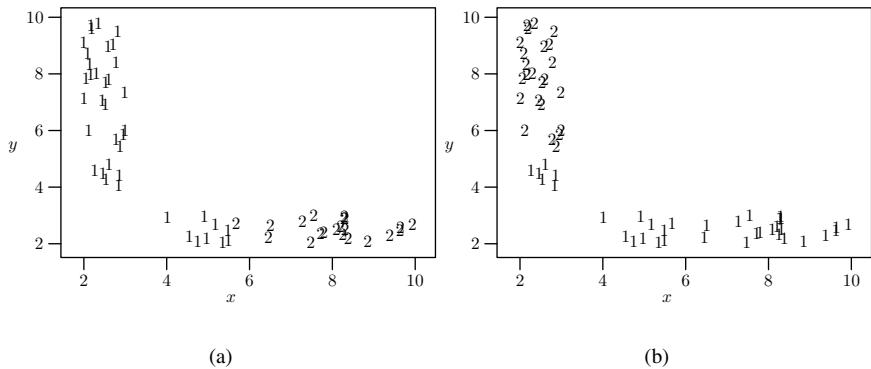


Figure 3.5. Examples of k -means-based categorization when $N = 2$. In (a), the values in the x dimension are categorized. In (b), the values in the y dimension are categorized. The label of each data point is plotted at that point.

where W_j is defined as

$$W_j = \sum_{x \in C_j} (x - c_j)^2, \quad j = 1, 2, \dots, N. \quad (3.6)$$

A least squares partition is a partition that minimizes the SSD W defined in equation (3.5). Fisher also proved that a least squares partition is contiguous, where a contiguous partition is defined for a set of completely ordered points as a partition such that each group satisfies the following condition (Fisher, 1958): if a, b, c have the order $a < b < c$ and if a and c are assigned to the same group, then b must also be assigned to that same group.

For least squares partitions, Fisher (1958) has proved a suboptimization lemma.

Lemma 3.4 (Fisher's suboptimization lemma). Let $A_1 : A_2$ denote a partition of set A into two disjoint subsets A_1 and A_2 and let P_i^* ($i = 1, 2$) denote a least squares partition of A_i into G_i subsets. Then, of the class of subpartitions of $A_1 : A_2$ employing G_1 subsets over A_1 and G_2 subsets over A_2 , a least squares subpartition is $P_1^* : P_2^*$.

To implement the clustering algorithm based on Fisher's suboptimization lemma, we can use dynamic programming (Fitzgibbon et al., 2000). Before applying the algorithm to the data set D , we need to sort x_1, x_2, \dots, x_n such that $x_1 \leq x_2 \leq \dots \leq x_n$. In the following discussion, we assume that $x_1 \leq x_2 \leq \dots \leq x_n$ and let D_i ($i = 1, 2, \dots, n$) be the set consisting of the first i data points, i.e.,

$$D_i = \{x_1, x_2, \dots, x_i\}, \quad i = 1, 2, \dots, n. \quad (3.7)$$

Let the sum $S(i, j)$ and squared sum $S_2(i, j)$ ($1 \leq i, j \leq n$) be defined as

$$S(i, j) = \sum_{l=i}^j x_l, \quad (3.8)$$

$$S_2(i, j) = \sum_{l=i}^j x_l^2, \quad (3.9)$$

$$D(i, j) = S_2(i, j) - \frac{S(i, j)^2}{j - i + 1}. \quad (3.10)$$

Let $D[g, i]$ be the minimum SSD over all the partitions that partition D_i into g groups. Then from Fisher's lemma, we can define $D[g, i]$ and $j(g, i)$ ($1 \leq g \leq N, g \leq i \leq n$) as

$$D[g, i] = \begin{cases} S_2(1, i), & \text{if } g = 1, \\ \min_{g \leq j \leq i} (D[g - 1, j - 1] + D(j, i)) & \text{if } g \geq 2. \end{cases} \quad (3.11)$$

$$j(g, i) = \begin{cases} i & \text{if } g = 1, \\ \arg \min_{g \leq j \leq i} (D[g - 1, j - 1] + D(j, i)) - 1 & \text{if } g \geq 2. \end{cases} \quad (3.12)$$

The $j(g, i)$ defined in equation (3.12) is a cutting index of the partition that partitions D_i into g groups, where a cutting index of a group is the largest subscript of elements in that group. A partition of g groups has $g - 1$ cutting indices, because the cutting index is fixed for the last group. For example, let $\{x_1, x_2, \dots, x_{i_1}\}$, $\{x_{i_1+1}, \dots, x_{i_2}\}$, and $\{x_{i_2+1}, \dots, x_i\}$ be a partition of D_i . Then the cutting indices are i_1 and i_2 . We can use cutting indices for least squares partitions, because least squares partitions are contiguous partitions (Fisher, 1958).

To use Fisher's suboptimization lemma to find a least squares partition of N groups, we first find the cutting index $j(2, i)$ ($2 \leq i \leq n$) that partitions D_i into two groups and find the corresponding SSD for the partition. After we find the cutting indices of a least squares partition that partitions D_i ($i \geq g$) into g groups and the corresponding SSD, we can use this information to find the cutting indices of a least squares partition that partitions D_i ($i \geq g + 1$) into $g + 1$ groups. According to this procedure, the cutting indices $P(1, i), P(2, i), \dots, P(g - 1, i)$ of a least squares partition that partitions D_i into g groups are given by

$$\begin{aligned} P(g - 1, i) &= j(g, i), \\ P(g - 2, i) &= j(g - 1, P(g - 1, i)), \\ &\vdots \\ P(1, i) &= j(2, P(2, i)). \end{aligned}$$

In particular, the $N - 1$ cutting indices P_1, P_2, \dots, P_{N-1} of a least squares partition that partitions D into N groups are given by

$$\begin{aligned} P_{N-1} &= j(N, n), \\ P_{N-2} &= j(N - 1, P_{N-1}), \\ &\vdots \\ P_1 &= j(2, P_2). \end{aligned}$$

Using Fisher's suboptimization lemma to find least squares partitions can save a lot of time. Fitzgibbon et al. (2000) introduce a dynamic programming algorithm based on Fisher's suboptimization lemma. Figures 3.6 and 3.7 provide the illustrations for the data set given in Figure 3.1. ■

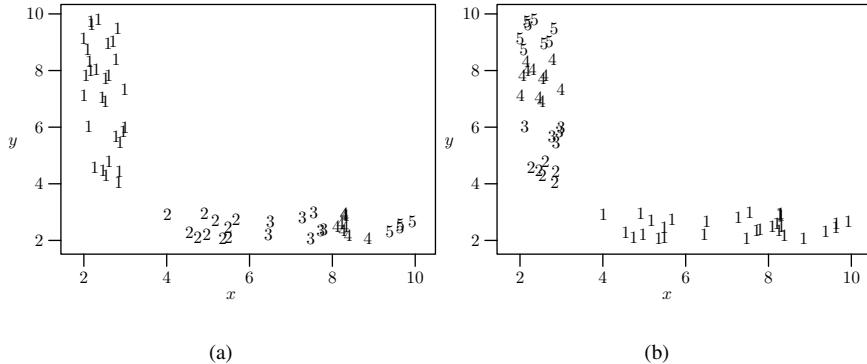


Figure 3.6. Examples of cluster-based categorization based on the least squares partition when $N = 5$. In (a), the values in the x dimension are categorized. In (b), the values in the y dimension are categorized. The label of each data point is plotted at that point.

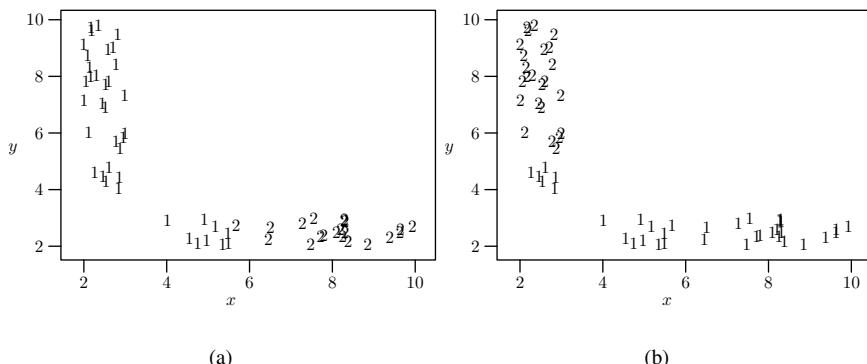


Figure 3.7. Examples of cluster-based categorization based on the least squares partition when $N = 2$. In (a), the values in the x dimension are categorized. In (b), the values in the y dimension are categorized. The label of each data point is plotted at that point.

Besides the least squares criterion, there are also other criteria to partition a one-dimensional data set into two groups, such as the mean-variance criterion (Engelman and Hartigan, 1969).

3.2.3 Automatic Categorization

In Subsection 3.2.1 and Subsection 3.2.2, we introduced direct categorization and cluster-based categorization. All these categorization methods need an important parameter, the number of categories. In most application problems, however, we have no information about the number of categories for each numerical variable, so we need to find a method to categorize numerical variable values automatically so that the optimal number of categories is discovered.

In the present subsection, we will introduce automatic categorization methods that can find the optimal number of categories for a given one-dimensional numerical data set. To achieve this goal, we need an index that indicates the quality of the resulting categorization. We can use clustering for this purpose.

Many validity criteria have been proposed to measure the quality of clustering results (Halkidi et al., 2002a,b; Davies and Bouldin, 1979; Halkidi et al., 2000; Dunn, 1974b) (see Chapter 17). The basic idea of automatic categorization is to apply a categorization method to a data set from $N = 1$ to N_{max} , where N is the number of categories used in the categorization method and N_{max} is the maximum number of categories, and then to calculate the validity index V_i for each categorization i , and finally to choose an optimal one to minimize or maximize V_i .

One of the disadvantages of the k -means algorithm is that the number of clusters must be supplied as a parameter. Ray and Turi (1999) introduced a compactness-separation-based validity measure to determine the number of clusters in k -means clustering. In the following example, we shall introduce this validity measure and apply it to the automatic categorization of numerical data.

Example 3.5 (Compactness-separation criterion). Let D be a one-dimensional data set. Suppose we have clustered the data set D into k clusters C_1, C_2, \dots, C_k . Let z_i be the cluster center for cluster C_i . Then the intracluster distance measure M_{intra} is defined as (Ray and Turi, 1999)

$$M_{intra} = \frac{1}{n} \sum_{i=1}^k \sum_{x \in C_i} \|x - z_i\|^2, \quad (3.13)$$

where n is the number of data points in D . The intercluster distance measure M_{inter} is defined as (Ray and Turi, 1999)

$$M_{inter} = \min_{1 \leq i < j \leq k} \|z_i - z_j\|^2. \quad (3.14)$$

Obviously, a good clustering result should have a small M_{intra} and a large M_{inter} ; thus Ray and Turi (1999) defined the validity measure V as

$$V = \frac{M_{intra}}{M_{inter}}. \quad (3.15)$$

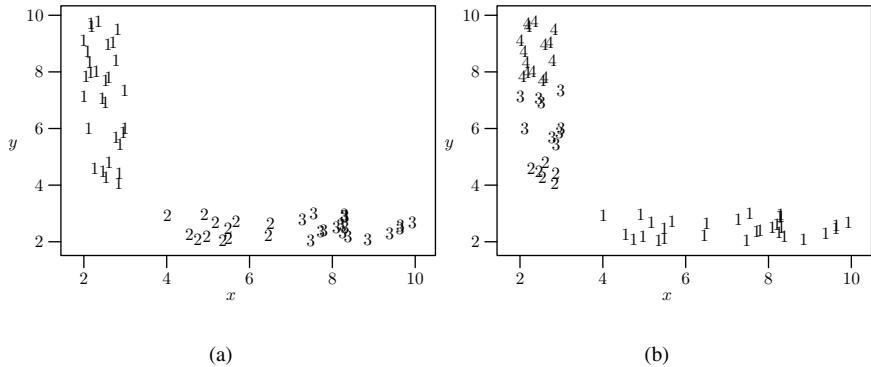


Figure 3.8. Examples of automatic categorization using the k -means algorithm and the compactness-separation criterion. In (a), the values in the x dimension are categorized. In (b), the values in the y dimension are categorized. The label of each data point is plotted at that point.

To determine an optimal number of clusters in the data set, we shall cluster the data set from $k = 2$ up to k_{max} , where k_{max} is the upper limit on the number of clusters, and then we calculate the validity measure V_k and choose the optimal number of clusters k_0 as the smallest k such that V_k has the minimum value, i.e.,

$$k_0 = \min\{j : V_j = \min_{1 \leq i \leq k} V_i, 1 \leq j \leq k\}. \quad (3.16)$$

To implement the algorithm is straightforward. The upper limit of the number of clusters k_{max} is an input parameter. Applying this criterion to the results of the k -means algorithm on the sample data set in Figure 3.1, we get the results in Figure 3.8. For each dimension of the sample data set, we also plot the validity measure V defined in equation (3.15) against the number of clusters k for each dimension, in Figure 3.9(a) and Figure 3.9(b), respectively. ■

Many clustering algorithms use the sum of squares objective function (Krzanowski and Lai, 1988; Friedman and Rubin, 1967; Scott and Symons, 1971b; Marriott, 1971). For these clustering algorithms, Krzanowski and Lai (1988) introduced a criterion to determine the optimal number of clusters. Since we only consider one-dimensional numerical data, we can use this criterion and other clustering algorithms (such as least squares partitions) together. In the following, we briefly introduce this criterion and apply it to automatic categorization.

Example 3.6 (Sum of squares criterion). Suppose D is a one-dimensional data set with k clusters C_1, C_2, \dots, C_k . Let \tilde{S}_k be the optimum value of the sum of squares objective function. Then the sum of squares criterion (SSC) validity measure is defined as (Krzanowski and Lai, 1988)

$$V_k = \frac{|\text{DIFF}(k)|}{|\text{DIFF}(k+1)|}, \quad (3.17)$$

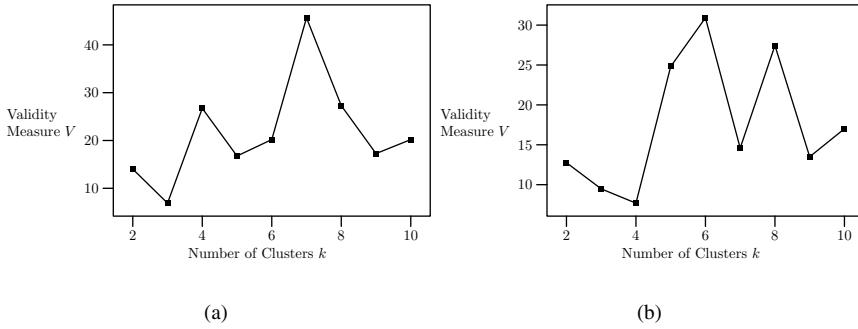


Figure 3.9. In (a), the values in the x dimension are categorized using the k -means algorithm from $k = 2$ to $k = 10$. For each k , the validity measure V defined in equation (3.15) is computed and plotted. In (b), the values in the y dimension are categorized using the k -means algorithm from $k = 2$ to $k = 10$. For each k , the validity measure V defined in equation (3.15) is computed and plotted.

where $\text{DIFF}(k)$ is defined as

$$\text{DIFF}(k) = (k - 1)^2 \tilde{S}_{k-1} - k^2 \tilde{S}_k. \quad (3.18)$$

The optimum number of clusters k_0 is the number k that maximizes V_k . The results of applying the SSC to the results of the least squares partition of the sample data set are given in Figure 3.10(a) and Figure 3.10(b), while the validity measure V_k defined in equation (3.17) against the number of clusters k is plotted in Figure 3.11(a) and Figure 3.11(b). ■

The above-introduced two methods to determine the optimum number of clusters can only be applied to certain data sets or clustering algorithms (and are not well defined when $k = 1$). In what follows, we introduce a general method (called the gap statistic (Tibshirani et al., 2001; SAS Institute Inc., 1983)) for estimating the number of clusters. This method is so general that it can be used with the output of any clustering algorithms.

Example 3.7 (Gap statistic). Suppose that D is a one-dimensional data set with k clusters C_1, C_2, \dots, C_k . Let W be defined as in equation (3.5). Then W is the pooled within-cluster sum of squares around the cluster means. The general idea of the gap statistic proposed by Tibshirani et al. (2001) is to standardize the graph of $\log(W_i)$ by comparing it to its expectation under an appropriate reference distribution of the data set, where W_i is W when $k = i$. For a given k , the gap statistic $\text{Gap}(k)$ is defined as (Tibshirani et al., 2001)

$$\text{Gap}(k) = E^*(\log(W_k)) - \log(W_k), \quad (3.19)$$

where $E^*(\log(W_k))$ is estimated by an average of B copies of $\log(W_k^*)$, each of which is computed from a Monte Carlo sample $X_1^*, X_2^*, \dots, X_B^*$ drawn from the reference distribution, i.e.,

$$E^*(\log(W_k)) = \frac{1}{B} \sum_{b=1}^B \log(W_{kb}^*). \quad (3.20)$$

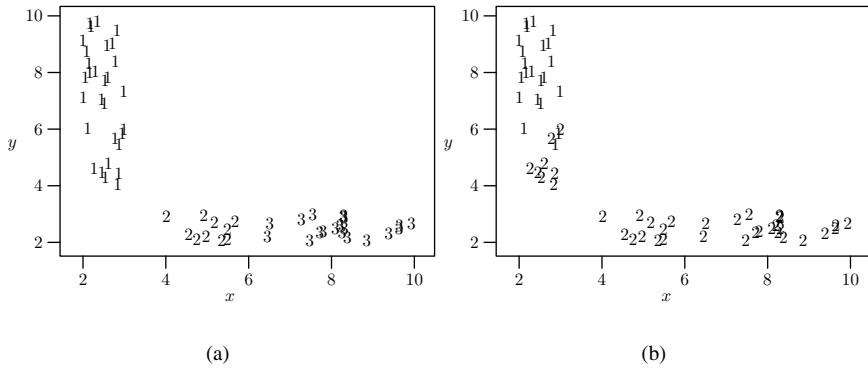


Figure 3.10. Examples of automatic categorization based on the least squares partition and the SSC. In (a), the values in the x dimension are categorized using the least squares partition algorithm. In (b), the values in the y dimension are categorized using the least squares partition algorithm. The label of each data point is plotted at that point.

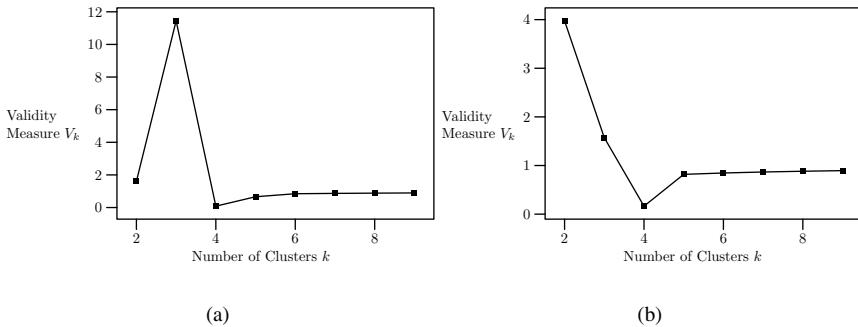


Figure 3.11. In (a), the values in the x dimension are categorized using the least squares partition algorithm from $k = 2$ to $k = 10$. For each k , the validity measure V_k defined in equation (3.17) is computed and plotted. In (b), the values in the y dimension are categorized using the least squares partition algorithm from $k = 2$ to $k = 10$. For each k , the validity measure V_k defined in equation (3.17) is computed and plotted.

Then the optimum number of clusters k_0 is defined to be the smallest k such that

$$k_0 = \min\{k : \text{Gap}(k) \geq \text{Gap}(k+1) - s_{k+1}\}, \quad (3.21)$$

where s_k is defined as

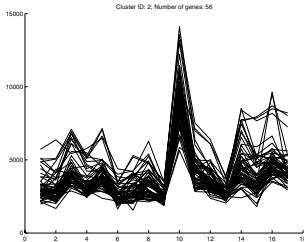
$$s_k = \sqrt{1 + \frac{1}{B}} \cdot \sqrt{\frac{1}{B} \sum_{b=1}^B (\log(W_{kb}^*) - \bar{l})^2}, \quad (3.22)$$

where \bar{l} is the mean of $\log(W_{kb}^*)$ defined as

$$\bar{l} = \frac{1}{B} \sum_{b=1}^B \log(W_{kb}^*). \quad \blacksquare \quad (3.23)$$

3.3 Summary

Some scale conversion techniques are briefly reviewed in this chapter. These scale conversions include interval to ordinal, interval to nominal, ordinal to nominal, nominal to ordinal, ordinal to interval, nominal to interval, and binarization. Readers are referred to (Anderberg, 1973) for detailed discussions of these conversions. In addition to these briefly described techniques, we also present categorization of numerical data, where several methods are discussed in detail.



Chapter 4

Data Standardization and Transformation

In many applications of cluster analysis, the raw data, or actual measurements, are not used directly unless a probabilistic model for pattern generation is available (Jain and Dubes, 1988). Preparing data for cluster analysis requires some sort of transformation, such as standardization or normalization.

Some commonly used data transformation methods for cluster analysis shall be discussed in this chapter. Some methods of data standardization are reviewed in Section 4.1.

For convenience, let $D^* = \{\mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_n^*\}$ denote the d -dimensional raw data set. Then the data matrix is an $n \times d$ matrix given by

$$(\mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_n^*)^T = \begin{pmatrix} x_{11}^* & x_{12}^* & \cdots & x_{1d}^* \\ x_{21}^* & x_{22}^* & \cdots & x_{2d}^* \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1}^* & x_{n2}^* & \cdots & x_{nd}^* \end{pmatrix}. \quad (4.1)$$

4.1 Data Standardization

Data standardization makes data dimensionless. It is useful for defining standard indices. After standardization, all knowledge of the location and scale of the original data may be lost. It is necessary to standardize variables in cases where the dissimilarity measure, such as the Euclidean distance, is sensitive to the differences in the magnitudes or scales of the input variables (Milligan and Cooper, 1988). The approaches of standardization of variables are essentially of two types: global standardization and within-cluster standardization.

Global standardization standardizes the variables across all elements in the data set. Within-cluster standardization refers to the standardization that occurs within clusters on each variable. Some forms of standardization can be used in global standardization and within-cluster standardization as well, but some forms of standardization can be used in global standardization only.

It is impossible to directly standardize variables within clusters in cluster analysis, since the clusters are not known before standardization. To overcome this difficulty, other approaches must be taken. Overall and Klett (1972) proposed an iterative approach that first

obtains clusters based on overall estimates and then uses these clusters to help determine the within-group variances for standardization in a second cluster analysis.

Milligan and Cooper (1988) present an in-depth examination of standardization of variables when using Euclidean distance as the dissimilarity measure. Before reviewing various methods for data standardization, we first remark that the choice of an appropriate standardization method depends on the original data set and the convention of the particular field of study.

To standardize the raw data given in equation (4.1), we can subtract a location measure and divide a scale measure for each variable. That is,

$$x_{ij}^* = \frac{x_{ij} - L_j}{M_j}, \quad (4.2)$$

where x_{ij} denotes the standardized value, L_j is the location measure, and M_j is the scale measure.

We can obtain various standardization methods by choosing different L_j and M_j in equation (4.2). Some well-known standardization methods are mean, median, standard deviation, range, Huber's estimate, Tukey's biweight estimate, and Andrew's wave estimate. Table 4.1 gives some forms of standardization, where \bar{x}_j^* , R_j^* , and σ_j^* , are the mean, range, and standard deviation of the j th variable, respectively, i.e.,

$$\bar{x}_j^* = \frac{1}{n} \sum_{i=1}^n x_{ij}^*, \quad (4.3a)$$

$$R_j^* = \max_{1 \leq i \leq n} x_{ij}^* - \min_{1 \leq i \leq n} x_{ij}^*, \quad (4.3b)$$

$$\sigma_j^* = \left[\frac{1}{n-1} \sum_{i=1}^n (x_{ij}^* - \bar{x}_j^*)^2 \right]^{\frac{1}{2}}. \quad (4.3c)$$

We now discuss in detail several common forms of standardization and their properties.

The z -score is a form of standardization used for transforming normal variants to standard score form. Given a set of raw data D^* , the z -score standardization formula is defined as

$$x_{ij} = Z_1(x_{ij}^*) = \frac{x_{ij}^* - \bar{x}_j^*}{\sigma_j^*}, \quad (4.4)$$

where \bar{x}_j^* and σ_j^* are the sample mean and standard deviation of the j th attribute, respectively.

The transformed variable will have a mean of 0 and a variance of 1. The location and scale information of the original variable has been lost. This transformation is also presented in (Jain and Dubes, 1988, p. 24).

One important restriction of the z -score standardization Z_1 is that it must be applied in global standardization and not in within-cluster standardization (Milligan and Cooper, 1988). In fact, consider the case where two well-separated clusters exist in the data. If a sample is located at each of the two centroids, then the within-cluster standardization would standardize the samples located at the centroids to zero vectors. Any clustering algorithm will group the two zero vectors together, which means that the two original samples will be grouped to a cluster. This gives a very misleading clustering result.

Table 4.1. Some data standardization methods, where \bar{x}_j^* , R_j^* , and σ_j^* are defined in equation (4.3).

Name	L_j	M_j
z -score	\bar{x}_j^*	σ_j^*
USTD	0	σ_j^*
Maximum	0	$\max_{1 \leq i \leq n} x_{ij}^*$
Mean	\bar{x}_j^*	1
Median	$x_{\frac{n+1}{2}j}^*$ if n is odd $\frac{1}{2}(x_{\frac{n}{2}j}^* + x_{\frac{n+2}{2}j}^*)$ if n is even	1
Sum	0	$\sum_{i=1}^n x_{ij}^*$
Range	$\min_{1 \leq i \leq n} x_{ij}^*$	R_j^*

The USTD (the weighted uncorrected standard deviation) standardization is similar to the z -score standardization and is defined as

$$x_{ij} = Z_2(x_{ij}^*) = \frac{x_{ij}^*}{\sigma_j^*}, \quad (4.5)$$

where σ_j^* is defined in equation (4.3c).

The variable transformed by Z_2 will have a variance of 1. Since the scores have not been centered by subtracting the mean, the location information between scores remains. Thus, the standardization Z_2 will not suffer the problem of the loss of information about the cluster centroids.

The third standardization method presented in Milligan and Cooper (1988) is to use the maximum score on the variable:

$$x_{ij} = Z_3(x_{ij}^*) = \frac{x_{ij}^*}{\max_{1 \leq i \leq n} x_{ij}^*}. \quad (4.6)$$

A variable X transformed by Z_3 will have a mean $\frac{\bar{X}}{\max(X)}$ and standard deviation $\frac{\sigma_X}{\max(X)}$, where \bar{X} and σ_X are the mean and standard deviation of the original variable. Z_3 is susceptible to the presence of outliers (Milligan and Cooper, 1988). If a large single observation on a variable is presented, Z_3 will standardize the remaining values to near 0. Z_3 seems to be meaningful only when the variable is a measure in a ratio scale (Milligan and Cooper, 1988).

Two standardizations that involve using the range of the variable have been presented in (Milligan and Cooper, 1988):

$$x_{ij} = Z_4(x_{ij}^*) = \frac{x_{ij}^*}{R_j^*}, \quad (4.7a)$$

$$x_{ij} = Z_5(x_{ij}^*) = \frac{x_{ij}^* - \min_{1 \leq i \leq n} x_{ij}^*}{R_j^*}, \quad (4.7b)$$

where R_j^* is the range of the j th attribute defined in equation (4.3b).

A variable X transformed by Z_4 and Z_5 will have means of $\frac{\bar{X}}{\max(X) - \min(X)}$ and $\frac{\bar{X} - \min(X)}{\max(X) - \min(X)}$, respectively, and have the same standard deviation $\frac{\sigma_X}{\max(X) - \min(X)}$. Both Z_4 and Z_5 are susceptible to the presence of outliers.

A standardization based on normalizing to the sum of the observations presented in (Milligan and Cooper, 1988) is defined as

$$x_{ij} = Z_6(x_{ij}^*) = \frac{x_{ij}^*}{\sum_{i=1}^n x_{ij}^*}. \quad (4.8)$$

The transformation Z_6 will normalize the sum of transformed values to unity and the transformed mean will be $\frac{1}{n}$. Thus, the mean will be constant across all the variables.

A very different approach of standardization that involves converting the scores to ranks is presented in (Milligan and Cooper, 1988) and is defined as

$$x_{ij} = Z_7(x_{ij}^*) = \text{Rank}(x_{ij}^*), \quad (4.9)$$

where $\text{Rank}(X)$ is the rank assigned to X .

A variable transformed by Z_7 will have a mean of $\frac{n+1}{2}$ and a variance of $(n + 1)\left(\frac{2n+1}{6} - \frac{n+1}{4}\right)$. The rank transformation reduces the impact of outliers in the data.

Conover and Iman (1981) suggested four types of rank transformation. The first rank transformation presented is ranked from smallest to largest, with the smallest score having rank one, the second smallest score having rank two, and so on. Average ranks are assigned in case of ties.

4.2 Data Transformation

Data transformation has something to do with data standardization, but it is more complicated than data standardization. Data standardization concentrates on variables, but data transformation concentrates on the whole data set. As such, data standardization can be viewed as a special case of data transformation. In this section, we present some data transformation techniques that can be used in cluster analysis.

4.2.1 Principal Component Analysis

The main purpose of principal component analysis (PCA) (Ding and He, 2004; Jolliffe, 2002) is to reduce the dimensionality of a high-dimensional data set consisting of a large number of interrelated variables and at the same time to retain as much as possible of the variation present in the data set. The principal components (PCs) are new variables that are uncorrelated and ordered such that the first few retain most of the variation present in all of the original variables.

The PCs are defined as follows. Let $\mathbf{v} = (v_1, v_2, \dots, v_d)'$ be a vector of d random variables, where ' is the transpose operation. The first step is to find a linear function $\mathbf{a}_1' \mathbf{v}$ of the elements of \mathbf{v} that maximizes the variance, where \mathbf{a}_1 is a d -dimensional vector $(a_{11}, a_{12}, \dots, a_{1d})'$, so

$$\mathbf{a}_1' \mathbf{v} = \sum_{i=1}^d a_{1i} v_i.$$

After finding $\mathbf{a}_1' \mathbf{v}, \mathbf{a}_2' \mathbf{v}, \dots, \mathbf{a}_{j-1}' \mathbf{v}$, we look for a linear function $\mathbf{a}_j' \mathbf{v}$ that is uncorrelated with $\mathbf{a}_1' \mathbf{v}, \mathbf{a}_2' \mathbf{v}, \dots, \mathbf{a}_{j-1}' \mathbf{v}$ and has maximum variance. Then we will find d such linear functions after d steps. The j th derived variable $\mathbf{a}_j' \mathbf{v}$ is the j th PC. In general, most of the variation in \mathbf{v} will be accounted for by the first few PCs.

To find the form of the PCs, we need to know the covariance matrix Σ of \mathbf{v} . In most realistic cases, the covariance matrix Σ is unknown, and it will be replaced by a sample covariance matrix (cf. Chapter 6). For $j = 1, 2, \dots, d$, it can be shown that the j th PC is given by $z_j = \mathbf{a}_j' \mathbf{v}$, where \mathbf{a}_j is an eigenvector of Σ corresponding to the j th largest eigenvalue λ_j .

In fact, in the first step, $z_1 = \mathbf{a}_1' \mathbf{v}$ can be found by solving the following optimization problem:

$$\text{maximize } \text{var}(\mathbf{a}_1' \mathbf{v}) \text{ subject to } \mathbf{a}_1' \mathbf{a}_1 = 1,$$

where $\text{var}(\mathbf{a}_1' \mathbf{v})$ is computed as

$$\text{var}(\mathbf{a}_1' \mathbf{v}) = \mathbf{a}_1' \Sigma \mathbf{a}_1.$$

To solve the above optimization problem, the technique of Lagrange multipliers can be used. Let λ be a Lagrange multiplier. We want to maximize

$$\mathbf{a}_1' \Sigma \mathbf{a}_1 - \lambda(\mathbf{a}_1' \mathbf{a}_1 - 1). \quad (4.10)$$

Differentiating equation (4.10) with respect to \mathbf{a}_1 , we have

$$\Sigma \mathbf{a}_1 - \lambda \mathbf{a}_1 = 0,$$

or

$$(\Sigma - \lambda I_d) \mathbf{a}_1 = 0,$$

where I_d is the $d \times d$ identity matrix.

Thus λ is an eigenvalue of Σ and \mathbf{a}_1 is the corresponding eigenvector. Since

$$\mathbf{a}_1' \Sigma \mathbf{a}_1 = \mathbf{a}_1' \lambda \mathbf{a}_1 = \lambda,$$

\mathbf{a}_1 is the eigenvector corresponding to the largest eigenvalue of Σ . In fact, it can be shown that the j th PC is $\mathbf{a}_j' \mathbf{v}$, where \mathbf{a}_j is an eigenvector of Σ corresponding to its j th largest eigenvalue λ_j (Jolliffe, 2002).

In (Ding and He, 2004), PCA is employed to reduce the dimensionality of the data set and then the k -means algorithm is applied in the PCA subspaces. Other examples of applying PCA in cluster analysis can be found in (Yeung and Ruzzo, 2001). Performing PCA is equivalent to performing singular value decomposition (SVD) on the covariance matrix of the data. ORCLUS uses the SVD (Kanth et al., 1998) technique to find out arbitrarily oriented subspaces with good clustering.

4.2.2 SVD

SVD is a powerful technique in matrix computation and analysis, such as solving systems of linear equations and matrix approximation. SVD is also a well-known linear projection technique and has been widely used in data compression and visualization (Andrews and Patterson, 1976a,b). In this subsection, the SVD method is briefly reviewed.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a numerical data set in a d -dimensional space. Then D can be represented by an $n \times d$ matrix X as

$$X = (x_{ij})_{n \times d},$$

where x_{ij} is the j -component value of \mathbf{x}_i .

Let $\bar{\mu} = (\bar{\mu}_1, \bar{\mu}_2, \dots, \bar{\mu}_d)$ be the column mean of X ,

$$\bar{\mu}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, \quad j = 1, 2, \dots, d,$$

and let \mathbf{e}_n be a column vector of length n with all elements equal to one. Then SVD expresses $X - \mathbf{e}_n \bar{\mu}$ as

$$X - \mathbf{e}_n \bar{\mu} = USV^T, \quad (4.11)$$

where U is an $n \times n$ column orthonormal matrix, i.e., $U^T U = I$ is an identity matrix, S is an $n \times d$ diagonal matrix containing the singular values, and V is a $d \times d$ unitary matrix, i.e., $V^H V = I$, where V^H is the conjugate transpose of V .

The columns of the matrix V are the eigenvectors of the covariance matrix C of X ; precisely,

$$C = \frac{1}{n} X^T X - \bar{\mu}^T \bar{\mu} = V \Lambda V^T. \quad (4.12)$$

Since C is a $d \times d$ positive semidefinite matrix, it has d nonnegative eigenvalues and d orthonormal eigenvectors. Without loss of generality, let the eigenvalues of C be ordered in decreasing order: $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$. Let σ_j ($j = 1, 2, \dots, d$) be the standard deviation of the j th column of X , i.e.,

$$\sigma_j = \left(\frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{\mu}_j)^2 \right)^{\frac{1}{2}}.$$

The trace Σ of C is invariant under rotation, i.e.,

$$\Sigma = \sum_{j=1}^d \sigma_j^2 = \sum_{j=1}^d \lambda_j.$$

Noting that $\mathbf{e}_n^T X = n \bar{\mu}$ and $\mathbf{e}_n^T \mathbf{e}_n = n$ from equations (4.11) and (4.12), we have

$$\begin{aligned} VS^T SV^T &= VS^T U^T USV^T \\ &= (X - \mathbf{e}_n \bar{\mu})^T (X - \mathbf{e}_n \bar{\mu}) \\ &= X^T X - \bar{\mu}^T \mathbf{e}_n^T X - X^T \mathbf{e}_n \bar{\mu} + \bar{\mu}^T \mathbf{e}_n^T \mathbf{e}_n \bar{\mu} \\ &= X^T X - n \bar{\mu}^T \bar{\mu} \\ &= n V \Lambda V^T. \end{aligned} \quad (4.13)$$

Since V is an orthonormal matrix, from equation (4.13), the singular values are related to the eigenvalues by

$$s_j^2 = n\lambda_j, \quad j = 1, 2, \dots, d.$$

The eigenvectors constitute the PCs of X , and uncorrelated features will be obtained by the transformation $Y = (X - \mathbf{e}_n\bar{\mu})V$. PCA selects the features with the highest eigenvalues.

Examples of applying SVD in clustering can be found in (Drineas et al., 2004), (Drineas et al., 1999), and (Thomasian et al., 1998). In particular, Drineas et al. (2004) proposed a fast SVD algorithm.

4.2.3 The Karhunen-Loëve Transformation

The Karhunen-Loëve (K-L) transformation is concerned with explaining the data structure through a few linear combinations of variables. Like PCA, the K-L transformation is also the optimal way to project d -dimensional points to lower dimensional points such that the error of projections (i.e., the sum of squared distances (SSD)) is minimal (Fukunaga, 1990).

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set in a d -dimensional space, and let X be the corresponding $n \times d$ matrix, i.e., $X = (x_{ij})_{n \times d}$ with x_{ij} the j -component value of \mathbf{x}_i .

\mathbf{x}_i ($i = 1, 2, \dots, n$) are d -dimensional vectors. They can be represented without error by the summation of d linearly independent vectors as

$$\mathbf{x}_i = \sum_{j=1}^d y_{ij} \phi_j^T = \mathbf{y}_i \Phi^T$$

or

$$X = Y \Phi^T, \tag{4.14}$$

where $\mathbf{y}_i = (y_{i1}, y_{i2}, \dots, y_{id})$, and

$$Y = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_d \end{pmatrix}, \quad \Phi = (\phi_1, \phi_2, \dots, \phi_d).$$

The $d \times d$ matrix Φ is the basis matrix and we may further assume that the rows of Φ form an orthonormal set, i.e.,

$$\phi_i \phi_j^T = \begin{cases} 1 & \text{for } i = j, \\ 0 & \text{for } i \neq j, \end{cases}$$

or

$$\Phi \Phi^T = I_{d \times d},$$

where $I_{d \times d}$ is the $d \times d$ identity matrix.

Then, from equation (4.14), the components of \mathbf{y}_j can be calculated by

$$\mathbf{y}_i = \mathbf{x}_i \Phi, \quad i = 1, 2, \dots, n,$$

or

$$Y = X\Phi.$$

Thus Y is simply an orthonormal transformation of X . ϕ_j is called the j th feature vector and y_{ij} is the j th component of the sample \mathbf{x}_i in the feature space.

In order to reduce dimensionality, we choose only m ($m < d$) feature vectors that can approximate X well. The approximation can be obtained by replacing those components of \mathbf{y}_j with preselected constants (Fukunaga, 1990, p. 402):

$$\hat{\mathbf{x}}_i(m) = \sum_{j=1}^m y_{ij}\phi_j^T + \sum_{j=m+1}^d b_{ij}\phi_j^T,$$

or

$$\hat{X}(m) = Y(1, m) \begin{pmatrix} \phi_1^T \\ \vdots \\ \phi_m^T \end{pmatrix} + B \begin{pmatrix} \phi_{m+1}^T \\ \vdots \\ \phi_d^T \end{pmatrix},$$

where $Y(1, m)$ is the $n \times m$ matrix formed by the first m columns of Y , i.e., $Y(1, m) = (y_{ij})_{n \times m}$, and B is an $n \times (m - d)$ matrix with its (i, j) th entry $b_{i,m+j}$.

Without loss of generality, we assume that only the first m components of each \mathbf{y}_j are calculated. Then the error of the resulting approximation is

$$\Delta \mathbf{x}_i(m) = \mathbf{x}_i - \hat{\mathbf{x}}_i(m) = \sum_{j=m+1}^d (y_{ij} - b_{ij})\phi_j^T,$$

or

$$\Delta X(m) = (Y(m+1, d) - B) \begin{pmatrix} \phi_{m+1}^T \\ \vdots \\ \phi_d^T \end{pmatrix},$$

where $Y(m+1, d)$ is the $n \times (m - d)$ matrix formed by the last $m - d$ columns of Y .

Note that \hat{X} and ΔX are random matrices, so the square error of the approximation is

$$\begin{aligned} \bar{\epsilon}^2(m) &= E\{\|\Delta X(m)\|^2\} \\ &= E\{\text{Tr}(\Delta X(m)\Delta X^T(m))\} \\ &= E\{\text{Tr}((Y(m+1, d) - B)(Y(m+1, d) - B)^T)\} \\ &= \sum_{i=1}^n \sum_{j=m+1}^d E\{(y_{ij} - b_{ij})^2\}. \end{aligned} \tag{4.15}$$

For every choice of constant terms b_{ij} , we obtain a value for $\bar{\epsilon}^2(m)$. The optimal choice for b_{ij} is the one that minimizes $\bar{\epsilon}^2(m)$. From equation (4.15), the optimal choice for b_{ij} is

$$\frac{\partial}{\partial b_{ij}} \bar{\epsilon}^2(m) = -2[E\{y_{ij}\} - b_{ij}] = 0,$$

which gives

$$b_{ij} = E\{y_{ij}\} = E\{\mathbf{x}_i\}\phi_j,$$

or

$$B = E\{Y(m+1, d)\} = E(X)(\phi_{m+1}, \dots, \phi_d).$$

Let Σ_X be the covariance matrix of X . Then we have

$$\begin{aligned}\Sigma_X &= (X - E\{X\})^T(X - E\{X\}) \\ &= [(\mathbf{x}_1^T, \dots, \mathbf{x}_n^T) - (E\{\mathbf{x}_1\}^T, \dots, E\{\mathbf{x}_n\}^T)] \left[\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} - \begin{pmatrix} E\{\mathbf{x}_1\} \\ \vdots \\ E\{\mathbf{x}_n\} \end{pmatrix} \right] \\ &= \sum_{i=1}^n (\mathbf{x}_i - E\{\mathbf{x}_i\})^T (\mathbf{x}_i - E\{\mathbf{x}_i\}).\end{aligned}$$

Thus the square error $\bar{\epsilon}^2(m)$ can be rewritten as

$$\begin{aligned}\bar{\epsilon}^2(m) &= \sum_{i=1}^n \sum_{j=m+1}^d E\{(y_{ij} - b_{ij})^2\} \\ &= \sum_{i=1}^n \sum_{j=m+1}^d \phi_j^T (\mathbf{x}_i - E\{\mathbf{x}_i\})^T (\mathbf{x}_i - E\{\mathbf{x}_i\}) \phi_j \\ &= \sum_{j=m+1}^d \phi_j^T \left(\sum_{i=1}^n (\mathbf{x}_i - E\{\mathbf{x}_i\})^T (\mathbf{x}_i - E\{\mathbf{x}_i\}) \right) \phi_j \\ &= \sum_{j=m+1}^d \phi_j^T \Sigma_X \phi_j.\end{aligned}\tag{4.16}$$

It can be shown that the optimum choice for the ϕ_j 's satisfies (Fukunaga, 1990)

$$\Sigma_X \phi_j = \lambda_j \phi_j.$$

That is, the ϕ_j 's should be the eigenvectors of Σ_X . Then equation (4.16) becomes

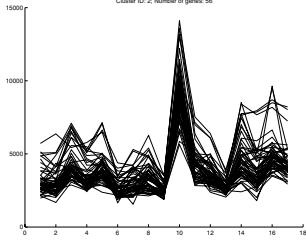
$$\bar{\epsilon}^2(m) = \sum_{j=M+1}^d \lambda_j.$$

Since the covariance matrix of X , Σ_X , is positive semidefinite, it has d nonnegative eigenvalues. If we choose the m eigenvectors that correspond to the largest m eigenvalues, then the square error will be minimized.

4.3 Summary

In this chapter, we have discussed some scale conversion, data standardization, and data transformation techniques. Usually, scale conversion and data standardization consider one variable, while data transformation focuses on the whole data set. Many ideas for scale conversions have also been presented and discussed by Anderberg (1973).

When a data set involves different types of variables, transformation is usually used to convert various variables into a certain variable. Another objective of data transformation is to ensure that each variable in the data set is given an appropriate weight in the analysis.



Chapter 5

Data Visualization

Data visualization techniques are extremely important in cluster analysis. In the final step of data-mining applications, visualization is both vital and a possible cause of misunderstanding.

This chapter introduces some visualization techniques. First, we introduce some nonlinear mappings, such as Sammon's mapping, multidimensional scaling (MDS), and self-organizing maps (SOMs). Then we introduce some methods for visualizing clustered data, including class-preserving maps, parallel coordinates, and tree maps. Finally, we introduce a technique for visualizing categorical data.

5.1 Sammon's Mapping

Sammon Jr. (1969) introduced a method for nonlinear mapping of multidimensional data into a two- or three-dimensional space. This nonlinear mapping preserves approximately the inherent structure of the data and thus is widely used in pattern recognition.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a set of vectors in a d -space and $\{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$ be the corresponding set of vectors in a d^* -space, where $d^* = 2$ or 3 . Let d_{is} be the distance between \mathbf{x}_i and \mathbf{x}_s and d_{is}^{*} be the distance between \mathbf{y}_i and \mathbf{y}_s . Let $\mathbf{y}_1^{(0)}, \mathbf{y}_2^{(0)}, \dots, \mathbf{y}_n^{(0)}$ be a random initial configuration:

$$\mathbf{y}_i^{(0)} = (y_{i1}^{(0)}, y_{i2}^{(0)}, \dots, y_{id^*}^{(0)})^T, \quad i = 1, 2, \dots, n.$$

Now the error E_{SAM} , which represents how well the present configuration of n points in the d^* -space fits the n points in the d -space, is defined as (Sammon Jr., 1969)

$$E_{SAM} = \frac{1}{\sum_{1 \leq i < s \leq n} d_{is}} \sum_{1 \leq i < s \leq n} \frac{(d_{is} - d_{is}^*)^2}{d_{is}} = \frac{1}{c} \sum_{1 \leq i < s \leq n} \frac{(d_{is} - d_{is}^*)^2}{d_{is}}, \quad (5.1)$$

where $c = \sum_{1 \leq i < s \leq n} d_{is}$. The mapping error E_{SAM} is a function of nd^* variables y_{ij} ($i = 1, 2, \dots, n$, $j = 1, 2, \dots, d^*$). The next step of Sammon's method is to adjust the configuration $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n$ such that the error E_{SAM} is minimized. Sammon Jr. (1969) proposed a steepest descent procedure to find the configuration that minimizes the error.

Let $E_{SAM}^{(t)}$ be the mapping error after the t th iteration, i.e.,

$$E_{SAM}^{(t)} = \frac{1}{c} \sum_{1 \leq i < s \leq n} \frac{(d_{is} - (d_{is}^*)^{(t)})^2}{d_{is}},$$

where

$$(d_{is}^*)^{(t)} = \left[\sum_{j=1}^{d^*} \left(y_{ij}^{(t)} - y_{sj}^{(t)} \right)^2 \right]^{\frac{1}{2}}.$$

The new configuration at the $(t + 1)$ th iteration is calculated as

$$y_{ij}^{(t+1)} = y_{ij}^{(t)} - \alpha \cdot \Delta_{ij}^{(t)},$$

where α is a constant that was empirically determined to be 0.3 or 0.4 and

$$\Delta_{ij}^{(t)} = \frac{1}{\left| \frac{\partial^2 E_{SAM}^{(t)}}{\partial [y_{ij}^{(t)}]^2} \right|} \cdot \frac{\partial E_{SAM}^{(t)}}{\partial y_{ij}^{(t)}}.$$

In particular, the partial derivatives are given by (Sammon Jr., 1969)

$$\frac{\partial E_{SAM}^{(t)}}{\partial y_{ij}^{(t)}} = \frac{-2}{c} \sum_{s=1, s \neq i}^n \left(\frac{d_{is} - d_{is}^*}{d_{is} d_{is}^*} \right) (y_{ij} - y_{sj})$$

and

$$\frac{\partial^2 E_{SAM}^{(t)}}{\partial [y_{ij}^{(t)}]^2} = \frac{-2}{c} \sum_{s=1, s \neq i}^n \frac{1}{d_{is} d_{is}^*} \cdot \left[(d_{is} - d_{is}^*) - \frac{(y_{ij} - y_{sj})^2}{d_{is}^*} \left(1 + \frac{d_{is} - d_{is}^*}{d_{is}^*} \right) \right].$$

The procedure to minimize the mapping error is sensitive to the factor or learning rate α and is only practically useful for problems with low values of n (De Backer et al., 1998). De Backer et al. (1998) introduced a better algorithm to minimize the error function (see Section 5.2). Kruskal (1971) pointed out that the minimization of the error function E_{SAM} is strongly related to the metric MDS procedure.

5.2 MDS

MDS (Carroll and Arabie, 1980; De Backer et al., 1998; Borg and Groenen, 1997; Cox and Cox, 1994) refers to a class of algorithms that visualize proximity relations of objects by distances between points in a low-dimensional Euclidean space. MDS algorithms are commonly used to visualize proximity data (i.e., pairwise dissimilarity values instead of feature vectors) by a set of representation points in a suitable embedding space. This section gives a brief introduction to the concept of MDS. For detailed discussions on this subject, readers are referred to monographs by Cox and Cox (1994) and Borg and Groenen (1997).

To introduce MDS, we first introduce some notation. Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a set of n objects in a d -dimensional feature space, and let d_{is} denote the Euclidean distance between \mathbf{x}_i and \mathbf{x}_s . Let d^* be the number of dimensions of the output space, \mathbf{x}_i^* be a d^* -dimensional point representing \mathbf{x}_i in the output space, and d_{is}^* denote the Euclidean distance between \mathbf{x}_i^* and \mathbf{x}_s^* in the output space. Let \mathbf{X} be an $n \times d$ -dimensional vector denoting the coordinates x_{ij} ($1 \leq i \leq n$, $1 \leq j \leq d$) as follows:

$$\mathbf{X} = (X_1, X_2, \dots, X_{nd})^T,$$

where $X_{(i-1)d+j} = x_{ij}$ for $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, d$. Let ∇f denote the gradient vector of function $f(\mathbf{X})$ evaluated at \mathbf{X} , i.e.,

$$\nabla f = (Y_1, Y_2, \dots, Y_{nd})^T,$$

where

$$Y_{(i-1)d+j} = \frac{\partial f(\mathbf{X})}{X_{(i-1)d+j}} = \frac{\partial f(\mathbf{X})}{x_{ij}}, \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, d.$$

Let H_f be the Hessian matrix of the function $f(\mathbf{X})$ evaluated as \mathbf{X} , i.e., $H_f = (h_{lm})$, where

$$h_{lm} = \frac{\partial^2 f(\mathbf{X})}{\partial X_l \partial X_m} = \frac{\partial^2 f(\mathbf{x})}{\partial x_{ij} \partial x_{st}}$$

for $l = (i - 1)d + j$ and $m = (s - 1)d + t$, $1 \leq i, s \leq n$, $1 \leq j, t \leq d$.

MDS can be either metric or nonmetric. If the dissimilarities are proportional to the distances, then it is metric. If the dissimilarities are assumed to be just ordinal, then it is nonmetric (Shepard, 1962; Kruskal, 1964). MDS is conducted such that the distances d_{ij}^* between the representative points match as much as possible some given dissimilarities between the points in the original space or input space.

In nonmetric MDS, where distances d_{ij} serve as dissimilarities in the input space, a loss function is defined and minimized through a gradient descent procedure (De Backer et al., 1998). For example, a loss function can be defined as (De Backer et al., 1998)

$$E_{MDS} = \frac{\sum_{1 \leq i < s \leq n} (d_{is}^* - \hat{d}_{is})^2}{\sum_{1 \leq i < s \leq n} \hat{d}_{is}^2}, \quad (5.2)$$

where \hat{d}_{is} are pseudodistances or target distances derived from the d_{is}^* with Kruskal's monotone regression procedure (Kruskal, 1964) and are calculated in such a way that their rank order matches as well as possible the rank order of d_{ij} and they are as close as possible to the d_{is}^* .

ALGORITHM 5.1. Nonmetric MDS.

Require: D : input data set; d_{is} : dissimilarities;

- 1: Define an initial configuration $\mathbf{X}^{(0)}$;

```

2: while  $E_{MDS}$  has not converged do
3:   Compute the distances  $d_{is}^*$  for the current configuration  $\mathbf{X}^{(t)}$ ;
4:   Compute the target distances  $\hat{d}_{is}$ ;
5:   Compute the gradient  $\nabla E_{MDS}^{(t)}$ ;
6:   Compute the learning rate  $\alpha(t)$ ;
7:    $\mathbf{X}^{(t+1)} \leftarrow \mathbf{X}^{(t)} - \alpha(t) \nabla E_{MDS}^{(t)}$ ;
8: end while
9: Output the  $\mathbf{X}$ .

```

The nonmetric MDS algorithm is described in Algorithm 5.1. In this algorithm, the learning rate $\alpha(t)$ is calculated as

$$\alpha(t) = \frac{\|\nabla E_{MDS}^{(t)}\|^2}{(\nabla E_{MDS}^{(t)})^T \cdot \nabla H_{E_{MDS}}^{(t)} \cdot \nabla E_{MDS}^{(t)}}.$$

In MDS, the minimization problem is nonconvex and sensitive to local minima. Klock and Buhmann (2000) developed a deterministic annealing approach to solving such minimization problems.

5.3 SOM

The SOM, introduced by Kohonen (1990, 1989), is a type of artificial neural network. SOMs reduce dimensions by producing a map of usually one or two dimensions that plots the similarities of the data by grouping similar objects together. SOMs are particularly useful for visualization and cluster analysis in that they can be used to explore the groupings and relations within high-dimensional data by projecting the data onto a two-dimensional image that clearly indicates regions of similarity.

The SOM architecture consists of two fully connected layers: an input layer and a Kohonen layer. The neurons in the Kohonen layer are arranged in a one- or two-dimensional lattice. Figure 5.1 displays the layout of a one-dimensional map where the output neurons are arranged in a one-dimensional lattice. The number of neurons in the input layer matches the number of attributes of the objects. Each neuron in the input layer has a feed-forward connection to each neuron in the Kohonen layer. The inputs are assumed to be normalized, i.e., $\|\mathbf{x}\| = 1$. Inputs to the Kohonen layer can be calculated as

$$y_j = \sum_{i=1}^d w_{ji} x_i, \quad (5.3)$$

where w_{ji} is the weight from the input neuron i to the output neuron j . Under a winner-takes-all paradigm, the neuron in the Kohonen layer with the biggest y_j will become the winning neuron or winner-takes-all neuron.

The algorithm responsible for the formation of the SOM first initializes the weights in the network by assigning them small random values. Then the algorithm proceeds to three essential processes: *competition*, *cooperation*, and *adaptation* (Haykin, 1999).

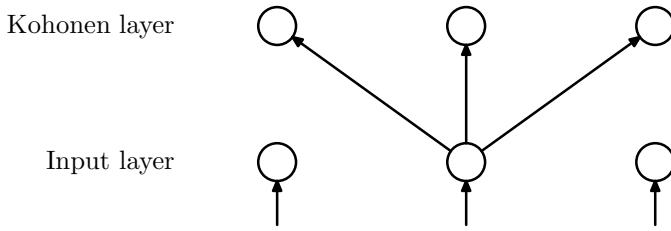


Figure 5.1. The architecture of the SOM.

Competitive process. Let $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ be an object selected at random from the input space, where d is the dimension of the input space. Let the weight vector of neuron j in the Kohonen layer be denoted by

$$\mathbf{w}_j = (w_{j1}, w_{j2}, \dots, w_{jd^*})^T, \quad j = 1, 2, \dots, d^*,$$

where d^* is the total number of neurons in the Kohonen layer. The best match of the input object \mathbf{x} with the weight vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_{d^*}$ can be found by comparing the inner products $\mathbf{w}_1^T \mathbf{x}, \mathbf{w}_2^T \mathbf{x}, \dots, \mathbf{w}_{d^*}^T \mathbf{x}$ and selecting the largest. In fact, the best matching criterion based on maximizing the inner products $\mathbf{w}_j^T \mathbf{x}$ is mathematically equivalent to minimizing the Euclidean distance between the vectors \mathbf{w}_j and \mathbf{x} (Haykin, 1999). Therefore, the index $i(\mathbf{x})$ of the winning neuron for the input object \mathbf{x} may be determined by

$$i(\mathbf{x}) = \arg \min_{1 \leq j \leq d^*} \|\mathbf{x} - \mathbf{w}_j\|.$$

Cooperative process. In the cooperative process, a topological neighborhood is defined so that the winning neuron locates the center of a topological neighborhood of cooperating neurons. Let $h_{j,t}$ denote the topological neighborhood centered on winning neuron t and $d_{t,j}$ denote the lateral distance between winning neuron t and excited neuron j . The topological neighborhood $h_{j,t}$ can be a unimodal function of the lateral distance $d_{t,j}$ satisfying the following two conditions (Haykin, 1999):

- (a) $h_{j,t}$ is symmetric about the maximum point defined by $d_{t,j} = 0$.
- (b) The amplitude of $h_{j,t}$ decreases monotonically with increasing lateral distance $d_{t,j}$ and decays to zero as $d_{t,j} \rightarrow \infty$.

For example, $h_{j,t}$ can be the Gaussian function

$$h_{j,t} = \exp\left(-\frac{d_{t,j}^2}{2\sigma^2}\right),$$

where σ is a parameter that measures the degree to which excited neurons in the neighborhood of the winning neuron participate in the learning process.

In the case of a one-dimensional lattice, the lateral distance $d_{t,j}$ can be defined as

$$d_{t,j} = |t - j|.$$

In the case of a two-dimensional lattice, the lateral distance $d_{t,j}$ can be defined as

$$d_{t,j} = \|\mathbf{r}_t - \mathbf{r}_j\|,$$

where \mathbf{r}_t and \mathbf{r}_j are discrete vectors defining the position of excited neuron j and the position of winning neuron t , respectively.

Adaptive process. In the adaptive process, the weight vector \mathbf{w}_j of neuron j changes according to the input object \mathbf{x} . Given the weight vector $\mathbf{w}_j^{(s)}$ of neuron j at time or iteration s , the new weight vector $\mathbf{w}_j^{(s+1)}$ at time $s + 1$ is defined by (Haykin, 1999)

$$\mathbf{w}_j^{(s+1)} = \mathbf{w}_j^{(s)} + \eta(s)h_{j,i(\mathbf{x})}(s)(\mathbf{x} - \mathbf{w}_j^{(s)}), \quad (5.4)$$

where $\eta(s)$ is the learning-rate parameter defined as

$$\eta(s) = \eta_0 \exp\left(-\frac{s}{\tau_2}\right), \quad s = 0, 1, 2, \dots,$$

and $h_{j,i(\mathbf{x})}(s)$ is the neighborhood function defined as

$$h_{j,i(\mathbf{x})}(s) = \exp\left(-\frac{d_{i(\mathbf{x}),j}^2}{2\sigma^2(s)}\right), \quad s = 0, 1, 2,$$

with $\sigma(s)$ specified by

$$\sigma(s) = \sigma_0 \left(-\frac{s}{\tau_1}\right).$$

The constants η_0 , σ_0 , τ_1 , and τ_2 can be configured as follows (Haykin, 1999):

$$\begin{aligned} \eta_0 &= 0.1, \\ \sigma_0 &= \text{the radius of the lattice}, \\ \tau_1 &= \frac{1000}{\log \sigma_0}, \\ \tau_2 &= 1000. \end{aligned}$$

ALGORITHM 5.2. The pseudocode of the SOM algorithm.

Require: D : the data set; d^* : the dimension of the feature map; η_0 , σ_0 , τ_1 , τ_2 : parameters;

- 1: Initialize weight vectors $\mathbf{w}_j^{(0)}$ for $j = 1, 2, \dots, d^*$ by selecting at random objects from D ;
- 2: **repeat**
- 3: Draw an object \mathbf{x} from D with a certain probability;
- 4: Find the winning neuron $i(\mathbf{x})$ at time step s by using the minimum-distance Euclidean criterion:

$$i(\mathbf{x}) = \arg \min_{1 \leq j \leq d^*} \|\mathbf{x} - \mathbf{w}_j^{(s)}\|;$$

-
- 5: Update the weight vectors of all neurons by the formula in equation (5.4);
 - 6: **until** No noticeable changes in the feature map
 - 7: Output the feature map.

The SOM algorithm is summarized in Algorithm 5.2. The SOM is one type of unsupervised competitive learning. Unlike supervised training algorithms such as backpropagation (Hertz et al., 1991), unsupervised learning algorithms have no expected outputs. One advantage of the SOM is that it constantly learns and changes with changing conditions and inputs. The SOM was implemented as a toolbox in MATLAB (Vesanto, 2000; Vesanto et al., 1999a,b). Eklund et al. (2003) illustrated how to use the SOM technique in financial performance analysis and benchmarking.

Kohonen's SOM and Sammon's nonlinear mapping (see Section 5.1) are topology- and distance-preserving mapping techniques commonly used for multivariate data projections. However, the computations for both techniques are high. To solve this problem, Konig (2000) proposed a two-stage neural projection approach for hierarchical mappings. This new approach is based on SOM and Sammon's nonlinear mapping and avoids recomputation of the mapping when additional data points or data sets are included.

5.4 Class-preserving Projections

The class-preserving projections introduced by Dhillon et al. (1998) map multi dimensional data onto two-dimensional planes and maintain the high-dimensional class structures. The main idea behind these class-preserving projections is to preserve interclass distances.

Let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ be a d -dimensional data set, $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k$ denote the class means, and n_1, n_2, \dots, n_k denote the class sizes. Let \mathbf{w}_1 and \mathbf{w}_2 be an orthonormal basis for the candidate two-dimensional plane of projection. Then the point \mathbf{x}_i and the mean \mathbf{z}_j are projected to the pairs $(\mathbf{w}_1^T \mathbf{x}_i, \mathbf{w}_2^T \mathbf{x}_i)$ and $(\mathbf{w}_1^T \mathbf{m}_j, \mathbf{w}_2^T \mathbf{m}_j)$, respectively. One way to obtain good separation of the projected classes is to maximize the difference between the projected means. In other words, the two vectors \mathbf{w}_1 and \mathbf{w}_2 are selected such that the objective function

$$C(\mathbf{w}_1, \mathbf{w}_2) = \text{Tr}(W^T SW) \quad (5.5)$$

is maximized, where $\text{Tr}(M)$ denotes the trace of the matrix M ,

$$W = [\mathbf{w}_1, \mathbf{w}_2], \quad \mathbf{w}_1^T \mathbf{w}_2 = 0, \quad \mathbf{w}_i^T \mathbf{w}_i = 1, \quad i = 1, 2,$$

and

$$S = \sum_{i=2}^k \sum_{j=1}^{i-1} n_i n_j (\mathbf{z}_i - \mathbf{z}_j)(\mathbf{z}_i - \mathbf{z}_j)^T.$$

The vectors \mathbf{w}_1 and \mathbf{w}_2 that maximize the objective function in equation (5.5) are the eigenvectors corresponding to the two largest eigenvalues of S (Dhillon et al., 1998).

The matrix S in equation (5.5) is positive semidefinite and can be interpreted as an interclass scatter matrix (see Subsection 6.1.3). Within-class scatter is ignored in these class-preserving projections.

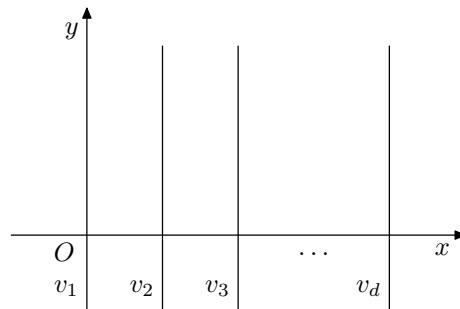


Figure 5.2. The axes of the parallel coordinates system.

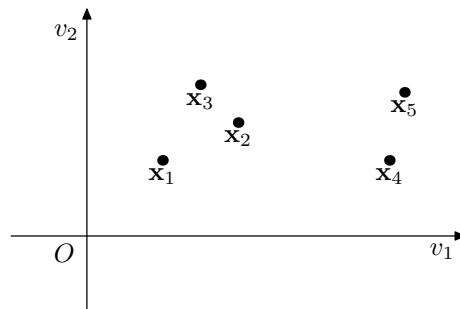


Figure 5.3. A two-dimensional data set containing five points.

5.5 Parallel Coordinates

Inselberg and Dimsdale (1990) proposed a visualization methodology called parallel coordinates for visualizing analytic and synthetic geometry in \mathbb{R}^d . This methodology induces a nonprojective mapping between sets in \mathbb{R}^d and sets in \mathbb{R}^2 and yields a graphical representation of multidimensional relations rather than just finite point sets.

The axes of the parallel coordinates system for Euclidean space \mathbb{R}^d are d copies of the real lines that are labeled v_1, v_2, \dots, v_d and placed on the xy -Cartesian coordinate plane equidistant and perpendicular to the x -axis. The real line labeled v_1 is coincident with the y -axis. Figure 5.2 depicts the d axes of the parallel coordinates system for Euclidean space \mathbb{R}^d on a Cartesian plane. A point $\mathbf{x} = (x_1, x_2, \dots, x_d)^T \in \mathbb{R}^d$ is represented by a polygonal line whose d vertices are at $(j-1, x_j)$ on the v_j -axis for $j = 1, 2, \dots, d$. The mapping between points in \mathbb{R}^d and polygonal lines with vertices on v_1, v_2, \dots, v_d is one-to-one.

Points in a two-dimensional space are represented by line segments between the v_1 -axis and the v_2 -axis. For example, the parallel coordinates plot of the five points in Figure 5.3 is given in Figure 5.4.

The parallel coordinates system can be used to plot time series and gene expression data (see Section 18.6). This methodology works well when the number of dimensions d is small. However, this methodology is rendered ineffective if the number of dimensions or the number of objects gets too high. Inselberg and Dimsdale (1990) also

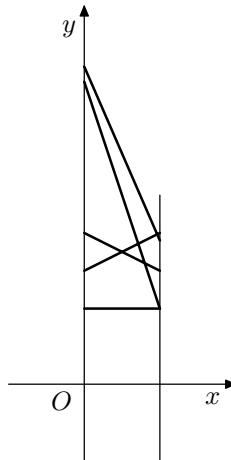


Figure 5.4. The parallel coordinates plot of the five points in Figure 5.3.

discussed multidimensional lines and hyperplanes for the purpose of visualization. The parallel coordinates methodology was implemented in the interactive hierarchical displays (IHDs) (Yang et al., 2003b).

Fua et al. (1999) proposed several extensions to the parallel coordinates display technique for large data sets. Other discussions of parallel coordinates can be found in (Wegman, 1990), (Miller and Wegman, 1991), (Andrienko and Andrienko, 2004), and (Johansson et al., 2006). Like the parallel coordinates, the star coordinates were proposed by Kandogan (2001) to visualize multidimensional clusters, trends, and outliers.

5.6 Tree Maps

Commonly, a tree or dendrogram is used to visualize the results of a hierarchical clustering algorithm. But this common visualization method suffers from the following problems (Wills, 1998):

- (a) It is hard to see the tree even for a moderately sized tree of 50 to 100 leaves.
- (b) The sizes of clusters are not obvious.
- (c) If the cut level is changed, the tree displayed often changes dramatically.
- (d) It offers no hint for future events if the number of clusters is increased.

As a solution to the above problem, a tree map (Shneiderman, 1992; Wills, 1998) is developed to visualize the tree structure of a hierarchical clustering by taking a specified rectangular area and recursively subdividing it up based on the tree structure. More specifically, this methodology looks at the first level of the tree and splits up the viewing area horizontally into m rectangles if the first node has m children. The area allocated to a rectangle is proportional to the size of the subtree underneath the corresponding child node. This methodology then looks at the next level of the tree and performs the same procedure for each node there, except it subdivides the area vertically.

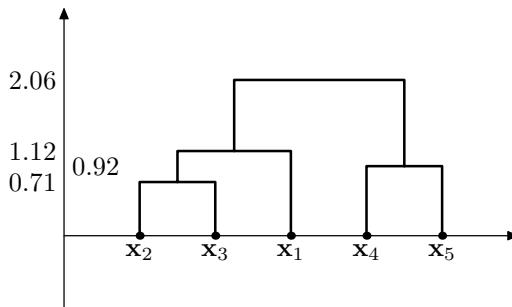


Figure 5.5. The dendrogram of the single-linkage hierarchical clustering of the five points in Figure 5.3.

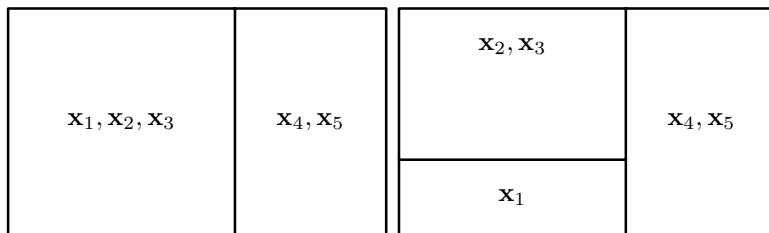


Figure 5.6. The tree maps of the dendrogram in Figure 5.5. The left tree map corresponds to cut levels between 1.12 and 2.06; the right tree map corresponds to cut levels between 0.92 and 1.12.

Taking the data set in Figure 5.3 as an example, we apply a single-linkage hierarchical algorithm to the data set and obtain the dendrogram shown in Figure 5.5. The numbers along the vertical line on the left are the heights of the dendrogram tree and its subtrees. Figure 5.6 gives two tree maps corresponding to different cut levels of the dendrogram shown in Figure 5.5.

The major advantage of the tree map methodology is its capability of visualizing large data sets. Wills (1998) used this methodology to cluster and visualize a large telecommunications data set summarizing call traffic. Pseudocode for the algorithm for drawing such a tree map is also presented by Wills (1998).

5.7 Categorical Data Visualization

Chang and Ding (2005) proposed a method for visualizing clustered categorical data so that users can adjust the clustering parameters based on the visualization. In this method, a special three-dimensional coordinate system is used to represent the clustered categorical data.

The main idea behind this visualization technique is that each attribute v_j of a cluster C_m has an attribute value A_{jl} (see Section 2.1 for categorical data notation) such that the probability of this attribute value in the cluster, $P(v_j = A_{jl}|C_m)$, is maximum and close to

Table 5.1. The coordinate system for the two clusters of the data set in Table 2.1.

v_1	v_2	v_3	v_4	v_5	v_6
$A\ 1$	$A\ 1$	$A\ 1$	$A\ 1$	$B\ \frac{1}{3}$	$B\ \frac{1}{3}$
				$C\ \frac{1}{3}$	$D\ \frac{1}{3}$
				$D\ \frac{1}{3}$	$C\ \frac{1}{3}$

v_1	v_2	v_3	v_4	v_5	v_6
$B\ 1$	$B\ 1$	$C\ \frac{1}{2}$	$C\ \frac{1}{2}$	$D\ \frac{1}{2}$	$C\ \frac{1}{2}$
		$D\ \frac{1}{2}$	$D\ \frac{1}{2}$	$C\ \frac{1}{2}$	$D\ \frac{1}{2}$

one. A cluster is represented by these attribute values. The three-dimensional coordinate system to plot an attribute value is constructed such that the x -axis represents the attribute indices (i.e., j denotes v_j), the y -axis represents the attribute value (or state) indices (i.e., l denotes A_{jl}), and the z -axis represents the probability that the attribute value is in the cluster. In other words, the cluster C_m is represented by d three-dimensional points $(j, l, P(A_{jl}|C_m))$ for $j = 1, 2, \dots, d$, where d is the number of attributes.

To display a set of clusters, the technique constructs a coordinate system such that interference among different clusters can be minimized in order to observe closeness. To obtain this effect, the technique first examines the attribute value with the highest proportion for each cluster, then summarizes the number of distinct attribute values for each attribute, and then sorts them in increasing order. Attributes with the same number of distinct attribute values are further ordered by the lowest value of their proportions. Attributes with the least number of distinct attribute values are put in the middle of the x -axis and the others are put at the two ends according to their orders. After this arrangement, the cluster C_m is represented by d three-dimensional points $(L_x(v_j), L_y(A_{jl}), P(A_{jl}|C_m))$ for $j = 1, 2, \dots, d$, where the function $L_x(v_j)$ returns the x -coordinate for the attribute v_j and the function $L_y(A_{jl})$ returns the y -coordinate for the attribute value A_{jl} .

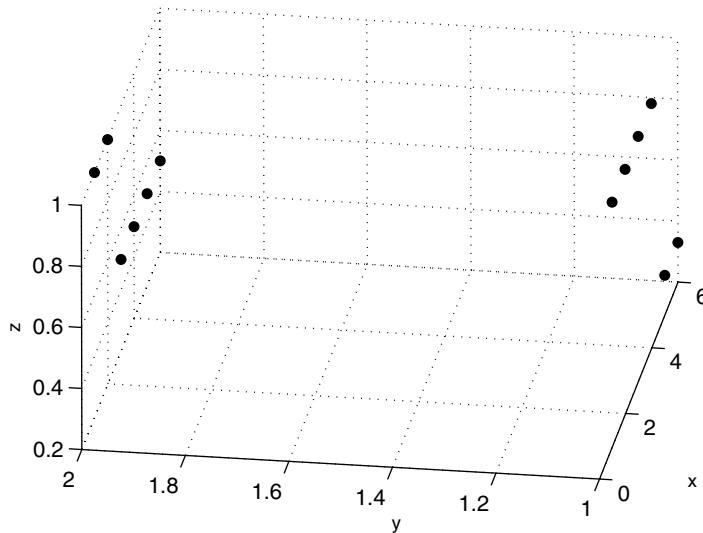
Consider the categorical data set in Table 2.1, for example. We see that there are two clusters: cluster C_1 contains $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_3$ and cluster C_2 contains $\mathbf{x}_4, \mathbf{x}_5$. To represent these two clusters, we can construct the coordinate system as follows:

1. For each cluster, summarize the attribute values and their proportions in the cluster in a table (see Table 5.1).
2. Examine the attribute value with the highest proportion for each cluster and sort them: v_1 and v_2 both have two distinct attribute values (i.e., A and B) of proportion 1; v_3 and v_4 both have two distinct attribute values (i.e., A and C) of proportions 1 and $\frac{1}{2}$, respectively; v_5 and v_6 both have two distinct attribute values (i.e., B and C). Thus, the order for the six attributes can be $v_1, v_2, v_3, v_4, v_5, v_6$.

Table 5.2. Coordinates of the attribute values of the two clusters in Table 5.1.

v_1	v_2	v_1	v_2	v_5	v_6
(1,1,1)	(2,1,1)	(3,1,1)	(4,1,1)	(5,1,0.33)	(6,1,0.33)

v_1	v_2	v_1	v_2	v_5	v_6
(1,2,1)	(2,2,1)	(3,2,0.5)	(4,2,0.5)	(5,2,0.5)	(6,2,0.5)

**Figure 5.7.** Plot of the two clusters in Table 5.1.

3. Determine coordinates for the attribute values (see Table 5.2).
4. Plot the coordinates in a three-dimensional space.

The visualization of the two categorical clusters in Table 5.1 is shown in Figure 5.7, from which we see that the points are scattered in two groups. It is easy to see that the data set in Table 2.1 contains two clusters. Additional examples of this visualization technique can be found in (Chang and Ding, 2004) and (Chang and Ding, 2005).

Hsu (2006) proposed a method to visualize categorical data and mixed-type data using the SOM. Beygelzimer et al. (2001) considered how to order categorical values to yield better visualization.

5.8 Other Visualization Techniques

Several other methods have been proposed for visualizing high-dimensional data. Andrews (1972) introduced a method to plot high-dimensional data with curves, one curve for each data item, obtained by using the components of the data vectors as coefficients of orthogonal sinusoids, which are then added together pointwise. Chernoff (1973) introduced a method to visualize multivariate data with faces. Each point in a d -dimensional space ($d \leq 18$) is represented by a facial caricature whose features such as length of nose are determined by components of the point. The major common drawback of these two methods is that they are not effective for visualization of large data sets. If a data set is large, all the objects are portrayed separately in a display, making it difficult to see the inherent structure of the data set.

Hofmann and Buhmann (1995) proposed three strategies derived in the maximum entropy framework for visualizing data structures. Pölzlauer et al. (2006) proposed two methods for depicting SOM based on vector fields. Vesanto (1999) presented a categorization of many visualization methods for the SOM. MDS and minimum spanning tree (MST) for visualizing hierarchical clustering are discussed by Kim et al. (2000b). Klock and Buhmann (2000) proposed a deterministic annealing approach to solving the MDS minimization problem.

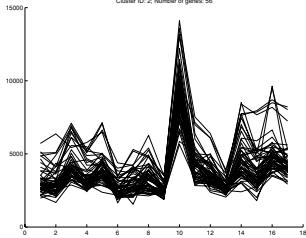
Somorjai et al. (2004) proposed a distance-based mapping for visualization of high-dimensional patterns and their relative relationships. The original distances between points are preserved exactly with respect to any two reference patterns in the relative distance plane (RDP), a special two-dimensional coordinate system. Faloutsos and Lin (1995) proposed an algorithm, called FastMap, to map objects into points in some m -dimensional space with the distances being preserved, where m is a user-specified parameter.

To visualize large-scale hierarchical data, Itoh et al. (2004) proposed a rectangle-packing algorithm that can provide good overviews of complete structures and the content of the data in one display space. Sprenger et al. (2000) proposed an algorithm called H-BLOB that can group and visualize cluster hierarchies at multiple levels of detail. H-BLOB is suited for the visualization of very large data sets. Koren and Harel (2003) proposed an embedded algorithm that can preserve the structure of a predefined partitional or hierarchical clustering to visualize clustered data.

Egan et al. (1998) introduced a tool called FCLUS (Fuzzing CLustering Simulation Tool) to visualize the results of fuzzy clusterings. This tool can also be used to demonstrate the computational method of the algorithm and thus can be used as a teaching tool in classrooms.

5.9 Summary

This chapter introduced some dimension reduction and visualization techniques. Visualization is an area that has attracted many researchers' attention recently. Many papers on visualization have been published. In this chapter, we focused on techniques for visualization of clustered data. For other visualization methods, readers are referred to (Keim and Kriegel, 1996), in which a general survey of visualization methods for data mining is presented.



Chapter 6

Similarity and Dissimilarity Measures

This chapter introduces some widely used similarity and dissimilarity measures for different attribute types. We start by introducing notions of *proximity matrices*, *proximity graphs*, *scatter matrices*, and *covariance matrices*. Then we introduce measures for several types of data, including numerical data, categorical data, binary data, and mixed-typed data, and some other measures. Finally, we introduce various similarity and distance measures between clusters and variables.

6.1 Preliminaries

A similarity coefficient indicates the strength of the relationship between two data points (Everitt, 1993). The more the two data points resemble one another, the larger the similarity coefficient is. Let $\mathbf{x} = (x_1, x_2, \dots, x_d)$ and $\mathbf{y} = (y_1, y_2, \dots, y_d)$ be two d -dimensional data points. Then the similarity coefficient between \mathbf{x} and \mathbf{y} will be some function of their attribute values, i.e.,

$$s(\mathbf{x}, \mathbf{y}) = s(x_1, x_2, \dots, x_d, y_1, y_2, \dots, y_d). \quad (6.1)$$

Similarity is usually symmetric, i.e., $s(\mathbf{x}, \mathbf{y}) = s(\mathbf{y}, \mathbf{x})$. Asymmetric similarity measures have also been discussed in (Constantine and Gower, 1978). A metric is a distance function f defined in a set E that satisfies the following four properties (Anderberg, 1973; Zhang and Srihari, 2003):

1. nonnegativity: $f(\mathbf{x}, \mathbf{y}) \geq 0$;
2. reflexivity: $f(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$;
3. commutativity: $f(\mathbf{x}, \mathbf{y}) = f(\mathbf{y}, \mathbf{x})$;
4. triangle inequality: $f(\mathbf{x}, \mathbf{y}) \leq f(\mathbf{x}, \mathbf{z}) + f(\mathbf{y}, \mathbf{z})$,

where \mathbf{x} , \mathbf{y} , and \mathbf{z} are arbitrary data points.

A dissimilarity function is a metric defined in a set. But by a similarity function, we mean a function $s(\cdot, \cdot)$ measured on any two data points in a data set that satisfies the following properties (Kaufman and Rousseeuw, 1990):

1. $0 \leq s(\mathbf{x}, \mathbf{y}) \leq 1$,
2. $s(\mathbf{x}, \mathbf{x}) = 1$,
3. $s(\mathbf{x}, \mathbf{y}) = s(\mathbf{y}, \mathbf{x})$,

where \mathbf{x} and \mathbf{y} are two arbitrary data points in the set.

Generally, there are many other similarity and dissimilarity structures. Let D be a data set. Hartigan (1967) lists 12 similarity structures for a similarity measure S defined on D :

1. S defined on $D \times D$ is a Euclidean distance.
2. S defined on $D \times D$ is a metric.
3. S defined on $D \times D$ is symmetric and real valued.
4. S defined on $D \times D$ is real valued.
5. S is a complete “similarity” order \leq_S on $D \times D$ (each pair of objects can be ordered).
6. S is a partial similarity order \leq_S on $D \times D$ (each comparable pair of objects can be ordered, but not all pairs of objects need to be comparable).
7. S is a tree on D (Hartigan, 1967).
8. S is a complete “relative similarity” order \leq_i on D for each i in D ($j \leq_i k$ means that j is no more similar to i than k is to i).
9. S is a partial relative similarity order \leq_i on D .
10. S is a similarity dichotomy on $D \times D$ in which $D \times D$ is divided into a set of similar pairs and a set of dissimilar pairs.
11. S is a similarity trichotomy on $D \times D$ in which $D \times D$ consists of similar pairs, dissimilar pairs, and the remaining pairs.
12. S is a partition of D into sets of similar objects.

6.1.1 Proximity Matrix

A proximity matrix (Jain and Dubes, 1988) is a matrix that contains the pairwise indices of proximity of a data set. Usually, proximity matrices are symmetrical. In what follows, a proximity index refers to either a similarity index or a dissimilarity index.

Given a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, each object of which is described by a d -dimensional feature vector, the distance matrix for D is defined as

$$M_{dist}(D) = \begin{pmatrix} 0 & d_{12} & \cdots & d_{1n} \\ d_{21} & 0 & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & 0 \end{pmatrix}, \quad (6.2)$$

where $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ with respect to some distance function $d(\cdot, \cdot)$.

The similarity matrix for D is defined as

$$M_{sim}(D) = \begin{pmatrix} 1 & s_{12} & \cdots & s_{1n} \\ s_{21} & 1 & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & 1 \end{pmatrix}, \quad (6.3)$$

where $s_{ij} = s(\mathbf{x}_i, \mathbf{x}_j)$ with respect to some similarity measure $s(\cdot, \cdot)$.

The distance matrix $M_{dist}(D)$ and similarity matrix $M_{sim}(D)$ of a data set D defined in (6.2) and (6.3) are two examples of a proximity matrix. If the distance function and similarity function are symmetrical, then the two proximity matrices are symmetrical.

6.1.2 Proximity Graph

A proximity graph is a weighted graph, where the nodes are the data points being clustered, and the weighted edges represent the proximity indices between points, i.e., the entries of the proximity matrix. A directed graph corresponds to an asymmetrical proximity matrix, while an undirected graph corresponds to a symmetrical proximity matrix.

6.1.3 Scatter Matrix

Given a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, each object of which is described by a d -dimensional feature vector, i.e., $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})$, the scatter matrix for D is defined as (Wilks, 1962)

$$M_t(D) = \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})^T (\mathbf{x}_i - \bar{\mathbf{x}}), \quad (6.4)$$

where $\bar{\mathbf{x}}$ is the arithmetic average, i.e.,

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i.$$

The scatter matrix $M_t(D)$ is also referred to as the matrix sum of squares. The trace of the scatter matrix $M_t(D)$ is said to be the statistical scatter of the data set D and is denoted by

$$\text{Tr}(M_t(D)) = \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T. \quad (6.5)$$

For a given cluster C of D , $M_t(C)$ is also called the within-scatter matrix of C . Let $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ be a partition of data set D . Then the within-cluster scatter matrix for this partition is defined as

$$M_w(\mathcal{C}) = \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{z}_i)^T (\mathbf{x} - \mathbf{z}_i), \quad (6.6)$$

where \mathbf{z}_i is the mean of cluster C_i , i.e.,

$$\mathbf{z}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}.$$

Similarly, the between-cluster scatter matrix for this partition is defined as

$$M_b(\mathcal{C}) = M_t(D) - M_w(\mathcal{C}), \quad (6.7)$$

where $M_t(D)$ is defined in (6.4).

6.1.4 Covariance Matrix

Covariance is a well-known concept in statistics. Let D be a data set with n objects, each of which is described by d attributes v_1, v_2, \dots, v_d . The attributes v_1, v_2, \dots, v_d are also referred to as variables. The covariance between two variables v_r and v_s is defined to be the ratio of the sum of the products of their deviation from the mean to the number of objects (Rummel, 1970), i.e.,

$$c_{rs} = \frac{1}{n} \sum_{i=1}^n (x_{ir} - \bar{x}_r)(x_{is} - \bar{x}_s),$$

where x_{ij} is the j th component of data point \mathbf{x}_i and \bar{x}_j is the mean of all data points in the j th variable, i.e.,

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, \quad j = 1, 2, \dots, d.$$

The covariance matrix is a $d \times d$ matrix in which the entry (r, s) contains the covariance between variable v_r and v_s , i.e.,

$$\Sigma = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1d} \\ c_{21} & c_{22} & \cdots & c_{2d} \\ \vdots & \vdots & & \vdots \\ c_{d1} & c_{d2} & \cdots & c_{dd} \end{pmatrix}. \quad (6.8)$$

From the definition of the covariance, the covariance matrix defined in equation (6.8) can be written as

$$\Sigma = \frac{1}{n} \mathbf{X}^T \mathbf{X},$$

where \mathbf{X}^T denotes the transpose of \mathbf{X} , and \mathbf{X} is an $n \times d$ matrix with the (i, j) th element $x_{ij} - \bar{x}_j$, i.e.,

$$\mathbf{X} = (x_{ij} - \bar{x}_j)_{n \times d} = \begin{pmatrix} \mathbf{x}_1 - \bar{x}_1 \mathbf{e}_d \\ \mathbf{x}_2 - \bar{x}_2 \mathbf{e}_d \\ \vdots \\ \mathbf{x}_d - \bar{x}_d \mathbf{e}_d \end{pmatrix}, \quad (6.9)$$

where \mathbf{e}_d is the d -dimensional identity vector, i.e., $\mathbf{e}_d = (1, 1, \dots, 1)$.

Sample covariance matrices are often used in multivariate analysis. These matrices are different from covariance matrices. For the given data set D above, the sample covariance matrix of D is defined to be a $d \times d$ matrix as (Jolliffe, 2002)

$$\mathbf{S} = \frac{n}{n-1} \Sigma = \frac{1}{n-1} \mathbf{X}^T \mathbf{X}. \quad (6.10)$$

6.2 Measures for Numerical Data

The choice of distances is important for applications, and the best choice is often achieved via a combination of experience, skill, knowledge, and luck. Here we list some commonly used distances.

6.2.1 Euclidean Distance

Euclidean distance is probably the most common distance we have ever used for numerical data. For two data points \mathbf{x} and \mathbf{y} in d -dimensional space, the Euclidean distance between them is defined to be

$$d_{euc}(\mathbf{x}, \mathbf{y}) = \left[\sum_{j=1}^d (x_j - y_j)^2 \right]^{\frac{1}{2}} = [(\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T]^{\frac{1}{2}}, \quad (6.11)$$

where x_j and y_j are the values of the j th attribute of \mathbf{x} and \mathbf{y} , respectively.

The squared Euclidean distance is defined to be

$$d_{seuc}(\mathbf{x}, \mathbf{y}) = d_{euc}(\mathbf{x}, \mathbf{y})^2 = \sum_{j=1}^d (x_j - y_j)^2 = (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T. \quad (6.12)$$

Note that the squared distance is in fact not a distance.

6.2.2 Manhattan Distance

Manhattan distance is also called “city block distance” and is defined to be the sum of the distances of all attributes. That is, for two data points \mathbf{x} and \mathbf{y} in d -dimensional space, the Manhattan distance between them is

$$d_{man}(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^d |x_k - y_k|. \quad (6.13)$$

If the data point \mathbf{x} or \mathbf{y} has missing values at some attributes, then the Manhattan distance can be defined as (Wishart, 2002)

$$d_{manw}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d \frac{w_j |x_j - y_j|}{\sum_{k=1}^d w_k}, \quad (6.14)$$

where $w_j = 1$ if both \mathbf{x} and \mathbf{y} have observations of the j th attribute and $w_j = 0$ if otherwise.

The Manhattan segmental distance is a variant of the Manhattan distance. In the Manhattan segmental distance, only a part of the whole dimension is used to calculate the distance. It is defined as (Aggarwal et al., 1999)

$$d_P(\mathbf{x}, \mathbf{y}) = \sum_{j \in P} \frac{|x_j - y_j|}{|P|}, \quad (6.15)$$

where P is a nonempty subset of $\{1, 2, \dots, d\}$.

6.2.3 Maximum Distance

The maximum distance is also called the “sup” distance. It is defined to be the maximum value of the distances of the attributes; that is, for two data points \mathbf{x} and \mathbf{y} in d -dimensional space, the maximum distance between them is

$$d_{max}(\mathbf{x}, \mathbf{y}) = \max_{1 \leq j \leq d} |x_j - y_j|. \quad (6.16)$$

6.2.4 Minkowski Distance

The Euclidean distance, Manhattan distance, and maximum distance are three particular cases of the Minkowski distance defined by

$$d_{min}(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^d |x_j - y_j|^r \right)^{\frac{1}{r}}, \quad r \geq 1. \quad (6.17)$$

r is called the order of the above Minkowski distance. Note that if we take $r = 2, 1$, and ∞ , we get the Euclidean distance, Manhattan distance, and maximum distance, respectively.

If the data set has compact or isolated clusters, the Minkowski distance works well (Mao and Jain, 1996); otherwise the largest-scale attribute tends to dominate the others. To avoid this, we should normalize the attributes or use weighting schemes (Jain et al., 1999).

6.2.5 Mahalanobis Distance

Mahalanobis distance (Jain and Dubes, 1988; Mao and Jain, 1996) can alleviate the distance distortion caused by linear combinations of attributes. It is defined by

$$d_{mah}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y}) \Sigma^{-1} (\mathbf{x} - \mathbf{y})^T}, \quad (6.18)$$

where Σ is the covariance matrix of the data set defined in equation (6.8). Therefore, this distance applies a weight scheme to the data.

Another important property of the Mahalanobis distance is that it is invariant under all nonsingular transformations. For example, let C be any nonsingular $d \times d$ matrix applied to the original data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ by

$$\mathbf{y}_i = C\mathbf{x}_i, \quad i = 1, 2, \dots, n.$$

Then the new covariance matrix becomes

$$\frac{1}{n}\mathbf{Y}^T\mathbf{Y} = \frac{1}{n}(\mathbf{X}C^T)^T(\mathbf{X}C^T),$$

where \mathbf{X} is defined in equation (6.9) and \mathbf{Y} is defined similarly for the transformed data set.

Then the Mahalanobis distance between \mathbf{y}_i and \mathbf{y}_j is

$$\begin{aligned} & d_{mah}(\mathbf{y}_i, \mathbf{y}_j) \\ &= \sqrt{(\mathbf{y}_i - \mathbf{y}_j) \left(\frac{1}{n}\mathbf{Y}^T\mathbf{Y} \right)^{-1} (\mathbf{y}_i - \mathbf{y}_j)^T} \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)C^T \left(\frac{1}{n}(\mathbf{X}C^T)^T(\mathbf{X}C^T) \right)^{-1} C(\mathbf{x}_i - \mathbf{x}_j)^T} \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j) \left(\frac{1}{n}\mathbf{X}^T\mathbf{X} \right)^{-1} (\mathbf{x}_i - \mathbf{x}_j)^T} \\ &= d_{mah}(\mathbf{x}_i, \mathbf{x}_j), \end{aligned}$$

which shows that the Mahalanobis distance is invariant under nonsingular transformations.

Morrison (1967) proposed a generalized Mahalanobis distance by including the weights of variables. Let λ_j ($j = 1, 2, \dots, d$) be the weight assigned to the j th variable, and let Λ be the $d \times d$ diagonal matrix containing the d weights, i.e.,

$$\Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_d \end{pmatrix}.$$

Then the generalized Mahalanobis distance is defined as (Morrison, 1967)

$$d_{gmah}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})\Lambda\Sigma^{-1}\Lambda(\mathbf{x} - \mathbf{y})^T}.$$

The Mahalanobis distance suffers from some disadvantages. For example, it involves high computation, since the covariance matrix is computed based on all data points in the data set.

6.2.6 Average Distance

As pointed out in (Legendre and Legendre, 1983), the Euclidean distance has the following drawback: two data points with no attribute values in common may have a smaller distance

than another pair of data points containing the same attribute values. In this case, the average distance was adopted (Legendre and Legendre, 1983).

The average distance is modified from the Euclidean distance. Given two data points \mathbf{x} and \mathbf{y} in d -dimensional space, the average distance is defined by

$$d_{ave}(\mathbf{x}, \mathbf{y}) = \left(\frac{1}{d} \sum_{j=1}^d (x_j - y_j)^2 \right)^{\frac{1}{2}}. \quad (6.19)$$

6.2.7 Other Distances

Chord distance (Orlóci, 1967), a modification of Euclidean distance, is defined as the length of the chord joining the two normalized points within a hypersphere of radius one. It can also be computed directly from nonnormalized data. The chord distance between two data points \mathbf{x} and \mathbf{y} is defined by

$$d_{chord}(\mathbf{x}, \mathbf{y}) = \left(2 - 2 \frac{\sum_{k=1}^d x_k y_k}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2} \right)^{\frac{1}{2}}, \quad (6.20)$$

where $\|\cdot\|_2$ is the L_2 -norm, i.e.,

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{k=1}^d x_k^2}.$$

The chord distance measure can solve the problem caused by the scale of measurements, and it is used to deal with the above-mentioned drawbacks of the Euclidean distance measure as well.

Geodesic distance (Legendre and Legendre, 1983) is a transformation of the chord distance and is defined to be the length of the shorter arc connecting the two normalized data points at the surface of the hypersphere of unit radius. It is defined as

$$d_{geo}(\mathbf{x}, \mathbf{y}) = \arccos \left(1 - \frac{d_{chord}(\mathbf{x}, \mathbf{y})}{2} \right). \quad (6.21)$$

Several other distance measures for numerical data are given in Table 6.1.

6.3 Measures for Categorical Data

Categorical data are data measured on nominal scales. Unlike numerical data, the computation of association measures between records described by nominal variables has received little attention. In this section, we review some similarity and dissimilarity measures for categorical data.

Table 6.1. Some other dissimilarity measures for numerical data.

Measure	$d(\mathbf{x}, \mathbf{y})$	Reference
Mean character difference	$\frac{1}{d} \sum_{j=1}^d x_j - y_j $	Czekanowski (1909)
Index of association	$\frac{1}{2} \sum_{j=1}^d \left \frac{x_j}{\sum_{l=1}^d x_l} - \frac{y_j}{\sum_{l=1}^d y_l} \right $	Whittaker (1952)
Canberra metric	$\sum_{j=1}^d \frac{ x_j - y_j }{(x_j + y_j)}$	Legendre and Legendre (1983)
Czekanowski coefficient	$1 - \frac{2 \sum_{j=1}^d \min\{x_j, y_j\}}{\sum_{j=1}^d (x_j + y_j)}$	Johnson and Wichern (1998)
Coefficient of divergence	$\left(\frac{1}{d} \sum_{j=1}^d \left(\frac{x_j - y_j}{x_j + y_j} \right)^2 \right)^{\frac{1}{2}}$	Legendre and Legendre (1983)

6.3.1 The Simple Matching Distance

The simple matching dissimilarity measure (Kaufman and Rousseeuw, 1990; Huang, 1997b, 1998) is a simple, well-known measure used to measure categorical data.

Let x and y be two categorical values. Then the simple matching distance (Kaufman and Rousseeuw, 1990) between x and y is given by

$$\delta(x, y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{if } x \neq y. \end{cases} \quad (6.22)$$

Let \mathbf{x} and \mathbf{y} be two categorical objects described by d categorical attributes. Then the dissimilarity between \mathbf{x} and \mathbf{y} measured by the simple matching distance is defined by

$$d_{sim}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d \delta(x_j, y_j). \quad (6.23)$$

Taking into account the frequencies of categories in the data set, we can define the dissimilarity measure as

$$d_{simf}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d \frac{(n_{x_j} + n_{y_j})}{n_{x_j} n_{y_j}} \delta(x_j, y_j), \quad (6.24)$$

where n_{x_j} and n_{y_j} are the numbers of objects in the data set that have categories x_j and y_j for attribute j , respectively.

6.3.2 Other Matching Coefficients

Besides the simple matching coefficient, other matching coefficients for nominal data are possible. Some matching coefficients extended from binary measures to nominal data have been presented in (Anderberg, 1973). These matching coefficients are also presented here.

Let \mathbf{x} and \mathbf{y} be two records, each of which is described by d nominal attributes, and let N_{a+d} be the number of attributes on which the two records match, N_d be the number of attributes on which the two records match in a “not applicable” category, and N_{b+c} be the number of attributes on which the two records do not match. Then

$$N_{a+d} = \sum_{j=1}^d [1 - \delta(x_j, y_j)],$$

where $\delta(\cdot, \cdot)$ is defined in (6.22). Similarly, we have

$$N_d = \sum_{j=1}^d [\delta(x_j, ?) + \delta(?, y_j) - \delta(x_j, ?)\delta(?, y_j)],$$

$$N_{b+c} = \sum_{j=1}^d \delta(x_j, y_j),$$

where “?” is a symbol for missing values, i.e., if $x_j = ?$, then \mathbf{x} has a missing value in the j th attribute.

The matching coefficients given in Table 6.2 are extended from binary measures.

Table 6.2. Some matching coefficients for nominal data.

Measure	$s(\mathbf{x}, \mathbf{y})$	Weighting of matches, mismatches
Russell and Rao	$\frac{N_{a+d} - N_d}{N_{a+d} + N_{b+c}}$	Equal weights
Simple matching	$\frac{N_{a+d}}{N_{a+d} + N_{b+c}}$	Equal weights
Jaccard	$\frac{N_{a+d} - N_d}{N_{a+d} - N_d + N_{b+c}}$	Equal weights
Unnamed	$\frac{2N_{a+d}}{2N_{a+d} + N_{b+c}}$	Double weight for matched pairs
Dice	$\frac{2N_{a+d} - 2N_d}{2N_{a+d} - 2N_d + N_{b+c}}$	Double weight for matched pairs
Rogers-Tanimoto	$\frac{N_{a+d}}{N_{a+d} + 2N_{b+c}}$	Double weight for unmatched pairs
Unnamed	$\frac{N_{a+d} - N_d}{N_{a+d} - N_d + 2N_{b+c}}$	Double weight for unmatched pairs
Kulczynski	$\frac{N_{a+d} - N_d}{N_{b+c}}$	Matched pairs excluded from denominator
Unnamed	$\frac{N_{a+d}}{N_{b+c}}$	Matched pairs excluded from denominator

6.4 Measures for Binary Data

In this section, we shall provide a short survey of various similarity measures for binary feature vectors. Some similarity and dissimilarity measures for binary data have been discussed in (Hubálek, 1982; Rogers and Tanimoto, 1960; Baulieu, 1989, 1997), and (Gower and Legendre, 1986).

Let \mathbf{x} and \mathbf{y} be two binary vectors in a d -dimensional space, and let A , B , C , D , and σ be defined as follows:

$$A = S_{11}(\mathbf{x}, \mathbf{y}), \quad (6.25a)$$

$$B = S_{01}(\mathbf{x}, \mathbf{y}), \quad (6.25b)$$

$$C = S_{10}(\mathbf{x}, \mathbf{y}), \quad (6.25c)$$

$$D = S_{00}(\mathbf{x}, \mathbf{y}), \quad (6.25d)$$

$$\sigma = \sqrt{(A + B)(A + C)(B + D)(C + D)}, \quad (6.25e)$$

where $S_{ij}(\mathbf{x}, \mathbf{y})$ ($i, j \in \{0, 1\}$) are defined in equations (2.7a)–(2.7d).

Let $s(\mathbf{x}, \mathbf{y})$ and $d(\mathbf{x}, \mathbf{y})$ be the similarity measure and dissimilarity measure between \mathbf{x} and \mathbf{y} , respectively. Table 6.3 gives eight similarity measures for binary feature vectors summarized by Tubbs (1989).

The ranges of various similarity measures are described in the third column of Table 6.3; however, there are some assumptions underlying these ranges. For example, $s(\bar{\mathbf{I}}, \bar{\mathbf{I}}) = 1$ in the Jaccard-Needham similarity measure, $s(\bar{\mathbf{I}}, \bar{\mathbf{I}}) = \frac{1}{2}$ in the Dice similarity measure, $s(\mathbf{I}, \mathbf{I}) = s(\bar{\mathbf{I}}, \bar{\mathbf{I}}) = 1$ and $s(\bar{\mathbf{I}}, \mathbf{I}) = s(\mathbf{I}, \bar{\mathbf{I}}) = -1$ in the correlation and Yule similarity measures, and $s(\bar{\mathbf{I}}, \bar{\mathbf{I}}) = \infty$ in the Kulzinsky similarity measure (Zhang and Srihari, 2003). The dissimilarity measures given in Table 6.3 have been normalized to $[0, 1]$.

Table 6.3. Similarity measures for binary vectors. $d(\mathbf{x}, \mathbf{y})$ is the corresponding dissimilarity measure.

Measure	$s(\mathbf{x}, \mathbf{y})$	Range of $s(\mathbf{x}, \mathbf{y})$	$d(\mathbf{x}, \mathbf{y})$
Jaccard	$\frac{A}{A+B+C}$	$[0, 1]$	$\frac{B+C}{A+B+C}$
Dice	$\frac{A}{2A+B+C}$	$[0, \frac{1}{2}]$	$\frac{B+C}{2A+B+C}$
Pearson	$\frac{AD-BC}{\sigma}$	$[-1, 1]$	$\frac{1}{2} - \frac{AD-BC}{2\sigma}$
Yule	$\frac{AD-BC}{AD+BC}$	$[-1, 1]$	$\frac{BC}{AD+BC}$
Russell-Rao	$\frac{A}{d}$	$[0, 1]$	$1 - \frac{A}{d}$
Sokal-Michener	$\frac{A+D}{d}$	$[0, 1]$	$\frac{2(B+C)}{A+2(B+C)+D}$
Rogers-Tanimoto	$\frac{A+D}{A+2(B+C)+D}$	$[0, 1]$	$\frac{2(B+C)}{A+2(B+C)+D}$
Rogers-Tanimoto-a	$\frac{A+D}{A+2(B+C)+D}$	$[0, 1]$	$\frac{2(d-A-D)}{2d-A-D}$
Kulzinsky	$\frac{A}{B+C}$	$[0, \infty]$	$\frac{B+C-A+d}{B+C+d}$

There are two types of binary similarity coefficients: symmetrical coefficients and asymmetrical coefficients. The difference between the two types of coefficients is that symmetrical coefficients take double zeros into account while asymmetrical coefficients exclude double zeros (Legendre and Legendre, 1983). Table 6.4 gives four binary similarity measures proposed by Sokal and Sneath (1963) and other binary similarity measures that take double zeros into account.

For some cases, a comparison of two binary feature vectors must exclude double zeros. For example, given two asymmetric binary feature vectors in which zero means lack of information, a similarity measure should exclude double zeros. These similarity mea-

Table 6.4. Some symmetrical coefficients for binary feature vectors.

Measure	$s(\mathbf{x}, \mathbf{y})$	Range of $s(\mathbf{x}, \mathbf{y})$
Simple matching	$\frac{A+D}{d}$	$[0, 1]$
Rogers-Tanimoto	$\frac{A+D}{A+2(B+C)+D}$	$[0, 1]$
Sokal-Sneath-a	$\frac{2(A+D)}{2A+B+C+2D}$	$[0, 1]$
Sokal-Sneath-b	$\frac{A+D}{B+C}$	$[0, \infty]$
Sokal-Sneath-c	$\frac{1}{4} \left(\frac{A}{A+B} + \frac{A}{A+C} + \frac{D}{B+D} + \frac{D}{C+D} \right)$	$[0, 1]$
Sokal-Sneath-d	$\frac{A}{\sqrt{(A+B)(A+C)}} \frac{D}{\sqrt{(B+D)(C+D)}}$	$[0, 1]$
Hamann	$\frac{A+D-B-C}{d}$	$[-1, 1]$
Yule	$\frac{AD-BC}{AD+BC}$	$[-1, 1]$
Pearson	$\frac{AD-BC}{\sigma}$	$[-1, 1]$

Table 6.5. Some asymmetrical coefficients for binary feature vectors.

Measure	$s(\mathbf{x}, \mathbf{y})$	Range of $s(\mathbf{x}, \mathbf{y})$
Jaccard	$\frac{A}{A+B+C}$	$[0, 1]$
Sørensen	$\frac{2A}{2A+B+C}$	$[0, 1]$
Russell-Rao	$\frac{A}{d}$	$[0, 1]$
Kulzinsky	$\frac{A}{B+C}$	$[0, \infty]$
Sokal-Sneath-e	$\frac{1}{2} \left(\frac{A}{A+B} + \frac{A}{A+C} \right)$	$[0, 1]$
Ochiai	$\frac{A}{\sqrt{(A+B)(A+C)}}$	$[0, 1]$

sures are asymmetrical coefficients. The best-known asymmetrical coefficient is Jaccard's coefficient. Table 6.5 gives some asymmetrical coefficients for binary feature vectors.

6.5 Measures for Mixed-type Data

In many applications, each instance in a data set is described by more than one type of attribute. In this case, the similarity and dissimilarity measures discussed before cannot be applied to this kind of data directly. Gower (1971) and Estabrook and Rogers (1966) propose some general measures, which will be discussed in this section.

6.5.1 A General Similarity Coefficient

The general similarity coefficient (Gower, 1971; Wishart, 2002), proposed by Gower (1971), has been widely implemented and used to measure the similarity for two mixed-type data points. This general similarity coefficient can also be applied to data points with missing values.

Let \mathbf{x} and \mathbf{y} denote two d -dimensional data points. Then the general similarity coefficient $s_{gower}(\mathbf{x}, \mathbf{y})$ is defined as

$$s_{gower}(\mathbf{x}, \mathbf{y}) = \frac{1}{\sum_{k=1}^d w(x_k, y_k)} \sum_{k=1}^d w(x_k, y_k) s(x_k, y_k), \quad (6.26)$$

where $s(x_k, y_k)$ is a similarity component for the k th attribute and $w(x_k, y_k)$ is either one or zero depending on whether or not a comparison is valid for the k th attribute of the two data points. They are defined respectively for different attribute types. Let x_k and y_k denote the k th attributes of \mathbf{x} and \mathbf{y} , respectively. Then $s(x_k, y_k)$ and $w(x_k, y_k)$ are defined as follows.

- For *quantitative* attributes x_k and y_k , $s(x_k, y_k)$ is defined as

$$s(x_k, y_k) = 1 - \frac{|x_k - y_k|}{R_k},$$

where R_k is the range of the k th attribute; $w(x_k, y_k) = 0$ if data point \mathbf{x} or \mathbf{y} has missing value at the k th attribute; otherwise $w(x_k, y_k) = 1$.

- For *binary* attributes x_k and y_k , $s(x_k, y_k) = 1$ if both data points \mathbf{x} and \mathbf{y} have the k th attribute “present”; otherwise $s(x_k, y_k) = 0$; $w(x_k, y_k) = 0$ if both data points \mathbf{x} and \mathbf{y} have the k th attribute “absent”; otherwise $w(x_k, y_k) = 1$.
- For *nominal* or *categorical* attributes x_k and y_k , $s(x_k, y_k) = 1$ if $x_k = y_k$; otherwise $s(x_k, y_k) = 0$; $w(x_k, y_k) = 0$ if data point \mathbf{x} or \mathbf{y} has missing value at the k th attribute; otherwise $w(x_k, y_k) = 1$.

From the definition of the general similarity coefficient, we see that $s_{gower}(\mathbf{x}, \mathbf{y})$ achieves minimum value zero if the two data points are identical and has maximum value one if the two data points are extremely different. Other general coefficients of similarity have been presented in (Estabrook and Rogers, 1966).

6.5.2 A General Distance Coefficient

To measure the distance of two data points or the means of two clusters, the general distance coefficient (Gower, 1971) is introduced. As in equation (6.26), the general distance coefficient between two data points \mathbf{x} and \mathbf{y} is defined as

$$d_{gower}(\mathbf{x}, \mathbf{y}) = \left(\frac{1}{\sum_{k=1}^d w(x_k, y_k)} \sum_{k=1}^d w(x_k, y_k) d^2(x_k, y_k) \right)^{\frac{1}{2}}, \quad (6.27)$$

where $d^2(x_k, y_k)$ is a squared distance component for the k th attribute and $w(x_k, y_k)$ is the same as in the general similarity coefficient, i.e., depending on whether or not a comparison is valid for the k th attribute, if both data points \mathbf{x} and \mathbf{y} have observations at the k th attribute, then $w(x_k, y_k) = 1$; otherwise $w(x_k, y_k) = 0$. For different types of attributes, $d^2(x_k, y_k)$ is defined differently, as described below.

- For *ordinal* and *continuous* attributes, $d(x_k, y_k)$ is defined as

$$d(x_k, y_k) = \frac{|x_k - y_k|}{R_k},$$

where R_k is the range of the k th attribute.

- For *quantitative* attributes, $d(x_k, y_k)$ can be defined simply as

$$d(x_k, y_k) = |x_k - y_k|.$$

It can also be normalized (Wishart, 2002) to be

$$d(x_k, y_k) = \frac{|x_k^2 - y_k^2|}{\sigma_k}$$

if we standardize x_k as

$$x_k^* = \frac{x_k - \mu_k}{\sigma_k},$$

where μ_k and σ_k are the mean and standard variation of the k th attribute, respectively.

- For *binary* attributes, $d(x_k, y_k) = 0$ if both i and j have the k th attributes “present” or “absent”; otherwise $d(x_k, y_k) = 1$.
- For *nominal* or *categorical* attributes, $d(x_k, y_k) = 0$ if $x_k = y_k$; otherwise $d(x_k, y_k) = 1$.

6.5.3 A Generalized Minkowski Distance

The generalized Minkowski distance (Ichino and Yaguchi, 1994; Ichino, 1988) is a distance measure designed for mixed-type data based on the Minkowski distance discussed in previous sections. This distance can deal with several feature types, including quantitative features (continuous and discrete), qualitative features (ordinal and nominal), and tree-structured features.

Let a record described by d features X_j ($j = 1, 2, \dots, d$) be represented by a Cartesian product set

$$\mathbf{E} = E_1 \times E_2 \times \cdots \times E_d,$$

where E_j is a feature value taken by a feature X_j , which can be an interval or a finite set depending on the feature type of X_j .

Let the feature space be denoted by

$$U^{(d)} = U_1 \times U_2 \times \cdots \times U_d,$$

where U_j is the domain of feature X_j . For different types of features, the domain U_j is specified as follows.

Quantitative feature. The height and blood pressure of a person and number of cities in a state are examples of this feature. E_j can be a single numerical value or an interval of values. Let the domain U_j be a finite interval $U_j = [a_j, b_j]$, where a_j and b_j are the minimum and the maximum possible values for the feature X_j .

Ordinal qualitative feature. Military rank is an example of this feature. Assume that the possible feature values are coded numerically. For example, the lowest rank value is coded by 1, the second lowest rank value is coded by 2, and so on. E_j can be a single value or an interval value, such as $E_j = 1 \in U_j$ or $E_j = [2, 4] \in U_j$. In this case, the domain U_j is a finite interval of the form $[a_j, b_j]$.

Nominal qualitative feature. Values taken by a nominal qualitative feature have no orders. For example, sex “male, female” and blood type “A, B, AB, O” are nominal values. E_j can be a finite set, such as $E_j = A, B$. The domain U_j is a finite set of possible values $\{a_{j1}, a_{j2}, \dots, a_{jm}\}$.

Tree-structured feature. This feature type is a special case of the nominal qualitative feature, when all terminal values are nominal values. E_j is a set of terminal values of the given tree. The domain U_j is a finite set of possible terminal values $\{a_{j1}, a_{j2}, \dots, a_{jm}\}$.

Let $\mathbf{A} = A_1 \times A_2 \times \dots \times A_d$ and $\mathbf{B} = B_1 \times B_2 \times \dots \times B_d$ be two records in $U^{(d)}$, and let the Cartesian join of \mathbf{A} and \mathbf{B} be defined as

$$\mathbf{A} \boxplus \mathbf{B} = (A_1 \boxplus B_1) \times (A_2 \boxplus B_2) \times \dots \times (A_d \boxplus B_d), \quad (6.28)$$

where $A_j \boxplus B_j$ is the Cartesian join of the j th feature values A_j and B_j , as specified below.

- If X_j is a quantitative or ordinal qualitative feature, then $A_j \boxplus B_j$ is a closed interval:

$$A_j \boxplus B_j = [\min\{A_{jL}, B_{jL}\}, \max\{A_{jU}, B_{jU}\}],$$

where A_{jL} and A_{jU} are the lower bound and the upper bound of the interval A_j , respectively.

- If X_j is a nominal qualitative feature, then $A_j \boxplus B_j$ is the union of A_j and B_j , i.e.,

$$A_j \boxplus B_j = A_j \cup B_j.$$

- If X_j is a tree-structured feature, let $N(A_j)$ denote the nearest parent node common to all terminal values in A_j . If $N(A_j) = N(B_j)$, $A_j \boxplus B_j = A_j \cup B_j$; if $N(A_j) \neq N(B_j)$, $A_j \boxplus B_j$ is the set of all terminal values branched from the node $N(A_j \cup B_j)$. $A_j \boxplus A_j$ is assumed to be A_j .

The Cartesian meet between \mathbf{A} and \mathbf{B} is defined as

$$\mathbf{A} \boxtimes \mathbf{B} = (A_1 \boxtimes B_1) \times (A_2 \boxtimes B_2) \times \dots \times (A_d \boxtimes B_d), \quad (6.29)$$

where $A_j \boxtimes B_j$ is the Cartesian meet of the j th feature values A_j and B_j , which is defined as the intersection of A_j and B_j , i.e.,

$$A_j \boxtimes B_j = A_j \cap B_j.$$

If at least one of $A_j \boxtimes B_j$ is empty, then $\mathbf{A} \boxtimes \mathbf{B}$ is an empty set. The mathematical model $(U^{(d)}, \boxplus, \boxtimes)$ is called the Cartesian space model.

Let $\phi(A_j, B_j)$ ($j = 1, 2, \dots, d$) be defined as

$$\phi(A_j, B_j) = |A_j \boxplus B_j| - |A_j \boxtimes B_j| + \gamma(2|A_j \boxtimes B_j| - |A_j| - |B_j|), \quad (6.30)$$

where $0 \leq \gamma \leq \frac{1}{2}$, and A_j denotes the length of the interval A_j if X_j is a continuous quantitative feature and the number of possible values in A_j if X_j is a discrete quantitative, qualitative, or tree-structured feature.

It can be shown that $\phi(A_j, B_j)$ defined in equation (6.30) is a metric distance (Ichino and Yaguchi, 1994). The generalized Minkowski distance of order p (≥ 1) between \mathbf{A} and \mathbf{B} is defined as

$$d_p(\mathbf{A}, \mathbf{B}) = \left(\sum_{j=1}^d \phi(A_j, B_j)^p \right)^{\frac{1}{p}}. \quad (6.31)$$

To remove the artifact of the measurement unit, $\phi(A_j, B_j)$ can be normalized to

$$\psi(A_j, B_j) = \frac{\phi(A_j, B_j)}{|U_j|}, \quad j = 1, 2, \dots, d,$$

where $|U_j|$ is the length of the domain U_j . Then $\psi(A_j, B_j) \in [0, 1]$ for all $j = 1, 2, \dots, d$. If $\psi(A_j, B_j)$ instead of $\phi(A_j, B_j)$ is used in equation (6.31), then the generalized Minkowski distance becomes

$$d_p(\mathbf{A}, \mathbf{B}) = \left(\sum_{j=1}^d \psi(A_j, B_j)^p \right)^{\frac{1}{p}}. \quad (6.32)$$

Weighting the variables further modifies the generalized Minkowski distance. Let $c_j (> 0)$ be the weight the j th feature such that $c_1 + c_2 + \dots + c_d = 1$. Then the generalized Minkowski distance becomes

$$d_p(\mathbf{A}, \mathbf{B}) = \left(\sum_{j=1}^d (c_j \psi(A_j, B_j))^p \right)^{\frac{1}{p}}. \quad (6.33)$$

The distance $d_p(\mathbf{A}, \mathbf{B})$ defined in equation (6.33) lies in $[0, 1]$. Examples of applications of the generalized Minkowski distance have been presented in (Ichino and Yaguchi, 1994) and (Ichino, 1988).

6.6 Measures for Time Series Data

A time series is a sequence of real numbers that represent the measurements of a real variable at equal time intervals (Gunopulos and Das, 2000). Classical time series analysis includes identifying patterns (e.g., trend analysis, seasonality analysis, autocorrelation, and autoregressive models (StatSoft, Inc., 2005)) and forecasting. From a databases perspective, Gunopulos and Das (2000) list a few important problems related to time series.

Similarity problem. This problem with time series data requires determining whether different time series have similar behaviors. Precisely, given two time series $\mathbf{x} = x_1, x_2, \dots, x_m$ and $\mathbf{y} = y_1, y_2, \dots, y_m$, how can we define and compute the distance $d(\mathbf{x}, \mathbf{y})$ or the similarity $s(\mathbf{x}, \mathbf{y})$? Readers are referred to (Yang and Shahabi, 2004), (Gunopulos and Das, 2000), (Gunopulos and Das, 2001), (Bollobás et al., 1997), (Keogh, 2001), and (Das et al., 1997) for more detailed discussions of this topic.

Indexing problem. This problem requires finding the best match to a query in a large database. An obvious solution is to retrieve and examine every sequence in the database. However, this method does not scale to large databases. This leads to the problem of indexing time series. For the indexing problem, interested readers are referred to (Hetland, 2004), (Chakrabarti et al., 2002), (Keogh et al., 2001), and (Rafiei and Mendelzon, 1997).

Subsequence similarity problem. This problem can be described as follows: given a template or a query Q , a reference database C , and a distance measure, find the location that best matches Q . For example, find out other days when a stock had similar movements as today. See (Keogh, 2001) for some discussions.

Clustering problem. Clustering time series is finding natural groupings of time series in a database under some similarity or dissimilarity measure. Due to the unique structure of time series, most classic clustering algorithms do not work well for time series data. Ramoni et al. (2002) introduce a Bayesian method for grouping time series into clusters so that the elements of each cluster have similar dynamics. Keogh and Kasetty (2003) present a survey for current research in this field. Readers are referred to the survey and references therein.

Rule discovery problem. The rule discovery problem is the problem of finding rules relating patterns in a time series to other patterns in the time series or patterns in one time series to patterns in another time series. For example, find rules such as “if stock X goes up and Y falls, then Z will go up next day.” Das et al. (1998), Tsumoto (1999), Caraça-Valente and López-Chavarriás (2000), and Chiu et al. (2003) address this problem.

Let \mathbf{x} and \mathbf{y} be two time series, and denote by $d(\mathbf{x}, \mathbf{y})$ a distance between \mathbf{x} and \mathbf{y} that will be defined. Like distance measures for other data types, the distance measure for time series should have the following properties (Keogh, 2001):

- $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ (symmetry),
- $d(\mathbf{x}, \mathbf{x}) = 0$ (constancy or self-similarity),
- $d(\mathbf{x}, \mathbf{y}) \geq 0$ and $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$ (positivity),
- $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$ (triangle inequality).

Most of the distances presented in this chapter so far can be applied to time series. In this section, we present some distance measures and briefly discuss their advantages and disadvantages.

6.6.1 The Minkowski Distance

Let \mathbf{x} and \mathbf{y} be two time series of length d , and let x_j and y_j be the values of \mathbf{x} and \mathbf{y} at time j ($1 \leq j \leq d$), respectively. Then the Minkowski distance between \mathbf{x} and \mathbf{y} is given by

$$d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^d (x_j - y_j)^p \right)^{\frac{1}{p}}, \quad (6.34)$$

where p is a positive real number. When $p = 1, 2$, and ∞ , we get the Manhattan, Euclidean, and maximum distances, respectively. The Minkowski distance is also referred to as the L_p norm.

Instead of using the Euclidean distance, one can use the squared Euclidean distance to optimize the distance calculations. The advantages of the Minkowski distance are that it is easy to compute and allows scalable solutions of other problems such as indexing and clustering (Gunopulos and Das, 2000). The disadvantage is that it cannot be applied to raw time series, since it does not allow for different baselines (e.g., a stock fluctuates around \$100 and another stock fluctuates around \$10), different scales, noise or short-term fluctuations, phase shifts in time, acceleration-deceleration along the time dimension, etc. (Gunopulos and Das, 2000). Applications of the Minkowski distance can be found in (Agrawal et al., 1993), (Yi and Faloutsos, 2000), and (Lee et al., 2000).

For some queries, different parts of the time series are more important. This leads to weighted distance measures. The weighted Euclidean distance, for example, is defined as

$$d_2(\mathbf{x}, \mathbf{y}, W) = \sqrt{\sum_{j=1}^d w_j (x_j - y_j)^2},$$

where w_j is the weight given to the j th component. The weights can be set by relevance feedback, which is defined as the reformulation of a search query in response to feedbacks provided by the user for the results of previous versions of the query (Wu et al., 2000a; Keogh, 2001). The basic idea of relevance feedback is that the weights and the shape of the query are updated by the user through ranking the displayed search results, and then the new query is executed. After several executions, the optimal query may be found.

6.6.2 Time Series Preprocessing

For most applications, the distortions of data should be removed. In this subsection, we present several commonly used transformations to remove distortions in time series data. In what follows, $\mathbf{x} = (x_1, x_2, \dots, x_d)$ and $\mathbf{y} = (y_1, y_2, \dots, y_d)$ shall denote two raw time series and \mathbf{x}^* and \mathbf{y}^* the transformed time series.

The Offset Transformation

The offset transformation removes the baseline of a time series. Precisely, the transformed time series $\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_d^*)$ is given by

$$x_j^* = x_j - \mu_{\mathbf{x}}, \quad j = 1, 2, \dots, d,$$

or

$$\mathbf{x}^* = \mathbf{x} - \mu_{\mathbf{x}},$$

where $\mu_{\mathbf{x}} = \frac{1}{d} \sum_{j=1}^d x_j$.

The Amplitude Transformation

The offset transformation can remove the baseline, but it cannot remove the amplitude of the time series. To remove amplitude, we can use the amplitude transformation, which is given by

$$x_j^* = \frac{x_j - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}, \quad j = 1, 2, \dots, d,$$

or

$$\mathbf{x}^* = \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}},$$

where $\mu_{\mathbf{x}} = \frac{1}{d} \sum_{j=1}^d x_j$ and $\sigma_{\mathbf{x}} = (\frac{1}{d} \sum_{j=1}^d (x_j - \mu_{\mathbf{x}})^2)^{\frac{1}{2}}$.

Examples of the application of this transformation to time series can be found in (Goldin and Kanellakis, 1995).

Remove Linear Trend

Intuitively, to remove the linear trend, one can find the best-fitting straight line to the time series and then subtract that line from the time series. For example, one can use the least square method to fit the time series. Let $x = kt + b$ be the best-fitting straight line of a time series $\mathbf{x} = (x_1, x_2, \dots, x_d)$ with coefficients k and b to be determined. Suppose x_j is observed at time t_j . Then the best-fitting straight line has the least square error, i.e.,

$$E = \sum_{j=1}^d (x_j - kt_j - b)^2.$$

To obtain the least square error, the unknown coefficients k and b must yield zero first derivatives, i.e.,

$$\begin{aligned}\frac{\partial E}{\partial k} &= 2 \sum_{j=1}^d (x_j - kt_j - b)(-t_j) = 0, \\ \frac{\partial E}{\partial b} &= 2 \sum_{j=1}^d (x_j - kt_j - b)(-1) = 0,\end{aligned}$$

which gives

$$\begin{aligned}k &= \frac{\sum_{j=1}^d x_j \sum_{j=1}^d t_j - d \sum_{j=1}^d x_j t_j}{\left(\sum_{j=1}^d t_j \right)^2 - d \sum_{j=1}^d t_j^2}, \\ b &= \frac{\sum_{j=1}^d x_j t_j \sum_{j=1}^d t_j - \sum_{j=1}^d x_j \sum_{j=1}^d t_j^2}{\left(\sum_{j=1}^d t_j \right)^2 - d \sum_{j=1}^d t_j^2}.\end{aligned}$$

Remove Noise

The noise in a time series can be removed by smoothing the time series. These methods include the moving average (Rafiei and Mendelzon, 1997) and collapsing the adjacent

segments into one segment (Gunopoulos and Das, 2000). The moving average is a well-known technique for smoothing time series. For example, a 3-day moving average is given by

$$x_j^* = \frac{x_{j-1} + x_j + x_{j+1}}{3}, \quad j = 2, \dots, d - 1.$$

Several transformations of time series are presented above. Although these transformations reduce the error rate in distance calculations, they have other drawbacks. For instance, subsequent computations become more complicated. In particular, in the indexing problem, feature extraction becomes more difficult, especially if the transformations depend on the particular time series in question (Gunopoulos and Das, 2001).

6.6.3 Dynamic Time Warping

Dynamic time warping is extensively used in speech recognition and allows acceleration-deceleration of signals along the time dimension (Berndt and Clifford, 1994). The basic idea behind dynamic time warping is that the sequences are extended by repeating elements and the distance is calculated between the extended sequences. Therefore, dynamic time warping can handle input sequences with different lengths. Dynamic time warping can reduce the error rate by an order of magnitude in the distance calculation, but it is slow (Keogh, 2001).

Mathematically, suppose we have two time series, $\mathbf{x} = (x_1, x_2, \dots, x_r)$ and $\mathbf{y} = (y_1, y_2, \dots, y_s)$, where the lengths r and s are not necessarily equal. Dynamic time warping is described as follows (Keogh and Pazzani, 2000). Let M be an $r \times s$ matrix with the (i, j) th element containing the squared Euclidean distance $d(x_i, y_j)$ (i.e., $d(x_i, y_j) = (x_i - y_j)^2$) between two points x_i and y_j . Each element (i, j) in M corresponds to the alignment between points x_i and y_j . Then each possible mapping from \mathbf{x} to \mathbf{y} can be represented as a warping path in the matrix M , where a warping path is a contiguous set of matrix elements.

Let $W = w_1, w_2, \dots, w_K$ be a warping path, where the k th element $w_k = (i_k, j_k)$. Then $\max\{r, s\} \leq K < r + s - 1$. Warping paths have some restrictions: monotonicity, continuity, and boundary conditions.

Monotonicity. Given $w_k = (i, j)$, then $w_{k-1} = (i', j')$, where $i \geq i'$ and $j \geq j'$. This ensures that the warping path W does not go down or to the left.

Continuity. Given $w_k = (i, j)$, then $w_{k-1} = (i', j')$, where $i \leq i' + 1$ and $j \leq j' + 1$. This restricts the warping path to adjacent cells and ensures that no elements may be skipped in a sequence.

Boundary conditions. The first and the last elements in W are fixed, i.e., $w_1 = (1, 1)$ and $w_K = (r, s)$. If a warping window is specified for warping paths, then only matrix elements (i, j) with $|i - j| \leq w$ are considered, where w is the size of the warping window.

There are exponentially many warping paths that satisfy the above conditions. An optimal warping path among them is the one that minimizes the warping cost, which is

defined as (Keogh and Pazzani, 2000)

$$DTW(\mathbf{x}, \mathbf{y}) = \frac{\sqrt{\sum_{l=1}^K w_l}}{K} = \frac{\sqrt{\sum_{l=1}^K d(x_{i_l}, y_{j_l})}}{K},$$

where $(i_l, j_l) = w_l$ for $l = 1, 2, \dots, K$.

The optimal path can be found by efficient dynamic programming. Let $\gamma(i, j)$ refer to the dynamic time warping distance between the subsequences x_1, x_2, \dots, x_i and y_1, y_2, \dots, y_j . Then the optimal path can be found using dynamic programming to evaluate the recurrence

$$\gamma(i, j) = d(x_i, y_j) + \min\{\gamma(i - 1, j), \gamma(i - 1, j - 1), \gamma(i, j - 1)\}.$$

The time complexity for the basic implementation is $O(rs)$. The dynamic time warping distance can be approximated by calculating the distance between the compressed or down-sampled representation of the time series. Keogh and Pazzani (2000) proposed piecewise dynamic time warping (PDTW), which reduces the time complexity to $O(c^2)$, where c is the compression rate. Further examples of applying dynamic time warping to time series can be found in (Yi et al., 1998) and (Keogh and Ratanamahatana, 2005).

6.6.4 Measures Based on Longest Common Subsequences

Longest common subsequence (LCS) measures allow gaps in sequences. These measures are often used in speech recognition and text pattern matching (Gunopulos and Das, 2001). Based on the types of scaling and baselines, Gunopulos and Das (2001) classify LCS-like measures into three categories: LCS without scaling (Yazdani and Ozsoyoglu, 1996), LCS with local scaling and baselines (Agrawal et al., 1995), and LCS with global scaling and baselines (Das et al., 1997; Chu and Wong, 1999). The LCS problem is also discussed in (Vlachos et al., 2003). In this subsection, we shall present some LCS-like measures.

LCS without Scaling

In order to match two image sequences, Yazdani and Ozsoyoglu (1996) proposed a modified version of the LCS method for matching the two sequences. This method does not require the sequences to be of the same length. This method is described as follows.

Let $\mathbf{x} = (x_1, x_2, \dots, x_M)$ and $\mathbf{y} = (y_1, y_2, \dots, y_N)$ be two time series of lengths M and N , respectively. An element x_i from the first sequence \mathbf{x} is said to match an element y_j from the second sequence \mathbf{y} if

$$|x_i - y_j| < \delta,$$

where δ is the matching distance, i.e., a threshold value. To determine the LCSSs of the two given sequences, a correspondence function that maps matching elements of the two given sequences should be found. This can be done by a dynamic programming algorithm with time complexity $O(MN)$ (Cormen et al., 2001).

Let $C(i, j)$ be the length of the LCS of \mathbf{x}_i and \mathbf{y}_j , where $\mathbf{x}_i = (x_1, x_2, \dots, x_i)$ and $\mathbf{y}_j = (y_1, y_2, \dots, y_j)$. Then $C(i, j)$ can be defined recursively as (Yazdani and Ozsoyoglu, 1996)

$$C(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ C(i - 1, j - 1) + 1 & \text{if } i, j > 0 \text{ and } x_i \text{ matches } y_j, \\ \max\{C(i - 1, j), C(i, j - 1)\} & \text{otherwise.} \end{cases}$$

Based on the recursive definition, the LCS can be found by a dynamic programming algorithm with time complexity $O(M + N)$, where M and N are the lengths of the two original sequences. Bozkaya et al. (1997) proposed a modified version of the edit distance for matching and retrieving sequences of different lengths.

LCS with Local Scaling

Agrawal et al. (1995) proposed a similarity model for time series with local scaling. In this model, two time series are said to be similar if they have enough nonoverlapping and time-ordered pairs of subsequences that are similar. A pair of subsequences are considered similar if one can be scaled and translated appropriately to approximately resemble the other.

Let \mathbf{x} and \mathbf{y} be two time sequences and x_i and y_j be the i th and j th elements of \mathbf{x} and \mathbf{y} , respectively. A total order on elements of \mathbf{x} is defined by the relationship $<$ with $x_i < x_j$ if and only if $i < j$. Two subsequences S and T of \mathbf{x} overlap if and only if $\text{first}(S) \leq \text{first}(T) \leq \text{last}(S)$ or $\text{first}(T) \leq \text{first}(S) \leq \text{last}(T)$, where $\text{first}(S)$ refers to the first element of S and $\text{last}(S)$ refers to the last element of S , and $\text{first}(T)$ and $\text{last}(T)$ are defined similarly.

The algorithm for determining the LCSs with local scaling consists of the following steps (Agrawal et al., 1995):

1. Find all pairs of atomic subsequences (i.e., subsequences of a certain minimum length) in \mathbf{x} and \mathbf{y} that are similar (**atomic matching**).
2. Stitch similar windows to form pairs of large similar subsequences (**window stitching**).
3. Find a nonoverlapping ordering of subsequence matches having the longest match length (**subsequence ordering**).

The first step, finding all atomic similar subsequence pairs, can be done by a spatial self-join (such as an R-tree) over the set of all atomic windows. The second step and the third step, i.e., window stitching and subsequence ordering, can be reduced to finding the longest paths in a directed acyclic graph (Agrawal et al., 1995).

LCS with Global Scaling

Das et al. (1997) proposed a model for measuring the similarity between two time series that takes into account outliers, different scaling functions, and variable sampling rates. It can be implemented in polynomial time using methods from computational geometry.

The similarity model is described as follows. Assume a time series is a finite sequence of integers. Let \mathcal{F} be a set of transformation functions for mapping integers to integers. For example, \mathcal{F} could consist of all linear functions $x \rightarrow ax + b$, all scaling functions $x \rightarrow ax$, all quadratic functions, etc. Two time series \mathbf{x} and \mathbf{y} are said to be \mathcal{F} -similar if there is a function $f \in \mathcal{F}$ such that a long subsequence \mathbf{x}' of \mathbf{x} can be approximately mapped to a long subsequence \mathbf{y}' of \mathbf{y} using f . Note that the subsequences \mathbf{x}' and \mathbf{y}' do not consist of consecutive points of \mathbf{x} and \mathbf{y} , respectively.

The linear transformation function f is derived from the subsequences, and not from the original sequences, by a polynomial algorithm based on the use of methods from computational geometry. Precisely, given a pair of time series \mathbf{x} and \mathbf{y} , the algorithm finds the linear transformation f that maximizes the length of the LCS of $f(\mathbf{x}), \mathbf{y}$. Once the function $f \in \mathcal{F}$ is found, determining the similarity between \mathbf{x} and \mathbf{y} is easy: form the sequence $f(\mathbf{x})$ by applying f to each element of \mathbf{x} and then locate the LCS of $f(\mathbf{x})$ and \mathbf{y} .

6.6.5 Measures Based on Probabilistic Models

Keogh and Smyth (1997) proposed a probabilistic model for measuring the similarity of time series data. The model integrates local and global shape information, can handle noise and uncertainty, and incorporates prior knowledge. For example, in a probabilistic model between time series \mathbf{x} and \mathbf{y} , an ideal prototype template \mathbf{x} can be “deformed” according to a prior probability distribution to generate the observed data \mathbf{y} . The model consists of local features, which are incorporated into a global shape sequence, and the degree of deformation of the local features and the degree of the elasticity of the global shape are governed by prior probability distributions.

Mathematically, let $Q = (q_1, q_2, \dots, q_k)$ be a query sequence of k local features such as peaks and plateaus. Let $l_i = (x_i, y_i)$ ($1 \leq i \leq k - 1$) be the observed distances between the centroids of feature i and feature $i + 1$, where x_i and y_i are temporal and amplitude distances, respectively. Then the candidate hypothesis is defined to be a set of observed deformations and distances corresponding to a set of candidate features as $D_h = \{d_1, d_2, \dots, d_k, l_1, l_2, \dots, l_{k-1}\}$, where d_i ($1 \leq i \leq k$) is the observed deformation between local feature q_i and the observed data at location i in the sequence (Keogh and Smyth, 1997). The candidate hypotheses are ranked by evaluating the likelihood $p(D_h|Q)$, where the generative probability model $p(D_h|Q)$ can be defined to varying levels of complexity in terms of the independence structure of the model and the functional forms of the component probability distributions.

For example, a simple generative probability model can be defined as

$$p(D_h|Q) = \prod_{i=1}^k p(d_i|q_i) \prod_{i=1}^{k-1} p(l_i|q_i),$$

where the model $p(d_i|q_i)$ is chosen based on prior knowledge of how features are deformed, such as an exponential model

$$p(d_i|Q) = \lambda_i e^{-\lambda_i d_i}.$$

The interfeature distance model $p(l_i|q_i)$ is a joint density on temporal and amplitude elasticity between features, such as y_i obeying a uniform distribution and y_i obeying a lognormal

distribution. Given these particular models, the loglikelihood is

$$\ln p(D_h|Q) \propto \sum_{i=1}^k \lambda_i d_i + \frac{1}{2} \sum_{i=1}^{k-1} \left(\frac{\ln x_i - \mu_i}{\sigma_i} \right)^2,$$

modulo a few extra conditions where this density is zero.

The probabilistic model naturally defines a distance metric that integrates both local and global evidence weighted appropriately according to prior knowledge. Ge and Smyth (2000) proposed another probabilistic generative modeling method based on segmental semi-Markov models to leverage both prior knowledge and training data.

6.6.6 Measures Based on Landmark Models

The landmark concept arises from psychology and cognitive science, on which humans and animals depend to organize their spatial memory. Perng et al. (2000) proposed a landmark model for time series, which leads to a landmark similarity that is consistent with human intuition and episodic memory. In the landmark model, landmarks in time series are defined to be those points (e.g., times, events) of greatest importance.

A point is called an n th-order landmark of a curve if the n th-order derivative is zero at the point. Given a sequence of landmarks, the curve can be reconstructed by segments of real-valued functions (Perng et al., 2000). The landmark model preserves all the peaks (i.e., local maxima) and valley (i.e., local minima) that are typically filtered out by both the discrete fourier transformation (DFT) (Faloutsos et al., 1994; Kahveci et al., 2002; Faloutsos et al., 1997; Rafiee and Mendelzon, 1998; Wu et al., 2000b) and the discrete wavelet transformation (DWT) (Chan and Fu, 1999; Popivanov and Miller, 2002).

Real-world data usually contain noise. In order to smooth the data, a process called the minimum distance/percentage principle (MDPP), which can be implemented in linear time, is introduced. Specifically, given a sequence of landmarks $(x_1, y_1), \dots, (x_n, y_n)$, a minimum distance D , and a minimum percentage P , remove landmarks (x_i, y_i) and (x_{i+1}, y_{i+1}) if

$$x_{i+1} - x_i < D \text{ and } \frac{2|y_{i+1} - y_i|}{|y_i| + |y_{i+1}|} < P.$$

The MDPP defined above preserves the offsets of each landmark. The landmark similarity in the landmark model is defined as follows: Let $L = (L_1, L_2, \dots, L_n)$ and $L' = (L'_1, L'_2, \dots, L'_n)$ be two sequences of landmarks, where $L_i = (x_i, y_i)$ and $L'_i = (x'_i, y'_i)$. The distance between the k th landmarks is defined as

$$\Delta_k(L, L') = \left(\delta_k^{(t)}(L, L'), \delta_k^{(a)}(L, L') \right),$$

where

$$\delta_k^{(t)}(L, L') = \begin{cases} \frac{2|(x_k - x_{k-1}) - (x'_k - x'_{k-1})|}{|x_k - x_{k-1}| + |x'_k - x'_{k-1}|} & \text{if } 1 < k \leq n, \\ 0 & \text{otherwise,} \end{cases}$$

$$\delta_k^{(a)}(L, L') = \begin{cases} 0 & \text{if } y_k = y'_k, \\ \frac{2|y_k - y'_k|}{|y_k| + |y'_k|} & \text{otherwise.} \end{cases}$$

The distance between the two sequences L and L' is defined as

$$\Delta(L, L') = (\|\delta^{(t)}(L, L')\|, \|\delta^{(a)}(L, L')\|),$$

where $\delta^{(t)}(L, L') = (\delta_1^{(t)}(L, L'), \delta_2^{(t)}(L, L'), \dots, \delta_n^{(t)}(L, L'))$, $\delta^{(a)}(L, L')$ is defined similarly, and $\|\cdot\|$ is a vector norm such as the l_∞ -norm.

The landmark similarity satisfies the triangle inequality, and it is invariant under a family of transformations. Several transformations are defined and discussed in (Perng et al., 2000).

6.6.7 Evaluation

Similarity and dissimilarity measures are key to clustering algorithms. Due to the unique structure of time series data, most classic clustering algorithms do not work well for time series. Thus, most of the contributions for time series clustering focus on providing a new similarity or dissimilarity measure as a subroutine to an existing clustering algorithm (Keogh and Kasetty, 2003). Many similarity and dissimilarity measures have been developed in the literature of sequence mining (Wang and Wang, 2000; Park and Chu, 1999; Park et al., 2001; Lee et al., 2000; Pratt and Fink, 2002; Park et al., 2000; Qu et al., 1998; Gavrilov et al., 2000; Jin et al., 2002; Indyk et al., 2000; Li et al., 1998; Kim et al., 2000a), so users often find it challenging to choose an appropriate measure.

In general, there are two approaches to evaluating a similarity measure: subjective evaluation and objective evaluation (Keogh and Kasetty, 2003). Subjective evaluation creates dendrograms of several time series from the domain of interest using different measures and then plots these dendrograms side by side (Keogh and Pazzani, 1998). A dendrogram is an attractive way to visualize a similarity measure. Another way of visualizing the quality of a similarity measure is to project the time series into two-dimensional space via methods like multidimensional scaling (MDS) or self-organizing map (SOM) (Debregeas and Hebrail, 1998).

Unlike the subjective evaluation, the objective evaluation of a similarity measure uses a database of labeled time series. The objective measurements of the quality of a proposed similarity measure can be obtained by running simple classification experiments. The synthetic data sets *Cylinder-Bell-Funnel* (Geurts, 2001) and *Control-Chart* (Blake and Merz, 1998), for example, can be used to run the experiments. Keogh and Kasetty (2003) conducted several experimental comparisons of 11 similarity measures using the 2 synthetic data sets.

6.7 Other Measures

Although the similarity and dissimilarity measures discussed above can be applied to data types such as transaction data, gene expression data, and documents, other types of similarity and dissimilarity measures have also been developed for specific types of data.

6.7.1 The Cosine Similarity Measure

The cosine similarity measure (Salton and McGill, 1983) is adopted to measure the similarity between transaction data. Let t_i and t_j be two transactions represented by a d -dimensional bit vector. Then the cosine similarity between t_i and t_j is given by

$$\text{Cos}(t_i, t_j) = \frac{\langle t_i, t_j \rangle}{\|t_i\| \cdot \|t_j\|}, \quad (6.35)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product and $\|\cdot\|$ denotes a vector norm. The cosine similarity measure is also presented in (Xiao and Dunham, 2001).

6.7.2 A Link-based Similarity Measure

A link-based similarity measure between two data points is defined based on the relation of the two data points with other data points in the data set. It can be defined in such a way that it does not depend on the distance. Below, we present the link-based similarity measure used in ROCK (Guha et al., 2000b).

Let $\text{sim}(p_i, p_j)$ be a normalized similarity function; it can be one of the known L_p -metrics or even nonmetric. Assume that sim takes a value in $[0, 1]$. Given a threshold θ between zero and one, p_i and p_j are said to be neighbors if the following condition is satisfied:

$$\text{sim}(p_i, p_j) \geq \theta.$$

Apart from the closeness or similarity between points p_i and p_j , the quantity $\text{link}(p_i, p_j)$ is defined to distinguish two clusters:

$$\text{link}(p_i, p_j) = |\{p : \text{sim}(p, p_i) \geq \theta \text{ and } \text{sim}(p, p_j) \geq \theta \forall p \text{ in the database}\}|. \quad (6.36)$$

Hence, the larger the $\text{link}(p_i, p_j)$, the more probable it is that p_i and p_j belong to the same cluster.

In particular, for market basket data, the similarity function can be defined as

$$\text{sim}(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}, \quad (6.37)$$

where T_1 and T_2 are two transactions and $|T_i|$ is the number of items in T_i .

The link-based similarity measure between two clusters C_i and C_j can be defined as

$$g(C_i, C_j) = \frac{\text{link}[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}},$$

where $\text{link}[C_i, C_j]$ is the number of cross links between clusters C_i and C_j , i.e.,

$$\text{link}[C_i, C_j] = \sum_{p_q \in C_i, p_r \in C_j} \text{link}(p_q, p_r).$$

Here $f(\theta)$ is a positive function such as

$$f(\theta) = \frac{1 - \theta}{1 + \theta}.$$

6.7.3 Support

Support (Ganti et al., 1999) is a similarity measure defined for categorical attributes.

Let A_1, \dots, A_d be a set of categorical attributes with domains $D_1 = \text{DOM}(A_1)$, $D_2 = \text{DOM}(A_2), \dots, D_d = \text{DOM}(A_d)$, respectively. Let D be a set of tuples with each tuple $t \in D_1 \times D_2 \times \dots \times D_d$. Let $a_i \in D_i$ and $a_j \in D_j$, $i \neq j$. The support $\sigma_D(a_i, a_j)$ of a_i and a_j with respect to D is defined by

$$\sigma_D(a_i, a_j) = \frac{|\{t \in D : t.A_i = a_i \text{ and } t.A_j = a_j\}|}{|D|}. \quad (6.38)$$

The attribute values a_i and a_j are said to be strongly connected with respect to D if

$$\sigma_D(a_i, a_j) > \alpha \cdot \frac{|D|}{|D_i| \cdot |D_j|},$$

where $\alpha > 0$ is a parameter.

Support can also be defined in another way: the support of a_i in D is the number of objects in D whose i th attribute is a_i , i.e.,

$$\sigma_D(a_i) = |\{t \in D : t.A_i = a_i\}|.$$

6.8 Similarity and Dissimilarity Measures between Clusters

Many clustering algorithms are hierarchical, i.e., is a sequence of nested partitions. In an agglomerative hierarchical algorithm, the two most similar groups are merged to form a large cluster at each step, and this processing is continued until the desired number of clusters is obtained. In a divisive hierarchical algorithm, the process is reversed by starting with all data points in one cluster and subdividing into smaller clusters. In either case, we need to compute the distance between an object and a cluster and the distance between two clusters.

In what follows, we always let $C_1 = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_r\}$ and $C_2 = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_s\}$ denote two clusters of size r and s from a partition, respectively.

6.8.1 The Mean-based Distance

A popular way to measure the dissimilarity between two clusters for numerical data is to measure the distance between the means of the two clusters. Suppose C_1 and C_2 are two clusters of a numerical data set. Then the mean-based distance between C_1 and C_2 with respect to $d(\cdot, \cdot)$ is defined as

$$D_{\text{mean}}(C_1, C_2) = d(\mu(C_1), \mu(C_2)), \quad (6.39)$$

where $\mu(C_1)$ and $\mu(C_2)$, are the means of clusters C_1 and C_2 , respectively, i.e.,

$$\mu(C_j) = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}, \quad j = 1, 2.$$

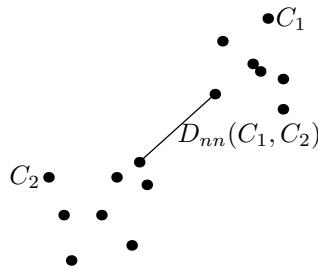


Figure 6.1. Nearest neighbor distance between two clusters.

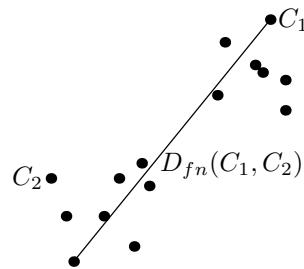


Figure 6.2. Farthest neighbor distance between two clusters.

6.8.2 The Nearest Neighbor Distance

Given a distance function $d(\cdot, \cdot)$, the nearest neighbor distance (Williams and Lambert, 1966) between C_1 and C_2 with respect to $d(\cdot, \cdot)$ is defined as

$$D_{nn}(C_1, C_2) = \min_{1 \leq i \leq r, 1 \leq j \leq s} d(\mathbf{y}_i, \mathbf{z}_j). \quad (6.40)$$

Figure 6.1 gives an example of the nearest neighbor distance in the two-dimensional case.

6.8.3 The Farthest Neighbor Distance

The farthest distance neighbor (Duran and Odell, 1974) between C_1 and C_2 with respect to $d(\cdot, \cdot)$ is defined as

$$D_{fn}(C_1, C_2) = \max_{1 \leq i \leq r, 1 \leq j \leq s} d(\mathbf{y}_i, \mathbf{z}_j). \quad (6.41)$$

Figure 6.2 gives an example of the farthest neighbor distance in the two-dimensional case.

6.8.4 The Average Neighbor Distance

The average neighbor distance (Duran and Odell, 1974) between C_1 and C_2 with respect to $d(\cdot, \cdot)$ is defined as

$$D_{ave}(C_1, C_2) = \frac{1}{rs} \sum_{i=1}^r \sum_{j=1}^s d(\mathbf{y}_i, \mathbf{z}_j). \quad (6.42)$$

The statistical distance (Duran and Odell, 1974) between C_1 and C_2 is defined as

$$D_{stat}(C_1, C_2) = \frac{rs}{r+s} (\bar{\mathbf{y}} - \bar{\mathbf{z}})(\bar{\mathbf{y}} - \bar{\mathbf{z}})^T, \quad (6.43)$$

where $\bar{\mathbf{y}}$ and $\bar{\mathbf{z}}$ are defined as

$$\bar{\mathbf{y}} = \frac{1}{r} \sum_{i=1}^r \mathbf{y}_i, \quad (6.44)$$

$$\bar{\mathbf{z}} = \frac{1}{s} \sum_{j=1}^s \mathbf{z}_j. \quad (6.45)$$

Let C be the cluster formed by merging C_1 and C_2 , i.e., $C = C_1 \cup C_2$, and let $M_{sca}(C)$, $M_{sca}(C_1)$, and $M_{sca}(C_2)$ be the within-scatter matrices, of clusters C , C_1 , and C_2 , respectively. Then it can be shown that (Duran and Odell, 1974)

$$M_{sca}(C) = M_{sca}(C_1) + M_{sca}(C_2) + \frac{rs}{r+s} (\bar{\mathbf{y}} - \bar{\mathbf{z}})(\bar{\mathbf{y}} - \bar{\mathbf{z}})^T, \quad (6.46)$$

where $\bar{\mathbf{y}}$ and $\bar{\mathbf{z}}$ are defined in (6.44) and (6.45) respectively.

The matrix $\frac{rs}{r+s} (\bar{\mathbf{y}} - \bar{\mathbf{z}})(\bar{\mathbf{y}} - \bar{\mathbf{z}})^T$ in (6.46) is called the between-scatter matrix, and the trace of this matrix is exactly the statistical distance between clusters C_1 and C_2 .

6.8.5 Lance-Williams Formula

In an agglomerative hierarchical algorithm, we need to compute the distances between old clusters and a new cluster formed by two clusters. Lance and Williams (1967a) propose a recurrence formula that gives the distance between a cluster C_k and a cluster C formed by the fusion of clusters C_i and C_j , i.e., $C = C_i \cup C_j$. The formula is given by

$$\begin{aligned} & D(C_k, C_i \cup C_j) \\ &= \alpha_i D(C_k, C_i) + \alpha_j D(C_k, C_j) \\ &\quad + \beta D(C_i, C_j) + \gamma |D(C_k, C_i) - D(C_k, C_j)|, \end{aligned} \quad (6.47)$$

where $D(\cdot, \cdot)$ is a distance between two clusters.

By a suitable choice of the parameters α_i , α_j , β , and γ in (6.47), we can obtain various intercluster distances used by hierarchical clustering algorithms. Table 6.6 gives some commonly used values for the parameters in the Lance-Williams formula. Some properties of equation (6.47) were investigated in DuBien and Warde (1979).

Jambu (1978) proposed a more general recurrence formula that contains more parameters than the Lance-Williams formula. The general recurrence formula is defined as

Table 6.6. Some commonly used values for the parameters in the Lance-Williams's formula, where $n_i = |C_i|$ is the number of data points in C_i , and $\Sigma_{ijk} = n_i + n_j + n_k$.

Algorithm	α_i	α_j	β	γ
Single-link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{-1}{2}$
Complete link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Ward's method	$\frac{n_i+n_j}{\Sigma_{ijk}}$	$\frac{n_j+n_k}{\Sigma_{ijk}}$	$\frac{-n_k}{\Sigma_{ijk}}$	0
Group average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Weighted group average	$\frac{1}{2}$	$\frac{1}{2}$	0	0
Centroid	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i n_j}{(n_i+n_j)^2}$	0
Median (weighted centroid)	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{-1}{4}$	0

$$\begin{aligned}
 & D(C_k, C_i \cup C_j) \\
 &= \alpha_i D(C_k, C_i) + \alpha_j D(C_k, C_j) \\
 &\quad + \beta D(C_i, C_j) + \gamma |D(C_k, C_i) - D(C_k, C_j)| \\
 &\quad + \delta_i h(C_i) + \delta_j h(C_j) + \epsilon h(C_k),
 \end{aligned} \tag{6.48}$$

where $D(\cdot, \cdot)$ is a distance between two clusters and $h(C_i)$ denotes the height in the dendrogram of C_i .

In Table 6.7, we let n_i , n_j , and n_k be the number of data points in C_i , C_j , and C_k , respectively, let Z_{ij} , Z_{ik} , Z_{jk} , and Z_{ijk} be

$$\begin{aligned}
 Z_{ij} &= \frac{\binom{n_i + n_j}{2}}{\binom{\Sigma_{ijk}}{2}}, \\
 Z_{ik} &= \frac{\binom{n_i + n_k}{2}}{\binom{\Sigma_{ijk}}{2}}, \\
 Z_{jk} &= \frac{\binom{n_j + n_k}{2}}{\binom{\Sigma_{ijk}}{2}}, \\
 Z_{ijk} &= \frac{\binom{n_i + n_j + n_k}{2}}{\binom{\Sigma_{ijk}}{2}},
 \end{aligned}$$

where $\Sigma_{ijk} = n_i + n_j + n_k$, and finally let Z_i , Z_j , and Z_k be

$$Z_i = \frac{\binom{n_i}{2}}{\binom{\Sigma_{ijk}}{2}}, \quad Z_j = \frac{\binom{n_j}{2}}{\binom{\Sigma_{ijk}}{2}}, \quad Z_k = \frac{\binom{n_k}{2}}{\binom{\Sigma_{ijk}}{2}}.$$

Different choices of parameters $(\alpha_i, \alpha_j, \beta, \gamma, \delta_i, \delta_j, \epsilon)$ give different clustering schemes. Table 6.7 summarizes some common parameters for the general recurrence formula. This table is also presented in (Gordon, 1996). Other values of β in the recurrence formula that might be more successful in recovering the underlying cluster structure have also been suggested by Milligan (1989) and Scheibler and Schneider (1985).

6.9 Similarity and Dissimilarity between Variables

Instead of performing cluster analysis on a set of records, one can perform cluster analysis on a set of variables that have been observed in some population. Situations exist in which several variables describe the same property; in these situations, one may first perform cluster analysis on the variables to reduce the number of variables.

6.9.1 Pearson's Correlation Coefficients

Cattell (1949) suggested three senses of pattern matching: matching for shape, matching for absolute agreement, and matching for effect. The correlation coefficient can be used to measure the agreement of shapes between two patterns. Correlation coefficients provide a measure of similarity between variables (Kaufman and Rousseeuw, 1990) if one wants to perform a cluster analysis on a set of variables that have been observed in some population.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set in which each object is described by d attributes v_1, v_2, \dots, v_d . Then the Pearson product-moment correlation coefficient between v_s and v_t is defined as

$$R(v_s, v_t) = \frac{\sum_{i=1}^n (x_{is} - m_s)(x_{it} - m_t)}{\sqrt{\sum_{i=1}^n (x_{is} - m_s)^2} \sqrt{\sum_{i=1}^n (x_{it} - m_t)^2}} \quad (6.49)$$

for $s, t = 1, 2, \dots, d$, where x_{is} and x_{it} are the s th and t th components of \mathbf{x}_i , respectively, and m_s is the mean in the s th attribute, i.e.,

$$m_s = \frac{1}{n} \sum_{i=1}^n x_{is}.$$

The coefficients defined in equation (6.49) lie in $[-1, 1]$. They can be converted to dissimilarities $d(v_s, v_t)$ as follows (Kaufman and Rousseeuw, 1990):

$$d(v_s, v_t) = \frac{1 - R(v_s, v_t)}{2},$$

Table 6.7. Some common parameters for the general recurrence formula proposed by Jambu (1978).

Name	α_i	α_j	β	γ	δ_i	δ_j	ϵ	Reference
Single-link	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0	(Sneath, 1957)
Complete link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	(McQuitty, 1960)
Ward's method (minimum variance)	$\frac{n_i+n_k}{\Sigma_{ijk}}$	$\frac{n_j+n_k}{\Sigma_{ijk}}$	$\frac{-n_k}{\Sigma_{ijk}}$	0	0	0	0	(Ward Jr., 1963) (Wishart, 1969)
Group average (UPGMA) ¹	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0	0	0	0	(McQuitty, 1967)
Weighted group average (WPGMA)	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0	0	(McQuitty, 1966) (McQuitty, 1967)
Centroid (UPGMC)	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i n_j}{(n_i+n_j)^2}$	0	0	0	0	(Gower, 1967)
Median (WPGMC, weighted centroid) ²	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{4}$	0	0	0	0	(Gower, 1967)

Continued on next page

¹“UPGMA” stands for “unweighted pair group method using arithmetic averages” (Jain and Dubes, 1988).
²“WPGMC” stand for “weighted pair group method using centroids”(Jain and Dubes, 1988).

Continued from previous page						
Name	α_i	α_j	β	γ	δ_i	δ_j
Flexible	$\frac{1-x}{2}$	$\frac{1-x}{2}$	$x(x < 1)$	0	0	0
Mean dissimilarity	$\frac{Z_{ik}}{Z_{ijk}}$	$\frac{Z_{ij}}{Z_{ijk}}$	0	$\frac{-Z_i}{Z_{ijk}}$	$\frac{-Z_j}{Z_{ijk}}$	(Podani, 1989)
Sum of squares	$\frac{n_i+n_k}{\Sigma_{ijk}}$	$\frac{n_j+n_k}{\Sigma_{ijk}}$	0	$\frac{-n_i}{\Sigma_{ijk}}$	$\frac{-n_j}{\Sigma_{ijk}}$	(Jambu, 1978) (Podani, 1989)

Table 6.8. The contingency table of variables u and v .

	B_1	B_2	\dots	B_q	Totals
A_1	n_{11}	n_{12}	\dots	n_{1q}	r_1
A_2	n_{21}	n_{22}	\dots	n_{2q}	r_2
\vdots	\vdots	\vdots	\dots	\vdots	\vdots
A_p	n_{p1}	n_{p2}	\dots	n_{pq}	r_p
Totals	c_1	c_2	\dots	c_q	n

or

$$d(v_s, v_t) = 1 - |R(v_s, v_t)|.$$

Pearson's correlation coefficient is the standard measure of the linear relationship between two variables. The absolute value of this coefficient gives the degree of relationship. When the variables have no linear relationship, the value is 0; when each variable is perfectly predicted by the other, the absolute value is 1. More specifically, a positive Pearson's correlation coefficient indicates a tendency for high values of one variable to be associated with high values of the other and low values to be associated with low values of the other. A negative sign indicates a tendency for high values of one variable to be associated with low values of the other and low values to be associated with high values of the other.

Pearson's correlation coefficient is very simple. Farris (1969) introduced the Cophe- netic correlation coefficient, which is also applicable for the purpose of clustering.

6.9.2 Measures Based on the Chi-square Statistic

The chi-square statistic is a common quantity in the analysis of the contingency table. It represents the joint distribution of two categorical variables (Anderberg, 1973). Measures of association can be defined based on the chi-square statistic.

Let D be a data set with n objects, each of which is described by d categorical attributes. Let u and v be two of the d categorical variables with domains $\{A_1, A_2, \dots, A_p\}$ and $\{B_1, B_2, \dots, B_q\}$. Then the contingency table that represents the joint distribution of variables u and v with respect to the data set is shown in Table 6.8. The n_{ij} entry in the table is the number of objects in D that fall in both the i th state of variable u and the j th state of variable v . c_j is the number of objects in D that fall in the j th state of variable v , and r_i is the number of objects in D that fall in the i th state of variable u .

The nominal measures of association are invariant to any permutation of rows and columns in the contingency table, but the ordinal measures of association are dependent on the prior ordering of rows and columns.

Based on the contingency table, the sample chi-square statistic is defined as

$$\chi^2 = \sum_{i=1}^p \sum_{j=1}^q \frac{(o_{ij} - e_{ij})^2}{e_{ij}},$$

where o_{ij} is the observed count in cell ij and e_{ij} is the corresponding expected count under the hypothesis of independence.

Table 6.9. Measures of association based on the chi-square statistic.

Measure	Definition
Mean-square contingency	$\phi^2 = \frac{\chi^2}{n}$
Tschuprow	$T = \left[\frac{\chi^2}{n\sqrt{(p-1)(q-1)}} \right]^{\frac{1}{2}}$
Cramér	$C = \left[\frac{\chi^2}{n \min\{(p-1), (q-1)\}} \right]^{\frac{1}{2}}$
Maung	$C = \left[\sum_{i=1}^k \frac{g_i^2}{k} \right]^{\frac{1}{2}}$
Pearson's coefficient of contingency	$P = \left(\frac{\chi^2}{n + \chi^2} \right)^{\frac{1}{2}}$

Let f_{ij} be the frequency of entry ij , i.e., $f_{ij} = \frac{n_{ij}}{n}$. Then under the hypothesis of independence the expected count in cell ij is

$$e_{ij} = f_{i\cdot} f_{\cdot j} n = \frac{r_i c_j}{n},$$

where

$$f_{i\cdot} = \sum_{j=1}^q f_{ij}, \quad f_{\cdot j} = \sum_{i=1}^p f_{ij}.$$

The observed count o_{ij} is $f_{ij}n = n_{ij}$. We have several forms of the chi-square statistic (Anderberg, 1973):

$$\chi^2 = \sum_{i=1}^p \sum_{j=1}^q \left(n_{ij} - \frac{r_j c_j}{n} \right)^2 \frac{n}{r_i c_j}, \quad (6.50a)$$

$$\chi^2 = n \sum_{i=1}^p \sum_{j=1}^q \frac{(f_{ij} - r_i c_j)^2}{f_{i\cdot} f_{\cdot j}}, \quad (6.50b)$$

$$\chi^2 = n \sum_{i=1}^p \sum_{j=1}^q \frac{f_{ij}^2}{f_{i\cdot} f_{\cdot j}} - n = n \sum_{i=1}^p \sum_{j=1}^q \frac{n_{ij}^2}{r_i c_j} - n. \quad (6.50c)$$

Since the value of χ^2 increases without bound as n increases, it is not a suitable measure of association. Several extensions of the chi-square statistics are presented in Table 6.9 (Anderberg, 1973), where g_i is the i th of k nonzero canonical correlations.

The mean-square contingency is simple, but it depends on the size of the contingency table (Anderberg, 1973). Other measures in the table normalize ϕ^2 to the range of 0 and 1. Even though the chi-square statistic and the measures derived from it are useful as tests of hypotheses, they may not be so useful as measures of association (Goodman and Kruskal, 1954). The chi-square-based measures are not preferred for the purpose of cluster analysis as they lack operational interpretation (Anderberg, 1973). These traditional measures have also been presented in (Goodman and Kruskal, 1954).

6.9.3 Measures Based on Optimal Class Prediction

Association between variables can be defined through measuring the power of one as a predictor for the other. Generally, there are two types of optimal predictions: asymmetrical optimal prediction and symmetrical optimal prediction. In this subsection, we shall introduce measures of association based on these two types of optimal predictions.

Measures Based on Asymmetrical Prediction

Let u and v be two variables as in subsection 6.9.2. The model of activity is as follows (Goodman and Kruskal, 1954): an object is chosen at random from the population and we are asked to guess its v class (or state) as much as we can, either

1. given no further information or
2. given its u class (or state).

In case 1, the best we can do is to choose the v class B_m with the largest marginal total, i.e., the value of m satisfying

$$f_{\cdot m} = \max_{1 \leq j \leq q} f_{\cdot j}.$$

In this case, the probability of error is

$$P_1 = 1 - f_{\cdot m}.$$

In case 2, suppose the u class for an object is known to be A_a for some $1 \leq a \leq p$. Then only row a of the contingency table (see Table 6.8) is of interest and we are best off guessing B_{m_a} for which

$$f_{am_a} = \max_{1 \leq j \leq q} f_{aj}.$$

Given that a is known, the probability of error is $P_1 = 1 - \frac{f_{am_a}}{f_{\cdot a}}$. Thus in case 2, the probability of error is

$$P_2 = \sum_{i=1}^p f_{i\cdot} \left(1 - \frac{f_{im_i}}{f_{i\cdot}} \right) = 1 - \sum_{i=1}^p f_{im_i}.$$

Goodman and Kruskal (1954) proposed a measure of association as

$$\lambda_v = \frac{P_1 - P_2}{P_1} = \frac{\sum_{i=1}^p f_{im_i} - f_{\cdot m}}{1 - f_{\cdot m}}. \quad (6.51)$$

The λ_v defined in equation (6.51) is the relative decrease in probability of error in guess B_b as between A_a unknown and A_a known. The measure λ_v has some important properties:

- λ_v is indeterminate if and only if the population lies in one v class; for example, the objects in a data set have the same state for variable v .

- Otherwise, $\lambda_v \in [0, 1]$.

Other properties of λ_v are discussed in (Goodman and Kruskal, 1954). Similarly, a measure of the predictive power of v for u can be defined as

$$\lambda_u = \frac{\sum_{j=1}^q f_{m_j j} - f_{m \cdot}}{1 - f_{m \cdot}},$$

where

$$f_{m \cdot} = \max_{1 \leq i \leq p} f_{i \cdot}, \quad f_{m_j j} = \max_{1 \leq i \leq p} f_{ij}.$$

Measures Based on Symmetrical Prediction

For symmetrical prediction, the model of activity is as follows: an object is chosen at random from the population and we are asked to guess its u class half the time and its v class half the time, either

1. given no further information, or
2. given the object's u class when we guess its v class and vice versa.

In case 1, the probability of error is

$$P_1 = 1 - \frac{1}{2}(f_{\cdot m} + f_{m \cdot}),$$

and in case 2, the probability of error is

$$P_2 = 1 - \frac{1}{2} \left(\sum_{i=1}^p f_{im_i} + \sum_{j=1}^q f_{m_j j} \right).$$

Considering the relative decrease in the probability of error, the measure is defined as

$$\lambda = \frac{P_1 - P_2}{P_1} = \frac{\frac{1}{2} \left(\sum_{i=1}^p f_{im_i} + \sum_{j=1}^q f_{m_j j} - f_{\cdot m} - f_{m \cdot} \right)}{1 - \frac{1}{2}(f_{\cdot m} + f_{m \cdot})},$$

or

$$\lambda = \frac{P_1 - P_2}{P_1} = \frac{\sum_{i=1}^p f_{im_i} + \sum_{j=1}^q f_{m_j j} - f_{\cdot m} - f_{m \cdot}}{2 - f_{\cdot m} - f_{m \cdot}}.$$

The measure λ has some similar properties as λ_u and λ_v :

- λ is determinant except when an entire object lies in a single cell of the contingency table.
- Otherwise, $\lambda \in [0, 1]$.

Anderberg (1973) introduced a measure that is closely related to λ as

$$D = P_1 - P_2 = \frac{1}{2} \left(\sum_{i=1}^p f_{im_i} + \sum_{j=1}^q f_{mj} - f_{m\cdot} - f_{\cdot m} \right),$$

which is the actual reduction in the probability of error. The measure D is always determinant and lies between 0 and 1 inclusive, i.e., $D \in [0, 1]$.

The measures presented in this subsection can be used to measure association between binary variables (Anderberg, 1973).

6.9.4 Group-based Distance

Mántaras (1991) proposed a distance between partitions that can be used directly to calculate the distance between two categorical variables in a data set. This distance is defined based on information theory.

Let X and Y be two categorical variables in a data set with n records, and let the i th record take values x_i and y_i on the X and Y variables, respectively. Suppose that X has k_1 ($k_1 \leq n$) states X_1, X_2, \dots, X_{k_1} , i.e., for each i ($1 \leq i \leq n$), $x_i = X_j$ for some j , and for each j ($1 \leq j \leq k_1$), there exists at least one i such that $x_i = X_j$, and Y has k_2 states Y_1, Y_2, \dots, Y_{k_2} . Let the probabilities P_s , P_t , P_{st} , $P_{t|s}$, and $P_{s|t}$ be defined as

$$\begin{aligned} P_s &= \frac{|\{i : x_i = X_s, 1 \leq i \leq n\}|}{n}, \\ P_t &= \frac{|\{i : y_i = Y_t, 1 \leq i \leq n\}|}{n}, \\ P_{st} &= \frac{|\{i : x_i = X_s \wedge y_i = Y_t, 1 \leq i \leq n\}|}{n}, \\ P_{t|s} &= \frac{P_{st}}{P_s}, \\ P_{s|t} &= \frac{P_{st}}{P_t} \end{aligned}$$

for $s = 1, 2, \dots, k_1$ and $t = 1, 2, \dots, k_2$, where $|S|$ denotes the number of elements in the set S .

Let $d(X, Y)$ be defined as

$$d(X, Y) = 2I(P_X \cap P_Y) - I(P_X) - I(P_Y), \quad (6.52)$$

where $I(P_X)$, $I(P_Y)$, and $I(P_X \cap P_Y)$ are defined as

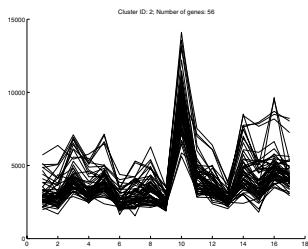
$$\begin{aligned} I(P_X) &= - \sum_{s=1}^{k_1} P_s \log_2 P_s, \\ I(P_Y) &= - \sum_{t=1}^{k_2} P_t \log_2 P_t, \\ I(P_X \cap P_Y) &= - \sum_{s=1}^{k_1} \sum_{t=1}^{k_2} P_{st} \log_2 P_{st}. \end{aligned}$$

It can be shown that $d(X, Y)$ defined in equation (6.52) is a metric distance measure (Mántaras, 1991). The normalization of the distance $d(X, Y)$ is given by (Mántaras, 1991)

$$d_N(X, Y) = \frac{d(X, Y)}{I(P_X \cap P_Y)} \in [0, 1]. \quad (6.53)$$

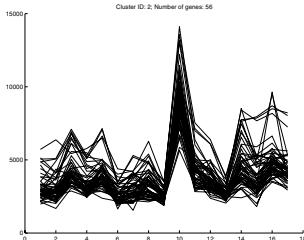
6.10 Summary

Some commonly used similarity and dissimilarity measures have been presented briefly in this chapter. In practice, the selection of similarity or dissimilarity measures depends on the type of variables as well as the measurement level of variables. For a comprehensive discussion of similarity or dissimilarity measures, readers are referred to (Sokal and Sneath, 1963) and (Sokal and Sneath, 1973). Additional discussions can be found in (Gower and Legendre, 1986), (Hubálek, 1982), (Hartigan, 1967), (Cattell, 1949), (Goodall, 1966), (Green and Rao, 1969), and (Baulieu, 1989).



Part II

Clustering Algorithms



Chapter 7

Hierarchical Clustering Techniques

Hard clustering algorithms are subdivided into hierarchical algorithms and partitional algorithms. A partitional algorithm divides a data set into a single partition, whereas a hierarchical algorithm divides a data set into a sequence of nested partitions. As we mentioned in Chapter 1, hierarchical algorithms are subdivided into agglomerative hierarchical algorithms and divisive hierarchical algorithms (see Figure 1.5).

Agglomerative hierarchical clustering starts with every single object in a single cluster. Then it repeats merging the closest pair of clusters according to some similarity criteria until all of the data are in one cluster. There are some disadvantages for agglomerative hierarchical clustering, such as (a) data points that have been incorrectly grouped at an early stage cannot be reallocated and (b) different similarity measures for measuring the similarity between clusters may lead to different results.

If we treat agglomerative hierarchical clustering as a bottom-up clustering method, then divisive hierarchical clustering can be viewed as a top-down clustering method. Divisive hierarchical clustering starts with all objects in one cluster and repeats splitting large clusters into smaller pieces. Divisive hierarchical clustering has the same drawbacks as agglomerative hierarchical clustering. Figure 7.1 gives an intuitive example of agglomerative hierarchical clustering and divisive hierarchical clustering.

Hierarchical algorithms can be expressed in terms of either graph theory or matrix algebra (Jain and Dubes, 1988). A dendrogram, a special type of tree structure, is often used to visualize a hierarchical clustering. Figure 7.1 is an example of a dendrogram.

7.1 Representations of Hierarchical Clusterings

A hierarchical clustering can be represented by either a picture or a list of abstract symbols. A picture of a hierarchical clustering is much easier for humans to interpret. A list of abstract symbols of a hierarchical clustering may be used internally to improve the performance of the algorithm. In this section, some common representations of hierarchical clusterings are summarized.

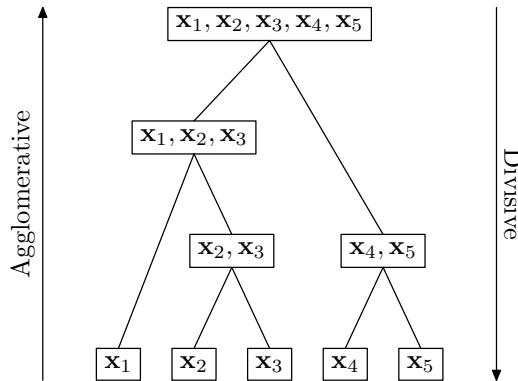


Figure 7.1. Agglomerative hierarchical clustering and divisive hierarchical clustering.

7.1.1 *n*-tree

A hierarchical clustering is generally represented by a tree diagram. An *n*-tree is a simple hierarchically nested tree diagram that can be used to represent a hierarchical clustering. Let $D = \{x_1, x_2, \dots, x_n\}$ be a set of objects. Then an *n*-tree on D is defined to be a set \mathcal{T} of subsets of D satisfying the following conditions (Bobisud and Bobisud, 1972; McMorris et al., 1983; Gordon, 1996):

1. $D \in \mathcal{T}$;
2. empty set $\Phi \in \mathcal{T}$;
3. $\{x_i\} \in \mathcal{T}$ for all $i = 1, 2, \dots, n$;
4. if $A, B \in \mathcal{T}$, then $A \cap B \in \{\Phi, A, B\}$.

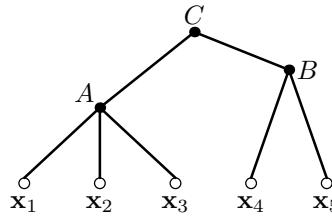
A 5-tree is illustrated in Figure 7.2. The terminal nodes or leaves depicted by an open circle represent a single data point. The internal nodes depicted by a filled circle represent a group or cluster. *n*-trees are also referred to as nonranked trees (Murtagh, 1984b). If an *n*-tree has precisely $n - 1$ internal nodes, then the tree is called a binary tree or a dichotomous tree.

Tree diagrams, such as *n*-trees and dendrograms (discussed later), contain many indeterminacies. For example, the order of the internal nodes and the order of leaves can be interchanged. Also, tree diagrams have many variations. For example, rotating the tree 90° gives a horizontal tree. Alternative properties of trees have been presented in (Hartigan, 1967) and (Constantinescu, 1966).

7.1.2 Dendrogram

A dendrogram is also called a valued tree (Gordon, 1996). A dendrogram is an *n*-tree in which each internal node is associated with a height satisfying the condition

$$h(A) \leq h(B) \Leftrightarrow A \subseteq B$$

**Figure 7.2.** A 5-tree.

for all subsets of data points A and B if $A \cap B \neq \Phi$, where $h(A)$ and $h(B)$ denote the heights of A and B , respectively.

As an illustration, Figure 7.3 shows a dendrogram with five data points. The dotted lines indicate the heights of the internal nodes. For each pair of data points $(\mathbf{x}_i, \mathbf{x}_j)$, let h_{ij} be the height of the internal node specifying the smallest cluster to which both \mathbf{x}_i and \mathbf{x}_j belong. Then a small value of h_{ij} indicates a high similarity between \mathbf{x}_i and \mathbf{x}_j . In the dendrogram given in Figure 7.3, for example, we have $h_{12} = 1$, $h_{23} = h_{13} = 3$, and $h_{14} = 4$.

The heights in the dendrogram satisfy the following ultrametric conditions (Johnson, 1967):

$$h_{ij} \leq \max\{h_{ik}, h_{jk}\} \quad \forall i, j, k \in \{1, 2, \dots, n\}. \quad (7.1)$$

In fact, the ultrametric condition (7.1) is also a necessary and sufficient condition for a dendrogram (Gordon, 1987).

Mathematically, a dendrogram can be represented by a function $c : [0, \infty) \rightarrow E(D)$ that satisfies (Sibson, 1973)

$$\begin{aligned} c(h) &\subseteq c(h') \text{ if } h \leq h', \\ c(h) &\text{ is eventually in } D \times D, \\ c(h + \delta) &= c(h) \text{ for some small } \delta > 0, \end{aligned}$$

where D is a given data set and $E(D)$ is the set of equivalence relations on D . As an example, the function c given below contains the information of the dendrogram given in Figure 7.3:

$$c(h) = \begin{cases} \{(i, i) : i = 1, 2, 3, 4, 5\} & \text{if } 0 \leq h < 1, \\ \{(i, i) : i = 3, 4, 5\} \cup \\ \{(i, j) : i, j = 1, 2\} & \text{if } 1 \leq h < 2, \\ \{(3, 3)\} \cup \\ \{(i, j) : i, j = 1, 2\} \cup \\ \{(i, j) : i, j = 4, 5\} & \text{if } 2 \leq h < 3, \\ \{(i, j) : i, j = 4, 5\} \cup \\ \{(i, j) : i, j = 1, 2, 3\} & \text{if } 3 \leq h < 4, \\ \{(i, j) : i, j = 1, 2, 3, 4, 5\} & \text{if } 4 \leq h. \end{cases} \quad (7.2)$$

Other characterizations of a dendrogram have been presented in (Johnson, 1967), (Jardine et al., 1967), and (Banfield, 1976). van Rijsbergen (1970) suggested an algorithm

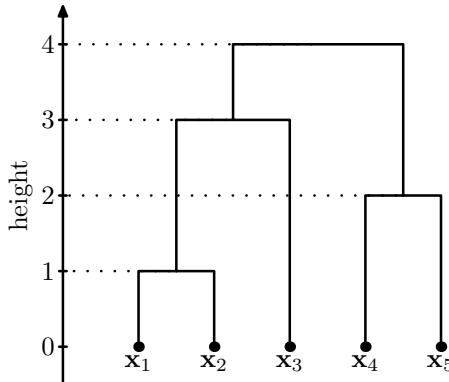


Figure 7.3. A dendrogram of five data points.

for finding the single-link dendrogram from the input dissimilarity matrix. Algorithms for plotting dendrograms have been discussed in (Rohlf, 1974), (Gower and Ross, 1969), and (Ross, 1969). Comparison of dendrograms has been discussed in (Sokal and Rohlf, 1962).

7.1.3 Banner

A banner (Rousseeuw, 1986) is a list of symbols and codes that represent a hierarchical structure. Banners can be constructed from dendrograms. In a banner, the heights in the dendrogram are represented on a horizontal axis. Each data point in the banner is assigned a line and a code that is repeated with a separator (such as “+”) along the line until truncated at the right-hand margin. The presence of a symbol (such as “*”) between two data points indicates that the two points are in the same group for this value of the height.

Figure 7.4 illustrates a banner that contains the information in the dendrogram given in Figure 7.3. In this banner, each data point is labeled by a two-number code. Alternative examples of banner representations are presented in (Kaufman and Rousseeuw, 1990, Chapter 6) and Gordon (1996).

7.1.4 Pointer Representation

A pointer representation (Sibson, 1973) is a pair of functions which contain information on a dendrogram. It is defined to be a pair of functions $\pi : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\}$ and $\lambda : \pi(\{1, 2, \dots, n\}) \rightarrow [0, \infty]$ that have the following properties:

$$\pi(n) = n, \quad \pi(i) > i \text{ for } i < n, \quad (7.3a)$$

$$\lambda(n) = \infty, \quad \lambda(\pi(i)) > \lambda(i) \text{ for } i < n, \quad (7.3b)$$

where n is the number of data points in D .

Given a dendrogram, the corresponding $\lambda(i)$ is the lowest level at which the i th object is no longer the last object in its cluster and $\pi(i)$ is the last object in the cluster that it joins.

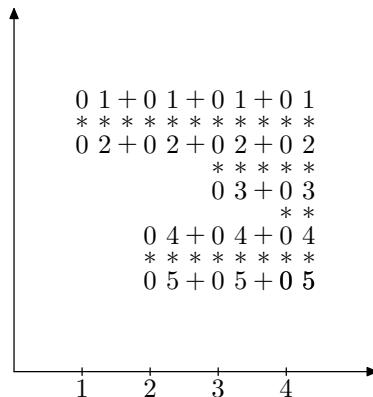


Figure 7.4. A banner constructed from the dendrogram given in Figure 7.3.

Mathematically, let c be the function that denotes a dendrogram. Then the corresponding pointer representation is defined by

$$\begin{aligned}\lambda(i) &= \inf\{h : \exists j > i \text{ such that } (i, j) \in c(h)\}, \\ \pi(i) &= \max\{j : (i, j) \in c(\lambda(i))\}\end{aligned}$$

for $i < n$.

The pair of functions λ and π illustrated in Table 7.1 is the pointer representation of the dendrogram given in Figure 7.3.

It can be shown that there is a one-to-one correspondence between dendrograms and pointer representations (Sibson, 1973). The pointer representation of a dendrogram allows a new object to be inserted in an efficient way. Usually, pointer representations are used internally in hierarchical algorithms in order to improve the performance. The pointer representation of a dendrogram is not helpful from a user's point of view; therefore, a so-called packed representation, which will be presented in the next section, is used for output.

Table 7.1. The pointer representation corresponding to the dendrogram given in Figure 7.3.

i	$\pi(i)$	$\lambda(i)$
1	2	1
2	3	3
3	5	4
4	5	2
5	5	∞

Table 7.2. The packed representation corresponding to the pointer representation given in Table 7.1.

i	$\tau(i)$	$v(i)$
1	1	1
2	2	3
3	3	4
4	4	2
5	5	∞

Table 7.3. A packed representation of six objects.

i	1	2	3	4	5	6
$\tau(i)$	4	1	3	5	2	6
$v(i)$	2	1.5	1	2.5	0.5	∞
$\lambda(i)$	1.5	0.5	1	2	2.5	∞
$\pi(i)$	5	6	5	5	6	6
$\lambda(\tau(i))$	2	1.5	1	2.5	0.5	∞
$\pi(\tau(i))$	5	5	5	6	6	6
$\tau^{-1}(\pi(\tau(i)))$	4	4	4	6	6	6

7.1.5 Packed Representation

Packed representations (Sibson, 1973) are developed in order to facilitate the output of dendrograms. A packed representation consists of two functions. Let λ, π be a pointer representation. Then the corresponding packed representation is defined as a pair of functions τ, v (Sibson, 1973),

$$\tau : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\}, \quad v : \{1, 2, \dots, n\} \rightarrow [0, +\infty)$$

which satisfy the following conditions:

$$\begin{aligned} \tau &\text{ is one-to-one and onto,} \\ \tau^{-1}(\pi(\tau(i))) &> i \text{ if } i < n, \\ v(i) &= \lambda(\tau(i)), \\ v(j) &\leq v(i) \text{ if } i \leq j < \tau^{-1}(\pi(\tau(i))). \end{aligned}$$

The packed representation defined above determines a dendrogram uniquely. In fact, the order of the objects in the dendrogram is specified by the function τ , i.e., the index of the object in position i is $\tau(i)$. Table 7.2 gives the packed representation corresponding to the pointer representation given in Table 7.1.

Another example of a packed representation is illustrated in Table 7.3. The dendrogram determined by this packed representation is shown in Figure 7.5.

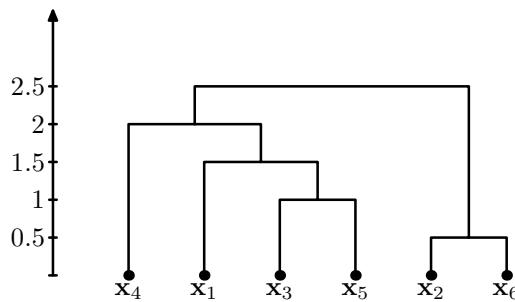


Figure 7.5. The dendrogram determined by the packed representation given in Table 7.3.

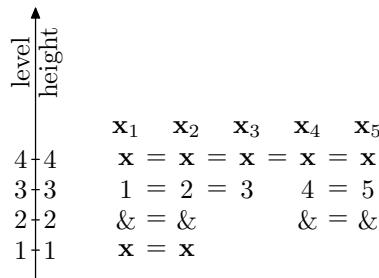


Figure 7.6. An icicle plot corresponding to the dendrogram given in Figure 7.3.

7.1.6 Icicle Plot

An icicle plot, proposed by Kruskal and Landwehr (1983), is another method for presenting a hierarchical clustering. It can be constructed from a dendrogram. The major advantage of an icicle plot is that it is easy to read off which objects are in a cluster during a live process of data analysis.

In an icicle plot, the height and the hierarchical level are represented along the vertical axis; each object is assigned a vertical line and labeled by a code that is repeated with separators (such as “&”) along the line from top to bottom until truncated at the level where it first joins a cluster, and objects in the same cluster are joined by the symbol “=” between two objects.

The list of symbols in Figure 7.6 is an icicle plot corresponding to the dendrogram given in Figure 7.3. Each object in this icicle plot is labeled by its name.

7.1.7 Other Representations

Other representations of hierarchical structures have been presented in (Sokal and Sneath, 1973), (Friedman and Rafsky, 1981), (Everitt and Nicholls, 1975), (Wirth et al., 1966), and (Hartigan and Wong, 1979), such as skyline plots (Wirth et al., 1966), silhouette plots (Rousseeuw, 1987), loop plots (see Figure 7.7) (Kruskal and Landwehr, 1983), and three-

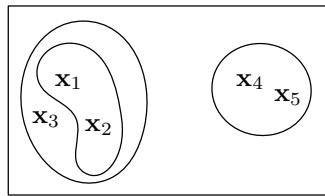


Figure 7.7. A loop plot corresponding to the dendrogram given in Figure 7.3.

dimensional plots (Kruskal and Landwehr, 1983). It seems, however, that these representations are only suitable for representing small data sets.

7.2 Agglomerative Hierarchical Methods

According to different distance measures between groups, agglomerative hierarchical methods can be subdivided into single-link methods, complete link methods, etc. Some commonly used hierarchical methods are given in Figure 7.8. The single, complete, average, and weighted average linkage methods are also referred to as graph methods, while Ward's method, the centroid method, and the median method are referred to as geometric methods (Murtagh, 1983), since in graph methods a cluster can be represented by a subgraph or interconnected points and in geometric methods a cluster can be represented by a center point.

Murtagh (1983) gives a survey for hierarchical clustering algorithms, especially for agglomerative hierarchical clustering algorithms. The performance of hierarchical clustering algorithms can be improved by incorporating efficient nearest neighbor searching algorithms into the clustering algorithms. For hierarchical methods, the storage requirements are reduced if each cluster is represented by a center point or a set of points, since

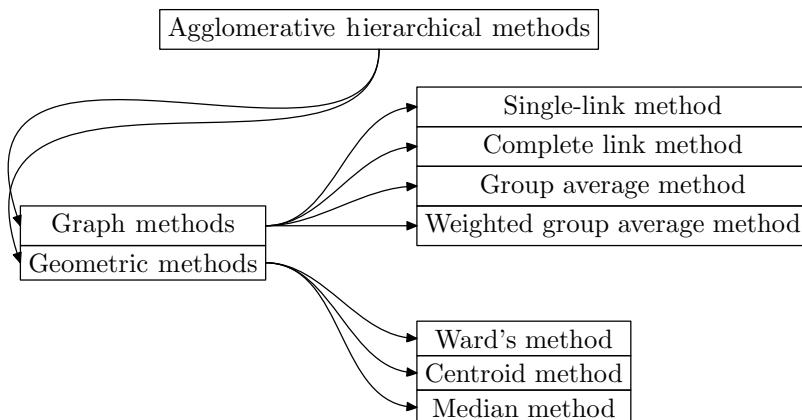


Figure 7.8. Some commonly used hierarchical methods.

Table 7.4. The cluster centers agglomerated from two clusters and the dissimilarities between two cluster centers for geometric hierarchical methods, where $\mu(C)$ denotes the center of cluster C .

Hierarchical Method	$\mu(C_i \cup C_j)$	Dissimilarity between C_i and C_j
Median	$\frac{\mu(C_i) + \mu(C_j)}{2}$	$\ \mu(C_i) - \mu(C_j)\ ^2$
Centroid	$\frac{ C_i \mu(C_i) + C_j \mu(C_j)}{ C_i + C_j }$	$\ \mu(C_i) - \mu(C_j)\ ^2$
Ward's	$\frac{ C_i \mu(C_i) + C_j \mu(C_j)}{ C_i + C_j }$	$\frac{ C_i C_j }{ C_i + C_j } \ \mu(C_i) - \mu(C_j)\ ^2$

$O(n^2)$ storage is required for a dissimilarity matrix, where n is the number of data points. Also for geometric hierarchical methods, the representative point of a cluster can be derived directly from that of the two clusters that form the cluster (see Table 7.4).

In agglomerative hierarchical clustering algorithms, the Lance-Williams formula (cf. Section 6.8) is used to calculate the dissimilarity between a cluster and a cluster formed by merging two other clusters. The single-link and complete link hierarchical clustering algorithms induce a metric on the data known as the ultrametric (Johnson, 1967). But the hierarchical structures produced by other clustering algorithms that use the Lance-Williams recurrence formula may violate the ultrametric inequality (Milligan, 1979).

The distances $D(C_k, C_i \cup C_j)$ are said to increase monotonically if $D(C_k, C_i \cup C_j) \geq D(C_i, C_j)$ at each level in the hierarchy. If an algorithm produces a monotonic hierarchy, then the algorithm induces a type of distance metric known as the ultrametric (Milligan, 1979). The centroid method and median method are examples of hierarchical algorithms that do not produce monotonic hierarchies.

Milligan (1979) has shown that the hierarchical clustering strategy $(\alpha_1, \alpha_2, \beta, \gamma)$ based on the Lance-Williams recurrence formula (Lance and Williams, 1967b) is monotonic, i.e.,

$$D(C_k, C_i \cup C_j) \geq D(C_i, C_j) \quad \forall i, j, k,$$

if the following conditions are satisfied:

1. $\gamma \geq 0 \vee (\gamma < 0 \wedge |\gamma| \leq \alpha_1, \alpha_2)$,
2. $\min\{\alpha_1, \alpha_2\} \geq 0$,
3. $\alpha_1 + \alpha_2 + \beta \geq 1$.

Also, Batagelj (1981) gives a necessary and sufficient condition for the hierarchical clustering strategy $(\alpha_1, \alpha_2, \beta, \gamma)$ to be monotonic:

1. $\gamma \geq -\min\{\alpha_1, \alpha_2\}$,
2. $\alpha_1 + \alpha_2 \geq 0$,
3. $\alpha_1 + \alpha_2 + \beta \geq 1$.

7.2.1 The Single-link Method

The single-link method is one of the simplest hierarchical clustering methods. It was first introduced by Florek et al. (1951) and then independently by McQuitty (1957) and Sneath (1957). The single-link method is also known by other names, such as the nearest neighbor method, the minimum method, and the connectedness method (Rohlf, 1982). The single-link method is invariant under monotone transformations (such as the logarithmic transformation) of the original data (Johnson, 1967).

It employs the nearest neighbor distance (cf. Section 6.8) to measure the dissimilarity between two groups. Let C_i , C_j , and C_k be three groups of data points. Then the distance between C_k and $C_i \cup C_j$ can be obtained from the Lance-Williams formula as follows:

$$\begin{aligned} & D(C_k, C_i \cup C_j) \\ &= \frac{1}{2} D(C_k, C_i) + \frac{1}{2} D(C_k, C_j) - \frac{1}{2} |D(C_k, C_i) - D(C_k, C_j)| \\ &= \min\{D(C_k, C_i), D(C_k, C_j)\}, \end{aligned} \quad (7.4)$$

where $D(\cdot, \cdot)$ is a distance between two clusters.

From equation (7.4), it is not difficult to verify that

$$D(C, C') = \min_{\mathbf{x} \in C, \mathbf{y} \in C'} d(\mathbf{x}, \mathbf{y}),$$

where C and C' are two nonempty, nonoverlapping clusters and $d(\cdot, \cdot)$ is the distance function by which the dissimilarity matrix is computed.

Rohlf (1982) has classified single-link algorithms into five different types:

1. connectedness algorithms,
2. algorithms based on an ultrametric transformation,
3. probability density estimation algorithms,
4. agglomerative algorithms,
5. algorithms based on the minimum spanning tree.

The connectedness algorithms are based on graph theory. In a connectedness algorithm, the data points are represented as vertices in a graph: a pair (i, j) of vertices are connected with an edge if and only if the distance between data points i and j $d_{ij} \leq \Delta$. The single-link clusters at level Δ correspond to the connected subgraphs of the graph. The connectedness algorithms require a considerable amount of computational effort. van Groenewoud and Ihm (1974) presented such an algorithm, whose total time complexity is $O(n^5)$, where n is the size of the data set.

More single-link algorithms will be presented in later sections of this chapter. What follows is a simple example that illustrates the idea of the single-link algorithm.

For the data set given in Figure 7.9, for example, the dissimilarity matrix computed using the Euclidean distance is described in Table 7.5. If a single-link hierarchical clustering algorithm is applied to this data set, then \mathbf{x}_1 and \mathbf{x}_2 will be agglomerated to form a big cluster

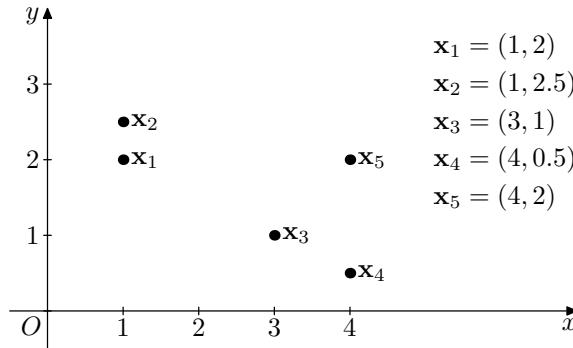


Figure 7.9. A two-dimensional data set with five data points.

Table 7.5. The dissimilarity matrix of the data set given in Figure 7.9. The entry (i, j) in the matrix is the Euclidean distance between \mathbf{x}_i and \mathbf{x}_j .

	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5
\mathbf{x}_1	0	0.5	2.24	3.35	3
\mathbf{x}_2	0.5	0	2.5	3.61	3.04
\mathbf{x}_3	2.24	2.5	0	1.12	1.41
\mathbf{x}_4	3.35	3.61	1.12	0	1.5
\mathbf{x}_5	3	3.04	1.41	1.5	0

at the first stage of the algorithm, since they have the least distance in the dissimilarity matrix. The distance between $\{\mathbf{x}_1, \mathbf{x}_2\}$ and \mathbf{x}_3 , \mathbf{x}_4 , and \mathbf{x}_5 now becomes

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3) = \min\{d(\mathbf{x}_1, \mathbf{x}_3), d(\mathbf{x}_2, \mathbf{x}_3)\} = 2.24,$$

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4) = \min\{d(\mathbf{x}_1, \mathbf{x}_4), d(\mathbf{x}_2, \mathbf{x}_4)\} = 3.35,$$

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_5) = \min\{d(\mathbf{x}_1, \mathbf{x}_5), d(\mathbf{x}_2, \mathbf{x}_5)\} = 3,$$

which can also be obtained from the formula given in (7.4). After \mathbf{x}_1 and \mathbf{x}_2 are merged, the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.24	3.35	3
\mathbf{x}_3	2.24	0	1.12	1.41
\mathbf{x}_4	3.35	1.12	0	1.5
\mathbf{x}_5	3	1.41	1.5	0

At the second stage of the algorithm, \mathbf{x}_3 and \mathbf{x}_4 will be merged, since they have the least distance. Then the distances between the group $\{\mathbf{x}_3, \mathbf{x}_4\}$ and the remaining groups

become

$$\begin{aligned}
 & D(\{\mathbf{x}_3, \mathbf{x}_4\}, \{\mathbf{x}_1, \mathbf{x}_2\}) \\
 &= \min\{d(\mathbf{x}_1, \mathbf{x}_3), d(\mathbf{x}_2, \mathbf{x}_3), d(\mathbf{x}_1, \mathbf{x}_4), d(\mathbf{x}_2, \mathbf{x}_4)\} \\
 &= \min\{D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3), D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4)\} = 2.24
 \end{aligned}$$

and

$$D(\{\mathbf{x}_3, \mathbf{x}_4\}, \mathbf{x}_5) = \min\{d(\mathbf{x}_3, \mathbf{x}_5), d(\mathbf{x}_4, \mathbf{x}_5)\} = 1.41.$$

After \mathbf{x}_3 and \mathbf{x}_4 are merged, the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4\}$	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.24	3
$\{\mathbf{x}_3, \mathbf{x}_4\}$	2.24	0	1.41
\mathbf{x}_5	3	1.41	0

At the third stage of the algorithm, $\{\mathbf{x}_3, \mathbf{x}_4\}$ and \mathbf{x}_5 will be merged. The dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.24
$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$	2.24	0

At the fourth stage, all the data points are merged into a single cluster. The dendrogram of this clustering is shown in Figure 7.10.

7.2.2 The Complete Link Method

Unlike the single-link method, the complete link method uses the farthest neighbor distance (cf. Section 6.8) to measure the dissimilarity between two groups. The complete link

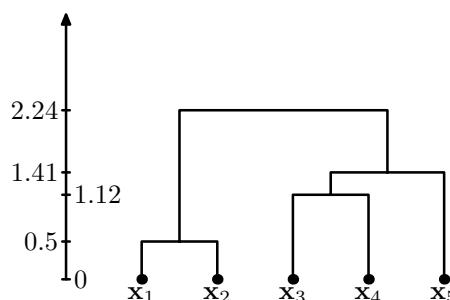


Figure 7.10. The dendrogram produced by applying the single-link method to the data set given in Figure 7.9.

method is also invariant under monotone transformations (Johnson, 1967). Let C_i , C_j , and C_k be three groups of data points. Then the distance between C_k and $C_i \cup C_j$ can be obtained from the Lance-Williams formula as follows:

$$\begin{aligned} & D(C_k, C_i \cup C_j) \\ &= \frac{1}{2}D(C_k, C_i) + \frac{1}{2}D(C_k, C_j) + \frac{1}{2}|D(C_k, C_i) - D(C_k, C_j)| \\ &= \max\{D(C_k, C_i), D(C_k, C_j)\}, \end{aligned} \quad (7.5)$$

where $D(\cdot, \cdot)$ is a distance between two clusters.

The distance defined in equation (7.5) has the following property:

$$D(C, C') = \max_{\mathbf{x} \in C, \mathbf{y} \in C'} d(\mathbf{x}, \mathbf{y}),$$

where C and C' are two nonempty, nonoverlapping clusters and $d(\cdot, \cdot)$ is the distance function by which the dissimilarity matrix is computed.

Applying the complete link method to the dissimilarity matrix given in Table 7.5, at the first stage we merge \mathbf{x}_1 and \mathbf{x}_2 . The distances between the group $\{\mathbf{x}_1, \mathbf{x}_2\}$ and the remaining three points are updated as

$$\begin{aligned} D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3) &= \max\{d(\mathbf{x}_1, \mathbf{x}_3), d(\mathbf{x}_2, \mathbf{x}_3)\} = 2.5, \\ D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4) &= \max\{d(\mathbf{x}_1, \mathbf{x}_4), d(\mathbf{x}_2, \mathbf{x}_4)\} = 3.61, \\ D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_5) &= \max\{d(\mathbf{x}_1, \mathbf{x}_5), d(\mathbf{x}_2, \mathbf{x}_5)\} = 3.04, \end{aligned}$$

which can also be obtained from the formula given in equation (7.5). After \mathbf{x}_1 and \mathbf{x}_2 are merged at the first stage of the algorithm, the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.5	3.61	3.04
\mathbf{x}_3	2.5	0	1.12	1.41
\mathbf{x}_4	3.61	1.12	0	1.5
\mathbf{x}_5	3.04	1.41	1.5	0

Again, at the second stage of the algorithm, \mathbf{x}_3 and \mathbf{x}_4 will be merged, since they have the least distance between them. After \mathbf{x}_3 and \mathbf{x}_4 are merged, the distances between the group $\{\mathbf{x}_3, \mathbf{x}_4\}$ and the remaining groups are updated as

$$\begin{aligned} & D(\{\mathbf{x}_3, \mathbf{x}_4\}, \{\mathbf{x}_1, \mathbf{x}_2\}) \\ &= \max\{d(\mathbf{x}_1, \mathbf{x}_3), d(\mathbf{x}_2, \mathbf{x}_3), d(\mathbf{x}_1, \mathbf{x}_4), d(\mathbf{x}_2, \mathbf{x}_4)\} \\ &= \max\{D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3), D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4)\} = 3.61 \end{aligned}$$

and

$$D(\{\mathbf{x}_3, \mathbf{x}_4\}, \mathbf{x}_5) = \max\{d(\mathbf{x}_3, \mathbf{x}_5), d(\mathbf{x}_4, \mathbf{x}_5)\} = 1.5.$$

After \mathbf{x}_3 and \mathbf{x}_4 are merged, the dissimilarity matrix becomes

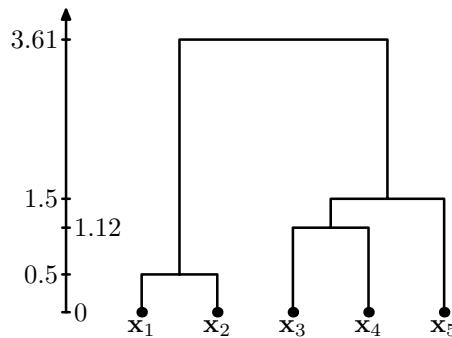


Figure 7.11. The dendrogram produced by applying the complete link method to the data set given in Figure 7.9.

	$\{x_1, x_2\}$	$\{x_3, x_4\}$	x_5
$\{x_1, x_2\}$	0	3.61	3.04
$\{x_3, x_4\}$	3.61	0	1.5
x_5	3.04	1.5	0

At the third stage of the algorithm, $\{x_3, x_4\}$ and x_5 must be merged, since they have the least distance. After x_4 and x_5 are merged, the distance between the two groups is

$$\begin{aligned}
 & D(\{x_1, x_2\}, \{x_3, x_4, x_5\}) \\
 &= \max\{d_{13}, d_{14}, d_{15}, d_{23}, d_{24}, d_{25}\} \\
 &= \max\{D(\{x_1, x_2\}, \{x_3, x_4\}), D(\{x_1, x_2\}, x_5)\} \\
 &= 3.61,
 \end{aligned}$$

where $d_{ij} = d(x_i, x_j)$ for $i = 1, 2$ and $j = 3, 4, 5$, and the dissimilarity matrix becomes

	$\{x_1, x_2\}$	$\{x_3, x_4, x_5\}$
$\{x_1, x_2\}$	0	3.61
$\{x_3, x_4, x_5\}$	3.61	0

The dendrogram produced by the complete link method is shown in Figure 7.11, which is the same as the dendrogram produced by the single-link method except for the heights.

7.2.3 The Group Average Method

The group average method is also referred as UPGMA, which stands for “unweighted pair group method using arithmetic averages” (Jain and Dubes, 1988). In the group average method, the distance between two groups is defined as the average of the distances between all possible pairs of data points that are made up of one data point from each group. Let C_i , C_j , and C_k be three groups of data points. Then the distance between C_k and $C_i \cup C_j$ can

be obtained from the Lance-Williams formula as follows:

$$\begin{aligned} & D(C_k, C_i \cup C_j) \\ &= \frac{|C_i|}{|C_i| + |C_j|} D(C_k, C_i) + \frac{|C_j|}{|C_i| + |C_j|} D(C_k, C_j), \end{aligned} \quad (7.6)$$

where $D(\cdot, \cdot)$ is a distance between two clusters.

Let C and C' be two nonempty, nonoverlapping clusters. Then in the group average method, we have

$$D(C, C') = \frac{1}{|C||C'|} \sum_{\mathbf{x} \in C, \mathbf{y} \in C'} d(\mathbf{x}, \mathbf{y}), \quad (7.7)$$

where $d(\cdot, \cdot)$ is the distance function by which the dissimilarity matrix is computed.

In fact, let C_1 , C_2 , and C_3 be three nonempty, mutually nonoverlapping clusters, and assume

$$D(C_i, C_j) = \frac{1}{n_i n_j} \Sigma(C_i, C_j), \quad 1 \leq i < j \leq 3, \quad (7.8)$$

where $n_i = |C_i|$, $n_j = |C_j|$, and $\Sigma(C_i, C_j)$ is the total between-clusters distance of C_i and C_j , that is,

$$\Sigma(C_i, C_j) = \sum_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}).$$

Now from equations (7.6) and (7.8), we have

$$\begin{aligned} & D(C_1, C_2 \cup C_3) \\ &= \frac{n_2}{n_2 + n_3} D(C_1, C_2) + \frac{n_3}{n_2 + n_3} D(C_1, C_3) \\ &= \frac{n_2}{n_2 + n_3} \cdot \frac{1}{n_1 n_2} \Sigma(C_1, C_2) + \frac{n_3}{n_2 + n_3} \cdot \frac{1}{n_1 n_3} \Sigma(C_1, C_3) \\ &= \frac{1}{n_1(n_2 + n_3)} \Sigma(C_1, C_2 \cup C_3), \end{aligned}$$

since $\Sigma(C_1, C_2) + \Sigma(C_1, C_3) = \Sigma(C_1, C_2 \cup C_3)$. This verifies equation (7.7).

Applying the group average method to the data set given in Figure 7.9, we note that again the first stage is to merge \mathbf{x}_1 and \mathbf{x}_2 . After \mathbf{x}_1 and \mathbf{x}_2 are merged, the distances between $\{\mathbf{x}_1, \mathbf{x}_2\}$ and the remaining three data points become

$$\begin{aligned} D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3) &= \frac{1}{2} d(\mathbf{x}_1, \mathbf{x}_3) + \frac{1}{2} d(\mathbf{x}_2, \mathbf{x}_3) = 2.37, \\ D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4) &= \frac{1}{2} d(\mathbf{x}_1, \mathbf{x}_4) + \frac{1}{2} d(\mathbf{x}_2, \mathbf{x}_4) = 3.48, \\ D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_5) &= \frac{1}{2} d(\mathbf{x}_1, \mathbf{x}_5) + \frac{1}{2} d(\mathbf{x}_2, \mathbf{x}_5) = 3.02, \end{aligned}$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.37	3.48	3.02
\mathbf{x}_3	2.37	0	1.12	1.41
\mathbf{x}_4	3.48	1.12	0	1.5
\mathbf{x}_5	3.02	1.41	1.5	0

Again, at the second stage of the algorithm, \mathbf{x}_4 and \mathbf{x}_3 will be merged. The distances between $\{\mathbf{x}_3, \mathbf{x}_4\}$ and the other clusters become

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4\}) = \frac{1}{2}D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4) + \frac{1}{2}D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3) = 2.93,$$

$$D(\{\mathbf{x}_3, \mathbf{x}_4\}, \mathbf{x}_5) = \frac{1}{2}d(\mathbf{x}_3, \mathbf{x}_5) + \frac{1}{2}d(\mathbf{x}_4, \mathbf{x}_5) = 1.46.$$

After \mathbf{x}_3 and \mathbf{x}_4 are merged into one cluster, the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4\}$	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.93	3.02
$\{\mathbf{x}_3, \mathbf{x}_4\}$	2.93	0	1.46
\mathbf{x}_5	3.02	1.46	0

At the third stage of the algorithm, $\{\mathbf{x}_3, \mathbf{x}_4\}$ and \mathbf{x}_5 must be merged, since they have the least distance. Then the distance between $\{\mathbf{x}_1, \mathbf{x}_2\}$ and $\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$ becomes

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}) \\ = \frac{2}{3}D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4\}) + \frac{1}{3}D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_5) \\ = 2.96.$$

Hence, the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.96
$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$	2.96	0

The distances can also be calculated by equation (7.7). In the last stage, for example, the distance between $\{\mathbf{x}_1, \mathbf{x}_2\}$ and $\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$ can be computed as

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}) \\ = \frac{1}{6}(d_{13} + d_{14} + d_{15} + d_{23} + d_{24} + d_{25}) = 2.96,$$

where $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$, i.e., the (i, j) th entry of the dissimilarity matrix.

The dendrogram of this clustering is shown in Figure 7.12.

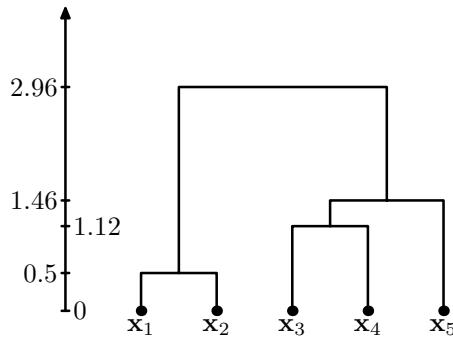


Figure 7.12. The dendrogram produced by applying the group average method to the data set given in Figure 7.9.

7.2.4 The Weighted Group Average Method

The weighted group average method is also referred to as the “weighted pair group method using arithmetic average” (Jain and Dubes, 1988). Using the Lance-Williams formula, the distance between clusters is

$$D(C_k, C_i \cup C_j) = \frac{1}{2} D(C_k, C_i) + \frac{1}{2} D(C_k, C_j),$$

where C_k , C_i , and C_j are three clusters in one level of clustering.

Applying the weighted group average method to the five-point data set given in Figure 7.9, the first stage is the same as in the other methods, i.e., we merge x_1 and x_2 . After x_1 and x_2 are merged, the distances between clusters are updated as

$$\begin{aligned} D(\{x_1, x_2\}, x_3) &= \frac{1}{2}(d(x_1, x_3) + d(x_2, x_3)) = 2.37, \\ D(\{x_1, x_2\}, x_4) &= \frac{1}{2}(d(x_1, x_4) + d(x_2, x_4)) = 3.48, \\ D(\{x_1, x_2\}, x_5) &= \frac{1}{2}(d(x_1, x_5) + d(x_2, x_5)) = 3.02, \end{aligned}$$

and the dissimilarity matrix becomes

	$\{x_1, x_2\}$	x_3	x_4	x_5
$\{x_1, x_2\}$	0	2.37	3.48	3.02
x_3	2.37	0	1.12	1.41
x_4	3.48	1.12	0	1.5
x_5	3.02	1.41	1.5	0

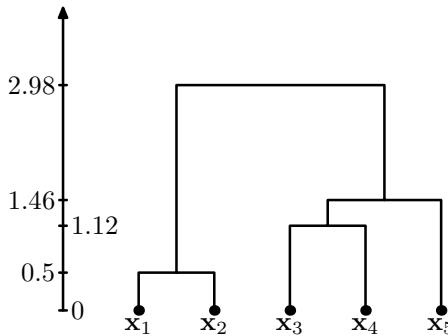


Figure 7.13. The dendrogram produced by applying the weighted group average method to the data set given in Figure 7.9.

At the second stage of this method, \mathbf{x}_3 and \mathbf{x}_4 will be merged. After \mathbf{x}_3 and \mathbf{x}_4 are merged, the distances between clusters are updated as

$$D(\{\mathbf{x}_3, \mathbf{x}_4\}, \{\mathbf{x}_1, \mathbf{x}_2\}) = \frac{1}{2}(2.37 + 3.48) = 2.93,$$

$$D(\{\mathbf{x}_3, \mathbf{x}_4\}, \mathbf{x}_5) = \frac{1}{2}(1.41 + 1.5) = 1.46,$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4\}$	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.93	3.02
$\{\mathbf{x}_3, \mathbf{x}_4\}$	2.93	0	1.46
\mathbf{x}_5	3.02	1.46	0

Clusters $\{\mathbf{x}_3, \mathbf{x}_4\}$ and \mathbf{x}_5 will be merged at the third stage of this method. The distance is updated as

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}) = \frac{1}{2}(2.93 + 3.02) = 2.98,$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.98
$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$	2.98	0

The whole process of this clustering can be represented by the dendrogram shown in Figure 7.13.

7.2.5 The Centroid Method

The centroid method is also referred to as the “unweighted pair group method using centroids” (Jain and Dubes, 1988). With the centroid method, the new distances between

clusters can be calculated by the following Lance-Williams formula:

$$\begin{aligned}
 & D(C_k, C_i \cup C_j) \\
 &= \frac{|C_i|}{|C_i| + |C_j|} D(C_k, C_i) + \frac{|C_j|}{|C_i| + |C_j|} D(C_k, C_j) \\
 &\quad - \frac{|C_i||C_j|}{(|C_i| + |C_j|)^2} D(C_i, C_j),
 \end{aligned} \tag{7.9}$$

where C_k , C_i , and C_j are three clusters in one level of clustering.

Let C and C' be any two nonoverlapping clusters, i.e., $C \cap C' = \emptyset$. Then it follows from equation (7.9) that

$$\begin{aligned}
 & D(C, C') \\
 &= \frac{1}{|C||C'|} \sum_{\mathbf{x} \in C, \mathbf{y} \in C'} d(\mathbf{x}, \mathbf{y}) - \frac{1}{2|C|^2} \sum_{\mathbf{x}, \mathbf{y} \in C} d(\mathbf{x}, \mathbf{y}) \\
 &\quad - \frac{1}{2|C'|^2} \sum_{\mathbf{x}, \mathbf{y} \in C'} d(\mathbf{x}, \mathbf{y}),
 \end{aligned} \tag{7.10}$$

where $d(\cdot, \cdot)$ is the distance function by which the dissimilarity matrix is calculated.

In fact, let C_1 , C_2 , and C_3 be three nonempty, mutually nonoverlapping clusters, and assume

$$D(C_i, C_j) = \frac{1}{n_i n_j} \Sigma(C_i, C_j) - \frac{1}{2n_i^2} \Sigma(C_i) - \frac{1}{2n_j^2} \Sigma(C_j) \tag{7.11}$$

for $1 \leq i < j \leq 3$, where $n_i = |C_i|$, $n_j = |C_j|$, $\Sigma(C_i, C_j)$ is the total between-clusters distance of C_i and C_j , that is,

$$\Sigma(C_i, C_j) = \sum_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}),$$

$\Sigma(C_i)$ is the total within-cluster distance of C_i , that is,

$$\Sigma(C_i) = \sum_{\mathbf{x}, \mathbf{y} \in C_i} d(\mathbf{x}, \mathbf{y}),$$

and $\Sigma(C_j)$ is defined similarly.

We should prove that

$$\begin{aligned}
 & D(C_1, C_2 \cup C_3) \\
 &= \frac{1}{n_1(n_2 + n_3)} \Sigma(C_1, C_2 \cup C_3) - \frac{1}{2n_1^2} \Sigma(C_1) \\
 &\quad - \frac{1}{2(n_2 + n_3)^2} \Sigma(C_2 \cup C_3).
 \end{aligned} \tag{7.12}$$

From equation (7.9), we have

$$\begin{aligned} & D(C_1, C_2 \cup C_3) \\ &= \frac{n_2}{n_2 + n_3} D(C_1, C_2) + \frac{n_3}{n_2 + n_3} D(C_1, C_3) \\ &\quad - \frac{n_2 n_3}{(n_2 + n_3)^2} D(C_2, C_3), \end{aligned}$$

Substituting equation (7.11) into the above equation and performing some simple manipulations, we have

$$\begin{aligned} & D(C_1, C_2 \cup C_3) \\ &= \frac{n_2}{n_2 + n_3} \left(\frac{1}{n_1 n_2} \Sigma(C_1, C_2) - \frac{1}{2n_1^2} \Sigma(C_1) - \frac{1}{2n_2^2} \Sigma(C_2) \right) \\ &\quad + \frac{n_3}{n_2 + n_3} \left(\frac{1}{n_1 n_3} \Sigma(C_1, C_3) - \frac{1}{2n_1^2} \Sigma(C_1) - \frac{1}{2n_3^2} \Sigma(C_3) \right) \\ &\quad - \frac{n_2 n_3}{(n_2 + n_3)^2} \left(\frac{1}{n_2 n_3} \Sigma(C_2, C_3) - \frac{1}{2n_2^2} \Sigma(C_2) - \frac{1}{2n_3^2} \Sigma(C_3) \right) \\ &= \frac{1}{n_1(n_2 + n_3)} \Sigma(C_1, C_2 \cup C_3) - \frac{1}{2n_1^2} \Sigma(C_1) \\ &\quad - \frac{1}{2(n_2 + n_3)^2} [\Sigma(C_2) + \Sigma(C_3) + 2\Sigma(C_2, C_3)] \\ &= \frac{1}{n_1(n_2 + n_3)} \Sigma(C_1, C_2 \cup C_3) - \frac{1}{2n_1^2} \Sigma(C_1) - \frac{1}{2(n_2 + n_3)^2} \Sigma(C_2 \cup C_3). \end{aligned}$$

In the above, we used

$$\Sigma(C_1, C_2) + \Sigma(C_1, C_3) = \Sigma(C_1, C_2 \cup C_3)$$

and

$$\Sigma(C_2) + \Sigma(C_3) + 2\Sigma(C_2, C_3) = \Sigma(C_2 \cup C_3).$$

In particular, if we take $d(\cdot, \cdot)$ in equation (7.10) as the squared Euclidean distance, then the distance $D(C, C')$ is exactly the squared Euclidean distance between the centroids of C and C' .

Actually, if $d(\cdot, \cdot)$ in equation (7.10) is the squared Euclidean distance, then we have

$$\begin{aligned} & D(C, C') \\ &= \frac{1}{|C||C'|} \sum_{\mathbf{x} \in C, \mathbf{y} \in C'} (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T - \frac{1}{2|C|^2} \sum_{\mathbf{x}, \mathbf{y} \in C} (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T \\ &\quad - \frac{1}{2|C'|^2} \sum_{\mathbf{x}, \mathbf{y} \in C'} (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x}\mathbf{x}^T - \frac{2}{|C||C'|} \sum_{\mathbf{x} \in C, \mathbf{y} \in C'} \mathbf{x}\mathbf{y}^T + \frac{1}{|C'|} \sum_{\mathbf{x} \in C'} \mathbf{x}\mathbf{x}^T \\
&\quad - \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x}\mathbf{x}^T + \frac{1}{|C|^2} \sum_{\mathbf{x}, \mathbf{y} \in C} \mathbf{x}\mathbf{y}^T - \frac{1}{|C'|} \sum_{\mathbf{y} \in C'} \mathbf{y}\mathbf{y}^T \\
&\quad + \frac{1}{|C'|^2} \sum_{\mathbf{x}, \mathbf{y} \in C'} \mathbf{x}\mathbf{y}^T \\
&= \frac{1}{|C|^2} \sum_{\mathbf{x}, \mathbf{y} \in C} \mathbf{x}\mathbf{y}^T + \frac{1}{|C'|^2} \sum_{\mathbf{x}, \mathbf{y} \in C'} \mathbf{x}\mathbf{y}^T - \frac{2}{|C||C'|} \sum_{\mathbf{x} \in C, \mathbf{y} \in C'} \mathbf{x}\mathbf{y}^T \\
&= \left(\frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x} - \frac{1}{|C'|} \sum_{\mathbf{x} \in C'} \mathbf{x} \right) \left(\frac{1}{|C|} \sum_{\mathbf{y} \in C} \mathbf{y} - \frac{1}{|C'|} \sum_{\mathbf{y} \in C'} \mathbf{y} \right)^T,
\end{aligned}$$

since $(\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T = \mathbf{x}\mathbf{x}^T - 2\mathbf{x}\mathbf{y}^T + \mathbf{y}\mathbf{y}^T$ and $\mathbf{x}\mathbf{y}^T = \mathbf{y}\mathbf{x}^T$.

Equation (7.10) provides another way to compute the distances between new clusters and old ones. Applying the centroid method to the data set given in Figure 7.9, the first stage is still the same as in other methods, i.e., \mathbf{x}_1 and \mathbf{x}_2 will be merged. After \mathbf{x}_1 and \mathbf{x}_2 are merged, the distances are updated as

$$\begin{aligned}
D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3) &= \frac{1}{2}(d(\mathbf{x}_1, \mathbf{x}_3) + d(\mathbf{x}_2, \mathbf{x}_3)) - \frac{1}{4}d(\mathbf{x}_1, \mathbf{x}_2) = 2.245, \\
D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4) &= \frac{1}{2}(d(\mathbf{x}_1, \mathbf{x}_4) + d(\mathbf{x}_2, \mathbf{x}_4)) - \frac{1}{4}d(\mathbf{x}_1, \mathbf{x}_2) = 3.355, \\
D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_5) &= \frac{1}{2}(d(\mathbf{x}_1, \mathbf{x}_5) + d(\mathbf{x}_2, \mathbf{x}_5)) - \frac{1}{4}d(\mathbf{x}_1, \mathbf{x}_2) = 2.895,
\end{aligned}$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.245	3.355	2.895
\mathbf{x}_3	2.245	0	1.12	1.41
\mathbf{x}_4	3.355	1.12	0	1.5
\mathbf{x}_5	2.895	1.41	1.5	0

At the second stage, \mathbf{x}_3 and \mathbf{x}_4 will be merged, and the distances are updated as

$$\begin{aligned}
D(\{\mathbf{x}_3, \mathbf{x}_4\}, \{\mathbf{x}_1, \mathbf{x}_2\}) &= \frac{1}{2}(2.245 + 3.355) - \frac{1}{4}(1.12) = 2.52, \\
D(\{\mathbf{x}_3, \mathbf{x}_4\}, \mathbf{x}_5) &= \frac{1}{2}(1.41 + 1.5) - \frac{1}{4}(1.12) = 1.175,
\end{aligned}$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4\}$	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.52	2.895
$\{\mathbf{x}_3, \mathbf{x}_4\}$	2.52	0	1.175
\mathbf{x}_5	2.895	1.175	0

At the third stage, \mathbf{x}_5 will be merged with $\{\mathbf{x}_3, \mathbf{x}_4\}$, and the distance is updated as

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}) = \frac{2}{3}(2.52) + \frac{1}{3}(2.895) - \frac{2}{9}(1.175) = 2.384,$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	2.39
$\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$	2.39	0

The distances can also be updated by equation (7.10). For example, the distance between $\{\mathbf{x}_1, \mathbf{x}_2\}$ and $\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$ in the last stage can be computed as

$$\begin{aligned} & D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}) \\ &= \frac{1}{6}(d_{13} + d_{14} + d_{15} + d_{23} + d_{24} + d_{25}) - \frac{1}{8}(2d_{12}) \\ &\quad - \frac{1}{18}(2d_{34} + 2d_{35} + 2d_{45}) \\ &= \frac{1}{6}(2.24 + 3.35 + 3 + 2.5 + 3.61 + 3.04) - \frac{1}{4}(0.5) \\ &\quad - \frac{1}{9}(1.12 + 1.41 + 1.5) \\ &= 2.957 - 0.125 - 0.448 \\ &= 2.384, \end{aligned}$$

where $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$.

The whole process of this clustering can be represented by the dendrogram shown in Figure 7.14.

7.2.6 The Median Method

The median method is also referred to as the “weighted pair group method using centroids” (Jain and Dubes, 1988) or the “weighted centroid” method. It was first proposed by Gower (1967) in order to alleviate some disadvantages of the centroid method. In the centroid method, if the sizes of the two groups to be merged are quite different, then the centroid of the new group will be very close to that of the larger group and may remain within that group (Everitt, 1993). In the median method, the centroid of a new group is independent of the size of the groups that form the new group.

A disadvantage of the median method is that it is not suitable for measures such as correlation coefficients, since interpretation in a geometrical sense is no longer possible (Lance and Williams, 1967a).

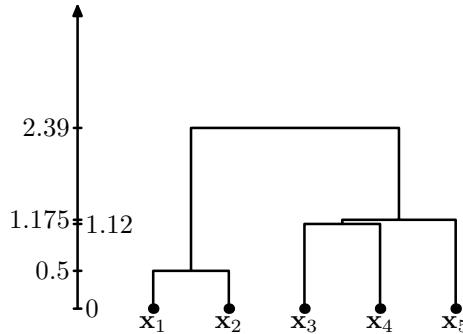


Figure 7.14. The dendrogram produced by applying the centroid method to the data set given in Figure 7.9.

In the median method, the distances between newly formed groups and other groups are computed as

$$D(C_k, C_i \cup C_j) = \frac{1}{2}D(C_k, C_i) + \frac{1}{2}D(C_k, C_j) - \frac{1}{4}D(C_i, C_j), \quad (7.13)$$

where C_k , C_i , and C_j are three clusters in one level of clustering.

We now take the data set given in Figure 7.9 as an example to illustrate the median method. The first stage of the median method is still the same as in other methods. After x_1 and x_2 are merged, the distances are updated as

$$\begin{aligned} D(\{x_1, x_2\}, x_3) &= \frac{1}{2}(d(x_1, x_3) + d(x_2, x_3)) - \frac{1}{4}d(x_1, x_2) = 2.245, \\ D(\{x_1, x_2\}, x_4) &= \frac{1}{2}(d(x_1, x_4) + d(x_2, x_4)) - \frac{1}{4}d(x_1, x_2) = 3.355, \\ D(\{x_1, x_2\}, x_5) &= \frac{1}{2}(d(x_1, x_5) + d(x_2, x_5)) - \frac{1}{4}d(x_1, x_2) = 2.895, \end{aligned}$$

and the dissimilarity matrix becomes

	$\{x_1, x_2\}$	x_3	x_4	x_5
$\{x_1, x_2\}$	0	2.245	3.355	2.895
x_3	2.245	0	1.12	1.41
x_4	3.355	1.12	0	1.5
x_5	2.895	1.41	1.5	0

At the second stage of this method, x_3 and x_4 will be merged. After x_3 and x_4 are merged, the distances between clusters are updated as

$$\begin{aligned} D(\{x_3, x_4\}, \{x_1, x_2\}) &= \frac{1}{2}(2.245 + 3.355) - \frac{1}{4}(1.12) = 2.52, \\ D(\{x_3, x_4\}, x_5) &= \frac{1}{2}(1.41 + 1.5) - \frac{1}{4}(1.12) = 1.175, \end{aligned}$$

and the dissimilarity matrix becomes

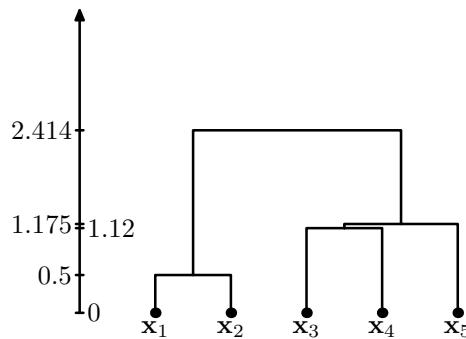


Figure 7.15. The dendrogram produced by applying the median method to the data set given in Figure 7.9.

	$\{x_1, x_2\}$	$\{x_3, x_4\}$	x_5
$\{x_1, x_2\}$	0	2.52	2.895
$\{x_3, x_4\}$	2.52	0	1.175
x_5	2.895	1.175	0

Clusters $\{x_3, x_4\}$ and x_5 will be merged at the third stage of this method. The distance is updated as

$$D(\{x_1, x_2\}, \{x_3, x_4, x_5\}) = \frac{1}{2}(2.52 + 2.895) - \frac{1}{4}(1.175) = 2.414,$$

and the dissimilarity matrix becomes

	$\{x_1, x_2\}$	$\{x_3, x_4, x_5\}$
$\{x_1, x_2\}$	0	2.414
$\{x_3, x_4, x_5\}$	2.414	0

The whole process of this clustering can be represented by the dendrogram shown in Figure 7.15.

7.2.7 Ward's Method

Ward Jr. (1963) and Ward Jr. and Hook (1963) proposed a hierarchical clustering procedure seeking to form the partitions P_n, P_{n-1}, \dots, P_1 in a manner that minimizes the loss of information associated with each merging. Usually, the information loss is quantified in terms of an error sum of squares (ESS) criterion, so Ward's method is often referred to as the “minimum variance” method.

Given a group of data points C , the ESS associated with C is given by

$$\text{ESS}(C) = \sum_{\mathbf{x} \in C} (\mathbf{x} - \mu(C))(\mathbf{x} - \mu(C))^T,$$

or

$$\begin{aligned}\text{ESS}(C) &= \sum_{\mathbf{x} \in C} \mathbf{x}\mathbf{x}^T - \frac{1}{|C|} \left(\sum_{\mathbf{x} \in C} \mathbf{x} \right) \left(\sum_{\mathbf{x} \in C} \mathbf{x} \right)^T \\ &= \sum_{\mathbf{x} \in C} \mathbf{x}\mathbf{x}^T - |C|\mu(C)\mu(C)^T,\end{aligned}\quad (7.14)$$

where $\mu(C)$ is the mean of C , that is,

$$\mu(C) = \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x}.$$

Suppose there are k groups C_1, C_2, \dots, C_k in one level of the clustering. Then the information loss is represented by the sum of ESSs given by

$$\text{ESS} = \sum_{i=1}^k \text{ESS}(C_i),$$

which is the total within-group ESS.

At each step of Ward's method, the union of every possible pair of groups is considered and two groups whose fusion results in the minimum increase in loss of information are merged.

If the squared Euclidean distance is used to compute the dissimilarity matrix, then the dissimilarity matrix can be updated by the Lance-Williams formula during the process of clustering as follows (Wishart, 1969):

$$\begin{aligned}D(C_k, C_i \cup C_j) \\ = \frac{|C_k| + |C_i|}{\Sigma_{ijk}} D(C_k, C_i) + \frac{|C_k| + |C_j|}{\Sigma_{ijk}} D(C_k, C_j) \\ - \frac{|C_k|}{\Sigma_{ijk}} D(C_i, C_j),\end{aligned}\quad (7.15)$$

where $\Sigma_{ijk} = |C_k| + |C_i| + |C_j|$.

To justify this, we suppose C_i and C_j are chosen to be merged and the resulting cluster is denoted by C_t , i.e., $C_t = C_i \cup C_j$. Then the increase in ESS is

$$\begin{aligned}\Delta \text{ESS}_{ij} &= \text{ESS}(C_t) - \text{ESS}(C_i) - \text{ESS}(C_j) \\ &= \left(\sum_{\mathbf{x} \in C_t} \mathbf{x}\mathbf{x}^T - |C_t|\mu_t\mu_t^T \right) - \left(\sum_{\mathbf{x} \in C_i} \mathbf{x}\mathbf{x}^T - |C_i|\mu_i\mu_i^T \right) \\ &\quad - \left(\sum_{\mathbf{x} \in C_j} \mathbf{x}\mathbf{x}^T - |C_j|\mu_j\mu_j^T \right) \\ &= |C_i|\mu_i\mu_i^T + |C_j|\mu_j\mu_j^T - |C_t|\mu_t\mu_t^T,\end{aligned}\quad (7.16)$$

where μ_t, μ_i , and μ_j are the means of clusters C_t, C_i , and C_j , respectively.

Noting that $|C_t|\mu_t = |C_i|\mu_i + |C_j|\mu_j$, squaring both sides of this equation gives

$$|C_t|^2\mu_t\mu_t^T = |C_i|^2\mu_i\mu_i^T + |C_j|^2\mu_j\mu_j^T + 2|C_i||C_j|\mu_i\mu_j^T,$$

or

$$\begin{aligned} |C_t|^2\mu_t\mu_t^T &= |C_i|^2\mu_i\mu_i^T + |C_j|^2\mu_j\mu_j^T + |C_i||C_j|(\mu_i\mu_i^T + \mu_j\mu_j^T) \\ &\quad - |C_i||C_j|(\mu_i - \mu_j)(\mu_i - \mu_j)^T \\ &= |C_i|(|C_i| + |C_j|)\mu_i\mu_i^T + |C_j|(|C_i| + |C_j|)\mu_j\mu_j^T \\ &\quad - |C_i||C_j|(\mu_i - \mu_j)(\mu_i - \mu_j)^T, \end{aligned} \quad (7.17)$$

since

$$2\mu_i\mu_j^T = \mu_i\mu_i^T + \mu_j\mu_j^T - (\mu_i - \mu_j)(\mu_i - \mu_j)^T.$$

Dividing both sides of equation (7.17) by $|C_t|$ and substituting $|C_t|\mu_t\mu_t^T$ into equation (7.16) give

$$\Delta\text{ESS}_{ij} = \frac{|C_i||C_j|}{|C_i| + |C_j|}(\mu_i - \mu_j)(\mu_i - \mu_j)^T. \quad (7.18)$$

Now considering the increase in ESS that would result from the potential fusion of groups C_k and C_t , from equation (7.18) we have

$$\Delta\text{ESS}_{kt} = \frac{|C_k||C_t|}{|C_k| + |C_t|}(\mu_k - \mu_t)(\mu_k - \mu_t)^T, \quad (7.19)$$

where $\mu_k = \mu(C_k)$ is the mean of group C_k .

Noting that $\mu_t = \frac{1}{|C_t|}(|C_i|\mu_i + |C_j|\mu_j)$ and $|C_t| = |C_i| + |C_j|$, using equation (7.17), we have

$$\begin{aligned} &(\mu_k - \mu_t)(\mu_k - \mu_t)^T \\ &= \frac{|C_i|}{|C_t|}(\mu_k - \mu_i)(\mu_k - \mu_i)^T + \frac{|C_j|}{|C_t|}(\mu_k - \mu_j)(\mu_k - \mu_j)^T \\ &\quad - \frac{|C_i||C_j|}{|C_t|^2}(\mu_i - \mu_j)(\mu_i - \mu_j)^T. \end{aligned}$$

Substituting the above equation into equation (7.19), and after simple manipulations, we get

$$\begin{aligned} \Delta\text{ESS}_{kt} &= \frac{|C_k||C_i|}{|C_k| + |C_t|}(\mu_k - \mu_i)(\mu_k - \mu_i)^T \\ &\quad + \frac{|C_k||C_j|}{|C_k| + |C_t|}(\mu_k - \mu_j)(\mu_k - \mu_j)^T \\ &\quad - \frac{|C_k||C_i||C_j|}{|C_k| + |C_t|}(\mu_i - \mu_j)(\mu_i - \mu_j)^T, \end{aligned}$$

and, using equation (7.18), we have

$$\begin{aligned} \Delta\text{ESS}_{kt} &= \frac{|C_k| + |C_i|}{|C_k| + |C_t|}\Delta\text{ESS}_{ki} + \frac{|C_k| + |C_j|}{|C_k| + |C_t|}\Delta\text{ESS}_{kj} \\ &\quad - \frac{|C_k|}{|C_k| + |C_t|}\Delta\text{ESS}_{ij}. \end{aligned} \quad (7.20)$$

Table 7.6. The dissimilarity matrix of the data set given in Figure 7.9, where the entry (i, j) in the matrix is the squared Euclidean distance between \mathbf{x}_i and \mathbf{x}_j .

	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5
\mathbf{x}_1	0	0.25	5	11.25	9
\mathbf{x}_2	0.25	0	6.25	13	9.25
\mathbf{x}_3	5	6.25	0	1.25	2
\mathbf{x}_4	11.25	13	1.25	0	2.25
\mathbf{x}_5	9	9.25	2	2.25	0

This proves equation (7.15). If we compute the dissimilarity matrix for a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ using the squared Euclidean distance, then the entry (i, j) of the dissimilarity matrix is

$$d_{ij}^2 = d(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T = \sum_{l=1}^d (x_{il} - x_{jl})^2,$$

where d is the dimensionality of the data set D .

If $C_i = \{\mathbf{x}_i\}$ and $C_j = \{\mathbf{x}_j\}$ in equation (7.18), then the increase in ESS that results from the fusion of \mathbf{x}_i and \mathbf{x}_j is

$$\Delta \text{ESS}_{ij} = \frac{1}{2} d_{ij}^2.$$

Since the objective of Ward's method is to find at each stage those two groups whose fusion gives the minimum increase in the total within-group ESS, the two points with minimum squared Euclidean distance will be merged at the first stage. Suppose \mathbf{x}_i and \mathbf{x}_j have minimum squared Euclidean distance. Then $C_i = \{\mathbf{x}_i\}$ and $C_j = \{\mathbf{x}_j\}$ will be merged. After C_i and C_j are merged, the distances between $C_i \cup C_j$ and other points must be updated.

Now let $C_k = \{\mathbf{x}_k\}$ be any other group. Then the increase in ESS that would result from the potential fusion of C_k and $C_i \cup C_j$ can be calculated from equation (7.20) as

$$\Delta \text{ESS}_{k(ij)} = \frac{2}{3} \frac{d_{ki}^2}{2} + \frac{2}{3} \frac{d_{kj}^2}{2} - \frac{1}{3} \frac{d_{ij}^2}{2}.$$

If we update the dissimilarity matrix using equation (7.15), then we have from the above equation that

$$\Delta \text{ESS}_{k(ij)} = \frac{1}{2} D(C_k, C_i \cup C_j).$$

Thus if we update the dissimilarity matrix using equation (7.15) during the process of clustering, then the two groups with minimum distance will be merged.

Taking the data set given in Figure 7.9 as an example, the dissimilarity matrix computed by the squared Euclidean distance is given in Table 7.6.

Initially, each single point forms a cluster and the total ESS is $\text{ESS}_0 = 0$. According to the above discussion, \mathbf{x}_1 and \mathbf{x}_2 will be merged at the first stage of Ward's method, and

the increase in ESS that results from the fusion of \mathbf{x}_1 and \mathbf{x}_2 is $\Delta\text{ESS}_{12} = \frac{1}{2}(0.25) = 0.125$; hence, the ESS becomes

$$\text{ESS}_1 = \text{ESS}_0 + \Delta\text{ESS}_{12} = 0.125.$$

Using equation (7.15), the distances are updated as

$$\begin{aligned} D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_3) &= \frac{2}{3}(d(\mathbf{x}_1, \mathbf{x}_3) + d(\mathbf{x}_2, \mathbf{x}_3)) - \frac{1}{3}d(\mathbf{x}_1, \mathbf{x}_2) = 7.42, \\ D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_4) &= \frac{2}{3}(d(\mathbf{x}_1, \mathbf{x}_4) + d(\mathbf{x}_2, \mathbf{x}_4)) - \frac{1}{3}d(\mathbf{x}_1, \mathbf{x}_2) = 16.08, \\ D(\{\mathbf{x}_1, \mathbf{x}_2\}, \mathbf{x}_5) &= \frac{2}{3}(d(\mathbf{x}_1, \mathbf{x}_5) + d(\mathbf{x}_2, \mathbf{x}_5)) - \frac{1}{3}d(\mathbf{x}_1, \mathbf{x}_2) = 12.08, \end{aligned}$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	7.42	16.08	12.08
\mathbf{x}_3	7.42	0	1.25	2
\mathbf{x}_4	16.08	1.25	0	2.25
\mathbf{x}_5	12.08	2	2.25	0

At the second stage of this method, \mathbf{x}_3 and \mathbf{x}_4 will be merged and the resulting increase in ESS is $\Delta\text{ESS}_{34} = \frac{1}{2}(1.25) = 0.625$. The total ESS becomes

$$\text{ESS}_2 = \text{ESS}_1 + \Delta\text{ESS}_{34} = 0.125 + 0.625 = 0.75.$$

After \mathbf{x}_3 and \mathbf{x}_4 are merged, the distances are updated as

$$\begin{aligned} D(\{\mathbf{x}_3, \mathbf{x}_4\}, \{\mathbf{x}_1, \mathbf{x}_2\}) &= \frac{3}{4}(7.42 + 16.08) - \frac{2}{4}(1.25) = 17, \\ D(\{\mathbf{x}_3, \mathbf{x}_4\}, \mathbf{x}_5) &= \frac{2}{3}(2 + 2.25) - \frac{1}{3}(1.25) = 2.42, \end{aligned}$$

and the dissimilarity matrix becomes

	$\{\mathbf{x}_1, \mathbf{x}_2\}$	$\{\mathbf{x}_3, \mathbf{x}_4\}$	\mathbf{x}_5
$\{\mathbf{x}_1, \mathbf{x}_2\}$	0	17	12.08
$\{\mathbf{x}_3, \mathbf{x}_4\}$	17	0	2.42
\mathbf{x}_5	12.08	2.42	0

At the third stage, $\{\mathbf{x}_3, \mathbf{x}_4\}$ and \mathbf{x}_5 will be merged. The resulting increase in ESS is $\Delta\text{ESS}_{(34)5} = \frac{1}{2}(2.42) = 1.21$. Then the total ESS becomes

$$\text{ESS}_3 = \text{ESS}_2 + \Delta\text{ESS}_{(34)5} = 0.75 + 1.21 = 1.96.$$

The distances are updated as

$$D(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}) = \frac{4}{5}(17) + \frac{3}{5}(12.08) - \frac{2}{5}(2.42) = 19.88,$$

and the dissimilarity matrix becomes

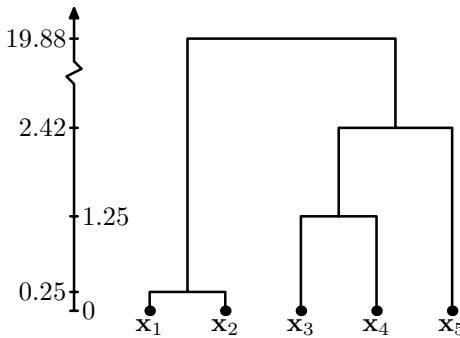


Figure 7.16. The dendrogram produced by applying Ward's method to the data set given in Figure 7.9.

	$\{x_1, x_2\}$	$\{x_3, x_4, x_5\}$
$\{x_1, x_2\}$	0	19.88
$\{x_3, x_4, x_5\}$	19.88	0

When all the data points are merged to form a single cluster, the increase in ESS will be $\Delta \text{ESS}_{(12)(345)} = \frac{1}{2}(19.88) = 9.94$ and the total ESS will be

$$\text{ESS}_4 = \text{ESS}_3 + \Delta \text{ESS}_{(12)(345)} = 1.96 + 9.94 = 11.9.$$

The whole process of this clustering can be represented by the dendrogram shown in Figure 7.16.

As pointed out by Anderberg (1973), a set of k clusters produced by Ward's method may or may not give the minimum possible ESS over all possible sets of k clusters formed from the n objects. However, the results of Ward's method are usually very good approximations of the optimal one. Kuiper and Fisher (1975) presented a comparison of Ward's method with the other five hierarchical clustering algorithms using the Monte Carlo method.

7.2.8 Other Agglomerative Methods

The seven agglomerative methods discussed in the previous subsections are most widely used in practice. In the Lance-Williams framework, there are some other agglomerative methods, such as the flexible method (Lance and Williams, 1967a), the sum of squares method (Jambu, 1978), and the mean dissimilarity method (Podani, 1989). Relevant discussions can be found in (Holman, 1992) and (Gordon, 1996).

There also exist agglomerative methods that cannot fit into the Lance-Williams framework. An example of such agglomerative methods is the bicriterion analysis proposed by Delattre and Hansen (1980).

7.3 Divisive Hierarchical Methods

The divisive hierarchical method proceeds the opposite way of the agglomerative hierarchical method. Initially, all the data points belong to a single cluster. The number of clusters

is increased by one at each stage of the algorithm by dividing an existing cluster into two clusters according to some criteria. A divisive hierarchical method may be adopted in which a single cluster is subdivided into smaller and smaller clusters. Divisive hierarchical clustering methods are essentially of two types: monothetic and polythetic (Everitt, 1993; Willett, 1988). A monothetic method divides the data on the basis of the possession or otherwise of a single specified attribute, while a polythetic method divides the data based on the values taken by all attributes.

It is not feasible to enumerate all possible divisions of a large (even moderate) cluster C to find the optimal partition, since there are $2^{|C|-1} - 1$ nontrivial different ways of dividing the cluster C into two clusters (Edwards and Cavalli-Sforza, 1965). Scott and Symons (1971a) have proposed an improved algorithm that requires examination of $2^d - 2$ partitions by assigning points in the hyperplane to the two clusters being considered, where d is the dimensionality of the data. Except for low-dimensional data, the algorithm proposed by Scott and Symons (1971a) is also very time-consuming. In fact, it turns out that the problem of finding an optimal bipartition for some clustering criteria is NP-hard (Gordon, 1996).

Another problem with divisive hierarchical algorithms is monotonicity, to be specified below. In a divisive hierarchy, one cluster is divided at a time, so what is the next cluster to be divided? This depends on the definition of a level. Such a level must be meaningful and monotone, which means that no subcluster may have a larger level than the level of its parent cluster.

However, it is possible to construct divisive hierarchical algorithms that do not need to consider all divisions and are monothetic. An algorithm called DIANA (DIvisive ANAlysis) presented in (Kaufman and Rousseeuw, 1990) is a divisive hierarchical clustering algorithm. It was based on the idea of Macnaughton-Smith et al. (1964). Other divisive hierarchical techniques are presented in (Edwards and Cavalli-Sforza, 1965) and (Späth, 1980).

7.4 Several Hierarchical Algorithms

Although the computational complexity of hierarchical algorithms is generally higher than that of partitional algorithms, many hierarchical algorithms have been designed and studied in (Kaufman and Rousseeuw, 1990), (Murtagh, 1983), and (Willett, 1988). In this section, we shall introduce some hierarchical algorithms, but we defer the introduction of hierarchical algorithms described in terms of graph theory (i.e., graph-based algorithms) to later chapters.

7.4.1 SLINK

The SLINK algorithm (Sibson, 1973) is a single-link hierarchical algorithm that can be carried out using arbitrary dissimilarity coefficients. In this algorithm, a compact representation of a dendrogram called a pointer representation, which offers economy in computation, is introduced.

Recall that a dendrogram is a nested sequence of partitions with associated numerical levels that can be defined as a function $c : [0, \infty) \rightarrow E(D)$ that satisfies

$$\begin{aligned} c(h) &\subseteq c(h') \text{ if } h \leq h', \\ c(h) &\text{ is eventually in } D \times D, \\ c(h + \delta) &= c(h) \text{ for some small } \delta > 0, \end{aligned}$$

where D is a given data set and $E(D)$ is the set of equivalence relations on D .

Recall also that a pointer representation is the pair of functions $\pi : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\}$ and $\lambda : \pi(\{1, 2, \dots, n\}) \rightarrow [0, \infty)$ that have the following properties:

$$\pi(n) = n, \quad \pi(i) > i \text{ for } i < n, \quad (7.21a)$$

$$\lambda(n) = \infty, \quad \lambda(\pi(i)) > \lambda(i) \text{ for } i < n, \quad (7.21b)$$

where n is the number of data points in D .

As discussed before, there is a one-to-one correspondence between dendrograms and pointer representations (Sibson, 1973). In particular, if c is a dendrogram, then the corresponding pointer representation is defined by

$$\begin{aligned} \lambda(i) &= \inf \{h : \exists j > i \text{ such that } (i, j) \in c(h)\}, \\ \pi(i) &= \max \{j : (i, j) \in c(\lambda(i))\} \end{aligned}$$

for $i < n$. Intuitively, $\lambda(i)$ is the lowest level at which the i th object is no longer the last object in its cluster, and $\pi(i)$ is the last object in the cluster that it joins. The pointer representation of a dendrogram ensures that a new object can be inserted in an efficient way.

ALGORITHM 7.1. The SLINK algorithm.

Require: n : number of objects; $d(i, j)$: dissimilarity coefficients;

```

1:  $\Pi[1] \leftarrow 1, \Lambda[1] \leftarrow \infty;$ 
2: for  $t = 1$  to  $n - 1$  do
3:    $\Pi[t + 1] \leftarrow t + 1, \Lambda[t + 1] \leftarrow \infty;$ 
4:    $M[i] \leftarrow d(i, t + 1)$  for  $i = 1, 2, \dots, t$ ;
5:   for  $i = 1$  to  $t$  do
6:     if  $\Lambda[i] \geq M[i]$  then
7:        $M[\Pi[i]] \leftarrow \min\{M[\Pi[i]], \Lambda[i]\};$ 
8:        $\Pi[i] \leftarrow M[i], \Pi[i] \leftarrow t + 1;$ 
9:     else
10:       $M[\Pi[i]] \leftarrow \min\{M[\Pi[i]], M[i]\};$ 
11:    end if
12:   end for
13:   for  $i = 1$  to  $t$  do
14:     if  $\Lambda[i] \geq \Lambda[\Pi[i]]$  then
15:        $\Pi[i] \leftarrow t + 1;$ 
16:     end if
17:   end for
```

18: **end for**

Let the dendrogram for the first t objects in the data set be c_t and its pointer representation be π_t, λ_t . Let $\mu_t(i)$ be defined recursively on i as

$$\mu_t(i) = \min\{d(i, t + 1), \min_{j, \pi_t(j)=i} \max\{\mu_t(j), \lambda_t(j)\}\},$$

which is defined for all $i = 1, 2, \dots, t$. It can be shown that the pointer representation for c_{t+1} is defined as (Sibson, 1973)

$$\pi_{t+1}(i) = \begin{cases} t + 1 & \text{if } i = t + 1, \\ t + 1 & \text{if } \mu_t(i) \text{ or } \mu_t(\pi_t(i)) \leq \lambda_t(i) \text{ for } i < t + 1, \\ \pi_t(i) & \text{otherwise,} \end{cases} \quad (7.22a)$$

$$\lambda_{t+1}(i) = \begin{cases} \min\{\mu_t(i), \lambda_t(i)\} & \text{if } i < t + 1, \\ \infty & \text{if } i = t + 1. \end{cases} \quad (7.22b)$$

If we start with $t = 1$ by $\pi_1(1) = 1, \lambda_1(1) = \infty$, then after $n - 1$ steps of the recursive process defined in (7.22), we shall obtain π_n, λ_n , which is the pointer representation of the dendrogram on the whole data set. Let Π, Λ , and M be three n -dimensional vectors such that Π, Λ contain π_t, λ_t in their first t entries in the t th step. Then the SLINK algorithm can be described as in Algorithm 7.1. The number of operations to find π_n, λ_n is $O(n^2)$.

7.4.2 Single-link Algorithms Based on Minimum Spanning Trees

A single-link cluster analysis can be carried out by using only the information contained in the minimum spanning tree (MST) (Gower and Ross, 1969). The performance of single-link cluster analysis can be improved by incorporating the MST algorithms into the clustering. To present the details, we start with a brief introduction of MSTs and some efficient algorithms for finding an MST.

The tree is a concept in graph theory. A tree is a connected graph with no cycles (Jain and Dubes, 1988). A spanning tree is a tree containing all vertices of the graph. When each edge in a graph is weighted by the dissimilarity between the two vertices that the edge connects, the weight of a tree is the sum of the edge weights in the tree. An MST of a graph G is a tree that has minimal weight among all other spanning trees of G .

A number of algorithms have been developed to find an MST. Most popular algorithms to find the MST proceed iteratively. Two popular algorithms for finding an MST have been discussed in (Gower and Ross, 1969). These algorithms are also presented in (Kruskal, 1956) and (Prim, 1957). In these algorithms, the edges belong to one of two sets A and B at any stage, where A is the set containing the edges assigned to the MST and B is the set of edges not assigned. Prim (1957) suggested an iterative algorithm that starts with any one of the given vertices and initially assigns to A the shortest segment starting from this vertex. Then the algorithm continues to assign to A the shortest segment from B that connects at least one segment from A without forming a closed loop among the segments already in A . The algorithm will stop when there are $n - 1$ segments in A . The MST produced by these algorithms may not be unique if there exist equal segments of minimum length.

Single-link cluster analysis can be performed by using a dissimilarity matrix that contains $\frac{n(n-1)}{2}$ distances, but it is impractical to record all $\frac{n(n-1)}{2}$ distances when n is large, where n is the number of data points. As such, the MST constructed from this data set provides a useful ancillary technique. To employ this technique in single-link cluster analysis, an MST should be constructed from the data set first.

Gower and Ross (1969) presented an approach to computing the MST from a data set. Let $D = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ be a data set, and let L_1 , L_2 , and L_3 be three n -dimensional vectors defined as follows:

$$\begin{aligned} L_1(i) &= j \text{ if } \mathbf{x}_i \text{ is the } j\text{th point to join } A, \\ L_2(i) &= j \text{ if } \mathbf{x}_i \text{ was linked to } \mathbf{x}_j \in A \text{ when it joined } A, \\ L_3(i) &= d(\mathbf{x}_i, \mathbf{x}_{L_2(i)}) \end{aligned}$$

for $i = 1, 2, \dots, n$, where $d(\cdot, \cdot)$ is a distance function.

The three vectors can be computed as follows. Initially, $L_1(1) = 1$, $L_1(i) = 0$ for $i = 2, 3, \dots, n$, and L_2 and L_3 are set to the zero vector, i.e., $A = \{\mathbf{x}_1\}$. At the first stage, let i_1 be defined as

$$i_1 = \arg \max_{1 \leq i \leq n} L_1(i)$$

and let j_1 be defined as

$$j_1 = \arg \min_{1 \leq j \leq n, L_1(j)=0} d(\mathbf{x}_{i_1}, \mathbf{x}_j).$$

Then \mathbf{x}_{j_1} will be assigned to A ; i.e., $L_1(2) = 0$ will be changed to $L_1(j_1) = 2$, and L_2 and L_3 will be updated as $L_2(j_1) = i_1$, and $L_3(j_1) = d(\mathbf{x}_{j_1}, \mathbf{x}_{i_1})$.

At the r th stage, let i_r and j_r be computed as

$$\begin{aligned} i_r &= \arg \max_{1 \leq i \leq n} L_1(i), \\ j_r &= \arg \min_{1 \leq j \leq n, L_1(j)=0} d(\mathbf{x}_{i_r}, \mathbf{x}_j). \end{aligned}$$

Hence, i_r is the point added to A at the previous stage. At this stage, \mathbf{x}_{j_r} will be added to A , and the vectors will be updated as $L_1(j_r) = r + 1$, $L_2(j_r) = i_r$, and $L_3(j_r) = d(\mathbf{x}_{j_r}, \mathbf{x}_{i_r})$.

After $n - 1$ stages, all the points will be added to A ; then the MST is found. Once the MST of the data set is found, the single-link cluster analysis can be performed from the MST. The algorithm does not need the full distance matrix at every level of clustering.

To find clusters at the first level, let δ be a distance threshold and L_0 be the largest multiple of δ that is less than the minimum distance, and let H be a set of links whose lengths lie between L_0 and $L_0 + \delta$. Let G be a list that contains the group members contiguously, marking the final member of each group with an indicator. For each link in H , find the endpoints in G , agglomerate the two groups in which the points are found, and shift down the intervening groups when necessary. Using the same procedure, a hierarchical system of agglomerative clusters can be easily constructed. This algorithm is also presented in (Rohlf, 1973), along with the FORTRAN code.

7.4.3 CLINK

Like the SLINK algorithm (Sibson, 1973), the CLINK algorithm (Defays, 1977) is also a hierarchical clustering algorithm based on a compact representation of a dendrogram.

But the CLINK algorithm is designed for the complete link method. The input of the CLINK algorithm is a fuzzy relation R , and the output is the pointer representation of the dendrogram.

A fuzzy relation R used in CLINK is defined as a collection of ordered pairs. For example, given a data set $D = \{\mathbf{x}_i : i = 1, 2, \dots, n\}$, the fuzzy relation R on D can be characterized as a membership function $R(\cdot, \cdot)$ that associates with each pair (i, j) the dissimilarity measure from \mathbf{x}_i to \mathbf{x}_j . The fuzzy relations R used in CLINK are reflexive and symmetric, i.e.,

$$\begin{aligned} R(i, i) &= 0, \quad 1 \leq i \leq n \text{ (reflexivity),} \\ R(i, j) &= R(j, i), \quad 1 \leq i, j \leq n \text{ (symmetry).} \end{aligned}$$

Let R and Q be two fuzzy relations defined on D . Then the min-max composition of R and Q is defined by

$$R \circ Q(i, j) = \min\{\max\{Q(i, k), R(k, j)\} : k = 1, 2, \dots, n\}$$

for $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, n$.

The r -fold composition $R \circ R \circ \dots \circ R$ is denoted by R^r . R is said to be transitive if $R^2 \supseteq R$. An ultrametric relation is a fuzzy relation that is reflexive, symmetric, and transitive. If a fuzzy relation R is reflexive and symmetric, then its transitive closure $\bar{R} = R^{n-1}$ may be obtained by a single-linkage clustering. A complete linkage clustering gives one or more minimal ultrametric relations (MURs) superior to R .

The goal of the CLINK algorithm is to find one of the MURs L superior to a reflexive symmetric fuzzy relation R . L and \bar{R} can be viewed as two extreme clusterings of D . In order to find such an L efficiently, the pointer representation (Sibson, 1973) is used in the CLINK algorithm. The pointer representation is a pair of functions (π, λ) defined as in equation (7.21). There is a one-to-one correspondence between pointer representations and ultrametric relations.

ALGORITHM 7.2. The pseudocode of the CLINK algorithm.

Require: n : number of objects; $R(i, j)$: dissimilarity coefficients;

```

1:  $\Pi[1] \Leftarrow 1, \Lambda[1] \Leftarrow \infty;$ 
2: for  $t = 1$  to  $n - 1$  do
3:    $\Pi[t + 1] \Leftarrow t + 1, \Lambda[t + 1] \Leftarrow \infty;$ 
4:    $M[i] \Leftarrow R(i, t + 1)$  for  $i = 1, 2, \dots, t$ ;
5:   for  $i = 1$  to  $t$  do
6:     if  $\Lambda[i] < M[i]$  then
7:        $M[\Pi[i]] \Leftarrow \max\{M[\Pi[i]], M[i]\};$ 
8:        $M[i] \Leftarrow \infty;$ 
9:     end if
10:   end for
11:   Set  $a \Leftarrow t$ ;
12:   for  $i = 1$  to  $t$  do
13:     if  $\Lambda[t - i + 1] \geq \Lambda[\Pi[t - i + 1]]$  then
```

```

14:   if  $M[t - i + 1] < M[a]$  then
15:      $a \Leftarrow t - i + 1;$ 
16:   end if
17:   else
18:      $M[t - i + 1] \Leftarrow \infty;$ 
19:   end if
20: end for
21: Set  $b \Leftarrow \Pi[a]$ ,  $c \Leftarrow \Lambda[a]$ ,  $\Pi[a] \Leftarrow t + 1$  and  $\Lambda[a] \Leftarrow M[a]$ ;
22: if  $a < t$  then
23:   while  $b < t$  do
24:     Set  $d \Leftarrow \Pi[b]$ ,  $e \Leftarrow \Lambda[b]$ ,  $\Pi[b] \Leftarrow t + 1$  and  $\Lambda[b] \Leftarrow c$ ;
25:     Set  $b \Leftarrow d$  and  $c \Leftarrow e$ ;
26:   end while
27:   if  $b = t$  then
28:     Set  $\Pi[b] \Leftarrow t + 1$  and  $\Lambda[b] \Leftarrow c$ ;
29:   end if
30: end if
31: for  $i = 1$  to  $t$  do
32:   if  $\Pi[\Pi[i]] = t + 1$  and  $\Lambda[i] = \Lambda[\Pi[i]]$  then
33:     Set  $\Pi[i] \Leftarrow t + 1$ ;
34:   end if
35: end for
36: end for

```

Suppose that L is an ultrametric relation on D . Then the corresponding pointer representation (π, λ) is defined as

$$\begin{aligned}\pi(i) &= \begin{cases} \max\{j : L(i, j) = \lambda(i)\} & \text{if } i < n, \\ n & \text{if } i = n, \end{cases} \\ \lambda(i) &= \begin{cases} \min\{L(i, j) : j > i\} & \text{if } i < n, \\ \infty & \text{if } i = n. \end{cases}\end{aligned}$$

Conversely, if (π, λ) is a pointer representation, then the corresponding ultrametric relation R is defined as

$$R(i, j) = \begin{cases} \lambda(i) & \text{if } j = \pi(i) > i, \\ \lambda(j) & \text{if } i = \pi(j) > j, \\ 0 & \text{if } i = j, \\ \infty & \text{otherwise.} \end{cases}$$

Let R_t be the restriction of R to the first t data points of D . If L_t is a MUR superior to R_t , then a MUR superior to R_{t+1} can be easily obtained from L_t . To do this, let (π_t, λ_t) be the pointer representation of a MUR L_t superior to R_t . The pseudocode of the CLINK algorithm is described in Algorithm 7.2. It is modified from the SLINK algorithm. These two algorithms cannot be further improved, because they require all pairwise dissimilarities to be considered (Murtagh, 1983).

7.4.4 BIRCH

Zhang et al. (1996) proposed an agglomerative hierarchical algorithm, called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), for clustering very large numerical data sets in Euclidean spaces. It is also the first clustering algorithm in the database area that takes account of noise.

In the algorithm of BIRCH, a clustering feature (CF) vector is used to summarize the information of each cluster. Given a cluster C of a d -dimensional data set, the CF vector for C is a triple defined as

$$\text{CF}(C) = (|C|, S_1, S_2),$$

where $|C|$ is the number of instances in C , and S_1 and S_2 are d -dimensional vectors defined as

$$S_1 = \sum_{\mathbf{x} \in C} \mathbf{x} = \left(\sum_{\mathbf{x} \in C} x_1, \sum_{\mathbf{x} \in C} x_2, \dots, \sum_{\mathbf{x} \in C} x_d \right),$$

$$S_2 = \sum_{\mathbf{x} \in C} \mathbf{x}^2 = \left(\sum_{\mathbf{x} \in C} x_1^2, \sum_{\mathbf{x} \in C} x_2^2, \dots, \sum_{\mathbf{x} \in C} x_d^2 \right),$$

where x_j ($1 \leq j \leq d$) is the value of the j th attribute of \mathbf{x} .

At the beginning, a CF tree is built dynamically as new data objects are inserted. A CF tree has three parameters: the branching factor B , the leaf factor L , and the threshold T . Each nonleaf node contains at most B subnodes of the form $[\text{CF}_i, \text{child}_i]$, each leaf node contains at most L entries of the form $[\text{CF}_i]$, and the diameter of each entry in a leaf node has to be less than T .

Outliers or noise are determined by considering the density of each entry in leaf nodes; i.e., low-density entries of leaf nodes are treated as outliers. Potential outliers are written out to disk in order to reduce the size of the tree. At certain points in the process, outliers are scanned to see if they can be reabsorbed into the current tree without causing the tree to grow in size.

After the CF tree is built, an agglomerative hierarchical clustering algorithm is applied directly to the nodes represented by their CF vectors. Then for each cluster, a centroid is obtained. Finally, a set of new clusters is formed by redistributing each data point to its nearest centroid.

BIRCH works well when clusters are of convex or spherical shape and uniform size. However, it is unsuitable when clusters have different sizes or nonspherical shapes (Guha et al., 1998).

7.4.5 CURE

Guha et al. (1998) proposed an agglomerative hierarchical clustering algorithm called CURE (Clustering Using REpresentatives) that can identify nonspherical shapes in large databases and wide variance in size. In this algorithm, each cluster is represented by a certain fixed number of points that are well scattered in the cluster. A combination of random sampling and partitioning is used in CURE in order to handle large databases.

CURE consists of six main steps: draw a random sample, partition the sample, partially cluster the partitions, eliminate the outliers, cluster the partial clusters, and label the data on the disk. Since CURE was developed for large databases, it begins by drawing a random sample from the database. Then the sample is partitioned and data points in each partition are partially clustered. After the outliers are eliminated, the preclustered data in each partition are clustered in a final pass to generate the final clusters.

In the first step, a random sample is drawn in order to handle large databases. In this step, the Chernoff bounds (Motwani and Raghavan, 1995) are used to analytically derive values for sample sizes such that the probability of missing clusters is low. The following theorem is proved.

Theorem 7.1. *For a cluster C , if the sample size s satisfies*

$$s \geq fn + \frac{n}{|C|} \log\left(\frac{1}{\delta}\right) + \frac{n}{|C|} \sqrt{\left(\log\frac{1}{\delta}\right)^2 + 2f|C| \log\frac{1}{\delta}},$$

then the probability that the sample contains fewer than $f|C|$ ($0 \leq f \leq 1$) points belonging to C is less than δ , $0 \leq \delta \leq 1$.

In the second step, a simple partitioning scheme is proposed for speeding up CURE when input sizes become large, since samples of larger sizes are required in some situations, such as when separation between clusters decreases and clusters become less densely packed. This is done by partitioning the sample space into p partitions, each of size $\frac{s}{p}$, where s is the sample size.

In the third step, each partition is clustered until the final number of clusters in each partition reduces to $\frac{s}{pq}$ for some constant $q > 1$. Since data sets almost always contain outliers, the fourth step of CURE is to eliminate outliers. The idea of this step is based on the fact that outliers tend, due to their large distances from other points, to merge with other points less and typically grow at a much slower rate than actual clusters in an agglomerative hierarchical clustering.

$\frac{s}{pq}$ clusters are generated for each partition in the third step. In the fifth step, a second clustering pass is run on the $\frac{s}{q}$ partial clusters for all the partitions. The hierarchical clustering algorithm is used only on points in a partition.

Since the input to CURE's clustering algorithm is a set of randomly sampled points from the original data set, the last step is to assign the appropriate cluster labels to the remaining data points such that each data point is assigned to the cluster containing the representative point closest to it.

The space complexity of CURE is linear in the input size n . The worst-case time complexity is $O(n^2 \log n)$, which can be further reduced to $O(n^2)$ in lower-dimensional spaces (e.g. two-dimensional space).

7.4.6 DIANA

DIANA (DIvisive ANALysis) presented in (Kaufman and Rousseeuw, 1990, Chapter 6) is a divisive hierarchical algorithm based on the proposal of Macnaughton-Smith et al.

(1964). It can be applied to all data sets that can be clustered by means of the agglomerative hierarchical algorithms.

The algorithm DIANA proceeds by a series of successive splits. At each step, the biggest cluster (i.e., the cluster with the largest diameter, which is defined to be the largest dissimilarity between two objects in it) is divided until at step $n - 1$ each object is in a single cluster.

Let C be a cluster. Then the diameter of C is defined to be

$$\text{Diam}(C) = \max_{\mathbf{x}, \mathbf{y} \in C} d(\mathbf{x}, \mathbf{y}). \quad (7.23)$$

The values of the diameter are also used as heights to represent the clustering structure in the dendograms or banners.

At each step, let C ($|C| \geq 2$) be the cluster to be divided, and let A and B be the clusters divided from C , i.e., $A \cap B = \emptyset$ and $A \cup B = C$. Initially, $A = C$ and $B = \emptyset$, and the algorithm DIANA finds A and B by moving points from A to B iteratively. At the first stage, a point \mathbf{y}_1 will be moved from A to B if it maximizes the function

$$D(\mathbf{x}, A \setminus \{\mathbf{x}\}) = \frac{1}{|A| - 1} \sum_{\mathbf{y} \in A, \mathbf{y} \neq \mathbf{x}} d(\mathbf{x}, \mathbf{y}), \quad (7.24)$$

where $d(\cdot, \cdot)$ can be any distance measure appropriate for the data.

Then A and B are updated as

$$\begin{aligned} A_{new} &= A_{old} \setminus \{\mathbf{y}_1\}, \\ B_{new} &= B_{old} \cup \{\mathbf{y}_1\}. \end{aligned}$$

In the next stage, the algorithm looks for other points in A that should be moved to B . Let $\mathbf{x} \in A$, and let the test function be defined as

$$\begin{aligned} &D(\mathbf{x}, A \setminus \{\mathbf{x}\}) - D(\mathbf{x}, B) \\ &= \frac{1}{|A| - 1} \sum_{\mathbf{y} \in A, \mathbf{y} \neq \mathbf{x}} d(\mathbf{x}, \mathbf{y}) - \frac{1}{|B|} \sum_{\mathbf{z} \in B} d(\mathbf{x}, \mathbf{z}). \end{aligned} \quad (7.25)$$

If a point \mathbf{y}_2 maximizes the function in equation (7.25) and the maximal value is strictly positive, then \mathbf{y}_2 will be moved from A to B . If the maximal value is negative or 0, the process is stopped and the division from C to A and B is completed.

Some variants of the process of dividing one cluster into two have been discussed in (Kaufman and Rousseeuw, 1990, Chapter 6). For example, the test function defined in equation (7.25) can be switched to

$$D(A \setminus \{\mathbf{x}\}, B \cup \{\mathbf{x}\}).$$

When a data point maximizes the above function, then it may be moved from A to B . A possible stopping rule is that $D(A_{new}, B_{new})$ no longer increases.

Since the algorithm DIANA uses the largest dissimilarity between two objects in a cluster as the diameter of the cluster, it is sensitive to outliers. Other techniques for splitting a cluster and examples of the algorithm have been presented in (Kaufman and Rousseeuw, 1990).

7.4.7 DISMEA

DISMEA, presented by Späth (1980) based on a divisive method proposed in Macqueen (1967), is a divisive hierarchical clustering algorithm that uses the k -means algorithm to subdivide a cluster into two. The divisive method produces a hierarchy that consists of n levels, where n is the size of the given data set. Starting with the whole data set, at each successive step, the cluster with the largest sum of squared distances (SSD) is divided into two clusters. This process is continued until every cluster contains a single data point.

More specifically, for a given data set D with n objects, the first step of the divisive method is to find a bipartition C_1, C_2 of D (i.e., $C_1 \neq \Phi, C_2 \neq \Phi, C_1 \cap C_2 = \Phi$, and $C_1 \cup C_2 = D$) such that the objective function

$$F(C_1, C_2; D) = \sum_{i=1}^2 \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mu(C_i)\|^2 \quad (7.26)$$

is minimized, where $\mu(C_i)$ is the centroid of cluster C_i , i.e.,

$$\mu(C_i) = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}.$$

At the succeeding steps, the cluster with the largest SSD is selected to be divided, where the SSD for a given cluster C is defined as

$$E(C) = \sum_{\mathbf{x} \in C} \|\mathbf{x} - \mu(C)\|^2,$$

where $\mu(C)$ is the centroid of cluster C .

For example, let C_1, C_2, \dots, C_j ($j < n$) be the clusters at a step. Then the next step is to divide C_{j_0} if

$$E(C_{j_0}) = \max_{1 \leq j \leq j_0} E(C_j).$$

One possible way to find the optimal bipartition is to examine all the $2^{|C|-1} - 1$ possible bipartitions and find an optimal one. However, this is impractical when the size of the cluster to be subdivided is large. Another approach is necessary to find an optimal bipartition.

Instead of examining all possible divisions, the algorithm DISMEA uses the k -means algorithm to subdivide a cluster into two. In practice, the maximum number of clusters k_{max} ($\leq n$) is specified in the algorithm DISMEA. The FORTRAN code for the algorithm DISMEA and some examples have been presented in (Späth, 1980).

7.4.8 Edwards and Cavalli-Sforza Method

Edwards and Cavalli-Sforza (1965) have suggested a divisive hierarchical algorithm by successfully splitting the objects into two groups to maximize the between-groups sum of squares. In this algorithm, the cluster density is measured by the variance, i.e., the within-cluster sum of squares divided by the number of points. At the beginning of the algorithm, the whole data set is divided into two groups according to the criterion mentioned above.

Then each of the two groups will be split into two groups according to the same criterion. One then continues splitting until each cluster contains only one point.

The technique adopted by Edwards and Cavalli-Sforza (1965) to split a group of data points into two subgroups is to enumerate all possible bipartitions and choose the one that minimizes the within-group sum of squares. Let D be a set of n data points. Then there are $2^{n-1} - 1$ different ways to divide D into two clusters. More specifically, let A_i, B_i be a partition of D for $i = 1, 2, \dots, 2^{n-1} - 1$, i.e., $A_i \cup B_i = D$, $A_i \cap B_i = \emptyset$, and $A_i \neq \emptyset, B_i \neq \emptyset$. Then the within-cluster sum of squares of the partition (A_i, B_i) is

$$WSS_i = \frac{1}{|A_i|} \sum_{\mathbf{x}, \mathbf{y} \in A_i} \|\mathbf{x} - \mathbf{y}\|^2 + \frac{1}{|B_i|} \sum_{\mathbf{x}, \mathbf{y} \in B_i} \|\mathbf{x} - \mathbf{y}\|^2 \quad (7.27)$$

for $i = 1, 2, \dots, 2^{n-1} - 1$.

The best partition is (A_{i_0}, B_{i_0}) such that

$$WSS_{i_0} = \min_{1 \leq i \leq 2^{n-1} - 1} WSS_i.$$

Edwards and Cavalli-Sforza (1965) also pointed out that the best partition may not be unique in some cases. For example, if the distance between any two data points is the same, say, d , then splitting n points into two clusters A, B such that $|A| = r, |B| = n - r$ will give a within-cluster sum of squares of

$$\frac{1}{r} \cdot \frac{1}{2} r(r-1)d^2 + \frac{1}{n-r} \cdot \frac{1}{2} (n-r)(n-r-1)d^2 = \frac{1}{2}(n-2)d^2,$$

which is independent of r .

For the Edwards and Cavalli-Sforza method, a major difficulty is that the initial division requires an examination of all $2^{n-1} - 1$ bipartitions, where n is the size of the original data set. This will take an enormous amount of computer time (Gower, 1967). Scott and Symons (1971a) suggested a refined algorithm that limits the consideration to $(2^d - 2) \binom{n}{d}$ partitions, where n is the size of the data set and d is the dimensionality of the data set. The improved algorithm is based on the results by Fisher (1958). Regarding the minimum variance partition for the univariate case, i.e., $d = 1$, Fisher (1958) defines a contiguous partition to be such that if x_i and x_j ($x_i \leq x_j$) belong to the same group, then every object x_k with $x_i \leq x_k \leq x_j$ also belongs to the group, and he proved that the optimal partition is contiguous.

For $d \geq 2$, Scott and Symons (1971a) generalized the definition of a contiguous partition into k groups to be such that if each member of a set of data points belongs to the same group, then every data point in the convex hull (Barber et al., 1996) of the set also belongs to the group. They also generalized Fisher's result as follows:

1. The minimum variance partition is contiguous.
2. For $d \geq 2$, the two groups of the minimum variance partition are separated by a $(d-1)$ -dimensional hyperplane containing d of the data points.

Thus for each of the $\binom{n}{d}$ choices of d data points, there are $2^d - 2$ ($d \geq 2$) possible assignments of the d points in the hyperplane into two groups. Since it is simple to decide

on which side of the hyperplane each of the remaining $n - d$ points will lie, there are a total of $(2^d - 2) \binom{n}{d}$ partitions that should be considered.

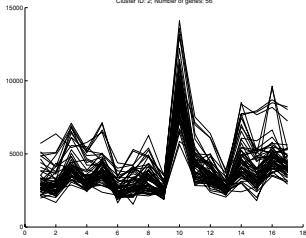
Note that, if $d \geq 2$, then the same partition will be considered many times. For example, when $d = 2$, the same partition will be considered twice. In fact, it has been shown that the number of distinct contiguous partitions is given by (Harding, 1967)

$$v_d(n) = \sum_{i=1}^d \binom{n}{i}.$$

7.5 Summary

Comprehensive discussions of hierarchical clustering methods can be found in (Gordon, 1987) and (Gordon, 1996). Techniques for improving hierarchical methods are discussed in (Murtagh, 1983), and (Murtagh, 1984a) discussed the complexities of some major agglomerative hierarchical clustering algorithms. A review of applying hierarchical clustering methods to document clustering is given in (Willett, 1988). Other discussions are provided in (Hodson, 1970), (Lance and Williams, 1967a), and (Lance and Williams, 1967c).

Posse (2001) proposed a hierarchical clustering method for large datasets using the MST algorithm to initialize the partition instead of the usual set of singleton clusters. Other hierarchical clustering algorithms can be found in (Rohlf, 1970) and (Day and Edelsbrunner, 1984).



Chapter 8

Fuzzy Clustering Algorithms

Hard (or crisp) clustering algorithms require that each data point of the data set belong to one and only one cluster. Fuzzy clustering extends this notion to associate each data point in the data set with every cluster using a membership function. Since the concept of fuzzy sets (Zadeh, 1965) was introduced, fuzzy clustering has been widely discussed, studied, and applied in various areas. Early work on applying fuzzy set theory in cluster analysis was proposed by Bellman et al. (1966) and Ruspini (1969).

Let D be a data set with n objects, each of which is described by d attributes, and let c be an integer between one and n . Then a fuzzy c -partition is defined by a $c \times n$ matrix $U = (u_{li})$ that satisfies

$$u_{li} \in [0, 1], \quad 1 \leq l \leq c, 1 \leq i \leq n, \quad (8.1a)$$

$$\sum_{l=1}^c u_{li} = 1, \quad 1 \leq i \leq n, \quad (8.1b)$$

$$\sum_{i=1}^n u_{li} > 0, \quad 1 \leq l \leq c, \quad (8.1c)$$

where u_{li} denotes the degree of membership of the object i in the l th cluster.

For each fuzzy c -partition, there is a corresponding hard c -partition. Let u_{li} ($l = 1, 2, \dots, c, i = 1, 2, \dots, n$) be the membership of any fuzzy c -partition. Then the corresponding hard c -partition of u_{li} can be defined as ω_{li} as follows (Xie and Beni, 1991):

$$\omega_{li} = \begin{cases} 1 & \text{if } l = \arg \max_{1 \leq j \leq c} u_{ji}, \\ 0 & \text{otherwise.} \end{cases}$$

8.1 Fuzzy Sets

The concept of the fuzzy set was first introduced by Zadeh (1965). A fuzzy set is defined to be a class of objects with a continuum of grades of membership. Each fuzzy set is

characterized by a membership function that assigns to each object a grade of membership ranging from 0 to 1. In this section, we will review some basic properties of the fuzzy set. Readers are referred to the original paper Zadeh (1965) or the reprinted paper Zadeh (1992).

Let $X = \{x\}$ be a space of points, where x denotes a generic element of X . A fuzzy set A in X is characterized by a membership (characteristic) function $f_A(x)$ that associates with each point in X a real number in the interval $[0, 1]$. Thus, when A is a set in the ordinary sense of the term, $f_A(x)$ can take on only two values, 0 and 1, with $f_A(x) = 1$ or 0 according as x does or does not belong to A . A larger value of $f_A(x)$ indicates a higher grade of membership of x in A .

Several definitions involving fuzzy sets are obvious extensions of the corresponding definitions for ordinary sets. Below we will introduce these definitions for fuzzy sets.

Emptiness. A fuzzy set A is empty if and only if its membership function $f_A(x)$ is identically zero on X , i.e., $f_A(x) \equiv 0$.

Equal. Two fuzzy sets A and B are equal, written as $A = B$, if and only if their membership functions are identical, i.e., $f_A(x) = f_B(x) \forall x \in X$.

Complementation. The complement of a fuzzy set A , denoted by A' , is defined by

$$f_{A'}(x) = 1 - f_A(x) \quad \forall x \in X.$$

Containment. A fuzzy set A is said to be contained in a fuzzy set B (or, equivalently, A is a subset of B , or A is smaller than or equal to B) if and only if $f_A(x) \leq f_B(x) \forall x \in X$. In symbols,

$$A \subset B \Leftrightarrow f_A \leq f_B.$$

Union. The union of two fuzzy sets A and B is again a fuzzy set C , written as $C = A \cup B$, whose membership function is defined by

$$f_C(x) = \max\{f_A(x), f_B(x)\}, \quad x \in X,$$

where $f_A(x)$ and $f_B(x)$ are membership functions of A and B , respectively. Let \vee stand for maximum. Then the union can be represented in abbreviated form as

$$f_C = f_A \vee f_B.$$

Like the union of ordinary sets, the union \cup of fuzzy sets has the associative property, i.e., $A \cup (B \cup C) = (A \cup B) \cup C$. It can be shown that the union of fuzzy sets A and B is the smallest fuzzy set containing both A and B .

Intersection. The intersection of two fuzzy sets A and B is again a fuzzy set C , written as $C = A \cap B$, whose membership function is defined by

$$f_C(x) = \min\{f_A(x), f_B(x)\} \quad \forall x \in X,$$

where $f_A(x)$ and $f_B(x)$ are membership functions of A and B , respectively. In abbreviated form,

$$f_C = f_A \wedge f_B,$$

where \wedge means minimum. It can also be shown that the intersection of two fuzzy sets A and B is the largest fuzzy set contained in both A and B . Like union \cup , intersection \cap has the associative property.

Based on the operations of union, intersection, and complementation, it is easy to extend many of the basic identities that hold for ordinary sets to fuzzy sets, such as the De Morgan laws:

$$(A \cup B)' = A' \cap B', \\ (A \cap B)' = A' \cup B',$$

and the distributive laws:

$$C \cap (A \cup B) = (C \cap A) \cup (C \cap B), \\ C \cup (A \cap B) = (C \cup A) \cap (C \cup B),$$

where A , B , and C are fuzzy sets.

8.2 Fuzzy Relations

Let D be a subset of a d -dimensional Euclidean space \mathbb{R}^d and c be a positive integer bigger than 1. A partition of D into c clusters can be represented by the indicator function $\mu_1, \mu_2, \dots, \mu_c$ such that for all $\mathbf{x} \in D$

$$\mu_i(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ is in the } i\text{th cluster,} \\ 0 & \text{otherwise,} \end{cases} \quad i = 1, 2, \dots, c.$$

A hard relation r in D is defined to be a function $r : D \times D \rightarrow \{0, 1\}$, where \mathbf{x} and \mathbf{y} in D are said to have a relation if $r(\mathbf{x}, \mathbf{y}) = 1$. A hard relation r in D is called an equivalence relation if and only if it satisfies reflexivity, symmetry, and transitivity, i.e., for all $\mathbf{x}, \mathbf{y} \in D$, the following conditions are satisfied (Yang, 1993):

1. Reflexivity: $r(\mathbf{x}, \mathbf{x}) = 1$.
2. Symmetry: $r(\mathbf{x}, \mathbf{y}) = r(\mathbf{y}, \mathbf{x})$.
3. Transitivity: If $r(\mathbf{x}, \mathbf{z}) = r(\mathbf{y}, \mathbf{z})$ for some $\mathbf{z} \in D$, then $r(\mathbf{x}, \mathbf{y}) = 1$.

Consider a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ containing n objects in \mathbb{R}^d . For convenience, denote $\mu_i(\mathbf{x}_j)$ by μ_{ij} , $i = 1, 2, \dots, c$, $j = 1, 2, \dots, n$, and $r(\mathbf{x}_j, \mathbf{x}_l)$ by r_{jl} , $j, l = 1, 2, \dots, n$. Let V_{cn} be the usual vector space of real $c \times n$ matrices and let μ_{ij} be the (i, j) th element of $U \in V_{cn}$. Then the hard c -partitions space for the data set D is defined as (Bezdek, 1981)

$$M_c = \left\{ U \in V_{cn} : \mu_{ij} \in \{0, 1\} \forall i, j; \quad \sum_{i=1}^c \mu_{ij} = 1 \forall j; \quad 0 < \sum_{j=1}^n \mu_{ij} \forall i \right\}.$$

For any $U = (\mu_{ij}) \in M_c$, there is a corresponding relation matrix $R = (r_{jl}) \in V_{nn}$ defined by

$$r_{jl} = \begin{cases} 1 & \text{if } \mu_{ij} = \mu_{il} = 1 \text{ for some } i, \\ 0 & \text{otherwise.} \end{cases}$$

Let $A = (a_{ij})$ and $B = (b_{ij})$ in V_{nn} , and define $A \leq B$ if and only if $a_{ij} \leq b_{ij}$ for all $i, j = 1, 2, \dots, n$. Define composition $R \circ R = (r'_{ij})$ by $r'_{ij} = \max_{1 \leq l \leq n} \min\{r_{il}, r_{lj}\}$. Then the set of all equivalence relations on D can be represented by

$$R_n = \{R \in V_{nn} : r_{ij} \in \{0, 1\} \forall i, j; \quad I \leq R; \quad R = R \circ R\}.$$

Thus, for any $U \in M_c$, there is a relation matrix $R \in R_n$ such that R is an equivalence relation corresponding to U .

The fuzzy set, first proposed by Zadeh (1965), is an extension to allow $\mu_i(\cdot)$ to be a membership function assuming values in the interval $[0, 1]$. By relaxing the conditions of M_c and R_n , we can obtain the fuzzy extension of M_c and R_n as

$$M_{fc} = \left\{ U \in V_{cn} : \mu_{ij} \in [0, 1] \forall i, j; \quad \sum_{i=1}^c \mu_{ij} = 1 \forall j; \quad 0 < \sum_{j=1}^n \mu_{ij} \forall i \right\}$$

and

$$R_{fn} = \{R \in V_{nn} : r_{ij} \in [0, 1] \forall i, j; \quad I \leq R; \quad R = R^T \text{ and } R \geq R \circ R\}.$$

Then M_{fc} is a fuzzy c -partitions space for D , and R_{fn} is the set of all similarity relations in D .

Fuzzy clustering based on fuzzy relations was first proposed by Tamura et al. (1971).

8.3 Fuzzy k -means

The fuzzy k -means algorithm (Bezdek, 1974b) is an extension of the k -means algorithm for fuzzy clustering. Give a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, the algorithm is based on minimization of the objective function

$$J_q(U, V) = \sum_{j=1}^n \sum_{i=1}^k u_{ij}^q d^2(\mathbf{x}_j, V_i) \tag{8.2}$$

with respect to U (a fuzzy k -partition of the data set) and to V (a set of k prototypes), where q is a real number greater than 1, V_i is the centroid of cluster i , u_{ij} is the degree of membership of object \mathbf{x}_j belonging to cluster i , $d^2(\cdot, \cdot)$ is an inner product metric, and k is the number of clusters. The parameter q controls the “fuzziness” of the resulting clusters (Bezdek, 1981).

ALGORITHM 8.1. The fuzzy k -means algorithm.

S_1 Choose initial centroids $V_i (i = 1, 2, \dots, k)$;

S_2 Compute the membership matrix as follows:

$$u_{ij} = \frac{\left[d^2(\mathbf{x}_j, V_i)\right]^{-\frac{1}{q-1}}}{\sum_{l=1}^k \left[d^2(\mathbf{x}_j, V_l)\right]^{-\frac{1}{q-1}}}, \quad i = 1, 2, \dots, k, j = 1, 2, \dots, n; \quad (8.3)$$

S_3 Compute new centroids \hat{V}_i ($i = 1, 2, \dots, k$) as

$$\hat{V}_i = \frac{\sum_{j=1}^n u_{ij}^q \mathbf{x}_j}{\sum_{j=1}^n u_{ij}^q},$$

and update the membership matrix (u_{ij}) to (\hat{u}_{ij}) according to equation (8.3);

S_4 If $\max_{ij} |u_{ij} - \hat{u}_{ij}| < \epsilon$, then stop; otherwise go to step S_3 , where ϵ is a termination criterion between 0 and 1.

The fuzzy clustering is carried out via an iterative optimization of equation (8.2) (Bezdek, 1974b). The procedure of the optimization is shown in Algorithm 8.1. The fuzzy k -means algorithm and its derivatives are also presented in (Gath and Geva, 1989).

For hyperellipsoidal clusters and clusters with variable densities and unequal sizes, Gath and Geva (1989) presented an “exponential” distance measure based on maximum likelihood estimation, i.e.,

$$d_e^2(\mathbf{x}_j, V_i) = \frac{\sqrt{\det(F_i)}}{P_i} \exp \left[\frac{(\mathbf{x}_j - V_i)^T F_i^{-1} (\mathbf{x}_j - V_i)}{2} \right],$$

where F_i is the fuzzy covariance matrix of the i th cluster and P_i is the *a priori* probability of selecting the i th cluster.

The above distance is used in the calculation of $h(i|\mathbf{x}_j)$, the probability of selecting the i th cluster given the j th object:

$$h(i|\mathbf{x}_j) = \frac{\frac{1}{d_e^2(\mathbf{x}_j, V_i)}}{\sum_{l=1}^k \frac{1}{d_e^2(\mathbf{x}_j, V_l)}}. \quad (8.4)$$

If we let $q = 2$ in equation (8.3), $h(i|\mathbf{x}_j)$ defined in equation (8.4) is similar to u_{ij} . Thus, if we substitute equation (8.4) instead of equation (8.3) in step S_2 of Algorithm 8.1, the fuzzy k -means algorithm becomes the FMLE (Fuzzy modification of the Maximum Likelihood Estimation) algorithm. In addition to computing the new centroids, Step S_3 of

Algorithm 8.1 needs to calculate P_i and F_i :

$$P_i = \frac{1}{n} \sum_{j=1}^n h(i|\mathbf{x}_j),$$

$$F_i = \frac{\sum_{j=1}^n h(i|\mathbf{x}_j)(\mathbf{x}_j - V_i)(\mathbf{x}_j - V_i)^T}{\sum_{j=1}^n h(i|\mathbf{x}_j)}.$$

Gath and Geva (1989) also pointed out that the FMLE algorithm does not perform well, as it seeks an optimum in a narrow local region due to the “exponential” distance incorporated in the algorithm.

Other generalizations of the fuzzy k -means algorithm are presented and discussed in (Yang, 1993).

8.4 Fuzzy k -modes

To describe the fuzzy k -modes algorithm (Huang and Ng, 1999), let us begin with some notation. Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a categorical data set with n objects, each of which is described by d categorical attributes A_1, A_2, \dots, A_d . Attribute A_j ($1 \leq j \leq d$) has n_j categories, i.e., $DOM(A_j) = \{a_{j1}, a_{j2}, \dots, a_{jn_j}\}$. Let the cluster centers be represented by $\mathbf{z}_l = (z_{l1}, z_{l2}, \dots, z_{ld})$ for $1 \leq l \leq k$, where k is the number of clusters. The simple matching distance measure between \mathbf{x} and \mathbf{y} in D is defined as

$$d_c(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d \delta(x_j, y_j), \quad (8.5)$$

where x_j and y_j are the j th components of \mathbf{x} and \mathbf{y} , respectively, and $\delta(\cdot, \cdot)$ is the simple matching distance (see Subsection 6.3.1).

Then the objective of the fuzzy k -modes clustering is to find W and Z that minimize

$$F_c(W, Z) = \sum_{l=1}^k \sum_{i=1}^n w_{li}^\alpha d_c(\mathbf{x}_i, \mathbf{z}_l), \quad (8.6)$$

subject to (8.1a), (8.1b), and (8.1c), where $\alpha > 1$ is the weighting component, $d_c(\cdot, \cdot)$ is defined in equation (8.5), $W = (w_{li})$ is the $k \times n$ fuzzy membership matrix, and $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ is the set of cluster centers. Note that $\alpha = 1$ gives the hard k -modes clustering, i.e., the k -modes algorithm.

To update the cluster centers given the estimate of W , Huang and Ng (1999) proved the following theorem.

Theorem 8.1. *The quantity $F_c(W, Z)$ defined in equation (8.6) is minimized if and only if $z_{lj} = a_{jr} \in DOM(A_j)$, where*

$$r = \arg \max_{1 \leq t \leq n_j} \sum_{i, x_{ij}=a_{jt}} w_{li}^\alpha,$$

i.e.,

$$\sum_{i, x_{ij} = a_{jr}} w_{li}^\alpha \geq \sum_{i, x_{ij} = a_{jt}} w_{li}^\alpha, \quad 1 \leq t \leq n_j,$$

for $1 \leq j \leq d$ and $1 \leq l \leq k$.

To update the fuzzy membership matrix W given the estimate of Z , Huang and Ng (1999) also presented the following theorem.

Theorem 8.2. Let $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ be fixed. Then the fuzzy membership matrix W that minimizes the quantity $F_c(W, Z)$ defined in equation (8.6) subject to (8.1a), (8.1b), and (8.1c) is given by

$$w_{li} = \begin{cases} 1 & \text{if } \mathbf{x}_i = \mathbf{z}_l, \\ 0 & \text{if } \mathbf{x}_i = \mathbf{z}_h, h \neq l, \\ \frac{1}{\sum_{h=1}^k \left[\frac{d(\mathbf{x}_i, \mathbf{z}_h)}{d(\mathbf{x}_i, \mathbf{z}_l)} \right]^{\frac{1}{\alpha-1}}} & \text{otherwise,} \end{cases} \quad 1 \leq l \leq k, 1 \leq i \leq n.$$

Based on the two theorems described above, the fuzzy k -modes algorithm can be implemented recursively (see Algorithm 8.2).

ALGORITHM 8.2. Fuzzy k -modes algorithm, where r is the maximum number of iterations.

```

1: Choose initial point  $Z_0 \in \mathbb{R}^{mk}$ ;
2: Determine the  $W_0$  such that the cost function  $F(W_0, Z_0)$  is minimized;
3: for  $t = 1$  to  $r$  do
4:   Determine the  $Z_1$  such that the cost function  $F(W_0, Z_1)$  is minimized;
5:   if  $F(W_0, Z_1) = F(W_0, Z_0)$  then
6:     stop;
7:   else
8:     Determine the  $W_1$  such that the cost function  $F(W_1, Z_1)$  is minimized;
9:     if  $F(W_1, Z_1) = F(W_0, Z_1)$  then
10:       stop;
11:     else
12:        $W_0 \Leftarrow W_1$ ;
13:     end if
14:   end if
15: end for

```

The fuzzy k -modes algorithm is described in Algorithm 8.2. The difficult part of the fuzzy k -modes algorithm is to minimize the cost function and to reduce the computational complexity. This issue is also addressed in (Huang and Ng, 1999).

8.5 The c -means Method

The c -means clustering algorithm (Bezdek, 1981; Bobrowski and Bezdek, 1991) is a clustering method that allows one piece of data to belong to two or more clusters. Let $X = \{x_1, x_2, \dots, x_M\}$ be a set of numerical data in \mathcal{R}^N . Let c be an integer, $1 < c < M$. Given X , we say that c fuzzy subsets $\{u_k : X \rightarrow [0, 1]\}$ are a c -partition of X if the following conditions are satisfied:

$$0 \leq u_{kj} \leq 1 \quad \forall k, j, \quad (8.7a)$$

$$\sum_{k=1}^c u_{kj} = 1 \quad \forall j, \quad (8.7b)$$

$$0 < \sum_{j=1}^M u_{kj} < n \quad \forall k, \quad (8.7c)$$

where $u_{kj} = u_k(x_j)$, $1 \leq k \leq c$ and $1 \leq j \leq M$. Let the cM values u_{kj} satisfying the above conditions be arrayed as a $c \times M$ matrix $U = [u_{kj}]$. Then the set of all such matrices are the *nondegenerate fuzzy c -partitions* of X :

$$M_{fcM} = \{U \in \mathcal{R}^N : u_{kj} \text{ satisfies conditions (8.7)} \quad \forall k \text{ and } j\}.$$

If all the u_{kj} 's are either 0 or 1, we have the subset of *hard c -partitions* of X :

$$M_{cM} = \{U \in M_{fcM} : u_{kj} = 0 \text{ or } 1 \quad \forall k \text{ and } j\}.$$

The u_{kj} 's can be treated as the membership of x_j in the k th cluster of X . Here c is assumed to be known. The well-known objective function for clustering in X is the classical within-group sum of squared errors (WGSS) function (Bobrowski and Bezdek, 1991), which is referred to as J_1 :

$$J_1(U, \mathbf{v}; X) = \sum_{k=1}^c \sum_{j=1}^M u_{kj} (\|x_j - v_k\|_I)^2,$$

where $\mathbf{v} = (v_1, v_2, \dots, v_c)$ is a vector of cluster centers, $v_k \in \mathcal{R}^N$ for $1 \leq k \leq c$ and $U \in M_{cM}$.

The goal is to find an optimal partition U^* of X such that the pairs (U^*, \mathbf{v}^*) are local minimizers of J_1 .

There is also a generalized form of the objective function, the fuzzy c -prototypes form:

$$J_m(U, P; X) = \sum_{k=1}^c \sum_{j=1}^M (u_{kj})^m D_{kj},$$

where $m \in [1, \infty)$ is a weighting exponent on each fuzzy membership, $U \in M_{fcM}$ is a fuzzy c -partition of X , $P = (P_1, P_2, \dots, P_c)$ are cluster prototypes, and D_{kj} is some measure of similarity (error, etc.) between P_k and x_j .

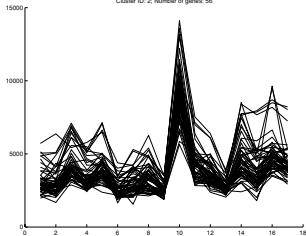
The measure of similarity D_{kj} can be defined in many ways, such as the inner product norms, $\|x_j - v_k\|_A^2 = (x_j - v_k)^T A (x_j - v_k)$, where A is a positive definite $N \times N$ matrix, and the Minkowski norms,

$$D_{kj} = \|x_j - v_k\|_p = \left[\sum_{s=1}^N |x_{js} - v_{ks}|^p \right]^{\frac{1}{p}}, \quad p \geq 1.$$

If we restrict $u_{kj} \in \{0, 1\}$ in (8.7), then we obtain a hard c -means algorithm (Pal and Biswas, 1997). The convergence properties of the fuzzy c -means algorithm are presented in (Hathaway and Bezdek, 1984) and Bezdek et al. (1992).

8.6 Summary

We introduced fuzzy sets and some fuzzy clustering algorithms in this chapter. Conventional clustering approaches assume that an object can belong to one and only one cluster. In practice, however, the separation of clusters is a fuzzy notion. Therefore, fuzzy clustering algorithms that combine fuzzy logic and cluster analysis techniques have special advantages over conventional clustering algorithms by allowing each object to be assigned to one or more clusters with some probabilities. Because of the important applications of fuzzy clustering analysis, Höppner et al. (1999) devoted a whole book to addressing the fuzzy clustering problem.



Chapter 9

Center-based Clustering Algorithms

Compared to other types of clustering algorithms, center-based algorithms are very efficient for clustering large databases and high-dimensional databases. Usually, center-based algorithms have their own objective functions, which define how good a clustering solution is. The goal of a center-based algorithm is to minimize its objective function. Clusters found by center-based algorithms have convex shapes and each cluster is represented by a center. Therefore, center-based algorithms are not good choices for finding clusters of arbitrary shapes. In this chapter, we shall present and discuss some center-based clustering algorithms and their advantages and disadvantages. We should mention that the expectation-maximization (EM) algorithm can be treated as a center-based algorithm, but we will defer the introduction of the EM algorithm to the chapter on model-based algorithms (Chapter 14).

9.1 The k -means Algorithm

The conventional k -means algorithm described in Algorithm 9.1, one of the most used clustering algorithms, was first described by Macqueen (1967). It was designed to cluster numerical data in which each cluster has a center called the mean. The k -means algorithm is classified as a partitional or nonhierarchical clustering method (Jain and Dubes, 1988). In this algorithm, the number of clusters k is assumed to be fixed. There is an error function in this algorithm. It proceeds, for a given initial k clusters, by allocating the remaining data to the nearest clusters and then repeatedly changing the membership of the clusters according to the error function until the error function does not change significantly or the membership of the clusters no longer changes. The conventional k -means algorithm (Hartigan, 1975; Hartigan and Wong, 1979) is briefly described below.

Let D be a data set with n instances, and let C_1, C_2, \dots, C_k be the k disjoint clusters of D . Then the error function is defined as

$$E = \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mu(C_i)), \quad (9.1)$$

where $\mu(C_i)$ is the centroid of cluster C_i . $d(\mathbf{x}, \mu(C_i))$ denotes the distance between \mathbf{x} and $\mu(C_i)$, and it can be one of the many distance measures discussed before, a typical choice of which is the Euclidean distance $d_{euc}(\cdot, \cdot)$ defined in (6.11).

ALGORITHM 9.1. The conventional k -means algorithm.

Require: Data set D , Number of Clusters k , Dimensions d :

- { C_i is the i th cluster}
- {1. Initialization Phase}
- 1: $(C_1, C_2, \dots, C_k) =$ Initial partition of D .
- {2. Iteration Phase}
- 2: **repeat**
- 3: $d_{ij} =$ distance between case i and cluster j ;
- 4: $n_i = \arg \min_{1 \leq j \leq k} d_{ij}$;
- 5: Assign case i to cluster n_i ;
- 6: Recompute the cluster means of any changed clusters above;
- 7: **until** no further changes of cluster membership occur in a complete iteration
- 8: Output results.

The k -means algorithm can be divided into two phases: the initialization phase and the iteration phase. In the initialization phase, the algorithm randomly assigns the cases into k clusters. In the iteration phase, the algorithm computes the distance between each case and each cluster and assigns the case to the nearest cluster.

We can treat the k -means algorithm as an optimization problem. In this sense, the goal of the algorithm is to minimize a given objective function under certain conditions. Let $D = \{\mathbf{x}_i, i = 1, 2, \dots, n\}$ be a data set with n instances and k be a given integer. The objective function can be defined as

$$P(W, Q) = \sum_{l=1}^k \sum_{i=1}^n w_{il} d_{euc}(\mathbf{x}_i, \mathbf{q}_l), \quad (9.2)$$

where $Q = \{\mathbf{q}_l, l = 1, 2, \dots, k\}$ is a set of objects, $d_{euc}(\cdot, \cdot)$ is the Euclidean distance defined in (6.11), and W is an $n \times k$ matrix that satisfies the following conditions:

1. $w_{il} \in \{0, 1\}$ for $i = 1, 2, \dots, n, l = 1, 2, \dots, k$,
2. $\sum_{l=1}^k w_{il} = 1$ for $i = 1, 2, \dots, n$.

The k -means algorithm can be formatted as the following optimization problem P (Selim and Ismail, 1984; Bobrowski and Bezdek, 1991): Minimize $P(W, Q)$ in (9.2) subject to conditions (1) and (2).

The optimization problem P can be solved by iteratively solving the following two subproblems (Huang, 1998):

- Subproblem P_1 : Fix $Q = \hat{Q}$ and solve the reduced problem $P(W, \hat{Q})$.
- Subproblem P_2 : Fix $W = \hat{W}$ and solve the reduced problem $P(\hat{W}, Q)$.

As to how to solve the subproblems P_1 and P_2 , we have the following theorems (Huang, 1998).

Theorem 9.1. *In subproblem P_1 , let $\hat{Q} = \{\hat{\mathbf{1}}_l, l = 1, 2, \dots, k\}$ be fixed. Then the function $P(W, \hat{Q})$ is minimized if and only if*

$$w_{il} = \begin{cases} 1 & \text{if } d_{euc}(\mathbf{x}_i, \hat{\mathbf{q}}_l) = \min_{1 \leq l \leq k} d_{euc}(\mathbf{x}_i, \hat{\mathbf{q}}_l), \\ 0 & \text{otherwise} \end{cases} \quad (9.3)$$

for $i = 1, 2, \dots, n$ and $l = 1, 2, \dots, k$.

Theorem 9.2. *In subproblem P_2 , let $\hat{W} = (\hat{w}_{il})$ be fixed. Then the function $P(\hat{W}, Q)$ is minimized if and only if*

$$\mathbf{q}_{lj} = \frac{\sum_{i=1}^n \hat{w}_{il} \mathbf{x}_{ij}}{\sum_{i=1}^n \hat{w}_{il}} \quad (9.4)$$

for $l = 1, 2, \dots, k$ and $j = 1, 2, \dots, d$.

ALGORITHM 9.2. The k -means algorithm treated as an optimization problem.

Require: Data set D , Number of Clusters k , Dimensions d :

- 1: Choose an initial Q^0 and solve $P(W, Q^0)$ to obtain W^0 ;
- 2: Let T be the number of iterations;
- 3: **for** $t = 0$ to T **do**
- 4: Let $\hat{W} \leftarrow W^t$ and solve $P(\hat{W}, Q)$ to obtain Q^{t+1} ;
- 5: **if** $P(\hat{W}, Q^t) = P(\hat{W}, Q^{t+1})$ **then**
- 6: Output \hat{W}, Q^t ;
- 7: Break;
- 8: **end if**
- 9: Let $\hat{Q} \leftarrow Q^{t+1}$ and solve $P(W^t, \hat{Q})$ to obtain W^{t+1} ;
- 10: **if** $P(W^t, \hat{Q}) = P(W^{t+1}, \hat{Q})$ **then**
- 11: Output W^t, \hat{Q} ;
- 12: Break;
- 13: **end if**
- 14: **end for**
- 15: Output W^{T+1}, Q^{T+1} .

The pseudocode of the optimization algorithm is described in Algorithm 9.2. The computational complexity of the algorithm is $O(nkd)$ per iteration (Phillips, 2002), where d is the dimension, k is the number of clusters, and n is the number of data points in the data set.

Since the sequence $P(\cdot, \cdot)$ generated by the algorithm is strictly decreasing, the algorithm will converge to a local minimum point after a finite number of iterations (Selim and Ismail, 1984). Convergence and some probability properties regarding the k -means

algorithm are also discussed in (Pollard, 1981), (Pollard, 1982), and (Serinko and Babu, 1992). García-Escudero and Gordaliza (1999) discussed the robustness properties of the k -means algorithm.

As one of the most often used clustering algorithms, the k -means algorithm has some important properties:

- It is efficient in clustering large data sets, since its computational complexity is linearly proportional to the size of the data sets.
- It often terminates at a local optimum (Anderberg, 1973; Selim and Ismail, 1984).
- The clusters have convex shapes, such as a ball in three-dimensional space (Anderberg, 1973).
- It works on numerical data.
- The performance is dependent on the initialization of the centers.

The k -means algorithm has some drawbacks (Peña et al., 1999). In particular, the performance is dependent on the initialization of the centers, as mentioned above. As a result, some methods for selecting good initial centers are proposed, for example, in (Babu and Murty, 1993) and (Bradley and Fayyad, 1998). Peña et al. (1999) provide a comparison of four initialization methods: a random method, Forgy's approach (Anderberg, 1973), Macqueen's approach (Macqueen, 1967), and Kaufman's approach (Kaufman and Rousseeuw, 1990). Other initialization methods are presented in (Khan and Ahmad, 2004).

In the iteration phase of the algorithm, the objects will be moved from one cluster to another in order to minimize the objective function. Tarsitano (2003) presents a computational study of the shortcomings and relative merits of 17 reallocation methods for the k -means algorithm.

Another drawback of the k -means algorithm is that it does not work effectively on high-dimensional data (Keim and Hinneburg, 1999). Also, working only on numerical data restricts some applications of the k -means algorithm.

The algorithm presented above is usually called the standard k -means algorithm. The standard k -means algorithm has several variations, such as the k -harmonic algorithm, the fuzzy k -means algorithm, and the Gaussian EM algorithm. Hamerly and Elkan (2002) investigated the properties of the standard k -means algorithm and its variations and alternatives.

9.2 Variations of the k -means Algorithm

Many clustering algorithms originating from the k -means algorithm are presented in (Faber, 1994), (Bradley and Fayyad, 1998), (Alsabti et al., 1998), and (Bottou and Bengio, 1995). These clustering algorithms were developed to improve the performance of the standard k -means algorithm. We will address some of these algorithms in subsequent sections.

9.2.1 The Continuous k -means Algorithm

The continuous k -means algorithm, proposed by Faber (1994), is faster than the standard k -means algorithm. It is different from the standard k -means algorithm in the following aspects. Firstly, in the continuous k -means algorithm, the prototypes (or reference points) are chosen as a random sample from the whole database, while in the standard k -means algorithm the initial points are chosen arbitrarily. Secondly, the data points are treated differently. During each complete iteration, the continuous k -means algorithm examines only a sample of the data points, while the standard k -means algorithm examines all the data points in sequence.

Theoretically, random sampling represents a return to Macqueen's original concept of the algorithm as a method of clustering data over a continuous space. In Macqueen's formulation, the error measure E_i for each region R_i is given by

$$E_i = \int_{R_i} \rho(\mathbf{x}) \|\mathbf{x} - \mathbf{z}_i\|^2 d\mathbf{x},$$

where $\rho(\cdot)$ is the probability distribution function, which is a continuous function defined over the space, and \mathbf{z}_i is the centroid of the region R_i . The sum of all the E_i 's is the total error measure.

A random sample of the data set can be a good estimate of the probability distribution function $\rho(\mathbf{x})$. Such a sample yields a representative set of centroids and a good estimate of the error measure without using all the data points in the original data set. Since both the reference points and the data points for updates are chosen by random sampling, the continuous k -means algorithm is generally faster than the standard k -means algorithm, and ten times faster than Lloyd's algorithm (Lloyd, 1982). Ways of further reducing the computer time are discussed in (Faber, 1994).

9.2.2 The Compare-means Algorithm

In order to accelerate the k -means algorithm, the algorithm compare-means (Phillips, 2002) uses a simple approach to avoid many unnecessary comparisons.

Let \mathbf{x} be a point in D and μ_i and μ_j be two means. By the triangle inequality, we have $d(\mathbf{x}, \mu_i) + d(\mathbf{x}, \mu_j) \geq d(\mu_i, \mu_j)$, so $d(\mathbf{x}, \mu_j) \geq d(\mu_i, \mu_j) - d(\mathbf{x}, \mu_i)$. Therefore, we have $d(\mathbf{x}, \mu_j) \geq d(\mathbf{x}, \mu_i)$ if $d(\mu_i, \mu_j) \geq 2d(\mathbf{x}, \mu_i)$. In this case, computing $d(\mathbf{x}, \mu_j)$ is unnecessary.

Since the number of clusters k is usually small, distances of all pairs of means are precomputed before each iteration. Then, before comparing a point \mathbf{x} to a mean μ_j , the above test is performed using the closest known mean to \mathbf{x} . The compare-means algorithm is described in Algorithm 9.3.

ALGORITHM 9.3. The compare-means algorithm.

Require: Data set D , Number of Clusters k , Dimensions d :

- { C_i is the i th cluster}
- {1. Initialization Phase}

```

1:  $(C_1, C_2, \dots, C_k)$  = Initial partition of  $D$ .
   {2. Iteration Phase}
2: repeat
3:   Calculate  $D_{ij} = d(\mu_i, \mu_j)$  for all  $i, j = 1, 2, \dots, k$  { $\mu_i$  is the mean of the  $i$ th cluster
   in the previous iteration};
4:   Let  $n_i$  be the subscript such that  $\mathbf{x}_i \in C_{n_i}$ ;
5:    $D_{min} \Leftarrow d(\mathbf{x}_i, \mu_{n_i})$ ;
6:   for  $j = 1$  to  $k$  do
7:     if  $D_{jn_i} < 2 * D_{min}$  and  $j \neq n_i$  then
8:        $dist = d(\mathbf{x}_i, \mu_j)$ ;
9:       if  $dist < D_{min}$  then
10:         $D_{min} \Leftarrow dist$ ;
11:         $n_i \Leftarrow j$ ;
12:      end if
13:    end if
14:   end for
15:   Assign case  $i$  to cluster  $n_i$ ;
16:   Recompute the cluster means of any changed clusters above;
17: until no further changes in cluster membership occur in a complete iteration
18: Output results;

```

The number of comparisons made by compare-means is harder to determine, but the overhead of compare-means is $\Theta(k^2d + nkd)$ (Phillips, 2002), where n is the number of records, k is the number of clusters, and d is the dimension.

9.2.3 The Sort-means Algorithm

The algorithm sort-means (Phillips, 2002) is an extension of compare-means. In this algorithm, the means are sorted in order of increasing distance from each mean in order to obtain a further speedup.

Let $D_{ij} = d(\mu_i, \mu_j)$ for $i, j = 1, 2, \dots, k$, where μ_i is the mean of the i th cluster. Let M be a $k \times k$ array in which row i ($m_{i1}, m_{i2}, \dots, m_{ik}$) is a permutation of $1, 2, \dots, k$ such that $d(\mu_i, \mu_{m_{i1}}) \leq d(\mu_i, \mu_{m_{i2}}) \leq \dots \leq d(\mu_i, \mu_{m_{ik}})$. An iteration of sort-means is described in Algorithm 9.4.

ALGORITHM 9.4. An iteration of the sort-means algorithm.

```

1: Calculate  $D_{ij} = d(\mu_i, \mu_j)$  for all  $i, j = 1, 2, \dots, k$  { $\mu_i$  is the mean of the  $i$ th cluster
   in the previous iteration};
2: Construct the array  $M$ ;
3: Let  $n_i$  be the subscript such that  $\mathbf{x}_i \in C_{n_i}$ ;
4:  $D_{inmin} \Leftarrow d(\mathbf{x}_i, \mu_{n_i})$ ;
5:  $D_{min} \Leftarrow D_{inmin}$ ;
6: for  $j = 2$  to  $k$  do
7:    $l \Leftarrow M_{n_i j}$ ;

```

```

8:   if  $D_{n_i l} \geq 2 * D_{inmin}$  then
9:     break;
10:    end if
11:     $dist = d(\mathbf{x}, \mu_l)$ ;
12:    if  $dist < D_{inmin}$  then
13:       $D_{min} \leftarrow dist$ ;
14:       $n_i \leftarrow l$ ;
15:    end if
16:  end for
17: Assign case  $i$  to cluster  $n_i$ ;
18: Recompute the cluster means of any changed clusters above;

```

For the sort-means algorithm, the running time of an iteration is $O(nd\gamma + k^2d + k^2 \log k)$ (Phillips, 2002), where n is the number of records, k is the number of clusters, d is the dimension, and γ is the average over all points \mathbf{x} of the number of means that are no more than twice as far as \mathbf{x} is from the mean \mathbf{x} was assigned to in the previous iteration.

9.2.4 Acceleration of the k -means Algorithm with the kd -tree

Pelleg and Moore (1999) proposed an algorithm for the k -means clustering problem using the kd -tree data structure. The kd -tree data structure, described in Appendix B, can be used to reduce the large number of nearest-neighbor queries issued by the traditional k -means algorithm. Hence, an analysis of the geometry of the current cluster centers can lead to a great reduction in the work needed to update the cluster centers. In addition, the initial centers of the k -means algorithm can be chosen by the kd -tree efficiently.

One way to use the kd -tree in the inner loop of the k -means algorithm is to store the centers in the tree; another way is to store the whole data set in the tree. The latter method is used in (Pelleg and Moore, 1999). To describe the application of the kd -tree in the k -means algorithm, let us start with an iteration of the k -means algorithm.

Let $C^{(i)}$ denote the set of centroids after the i th iteration. Before the first iteration, $C^{(0)}$ is initialized to a set of random values. The stop criterion of the algorithm is that $C^{(i)}$ and $C^{(i-1)}$ are identical. In each iteration of the algorithm, the following two steps are performed:

1. For each data point \mathbf{x} , find the center in $C^{(i)}$ that is closest to \mathbf{x} and associate \mathbf{x} with this center.
2. Update $C^{(i)}$ to $C^{(i+1)}$ by taking, for each center, the center of mass of all the data points associated with this center.

Pelleg's algorithm involves modifying the second step in the iteration. The procedure to update the centroids in $C^{(i)}$ is recursive and has a parameter, a hyperrectangle h . The procedure starts with the initial value of h being the hyperrectangle containing all the input points. If the procedure can find $owner_{C^{(i)}}(h)$, it updates its counters using the center of mass and number of points that are stored in the kd -tree node corresponding to h ; otherwise, it splits h by recursively calling itself with the children of h . Hence, given a set of centroids

C and a hyperrectangle h , $\text{owner}_C(h)$ is defined to be a center \mathbf{z} in C such that any point in h is closer to \mathbf{z} than to any other center in C , if such a center exists.

Performance comparison with BIRCH (Zhang et al., 1996) was presented in (Pelleg and Moore, 1999). Pelleg's method performs badly for high-dimensional (e.g., > 8) data but scales very well with the number of centers. Interested readers are referred to (Alsabti et al., 1998) and (Kanungo et al., 2002) for other examples of applying kd -tree in the k -means algorithm.

9.2.5 Other Acceleration Methods

We presented some methods previously for improving the performance of the k -means algorithm. Besides the above-mentioned methods, several other extensions of the standard k -means are proposed in order to improve the speed and quality of the k -means algorithm.

In order to improve the performance of the k -means algorithm in terms of solution quality and robustness, Chen et al. (2004) proposed a clustering algorithm that integrates the concepts of hierarchical approaches and the k -means algorithm. The initialization phase of this algorithm is similar to that of the k -means algorithm except that the number of initial centers m is larger than the number of clusters k . The iteration phase is the same as that of the k -means algorithm. The last phase is to merge clusters until k clusters are formed. The pair of clusters with the smallest score values will be merged into one cluster. The score between clusters C_i and C_j is defined as

$$\text{Score}(C_i, C_j) = \frac{|C_i||C_j|}{|C_i| + |C_j|} d_{\text{euc}}^2(\mu(C_i), \mu(C_j)),$$

where $\mu(C_i)$ and $\mu(C_j)$ are the centers of clusters C_i and C_j , respectively, and $d_{\text{euc}}(\cdot, \cdot)$ is the Euclidean distance.

Matoušek (2000) proposed a $(1 + \epsilon)$ -approximate (i.e., the objective function value of the approximation is no worse than $(1 + \epsilon)$ times the minimum value of the objective function) k -clustering algorithm whose complexity is

$$O(n \log^k n \epsilon^{-2k^2d})$$

for $k \geq 3$, where $\epsilon > 0$.

Har-Peled and Mazumdar (2004) proposed a similar approximation algorithm for the k -means by applying the k -means algorithm to, instead of the original data set D , a small weighted set $S \subset D$, of size $O(k\epsilon^{-d} \log n)$, where ϵ is a positive number, n is the number of objects in D , d is the dimensionality, and k is the number of clusters. It has been shown that the complexity of the approximation algorithm is

$$O\left(n + k^{k+2}\epsilon^{-(2d+1)k} \log^{k+1} n \log^k \frac{1}{\epsilon}\right),$$

which is linear to n for fixed k and ϵ . Details of this algorithm are omitted, but interested readers are referred to (Har-Peled and Mazumdar, 2004) for how the core set is computed.

Su and Chou (2001) proposed a modified version of the k -means algorithm that adopts a nonmetric distance measure based on the idea of “point symmetry.” Precisely, for a given

data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ and a center \mathbf{z} , the point symmetry distance between an object \mathbf{x}_i and the center \mathbf{z} is defined as

$$d(\mathbf{x}_i, \mathbf{z}) = \min_{1 \leq j \leq n, j \neq i} \frac{\|(\mathbf{x}_i - \mathbf{z}) + (\mathbf{x}_j - \mathbf{z})\|}{\|\mathbf{x}_i - \mathbf{z}\| + \|\mathbf{x}_j - \mathbf{z}\|}.$$

An application of this algorithm for human face detection is presented in Su and Chou (2001).

In order to handle high-dimensional data, Stute and Zhu (1995) proposed a modified version of the k -means algorithm based on the projection pursuit, and Agarwal and Mustafa (2004) proposed an extension of the k -means algorithm for projective clustering in arbitrary subspaces with techniques to avoid local minima. Kantabutra and Couch (2000) implemented a parallel k -means algorithm to handle large databases.

9.3 The Trimmed k -means Algorithm

The trimmed k -means algorithm (Cuesta-Albertos et al., 1997), based on “impartial trimming,” is a procedure that is more robust than the standard k -means algorithm. The main idea of the trimmed k -means algorithm is presented in this section.

The k -means algorithm can be viewed as a procedure based on the minimization of the expected value of a “penalty function” Φ of the distance to k -sets (sets of k points) through the following problem: Given an \mathbb{R}^d -valued random vector X , find the k -set $M = \{\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_k\}$ in \mathbb{R}^d such that

$$V_\Phi(M) = \int \Phi \left(\inf_{i=1,2,\dots,k} \|X - \mathbf{m}_i\| \right) dP$$

is minimized.

The trimmed k -means procedure based on the methodology of “impartial trimming,” which is a way to obtain a trimmed set with the lowest possible variation at some given level α , is formulated as follows (Cuesta-Albertos et al., 1997).

Let $\alpha \in (0, 1)$, the number of clusters k , and the penalty function Φ be given. For every set A such that $P(A) \geq 1 - \alpha$ and every k -set $M = \{\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_k\}$ in \mathbb{R}^d , the variation of M given A is defined as

$$V_\Phi^A(M) = \frac{1}{P(A)} \int_A \Phi \left(\inf_{i=1,2,\dots,k} \|X - \mathbf{m}_i\| \right) dP.$$

$V_\Phi^A(M)$ measures how well the set M represents the probability mass of P living on A . To find the best representation of the “more adequate” set containing a given amount of probability mass, we can minimize $V_\Phi^A(M)$ on A and M in the following way:

1. Obtain the k -variation given A , $V_{k,\Phi}^A$, by minimizing with respect to M :

$$V_{k,\Phi}^A = \inf_{M \subset \mathbb{R}^d, |M|=k} V_\Phi^A(M).$$

2. Obtain the trimmed k -variation $V_{k,\Phi,\alpha}$ by minimizing with respect to A :

$$V_{k,\Phi,\alpha} = V_{k,\Phi,\alpha}(X) = V_{k,\Phi,\alpha}(P_X) = \inf_{A \in \beta^d, P(A) \geq 1-\alpha} V_{k,\Phi}^A.$$

The goal of the algorithm is to obtain a trimmed set A_0 and a k -set M_0 , if both of them exist, through the condition

$$V_{\Phi}^{A_0}(M_0) = V_{k,\Phi,\alpha}.$$

The trimmed k -means algorithm described above can be generalized as follows. Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a sample of independently identically distributed random variables in \mathbb{R}^d with common distribution F . Let $\Phi : \mathbb{R}^+ \rightarrow \mathbb{R}$ be a suitable nondecreasing penalty function and $1 - \gamma \in (0, 1)$ be a trimming level. Then the generalized trimmed k -means of D is a k -set $\{\mathbf{m}_1^*, \mathbf{m}_2^*, \dots, \mathbf{m}_k^*\} \subset \mathbb{R}^d$ solving the double optimization problem

$$\min_Y \min_{\{\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_k\} \subset \mathbb{R}^d} \frac{1}{\lfloor n\gamma \rfloor} \sum_{\mathbf{x} \in Y} \Phi \left(\inf_{1 \leq j \leq k} \|\mathbf{x} - \mathbf{m}_j\| \right),$$

where Y ranges in the class of the subsets of D with $\lfloor n\gamma \rfloor$ data points, and $\lfloor x \rfloor$ denotes the smallest integer greater than or equal to x .

The properties of existence and consistency of the trimmed k -means are shown to hold under certain conditions. Details of the theorems are omitted here; interested readers are referred to (Cuesta-Albertos et al., 1997). A central limit theory for the generalized trimmed k -means algorithm is given in (García-Escudero et al., 1999b). García-Escudero and Gordaliza (1999) investigated the performance of the generalized k -means algorithm and the generalized trimmed k -means algorithm from the viewpoint of Hampel's robustness criteria (Hampel, 1971). Further discussions of the trimmed k -means algorithm are given in (García-Escudero et al., 1999a).

9.4 The x -means Algorithm

In the k -means algorithm, the number of clusters k is an input parameter specified by the user. In order to reveal the true number of clusters underlying the distribution, Pelleg and Moore (2000) proposed an algorithm, called x -means, by optimizing the Bayesian information criterion (BIC) or the Akaike information criterion (AIC) measure (Bozdogan, 1987).

In the x -means algorithm, the BIC or Schwarz criterion (Kass and Raftery, 1995; Schwarz, 1978) is used globally and locally in order to find the best number of clusters k . Given a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ containing n objects in a d -dimensional space and a family of alternative models $M_j = \{C_1, C_2, \dots, C_k\}$, (e.g., different models correspond to solutions with different values of k), the posterior probabilities $P(M_j | D)$ are used to score the models. The Schwarz criterion can be used to approximate the posteriors.

The Schwarz criterion is defined as

$$BIC(M_j) = \hat{l}_j(D) - \frac{p_j}{2} \log n,$$

where $\hat{l}_j(D)$ is the loglikelihood of D according to the j th model and taken at the maximum likelihood point, and p_j is the number of parameters in M_j . The model with the largest score is selected.

Under the identical spherical Gaussian distribution, the maximum likelihood estimate for variance is

$$\hat{\sigma}^2 = \frac{1}{n - k} \sum_{i=1}^n (\mathbf{x}_i - \mu_{(i)})^2,$$

where $\mu_{(i)}$ is the centroid associated with the object \mathbf{x}_i , i.e., (i) denotes the index of the centroid that is closest to \mathbf{x}_i . The point probabilities are

$$\hat{P}(\mathbf{x}_i) = \frac{|C_{(i)}|}{n} \cdot \frac{1}{\sqrt{2\pi}\hat{\sigma}^d} \exp\left(-\frac{1}{2\hat{\sigma}^2} \|\mathbf{x}_i - \mu_{(i)}\|^2\right).$$

Thus, the loglikelihood of the data is

$$l(D) = \log \prod_{i=1}^n P(\mathbf{x}_i) = \sum_{i=1}^n \left(\log \frac{1}{\sqrt{2\pi}\hat{\sigma}^d} - \frac{1}{2\hat{\sigma}^2} \|\mathbf{x}_i - \mu_{(i)}\|^2 + \log \frac{|C_{(i)}|}{n} \right).$$

The number of free parameters p_j is $k - 1 + dk + 1 = (d + 1)k$.

The Schwarz criterion is used in x -means globally to choose the best model it encounters and locally to guide all centroid splits. The algorithm can be briefly described as follows.

Given a range for k , $[k_{min}, k_{max}]$, the x -means algorithm starts with $k = k_{min}$ and continues to add centroids when they are needed until the upper bound is reached. New centroids are added by splitting some centroids into two according to the Schwarz criterion. During the process, the centroid set with the best score is recorded as the one that is the final output. The algorithm can be implemented efficiently using ideas of “blacklisting”(Pelleg and Moore, 1999) and kd -trees.

9.5 The k -harmonic Means Algorithm

k -harmonic means (Zhang et al., 2000a, 1999) is developed from the k -means algorithm and it is essentially insensitive to the initialization of centers.

We know that the error function (or performance function) of the k -means algorithm can be written as

$$E = \sum_{i=1}^n \min\{d(\mathbf{x}_i, \mu_j), j = 1, 2, \dots, k\}, \quad (9.5)$$

where μ_j is the mean of the j th cluster.

Then the error function of the k -harmonic means algorithm is obtained by replacing the minimum function $\min(\cdot)$ by the harmonic average (or harmonic mean) function $\text{HA}(\cdot)$

and using the squared Euclidean distance, i.e.,

$$\begin{aligned} E &= \sum_{i=1}^n \text{HA}(\{d_{seuc}(\mathbf{x}_i, \mu_j), j = 1, 2, \dots, k\}) \\ &= \sum_{i=1}^n \frac{k}{\sum_{j=1}^k \frac{1}{(\mathbf{x}_i - \mu_j)^T (\mathbf{x}_i - \mu_j)}}, \end{aligned} \quad (9.6)$$

where μ_j is the mean of the j th cluster, $d_{seuc}(\cdot, \cdot)$ is the squared Euclidean distance, and $\text{HA}(\cdot)$ is the harmonic average defined as

$$\text{HA}(\{a_i : i = 1, 2, \dots, m\}) = \frac{m}{\sum_{i=1}^m a_i^{-1}}. \quad (9.7)$$

The recursive formula for the k -harmonic means algorithm can be obtained by taking partial derivatives of the error function (9.6) with respect to the means $\mu_l, l = 1, 2, \dots, k$, and setting them to zero. That is,

$$\frac{\partial E}{\partial \mu_l} = -k \sum_{i=1}^n \frac{2(\mathbf{x}_i - \mu_l)}{d_{il}^4 \left(\sum_{j=1}^k d_{ij}^{-2} \right)^2} = \mathbf{0}, \quad (9.8)$$

where $d_{ij} = d_{euc}(\mathbf{x}_i, \mu_j) = [(\mathbf{x}_i - \mu_j)^T (\mathbf{x}_i - \mu_j)]^{\frac{1}{2}}$.

By solving equation (9.8), we obtain new centers $\mu_l^*, l = 1, 2, \dots, k$ as follows:

$$\mu_l^* = \frac{\sum_{i=1}^n d_{il}^{-4} \left(\sum_{j=1}^k d_{ij}^{-2} \right)^{-2} \mathbf{x}_i}{\sum_{i=1}^n d_{il}^{-4} \left(\sum_{j=1}^k d_{ij}^{-2} \right)^{-2}}. \quad (9.9)$$

Then given a set of initial centers, we can obtain new centers by (9.9). This recursion is continued until the centers stabilize.

In order to reduce the sensitivity of the convergence quality to the initial centers, Zhang et al. (2000) proposed a generalized k -harmonic means algorithm as follows:

$$\mu_l^* = \frac{\sum_{i=1}^n d_{il}^{-s} \left(\sum_{j=1}^k d_{ij}^{-2} \right)^{-2} \mathbf{x}_i}{\sum_{i=1}^n d_{il}^{-s} \left(\sum_{j=1}^k d_{ij}^{-2} \right)^{-2}} \quad (9.10)$$

for $l = 1, 2, \dots, k$, where s is a parameter. Unfortunately, no method has been developed to choose the parameter s .

9.6 The Mean Shift Algorithm

The mean shift algorithm (Fukunaga and Hostetler, 1975; Cheng, 1995; Comaniciu and Meer, 2002, 1999) is a simple iterative procedure that shifts each data point to the average of data points in its neighborhood. To introduce the mean shift algorithm, let us start with some definitions and notation.

Definition 9.3 (Profile). A profile k is a function $k : [0, \infty] \rightarrow [0, \infty]$ satisfying the following conditions:

1. k is nonincreasing,
2. k is piecewise continuous, and
3. $\int_0^\infty k(r)dr < \infty$.

Definition 9.4 (Kernel). A function $K : \mathbb{R}^d \rightarrow \mathbb{R}$ is said to be a kernel if there exists a profile k such that

$$K(\mathbf{x}) = k(\|\mathbf{x}\|^2),$$

where $\|\cdot\|$ denotes the Euclidean norm.

Let $\alpha > 0$. If K is a kernel, then

$$\begin{aligned} (\alpha K)(\mathbf{x}) &= \alpha K(\mathbf{x}), \\ K_\alpha(\mathbf{x}) &= K\left(\frac{\mathbf{x}}{\alpha}\right), \\ (K^\alpha)(\mathbf{x}) &= (K(\mathbf{x}))^\alpha \end{aligned}$$

are all kernels.

Definition 9.5 (The mean shift algorithm). Let $D \subset \mathbb{R}^d$ be a finite data set, K a kernel, and $w : D \rightarrow (0, \infty)$ a weight function. The sample mean with kernel K at $\mathbf{x} \in \mathbb{R}^d$ is defined as

$$m(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in D} K(\mathbf{y} - \mathbf{x})w(\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y} \in D} K(\mathbf{y} - \mathbf{x})w(\mathbf{y})}.$$

Let $T \subset \mathbb{R}^d$ be a finite set of cluster centers. The evolution of T in the form of iterations $T \leftarrow m(T)$ with $m(T) = \{m(\mathbf{y}) : \mathbf{y} \in T\}$ is called the mean shift algorithm.

The mean shift algorithm is a very general iterative procedure to the extent that some well-known clustering algorithms are its special cases. The maximum-entropy clustering (MEC) algorithm (Rose et al., 1990), for example, is a mean shift algorithm when T and D are separate sets, $G^\beta(\mathbf{x}) = e^{-\beta\|\mathbf{x}\|^2}$ is the kernel, and

$$w(\mathbf{y}) = \frac{1}{\sum_{\mathbf{t} \in T} G^\beta(\mathbf{y} - \mathbf{t})}, \quad \mathbf{y} \in D.$$

In addition, the well-known k -means algorithm is a limiting case of the mean shift algorithm (Cheng, 1995).

We now introduce some definitions in order to describe the convergence properties of the mean shift algorithm.

Definition 9.6 (Direction). A direction in \mathbb{R}^d is a point on the unit sphere, i.e., \mathbf{a} is a direction if and only if $|\mathbf{a}| = 1$.

Definition 9.7 (Projection). The projection in the direction \mathbf{a} is defined as the mapping $\pi_{\mathbf{a}} : \mathbb{R}^d \rightarrow \mathbb{R}$ with $\pi_{\mathbf{a}}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{a} \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product.

Definition 9.8 (Convex hull). The convex hull $h(Y)$ of a set $Y \subset \mathbb{R}^d$ is defined as

$$\bigcap_{\|\mathbf{a}\|=1} \{\mathbf{x} \in \mathbb{R}^d : \min \pi_{\mathbf{a}}(Y) \leq \pi_{\mathbf{a}}(\mathbf{x}) \leq \max \pi_{\mathbf{a}}(Y)\}.$$

Definition 9.9 (Translation). $h(D) \supseteq h(m(D)) \supseteq h(m(m(D))) \supseteq \dots$, i.e., a translation is a transformation of the data so that the origin is in all the convex hulls of data.

Definition 9.10 (Radius). Suppose after a translation, the origin is in all the convex hulls of data. Then the radius of data is

$$\rho(D) = \max\{\|\mathbf{x}\| : \mathbf{x} \in D\}.$$

Definition 9.11 (Diameter). The diameter of data is defined as

$$d(D) = \sup_{\|\mathbf{a}\|} (\max \pi_{\mathbf{a}}(D) - \min \pi_{\mathbf{a}}(D)).$$

Regarding the convergence of the mean shift algorithm, Cheng (1995) proved the following two theorems.

Theorem 9.12 (Convergence with broad kernels). Let k be the profile of the kernel used in a blurring process and S_0 be the initial data. If $k(d^2(S_0)) \geq \kappa$ for some $\kappa > 0$, then the diameter of the data approaches zero. The convergence rate is at least as fast as

$$\frac{d(m(D))}{d(D)} \leq 1 - \frac{\kappa}{4k(0)}.$$

Theorem 9.13 (Convergence with truncated kernels). If data points cannot move arbitrarily close to each other and $K(\mathbf{x})$ is either zero or larger than a fixed positive constant, then the blurring process reaches a fixed point in finitely many iterations.

The mean shift algorithm is not only an intuitive and basic procedure but also a deterministic process. It is more efficient than gradient descent or ascent methods in terms

of adapting to the right step size (Cheng, 1995). There are also some factors that make the mean shift algorithm not popular. For example, the computational cost of an iteration of the mean shift algorithm is $O(n^2)$ (Cheng, 1995), where n is the number of data points in the data set. The mean shift algorithm is also not suitable for high-dimensional data sets and large data sets. Other discussions of the mean shift algorithm can be found in Fashing and Tomasi (2005), Yang et al. (2003a), Chen and Meer (2005), and Georgescu et al. (2003).

9.7 MEC

The MEC algorithm, based on statistical physics, was introduced by Rose et al. (1990). The MEC algorithm is a fuzzy clustering algorithm and the fuzzy membership is obtained by maximizing the entropy at a given average variance. A deterministic annealing process is derived from the relationship between the corresponding Lagrange multiplier and the “temperature” so that the free energy is minimized at each temperature.

The energy or cost contributed to the cluster C_j by a data point \mathbf{x} is denoted by $E_j(\mathbf{x})$. In MEC, the energy $E_j(\mathbf{x})$ is defined as

$$E_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{z}_j\|^2,$$

where $\|\cdot\|$ is the Euclidean norm and \mathbf{z}_j is the centroid of C_j . The average total energy for a given partition is defined as

$$E = \sum_{\mathbf{x} \in D} \sum_{j=1}^k P(\mathbf{x} \in C_j) E_j(\mathbf{x}), \quad (9.11)$$

where $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ is the data set under consideration, k is the number of clusters, and $P(\mathbf{x} \in C_j)$, $j = 1, 2, \dots, k$, are the association probabilities or fuzzy memberships. The association probabilities that maximize the entropy under the constraint (9.11) are Gibbs distributions defined as

$$P(\mathbf{x} \in C_j) = \frac{e^{-\beta E_j(\mathbf{x})}}{Z_{\mathbf{x}}}, \quad (9.12)$$

where $Z_{\mathbf{x}}$ is the partition function defined as

$$Z_{\mathbf{x}} = \sum_{j=1}^k e^{-\beta E_j(\mathbf{x})}.$$

The parameter β is the Lagrange multiplier determined by the given value of E in equation (9.11). The total partition function is defined as

$$Z = \prod_{\mathbf{x} \in D} Z_{\mathbf{x}}.$$

Based on the partition function, the free energy is defined as

$$F = -\frac{1}{\beta} \ln Z = -\frac{1}{\beta} \sum_{\mathbf{x} \in D} \ln \left(\sum_{j=1}^k e^{-\beta \|\mathbf{x} - \mathbf{z}_j\|^2} \right). \quad (9.13)$$

The set of centroids \mathbf{z}_j that optimizes the free energy satisfies

$$\frac{\partial F}{\partial \mathbf{z}_j} = 0 \quad \forall j$$

or

$$\sum_{\mathbf{x} \in D} \frac{(\mathbf{x} - \mathbf{z}_j) e^{-\beta \|\mathbf{x} - \mathbf{z}_j\|^2}}{\sum_{l=1}^k e^{-\beta \|\mathbf{x} - \mathbf{z}_l\|^2}} = 0 \quad \forall j,$$

which leads to

$$\mathbf{z}_j = \frac{\sum_{\mathbf{x} \in D} \mathbf{x} P(\mathbf{x} \in C_j)}{\sum_{\mathbf{x} \in D} P(\mathbf{x} \in C_j)}.$$

The MEC algorithm is an iterative process $\mathbf{z}^{(r)} \rightarrow \mathbf{z}^{(r+1)}$, $r = 1, 2, \dots$. Note that the MEC algorithm is a special case of the mean shift algorithm (Cheng, 1995) and the k -means algorithm is the limiting case of the MEC algorithm when β approaches infinity (Cheng, 1995).

9.8 The k -modes Algorithm (Huang)

The k -modes algorithm (Huang, 1997b, 1998) comes from the k -means algorithm (see Section 9.1), and it was designed to cluster categorical data sets. The main idea of the k -modes algorithm is to specify the number of clusters (say, k) and then to select k initial modes, followed by allocating every object to the nearest mode.

The k -modes algorithm uses the simple match dissimilarity measure (see Section 6.3.1) to measure the distance of categorical objects. The mode of a cluster is defined as follows.

Let D be a set of categorical objects described by d categorical attributes, A_1, A_2, \dots, A_d . Let $X \subseteq D$. Then the mode of X is defined to be a vector $\mathbf{q} = (q_1, q_2, \dots, q_d)$ such that the function

$$D(X, \mathbf{q}) = \sum_{\mathbf{x} \in X} d_{sim}(\mathbf{x}, \mathbf{q}) \tag{9.14}$$

is minimized, where $d_{sim}(\cdot, \cdot)$ is defined in (6.23).

Hence, according to this definition, the mode is not necessarily an element of that data set. The following theorem (Huang, 1998) shows how to minimize the function given in (9.14).

Theorem 9.14. *Let the domain of A_j be $\text{DOM}(A_j) = \{A_{j1}, A_{j2}, \dots, A_{jn_j}\}$ for $j = 1, 2, \dots, d$, and let $X \subseteq D$. Let $f_{jr}(X)$ ($1 \leq j \leq d$, $1 \leq r \leq n_j$) be the number of objects in X that take value A_{jr} at the j th attribute, i.e.,*

$$f_{jr}(X) = |\{\mathbf{x} \in X : \mathbf{x}_j = A_{jr}\}|. \tag{9.15}$$

Then the function given in (9.14) is minimized if and only if $q_j \in \text{DOM}(A_j)$ for $j = 1, 2, \dots, d$, and

$$f_{jr_j}(X) \geq f_{jl}(X) \forall l \neq r_j, j = 1, 2, \dots, d,$$

where r_j is the subscript defined as $q_j = A_{jr_j}$ for $j = 1, 2, \dots, d$.

Theorem 9.14 provides us with a way to find \mathbf{q} for a given data set X . This theorem also implies that the mode of a data set is not necessarily unique.

Since the k -modes algorithm comes from the k -means algorithm, it can also be treated as an optimization problem. The objective function for the k -modes algorithm can be defined as in (9.2) by changing the Euclidean distance to the simple matching distance, i.e.,

$$P(W, Q) = \sum_{l=1}^k \sum_{i=1}^n w_{il} d_{sim}(\mathbf{x}_i, \mathbf{q}_l), \quad (9.16)$$

where $Q = \{\mathbf{q}_l, l = 1, 2, \dots, k\}$ is a set of objects, $d_{sim}(\cdot, \cdot)$ is the simple matching distance defined in (6.23), and W is an $n \times k$ matrix that satisfies the following conditions:

1. $w_{il} \in \{0, 1\}$ for $i = 1, 2, \dots, n, l = 1, 2, \dots, k$,
2. $\sum_{l=1}^k w_{il} = 1$ for $i = 1, 2, \dots, n$.

Thus, Algorithm 9.5 can be used for the k -modes algorithm by using the objective function defined in (9.16). But this algorithm is not efficient, since we need to calculate the total cost P of the whole data set each time a new Q or W is obtained. To make the computation more efficient, we use the algorithm described in Algorithm 9.5 (Huang, 1998).

ALGORITHM 9.5. The k -modes algorithm.

Require: Data set D , Number of Clusters k , Dimensions d :

- 1: Select k initial modes $Q = \{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_k\}$, and \mathbf{q}_l for cluster l ;
- 2: **for** $i = 1$ to n **do**
- 3: Find an l such that $d_{sim}(\mathbf{x}_i, \mathbf{q}_l) = \min_{1 \leq t \leq k} d_{sim}(\mathbf{x}_i, \mathbf{q}_t)$;
- 4: Allocate \mathbf{x}_i to cluster l ;
- 5: Update the mode \mathbf{q}_l for cluster l ;
- 6: **end for**
- 7: **repeat**
- 8: **for** $i = 1$ to n **do**
- 9: Let l_0 be the index of the cluster to which \mathbf{x}_i belongs;
- 10: Find an l_1 such that $d_{sim}(\mathbf{x}_i, \mathbf{q}_{l_1}) = \min_{1 \leq t \leq k, t \neq l_0} d_{sim}(\mathbf{x}_i, \mathbf{q}_t)$;
- 11: **if** $d_{sim}(\mathbf{x}_i, \mathbf{q}_{l_1}) < d_{sim}(\mathbf{x}_i, \mathbf{q}_{l_0})$ **then**
- 12: Reallocate \mathbf{x}_i to cluster l_1 ;
- 13: Update \mathbf{q}_{l_0} and \mathbf{q}_{l_1} ;
- 14: **end if**
- 15: **end for**
- 16: **until** No changes in cluster membership
- 17: Output results.

The proof of convergence for this algorithm is not available (Anderberg, 1973), but its practical use has shown that it always converges (Huang, 1998).

The k -modes algorithm is very popular for clustering categorical data. It has some important properties:

- It is efficient for clustering large data sets.
- It also produces locally optimal solutions that are dependent on initial modes and the order of objects in the data set (Huang, 1998).
- It works only on categorical data.

9.8.1 Initial Modes Selection

Since the clustering results and convergence speed of the k -modes algorithm are dependent on the initial modes, the selection of initial modes is an important issue in the k -modes algorithm. Good initial modes lead to fast convergence and good results, while bad initial modes lead to slow convergence. Many initial modes selection methods have been discussed in the literature. A commonly used approach, called the direct method, is to choose the first k distinct objects as initial modes. For example, for a given data set $D = \{\mathbf{x}_i, i = 1, 2, \dots, n\}$, we choose $\mathbf{q}_l = \mathbf{x}_l$ for $l = 1, 2, \dots, k$ as modes if $\mathbf{x}_l \neq \mathbf{x}_t$ for all $1 \leq l < t \leq k$. Another approach, called the diverse modes method, is to spread the initial modes over the whole data set by assigning the most frequent categories equally to the initial modes (Huang, 1998). Given a data set D , we first sort each column of its symbol table T_s such that each column of its corresponding frequency table of D is in decreasing order. In other words for each j , we sort $A_{j1}, A_{j2}, \dots, A_{jn_j}$ such that $f_{j1}(D) \geq f_{j2} \geq \dots \geq f_{jn_j}(D)$. Secondly, the most frequent categories are equally assigned to the initial modes $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_k$. For example, $A_{11}, A_{21}, \dots, A_{d1}$ are in different initial modes. Finally, we start with \mathbf{q}_1 , select the record most similar to \mathbf{q}_1 , and replace \mathbf{q}_1 as the first initial mode. After \mathbf{q}_i ($i = 1, 2, \dots, l$) are replaced, we select the record in D most similar to \mathbf{q}_{l+1} and replace \mathbf{q}_{l+1} with that record as the $(l + 1)$ th initial mode. We keep doing this until \mathbf{q}_k is replaced.

The last step is taken to avoid the occurrence of an empty cluster. The initial modes found by this method are diverse in the data set. These initial modes can lead to better clustering results, but this costs time.

9.9 The k -modes Algorithm (Chaturvedi et al.)

Chaturvedi et al. (2001) proposed a nonparametric bilinear model to derive clusters from categorical data. The clustering procedure is analogous to the traditional k -means algorithm (Macqueen, 1967). To describe the algorithm, let us begin with the bilinear clustering model.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set with n objects, each of which is described by d categorical attributes. Let k be the number of clusters. Then the bilinear clustering model is (Chaturvedi et al., 2001)

$$C = SW + error, \quad (9.17)$$

where C is an $n \times d$ data matrix; S is an $n \times k$ binary indicator matrix for membership of the n objects in k mutually exclusive, nonoverlapping clusters (i.e., the (i, j) th entry of S is 1 if \mathbf{x}_i belongs to the j th clusters, and 0 otherwise); and W is the matrix of centroids.

The data matrix C in equation (9.17) is known, whereas both S and W are unknown and must be estimated. The algorithm iterates as follows: estimate S given estimates of W , and then revise the estimates of W given the new estimates of S . This process will be repeated until the quality of clustering is not improved. In this algorithm, the quality of clustering is indicated by an L_0 loss function.

Let $\hat{C} = SW$. Then the L_p -norm-based loss function is defined as

$$L_p = \sum_{i=1}^n \sum_{j=1}^d |c_{ij} - \hat{c}_{ij}|^p$$

for positive values of $p \rightarrow 0$, where c_{ij} and \hat{c}_{ij} are the (i, j) th entries of C and \hat{C} , respectively. L_0 is the limiting case as $p \rightarrow 0$ and simply counts the number of mismatches in the matrices C and \hat{C} , i.e.,

$$L_0 = \sum_{i=1}^n \sum_{j=1}^d \delta(c_{ij}, \hat{c}_{ij}),$$

where $\delta(\cdot, \cdot)$ is defined in equation (6.22).

The goal of the algorithm is to minimize L_0 . The matrices S and W are estimated iteratively until the value of the L_0 loss function is not improved. The detailed estimation procedure is described as follows.

To estimate $S = (s_{il})$ given the estimates of $W = (w_{lj})$, we consider the functions

$$f_i = \sum_{j=1}^d \left(c_{ij} - \sum_{l=1}^k s_{il} w_{lj} \right)^0 = \sum_{l=1}^k s_{il} \left(\sum_{j=1}^d \delta(c_{ij}, w_{lj}) \right)$$

for $i = 1, 2, \dots, n$. Then $L_0 = \sum_{i=1}^n f_i$. To minimize L_0 , we can separately minimize f_i . To minimize f_i , we try all the d patterns $s_{il} = 1$ for $l = 1, 2, \dots, d$ and choose $\hat{s}_{il_0} = 1$ if

$$\sum_{j=1}^d \delta(c_{ij}, w_{l_0 j}) = \min_{1 \leq l \leq k} \sum_{j=1}^d \delta(c_{ij}, w_{lj}).$$

To estimate $W = (w_{lj})$ given the estimates of $S = (s_{il})$, we can estimate w_{lj} separately. Precisely, let C_l be the l th cluster, i.e., $C_l = \{\mathbf{x}_i : s_{il} = 1, 1 \leq i \leq n\}$, and consider the mode of $\{x_j : \mathbf{x} \in C_l\}$, where x_j is the j th attribute value of \mathbf{x} . Let \hat{w}_{lj} be the mode of $\{x_j : \mathbf{x} \in C_l\}$.

Although the above k -modes algorithm is faster than other procedures in some cases, such as the latent class procedure (Goodman, 1974), it has some disadvantages. Firstly, it can only guarantee a locally optimal solution. Secondly, the number of clusters k is required. In order to achieve a globally optimal solution, Gan et al. (2005) proposed a genetic k -modes algorithm based on the k -modes algorithm and the genetic algorithm.

9.10 The k -probabilities Algorithm

The k -probabilities algorithm (Wishart, 2002) is an extension of the k -modes algorithm. It was designed for clustering mixed-type data sets. The k -probabilities algorithm uses the

general distance coefficient (Gower, 1971) measure between two records, and it uses the squared distance to compute the distance between a case and a cluster; for instance, the distance d_{ip} between any case i and a cluster p is defined as

$$d_{ip}^2 = \sum_k \frac{w_{ipk}(x_{ik} - \mu_{pk})^2}{\sum_k w_{ipk}}, \quad (9.18)$$

where x_{ik} is the value of the k th variable for case i , μ_{pk} is the mean of the k th variable for cluster p , and w_{ipk} is a weight of 1 or 0 depending on whether or not the comparison between case i and cluster p is valid for the k th variable, i.e., $w_{ipk} = 1$ if we can compare the k th variable between case i and cluster p ; otherwise $w_{ipk} = 0$. Notice that for nominal variables, the mean μ_{pk} is a vector φ_{pks} of probabilities for each state s of the k th variable within cluster p .

The object function of this algorithm is

$$E = \sum_p E_p, \quad (9.19)$$

where E_p is the Euclidean sum of squares defined as

$$E_p = \sum_{i \in p} n_i \sum_k \frac{w_k(x_{ik} - \mu_{pk})^2}{\sum_k w_k}, \quad (9.20)$$

where x_{ik} and μ_{pk} are the same as in equation (9.18), n_i is a differential weight for case i (normally 1), and w_k is a differential weight for the k th variable, where $w_k = 0$ if x_{ik} or μ_{pk} has a missing value at the k th variable.

The object of the k -probabilities algorithm is to minimize the total Euclidean sum of squares E in equation (9.19). The algorithm starts with an initial partition of the data set into k clusters, and then reassigns the cases to another cluster such that the total Euclidean sum of squares E is minimized. To minimize E , a case i should only be reassigned from cluster p to cluster q if (Wishart, 1978)

$$E_p + E_q > E_{p-i} + E_{q+i},$$

which is equivalent to

$$I_{p-i,i} > I_{q,i}.$$

ALGORITHM 9.6. The k -probabilities algorithm.

Require: Data set D , No. of Clusters: k , Dimensions: d :

- { C_i is the i th cluster}
- {1. Initialization Phase}
- 1: (C_1, C_2, \dots, C_k) = Initial partition of D .
- {2. Reallocation Phase}
- 2: **repeat**
- 3: **for** $i = 1$ to $|D|$ **do**

```

4:   Compute  $I_{p-i,i}$  { $p$  is the current cluster number of case  $i$ };
5:   for  $q = 1$  to  $k$ ,  $q \neq p$  do
6:     compute  $I_{q+i,i}$ ;
7:     if  $I_{p-i,i} > I_{q+i,i}$  then
8:       Assign case  $i$  to cluster  $n_i$ ;
9:       break;
10:      end if
11:    end for
12:  end for
13: Recompute the cluster means of any changed clusters above;
14: until no further changes in cluster membership occur in a complete iteration

```

The pseudocode of the k -probabilities algorithm is given in Algorithm 9.6. Gupta et al. (1999) also proposed an algorithm that extends the k -means algorithm to cluster categorical data through defining the objective function based on the new Condorcet criterion (Michaud, 1997).

9.11 The k -prototypes Algorithm

The k -prototypes algorithm (Huang, 1998) comes from the k -means and k -modes algorithm; it was designed to cluster mixed-type data sets. A related work is (Huang, 1997a). In the k -prototypes algorithm, the prototype is the center of a cluster, just as the mean and mode are the centers of a cluster in the k -means and k -modes algorithms, respectively.

Let two mixed-type objects be described by attributes $A_1^r, A_2^r, \dots, A_p^r, A_{p+1}^c, \dots, A_m^c$, where the first p attributes are numerical while the remaining $m-p$ attributes are categorical. Let $X = [x_1, x_2, \dots, x_m]$, and $Y = [y_1, y_2, \dots, y_m]$, where x_i , and y_i ($1 \leq i \leq p$) take numerical values while the rest take categorical values. Then the dissimilarity measure between X and Y can be

$$d(X, Y) = \sum_{j=1}^p (x_j - y_j)^2 + \gamma \sum_{j=p+1}^m \delta(x_j, y_j),$$

where γ is a balance weight used to avoid favoring either type of attribute. In the definition of the dissimilarity measure, the squared Euclidean distance is used to measure the numerical attributes and the simple matching dissimilarity (Kaufman and Rousseeuw, 1990) measure is used to measure the categorical attributes.

The goal of the k -prototypes algorithm is to minimize the cost function

$$P(W, Q) = \sum_{l=1}^k (P_l^r + \gamma P_l^c),$$

where

$$P_l^r = \sum_{i=1}^n w_{i,l} \sum_{j=1}^p (x_{i,j} - q_{l,j})^2,$$

$$P_l^c = \sum_{i=1}^n w_{i,l} \sum_{j=p+1}^m \delta(x_{i,j}, q_{l,j}).$$

ALGORITHM 9.7. The k -prototypes algorithm.

Require: k : the number of clusters;

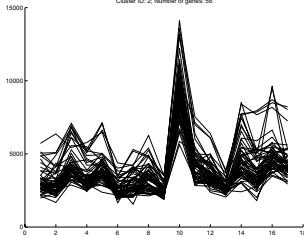
- 1: Select k initial prototypes from the database, one for each cluster;
- 2: Allocate each object in the database to a cluster whose prototype is the nearest to it according to the dissimilarity measure, and update the prototype of the cluster after each allocation;
- 3: **repeat**
- 4: Retest the similarity between each object and the prototype; if an object is found that is nearest to another prototype rather than the current one, reallocate the object to the nearest cluster;
- 5: Update the prototypes of both clusters;
- 6: **until** no further changes in the cluster membership

The k -prototypes algorithm (Algorithm 9.7) is the same as the k -probabilities algorithm (see Algorithm 9.6) except for the reallocation phase. The complexity of the k -prototypes algorithm is $O((t + 1)kn)$, where n is the number of data points in the data set, k is the number of clusters, and t is the number of iterations of the reallocation process.

9.12 Summary

A popular center-based clustering algorithm, the k -means algorithm, and its variations, has been presented in this chapter. To handle categorical data, two versions of the k -modes algorithm are also presented. Center-based algorithms are easy to implement and the results are easy to interpret. In addition, center-based algorithms are faster than hierarchical algorithms in general. Therefore, they are popular for clustering large databases. Zhang and Hsu (2000) discussed accelerating center-based clustering algorithms by parallelism.

In center-based clustering algorithms, each cluster has one center. Instead of generating a cluster center as a point, Bradley and Mangasarian (2000) proposed a clustering algorithm, called the k -plane algorithm, in which the entity of the center is changed from a point to a plane. In some cases, the k -plane algorithm outperforms the k -means algorithm (Bradley and Mangasarian, 2000).



Chapter 10

Search-based Clustering Algorithms

Hard clustering is a special case of fuzzy clustering. A fuzzy clustering problem can be considered as an optimization problem (Dunn, 1974a; Ng and Wong, 2002)

$$\min_{W, Z} F(W, Z) = \sum_{l=1}^k \sum_{i=1}^n w_{li}^\alpha d(\mu_l, x_i)$$

such that

$$0 \leq w_{li} \leq 1 \text{ for any } 1 \leq l \leq k, 1 \leq i \leq n, \quad (10.1a)$$

$$\sum_{l=1}^k w_{li} = 1 \text{ for } i = 1, 2, \dots, n, \quad (10.1b)$$

$$0 < \sum_{i=1}^n w_{li} < n, 1 \leq l \leq k, \quad (10.1c)$$

where $\alpha \geq 1$ is a parameter, n is the number of data points in the data set, k is the number of clusters, x_i is the i th data point in the data set, μ_l is the cluster center of the l th cluster, $W = [w_{li}]$ is a $k \times n$ fuzzy matrix, $Z = [\mu_l]$ is a matrix containing the cluster center, and $d(\mu_l, x_i)$ is a dissimilarity measure between centers μ_l and x_i .

Most of the clustering algorithms may not be able to find the global optimal cluster that fits the data set; these algorithms will stop if they find a local optimal partition of the data set. For example, the fuzzy k -means (Selim and Ismail, 1984), fuzzy ISODATA (Bezdek, 1980), and fuzzy c -means (Hathaway and Bezdek, 1984; Selim and Ismail, 1986) algorithms are convergent, but they may stop at a local minimum of the optimization problem. The algorithms in the family of search-based clustering algorithms can explore the solution space beyond the local optimality in order to find a globally optimal clustering that fits the data set. In this chapter, we present some search-based clustering algorithms, including those based on simulated annealing and tabu search.

10.1 Genetic Algorithms

Genetic algorithms (GAs) were first proposed by Holland (1975) as a family of computational models inspired by the analogy of evolution and population genetics. GAs are inherently parallel and particularly suitable for solving complex optimization problems. Filho et al. (1994) presented a survey of GAs along with a simple GA written in the C language.

Usually, there are only two main components of GAs that are problem dependent: the problem encoding and the evaluation function (e.g., objective function). Even for the same problem, one can use different encodings. For example, in the genetic k -means algorithm, Krishna and Narasimha (1999) employed *string-of-group-numbers encoding*, while Maulik and Bandyopadhyay (2000) encoded the strings such that each string is a sequence of real numbers representing the cluster centers.

In GAs, the parameters of the search space are encoded in the form of *strings* called *chromosomes*. A GA maintains a *population* (set) of N coded strings for some fixed *population size* N and evolves over *generations*. During each generation, three genetic operators, i.e., *natural selection*, *crossover*, and *mutation*, are applied to the current population to produce a new population. Each string in the population is associated with a fitness value depending on the value of the objective function. Based on the principle of survival of the fittest, a few strings in the current population are selected and each is assigned a number of copies, and then a new generation of strings are yielded by applying crossover and mutation to the selected strings.

In general, a typical GA has the following five basic components: *encoding*, *initialization*, *selection*, *crossover*, and *mutation*. Encoding is dependent on the problem under consideration. In the initialization phase, a population (set) of strings will be randomly generated. After the initialization phase, there is an iteration of generations. The number of generations is specified by the user. In GAs, the best string obtained so far is stored in a separate location outside the population and the final output is the best string among all possible strings inspected during the whole process.

GAs have been successfully applied to clustering (Jiang and Ma, 1996; Cheng et al., 2002; Maulik and Bandyopadhyay, 2000; Greene, 2003) and classification (Goldberg, 1989; Bandyopadhyay et al., 1995). Some clustering algorithms related to GAs are briefly presented below.

Murthy and Chowdhury (1996) proposed a GA in an attempt to reach the optimal solution for the clustering problem. In this algorithm, the evaluation function is defined as the sum of squared Euclidean distances of the data points from their respective cluster centers. In addition, single-point crossover (Michalewicz, 1992), i.e., the crossover operator between two strings, is performed at one position, and elitist strategies, i.e., the best string is carried from the previous population to the next, are used.

Tseng and Yang (2001) proposed a genetic approach called CLUSTERING to the automatic clustering problem. CLUSTERING is suitable for clustering data with compact spherical clusters, and the number of clusters can be controlled indirectly by a parameter w . The algorithm will produce a larger number of compact clusters with a small value of w and it will produce a smaller number of looser clusters with a large value of w . A genetic-based clustering algorithm aimed at finding nonspherical clusters was proposed by Tseng and Yang (2000).

Garai and Chaudhuri (2004) proposed a genetically guided hierarchical clustering algorithm that can find arbitrarily shaped clusters. The algorithm consists of two phases. At first, the original data set is decomposed into a number of fragmented groups in order to spread the GA search process at the latter phase over the entire space. Then the hierarchical cluster merging algorithm (HCMA) is used. During the merging process, a technique called the adjacent cluster checking algorithm (ACCA) is used to test the adjacency of two segmented clusters so that they can be merged into one cluster.

Krishna and Narasimha (1999) and Bandyopadhyay and Maulik (2002) proposed two different clustering algorithms based on GAs and the popular k -means algorithm. In the genetic k -means algorithm (GKA), Krishna and Narasimha (1999) used the k -means operator instead of the crossover operator to accelerate the convergence, while in KGA-clustering, Bandyopadhyay and Maulik (2002) used the single-point crossover operator.

Cowgill et al. (1999) proposed a genetic-based clustering algorithm called COWCLUS. In COWCLUS, the evaluation function is the variance ratio (VR) defined in terms of external cluster isolation and internal cluster homogeneity. The goal of the algorithm is to find the partition with maximum VR.

Further examples of applying GAs to clustering can be found in (Krishna and Narasimha, 1999), (Lu et al., 2004a), (Lu et al., 2004b), (Gan et al., 2005), (Babu and Murty, 1993), (Chiou and Lan, 2001), (Liu et al., 2004), (Hall et al., 1999), and (Zhao et al., 1996).

10.2 The Tabu Search Method

Tabu search (Glover, 1989, 1990; Glover et al., 1993) is a heuristic algorithm that can be used to solve combinatorial optimization problems. A memory-based strategy is introduced in tabu search in order to prevent the solution search from becoming trapped at a local optimal solution. Thus, tabu search can find global optimization, while other methods, such as hill climbing, may be trapped in local optimal solutions.

The basic elements of the tabu search method are configuration, move, neighborhood, candidate subset, tabu restrictions, and aspiration criteria. Tabu is a subset of moves that are classified as forbidden. This is a chief mechanism for exploiting memory in tabu search.

Configuration. Configuration is an assignment of values to variables. It is an initial solution to the optimization problem.

Move. Move is a procedure by which a new trial solution is generated from the current one. It characterizes the process of generating a feasible solution to the combinatorial problem that is related to the current configuration. In a clustering problem, for example, a move indicates that an element changes its cluster to another one to which it is newly assigned.

Neighborhood. Neighborhood is the set of all neighbors that are “adjacent solutions” that can be reached from any current solution. Neighbors that do not satisfy the given customary feasible conditions may also be included in the set.

Candidate subset. Candidate subset is a subset of the neighborhood. If the set of all neighbors is too large, one could operate with a subset of the neighborhood instead of the entire neighborhood.

Tabu restrictions. Tabu restrictions are certain conditions imposed on moves that make some of them forbidden. These forbidden moves are known as tabu. The tabu list is a list in which tabu moves are stored. For example, the tabu list may contain each element index and its cluster index (Sung and Jin, 2000).

Aspiration criteria. Aspiration criteria are rules that override tabu restrictions. If a certain move is forbidden by some tabu restrictions, but it satisfies the aspiration criteria, then this move is allowable.

Given the above basic elements, the tabu search method can be summarized as follows: start with an initial configuration, evaluate its corresponding criterion function, and then follow a certain set of candidate moves. If the best move is not tabu, or if the best move is tabu but satisfies the aspiration criterion, then pick that move and consider it to be the next current configuration. Otherwise, pick the first non-tabu move and consider it to be the next current configuration. The tabu list has a certain size, which frees the first move on the tabu from being tabu when the length of the tabu reaches the size and a new move enters the tabu list.

The tabu search method has also been applied to solve the problem of clustering. Sung and Jin (2000) proposed a heuristic clustering algorithm that combines the tabu search heuristic with two complementary functional procedures, called packing and releasing procedures. The packing procedure binds a subset of objects together as a single “object”, and the releasing procedure separates packed objects from each other. Xu et al. (2002) developed a fuzzy tabu search (FTS) technique for solving the clustering problem. In the FTS technique, a fuzzy method is used to determine the tabu period and select neighborhoods.

Delgado et al. (1997) also proposed a tabu search algorithm for the fuzzy clustering problem by applying a tabu search technique to the well-known fuzzy c -means algorithm.

10.3 Variable Neighborhood Search for Clustering

Variable neighborhood search (VNS) (Hansen and Mladenović, 2001b) is a metaheuristic proposed for solving combinatorial problems. This metaheuristic is obtained by proceeding to a systematic change of neighborhood within a local search algorithm.

Let D be a set of objects and \mathcal{P}_D be the solution space (the set of all partitions of D). For any two solutions $P'_D, P''_D \in \mathcal{P}_D$, let $\rho(P'_D, P''_D)$ be a distance between them. The metric is defined in order to induce the set of neighborhoods. To do this, each solution P_D is equivalently represented as a k -star graph G_D , where vertices correspond to objects and centroids and objects from the same cluster are connected to the same vertex. Each such graph has n edges including loops (if some centroids coincide with objects). Then the distance between P'_D and P''_D can be defined as $\rho(P'_D, P''_D) = l$ if and only if their corresponding graphs G'_D and G''_D differ in l of their edges. Then the set of neighborhoods can be induced from the distance as follows:

$$P''_D \in \mathcal{N}(P'_D) \Leftrightarrow \rho(P'_D, P''_D) = l.$$

Usually, the set of neighborhoods in VNS is induced from one metric function introduced into the solution space, e.g., $\rho(P'_D, P''_D)$. In the VNS algorithm, the search is centered

around the same solution until another solution better than the incumbent is found. This is not a trajectory method like simulated annealing or tabu search.

ALGORITHM 10.1. The VNS heuristic.

- S_1 (*Initialization*) Let P_D be an initial partition and f_{opt} be the corresponding objective function value. Set a value for a parameter l_{max} ;
- S_2 (*Termination*) If the stopping condition is met (e.g., maximum CPU time allowed, maximum number of iterations), stop;
- S_3 (*First neighborhood*) Set $l = 1$;
- S_4 (*Inner loop*) If $l > l_{max}$, return to step S_2 ;
- S_5 (*Perturbation*) Draw a point P'_D randomly from $\mathcal{N}(P_D)$, i.e., reassign any l objects from D to other clusters than their own;
- S_6 (*Local search*) Apply a local search (such as J -means) with P'_D as initial solution, denoting the resulting solution and objective function value by P''_D and f'' , respectively;
- S_7 (*Move or not*) If $f'' < f_{opt}$, then recenter the search around the better solution found ($f_{opt} = f''$ and $P_D = P''_D$) and go to step S_3 ; otherwise, set $l = l + 1$ and go to step S_4 .

The basic steps of the VNS heuristic for clustering are shown in Algorithm 10.1. VNS is also presented in (Mladenović and Hansen, 1997) and (Hansen and Mladenović, 2001a).

10.4 Al-Sultan's Method

In the problem of clustering n objects into k clusters such that the distance between points within a cluster and its center is minimized, many local optimal solutions exist. In order to find the global optimum, Al-Sultan (1995) proposed a clustering algorithm based on a tabu search technique.

Given a set of objects $D = \{\mathbf{x}_i | i = 1, 2, \dots, n\}$, the clustering problem discussed by Al-Sultan (1995) is mathematically summarized as follows:

Minimize

$$J(W, Z) = \sum_{i=1}^n \sum_{j=1}^k w_{ij} \|\mathbf{x}_i - \mathbf{z}_j\|^2, \quad (10.2)$$

subject to

$$\sum_{j=1}^k w_{ij} = 1, \quad i = 1, 2, \dots, n,$$

$$w_{ij} = 0 \text{ or } 1, \quad i = 1, 2, \dots, n, j = 1, 2, \dots, k,$$

where n is the number of objects, k is the number of clusters, $\mathbf{z}_j \in \mathbb{R}^d$ is the center of the j th cluster, and w_{ij} is the weight associated with cluster j , i.e.,

$$w_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ is allocated to cluster } j, \\ 0 & \text{otherwise.} \end{cases}$$

To describe the algorithm, let A be an n -dimensional array whose i th element A_i is a number representing the cluster to which the i th object is assigned. Given A , the matrix W can be determined as follows: $w_{ij} = 1$ if $A_i = j$; $w_{ij} = 0$ otherwise. In this algorithm, the center of each cluster \mathbf{z}_j is computed as the centroid of the objects allocated to that cluster, i.e.,

$$\mathbf{z}_j = \frac{\sum_{i=1}^n w_{ij} \mathbf{x}_i}{\sum_{i=1}^n w_{ij}}.$$

Thus, as long as such an array A (configuration) is given, the objective function (10.2) can be evaluated. Let A_t , A_c , and A_b denote the trial, current, and best configuration and J_t , J_c , and J_b denote the corresponding objective function values.

ALGORITHM 10.2. Al-Sultan's tabu search-based clustering algorithm.

- S_1 Let A_c be an arbitrary configuration and J_c be the corresponding objective function value. Let $A_b = A_c$ and $J_b = J_c$; set values for parameters $MTLS$ (tabu list size), P (probability threshold), NTS (number of trial solutions), and $ITMAX$ (maximum number of iterations); and let $N = 1$ and TLL (tabu list length) = 0;
- S_2 Generate NTS trial solutions A_t^1, \dots, A_t^{NTS} from A_c and evaluate the corresponding objective function values J_t^1, \dots, J_t^{NTS} ;
- S_3 Sort $J_t^1, J_t^2, \dots, J_t^{NTS}$ such that $J_t^{[1]} \leq J_t^{[2]} \leq \dots \leq J_t^{[NTS]}$; if $J_t^{[1]}$ is not tabu or if it is tabu but $J_t^{[1]} < J_b$, then let $A_c = A_t^{[1]}$ and $J_c = J_t^{[1]}$, and go to step S_4 ; otherwise let $A_c = A_t^{[L]}$ and $J_c^{[L]}$, where $J_t^{[L]}$ is the best objective function value of $J_t^{[2]}, \dots, J_t^{[NTS]}$ that is not tabu, and go to step S_4 ; if all $J_t^{[1]}, \dots, J_t^{[NTS]}$ are tabu, then go to step S_2 ;
- S_4 Insert A_c at the bottom of the tabu list; if $TLL = MTLS$, then delete the first element in the tabu list; otherwise, let $TLL = TLL + 1$; if $J_c < J_b$, let $A_b = A_c$ and $J_b = J_c$; if $N = ITMAX$, then stop; otherwise, let $N = N + 1$ and go to step S_2 .

The tabu search-based algorithm for clustering n objects into k clusters is summarized in Algorithm 10.2. In step S_2 of Algorithm 10.2, a number of trial configurations are generated from the current configuration A_c . The generation method adopted by Al-Sultan (1995) is described as follows: Given A_c and a probability threshold P , for each $i = 1, 2, \dots, n$, draw a random number $R \sim u(0, 1)$. If $R < P$, then let $A_t(i) = A_c(i)$; otherwise, draw an integer \hat{l} randomly from the set $\{l | 1 \leq l \leq k, l \neq A_c(i)\}$ and let $A_c(i) = \hat{l}$.

In this algorithm, three parameters need to be assigned values before the execution of the algorithm. In order to obtain better results, the algorithm is performed in two phases. In the first phase, the algorithm is performed to find the best parameter settings by considering test problems. In the second phase, the best values are then used for testing the performance of the algorithm.

10.5 Tabu Search-based Categorical Clustering Algorithm

Ng and Wong (2002) proposed a tabu search-based fuzzy k -means type algorithm, i.e., the tabu search-based fuzzy k -prototypes algorithm, in order to find a global solution of the fuzzy clustering problem.

The fuzzy k -prototypes algorithm integrates the fuzzy k -means algorithm and the fuzzy k -modes algorithm. It is designed to deal with mixed numeric and categorical data. The fuzzy k -prototypes algorithm can be summarized as follows.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a set of n objects, each of which is described by d mixed numeric and categorical attributes, i.e., $\mathbf{x}_i = (\mathbf{x}_i^{(n)}, \mathbf{x}_i^{(c)})$, where $\mathbf{x}_i^{(n)}$ represents the numeric attributes and $\mathbf{x}_i^{(c)}$ represents the categorical attributes. Let $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ denote k cluster centers; similarly, $\mathbf{z}_j = (\mathbf{z}_j^{(n)}, \mathbf{z}_j^{(c)})$, in which $\mathbf{z}_j^{(n)}$ represents the numeric attributes and $\mathbf{z}_j^{(c)}$ represents the categorical attributes.

In the fuzzy k -prototypes algorithm, the combined dissimilarity measure is used to measure the dissimilarity between two objects, i.e.,

$$d(\mathbf{x}_l, \mathbf{x}_i) = d_n(\mathbf{x}_l^{(n)}, \mathbf{x}_i^{(n)}) + \beta d_c(\mathbf{x}_l^{(c)}, \mathbf{x}_i^{(c)}), \quad (10.3)$$

where $d_n(\cdot, \cdot)$ is the Euclidean distance and $d_c(\cdot, \cdot)$ is the simple matching distance, and β is a weight used to balance the numeric and categorical parts.

Then the goal of the fuzzy k -prototypes algorithm is to minimize the objective function

$$F(W, Z) = \sum_{l=1}^k \sum_{i=1}^n w_{li}^\alpha \left[d_n(\mathbf{z}_l^{(n)}, \mathbf{x}_i^{(n)}) + \beta d_c(\mathbf{z}_l^{(c)}, \mathbf{x}_i^{(c)}) \right], \quad (10.4)$$

subject to the same conditions (10.1).

In the fuzzy k -prototypes algorithm, the matrices W and Z are updated iteratively as follows. Let Z be fixed, i.e., \mathbf{z}_l ($l = 1, 2, \dots, k$) are given. Then W can be formulated as follows:

$$w_{li} = \begin{cases} 1 & \text{if } \mathbf{x}_i = \mathbf{z}_l, \\ 0 & \text{if } \mathbf{x}_i = \mathbf{z}_h \text{ but } h \neq l, \\ \left[\sum_{h=1}^k \left(\frac{d(\mathbf{z}_l, \mathbf{x}_i)}{d(\mathbf{z}_h, \mathbf{x}_i)} \right)^{\frac{1}{\alpha-1}} \right]^{-1} & \text{if } \mathbf{x}_i \neq \mathbf{z}_l \text{ and } \mathbf{x}_i \neq \mathbf{z}_h, 1 \leq h \leq k, \end{cases} \quad (10.5)$$

where $d(\cdot, \cdot)$ is defined in (10.3).

If W is fixed, then the matrix Z can be updated in the following way. Let m be the number of numeric attributes. Then there are $d - m$ categorical attributes, i.e., $\mathbf{x}_i =$

$(\mathbf{x}_i^{(n)}, \mathbf{x}_i^{(c)}) = (x_{i1}^{(n)}, \dots, x_{im}^{(n)}, x_{i,m+1}^{(c)}, \dots, x_{id}^{(c)})$. The numeric part of Z is computed as

$$z_{lj} = \frac{\sum_{i=1}^n w_{li} x_{ij}^{(n)}}{\sum_{i=1}^n w_{li}}$$

for $1 \leq l \leq k$ and $1 \leq j \leq m$. The categorical part of Z is updated as follows. Suppose the j th attribute has n_j categories: $A_{j1}, A_{j2}, \dots, A_{jn_j}$ for $j = m + 1, m + 2, \dots, d$. Then $F(W, Z)$ is minimized if and only if

$$z_{lj} = A_{jr}, \quad m + 1 \leq j \leq d,$$

where r is determined as

$$r = \arg \max_{1 \leq t \leq n_j} \sum_{i, x_{ij}^{(c)} = A_{jt}} w_{li}^\alpha.$$

In the tabu search–based fuzzy k -prototypes algorithm proposed by Ng and Wong (2002), the cluster center Z is generated by a method used by Al-Sultan and Fedjki (1997). The fuzzy partition matrix W is updated by equation (10.5).

In this algorithm, one uses Z^t , Z^u , and Z^b to denote the trial, current, and best cluster centers and F^t , F^u , and F^b to denote the corresponding objective function values. The trial cluster centers Z^t are generated through moves from the current cluster centers Z^u . The best cluster centers found so far are saved in Z^b .

Generation of neighborhoods is one of the most distinctive features of the tabu search method. In this algorithm, the numeric part and the categorical part with ordinal attributes of the neighborhood of the center \mathbf{z}^u are defined as

$$\begin{aligned} N^{(n)}(\mathbf{z}^u) &= \{\mathbf{y}^{(n)} = (y_1, y_2, \dots, y_l) | y_i = \mathbf{z}_i^u + \phi\theta, \\ &\quad i = 1, 2, \dots, l \text{ and } \theta = -1, 0, \text{ or } 1\}, \end{aligned}$$

where ϕ is a small step size. The categorical part with nominal attribute cannot use the method described above. In this algorithm, the nominal part of the neighborhood of \mathbf{z}^u is defined as

$$N^{(c)}(\mathbf{z}^u) = \{\mathbf{y}^{(c)} = (y_1, y_2, \dots, y_l) | d_c(\mathbf{y}, \mathbf{z}^u) < \theta\},$$

where $0 \leq \theta \leq l$. The greater the value of θ , the larger the solution space to be examined. The neighbors of \mathbf{z}^u can be generated by picking randomly from $N^{(n)}(\mathbf{z}^u)$ and $N^{(c)}(\mathbf{z}^u)$.

Several parameters have been used in this algorithm, e.g., $NTLM$ (tabu list size), P (probability threshold), NH (number of trial solutions), $IMAX$ (maximum number of nonimproving moves for each center), and γ (reduction factor for $IMAX$). Different sets of these parameters have been tested by Ng and Wong (2002).

10.6 J-means

J-means (Hansen and Mladenović, 2001a) is a local search heuristic that is proposed for solving the minimum sum of squares clustering problem, i.e., the clustering problem described in equation (10.2).

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a set of objects and \mathcal{P}_D denote the set of all partitions of D . The problem in equation (10.2) can also be presented as

$$\min_{P_D \in \mathcal{P}_D} \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{z}_i\|^2, \quad (10.6)$$

where k is the number of clusters, $\|\cdot\|$ denotes the Euclidean norm, and \mathbf{z}_i is the centroid of cluster C_i defined as

$$\mathbf{z}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$$

for $i = 1, 2, \dots, k$.

To describe the *J*-means heuristic, let us introduce some terms first. Occupied points are referred to as the existing points (i.e., points in D) that coincide with a cluster centroid within a small tolerance. In this algorithm, a neighboring solution of the current one is obtained by relocating the centroid \mathbf{z}_i of a cluster C_i (and not an existing point) to some unoccupied point location and relocating all points in C_i to their closest centroid. The jump neighborhood of the current solution consists of all possible such moves.

ALGORITHM 10.3. The *J*-means algorithm.

*S*₁ (*Initialization*) Let $P_D = \{C_1, C_2, \dots, C_k\}$ be an initial partition of D , \mathbf{z}_i be the centroid of cluster C_i , and f_{opt} be the current objective function value;

*S*₂ (*Occupied points*) Find unoccupied points, i.e., points in D that do not coincide with a cluster centroid within a small tolerance;

*S*₃ (*Jump neighborhood*) Find the best partition P'_D and the corresponding objective function value f' in the jump neighborhood of the current solution P_D :

*S*₃₁ (*Exploring the neighborhood*) For each j ($j = 1, 2, \dots, n$), repeat the following steps: (a) *Relocation*. Add a new cluster centroid \mathbf{z}_{k+1} at some unoccupied point location \mathbf{x}_j and find the index i of the best centroid to be deleted; let v_{ij} denote the change in the objective function value; (b) *Keep the best*. Keep the pair of indices i' and j' where v_{ij} is minimum;

*S*₃₂ (*Move*) Replace the centroid $\mathbf{z}_{i'}$ by $\mathbf{x}_{j'}$ and update cluster membership accordingly to get the new partition P'_D ; set $f' := f_{opt} + v_{i'j'}$;

*S*₄ (*Termination or move*) If $f' > f_{opt}$, stop; otherwise, move to the best neighboring solution P'_D ; set P'_D as current solution and return to step *S*₂.

The main procedure of the *J*-means algorithm is summarized in Algorithm 10.3. The *relocation* step *S*₃₁(a) can be implemented in $O(n)$ time (Hansen and Mladenović, 2001a), and the efficiency of the algorithm is largely dependent on this fact. After step *S*₃, each jump neighborhood solution can be improved by using the *k*-means algorithm, the *H*-means

algorithm (Howard, 1966), or some other heuristic. The resulting hybrid heuristic is called *J*-means+.

A comparison of six local search methods (*k*-means, *H*-means (Howard, 1966), modification of *H*-means (Hansen and Mladenović, 2001a), *Hk*-means, *J*-means, and *J*-Means+) has been presented by Hansen and Mladenović (2001a). The results show that the well-known *k*-means and *H*-means heuristics for minimum sum of squares clustering can give very poor clustering results when the number of clusters is large.

Fuzzy *J*-means (Belacel et al., 2002) is an extension of the *J*-means heuristic for fuzzy clustering.

10.7 GKA

Krishna and Narasimha (1999) proposed a hybrid algorithm called GKA, modified from the GA, which can find a globally optimal partition of a given data set into a specified number of clusters. Using the finite Markov chain theory, GKA is proved to converge to the global optimum. In practice, GKA also searches faster than some evolutionary algorithms used for clustering.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a set of n objects in a d -dimensional space, and let C_1, C_2, \dots, C_k be k clusters of D . Let w_{ij} be defined as

$$w_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C_j, \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, k$. Then the matrix $W = (w_{ij})$ has the following properties:

$$w_{ij} \in \{0, 1\} \text{ and } \sum_{j=1}^k w_{ij} = 1. \quad (10.7)$$

Let the within-cluster variation of C_j be defined as

$$S^{(j)}(W) = \sum_{i=1}^n w_{ij} \sum_{l=1}^d (x_{il} - z_{jl}),$$

and the total within-cluster variation, also called the squared Euclidean (SE) measure, be defined as

$$S(W) = \sum_{j=1}^k S^{(j)}(W) = \sum_{j=1}^k \sum_{i=1}^n w_{ij} \sum_{l=1}^d (x_{il} - z_{jl}), \quad (10.8)$$

where x_{il} is the l th-dimensional value of object \mathbf{x}_i and z_{jl} is the l th-dimensional value of \mathbf{z}_j , the center of C_j , defined as

$$z_{jl} = \frac{\sum_{i=1}^n w_{ij} x_{il}}{\sum_{i=1}^n w_{ij}} \quad (10.9)$$

for $l = 1, 2, \dots, d$.

The objective is to find $W^* = (w_{ij}^*)$ such that

$$S(W^*) = \min_W S(W).$$

There are three genetic operators used in GKA: the selection operator, the distance-based mutation operator, and the k -means operator. The evolution takes place until a termination condition is reached. To describe GKA, we start with the coding and initialization schemes and introduce the genetic operators.

All matrices satisfying equation (10.7) constitute the search space. A natural way to code such a W into a string s_W is to consider a chromosome of length n and allow each allele to take values from $\{1, 2, \dots, k\}$. What GKA maintains is a population of such strings. The initial population $\mathcal{P}(0)$ is selected by initializing each allele in the population to a cluster number randomly selected from the uniform distribution over the set $\{1, 2, \dots, k\}$. Illegal strings (i.e., strings representing a partition in which some clusters are empty) should be avoided.

In GKA, the selection operator randomly selects a chromosome from the previous population according to the distribution given by

$$P(s_i) = \frac{F(s_i)}{\sum_{i=1}^N F(s_i)},$$

where N is the population size, and $F(s_i)$ represents the fitness value of the string s_i in the population and is defined by

$$F(s_W) = \begin{cases} f(s_W) - (\bar{f} - c\sigma) & \text{if } f(s_W) - (\bar{f} - c\sigma) \geq 0, \\ 0 & \text{otherwise,} \end{cases}$$

where $f(s_W) = -S(W)$, \bar{f} and σ denote the average value and standard deviation of $f(s_W)$ in the current population, respectively, and c is a constant between 1 and 3.

The mutation operator changes an allele value depending on the distance between the cluster center and the corresponding data point. To apply the mutation operator to the allele $s_W(i)$ corresponding to object \mathbf{x}_i , let $d_j = d_{euc}(\mathbf{x}_i, \mathbf{z}_j)$. Then $s_W(i)$ is replaced with a value chosen randomly from the distribution

$$p_j = P(s_W(i) = j) = \frac{c_m d_{max} - d_j}{k c_m d_{max} - \sum_{l=1}^k d_l}, \quad (10.10)$$

where $c_m > 1$ and $d_{max} = \max_{1 \leq j \leq k} d_j$. To avoid empty clusters, an allele is mutated only when $d_{s_W(i)} > 0$. The pseudocode is given in Algorithm 10.4.

ALGORITHM 10.4. Mutation (s_W).

Require: P_m : mutation probability; n ; k ;

- 1: **for** $i = 1$ to n **do**
- 2: **if** $drand() < P_m$ **then**

```

3:   { $d_{rand}()$  returns a uniformly distributed random number in  $[0, 1]$ }
4:   Calculate  $d_j = d_{euc}(\mathbf{x}_i, \mathbf{z}_j)$  for  $j = 1, 2, \dots, k$ ;
5:   if  $d_{s_w(i)} > 0$  then
6:      $d_{max} = \max\{d_1, d_2, \dots, d_k\}$ ;
7:     Calculate  $p_j$  using Equation (10.10) for  $j = 1, 2, \dots, k$ ;
8:      $s_w(i) =$  a number randomly chosen from  $\{1, 2, \dots, k\}$  according to the distribution  $\{p_1, p_2, \dots, p_k\}$ .
9:   end if
10:  end if
11: end for

```

Since the algorithm with the selection and mutation operators may take more time to converge, a one-step k -means algorithm (named k -means operator (KMO)) is introduced. Let s_w be a string; applying KMO to s_w yields $s_{\hat{W}}$. KMO calculates the centers using equation (10.9) for the given matrix W and then forms \hat{W} by reassigning each data point to the cluster with the nearest center. KMO may result in illegal strings, i.e., partitions with empty clusters; this can be avoided by techniques such as placing in each empty cluster an object from the cluster with maximum within-cluster variation (Klein and Dubes, 1989).

ALGORITHM 10.5. The pseudocode of GKA.

Require: Mutation probability P_m , population size, N and maximum number of generation MAX_GEN ;

```

1: Initialize population  $\mathcal{P}$ ;
2:  $geno = MAX\_GEN$ ;
3:  $s^* = \mathcal{P}_1 \{ \mathcal{P}_i \text{ is the } i\text{th string in } \mathcal{P} \}$ ;
4: while  $geno > 0$  do
5:    $\hat{\mathcal{P}} = selection(\mathcal{P})$ ;
6:    $\mathcal{P}_i = Mutation(\hat{\mathcal{P}}_i)$  for  $i = 1, 2, \dots, N$ ;
7:    $k\text{-Means}(\mathcal{P}_i)$  for  $i = 1, 2, \dots, N$ ;
8:   Select  $s$  such that the corresponding matrix  $W_s$  has minimum SE measure;
9:   if  $S(W_{s^*}) > S(W_s)$  then
10:     $s^* = s$ ;
11:     $geno = geno - 1$ ;
12:  end if
13: end while
14: Output  $s^*$ .

```

Combining the three operators, the GKA can be described in Algorithm 10.5. In GKA, the complexity of evaluating the SE measure of a given solution string is $O(nd)$, the complexity of the mutation operator is $O(n^2d)$, and the complexity of the KMO is $O(nkd)$.

In order to make GKA faster, Lu et al. (2004b) proposed FGKA (fast GKA) by modifying the operators in GKA. In addition, Lu et al. (2004a) proposed another algorithm called IGKA (incremental GKA) and applied it in gene expression data analysis. The main idea of IGKA is to calculate the SE measure and to cluster the centroids incrementally

whenever the mutation probability is small. The details of the two algorithms are omitted here; interested readers are referred to (Lu et al., 2004b) and (Lu et al., 2004a), respectively.

10.8 The Global k -means Algorithm

Likas and Verbeek (2003) proposed a deterministic global clustering algorithm, called the global k -means algorithm, to minimize the clustering error, which employs the k -means algorithm as a local search procedure. The global k -means algorithm proceeds in an incremental way: to solve a clustering problem with k clusters, all intermediate clustering problems with $1, 2, \dots, k - 1$ clusters are sequentially solved.

The main idea behind the global k -means algorithm is that an optimal solution of a clustering problem with k clusters can be obtained by carrying out a series of local searches using the k -means algorithm. At each local search, the $k - 1$ cluster centers are always initialized at their optimal positions, i.e., the cluster centers of the clustering problem with $k - 1$ clusters. The k th cluster center is initialized to several positions within the data space. The optimal solution of the clustering problem with 1 cluster is trivial and thus known.

More specifically, the global k -means algorithm is described as follows. Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a given data set in a d -dimensional space. The aim of the clustering problem with k clusters is to partition D into k clusters C_1, C_2, \dots, C_k such that the clustering criterion

$$E(\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_k) = \sum_{j=1}^k \sum_{\mathbf{x} \in C_j} d_{euc}^2(\mathbf{x}, \mathbf{m}_j)$$

is optimized, where \mathbf{m}_j is the centroid of cluster C_j and $d_{euc}(\cdot, \cdot)$ is the Euclidean distance.

Suppose $\{\mathbf{m}_1^*(j), \mathbf{m}_2^*(j), \dots, \mathbf{m}_j^*(j)\}$ is the optimal solution of the clustering problem with j cluster. Then the optimal solution of the clustering problem with $j + 1$ clusters is obtained as follows: Perform n runs of the k -means algorithm, where the i th run starts with the initial state $\{\mathbf{m}_1^*(j), \mathbf{m}_2^*(j), \dots, \mathbf{m}_j^*(j), \mathbf{x}_i\}$. The best solution obtained from the n runs is considered as the solution of the clustering problem with $j + 1$ clusters.

In order to speed up the global k -means algorithm, some techniques, such as initialization with kd -trees, are used. Hussein (2003) proposed another global k -means algorithm based on a greedy approach, called the greedy k -means algorithm, in order to avoid some drawbacks of the global k -means algorithm presented in this section.

10.9 The Genetic k -modes Algorithm

The genetic k -modes algorithm (Gan et al., 2005), GKMODE, is similar to GKA except that a k -modes operator is used instead of KMO and, most important, illegal strings are permitted. As in GKA, GKMODE has five basic elements: *coding*, *initialization*, *selection*, *mutation*, and *k -modes operator*. The search space is the space of all binary membership matrices W that satisfy (10.7). Coding in GKMODE is exactly the same as in GKA. The initial population $\mathcal{P}(0)$ is randomly generated as in FGKA (Lu et al., 2004b). We now describe the genetic operators used in GKMODE in detail.

10.9.1 The Selection Operator

To describe the selection operator, let us start with the definition of the fitness value of a string. The fitness value of a string s_W depends on the value of the loss function $L_0(W)$, the limiting case of $L_p(W)$ as $p \rightarrow 0$. Since the objective is to minimize the loss function $L_0(W)$, a string with a relatively small loss must have a relatively high fitness value. In addition, illegal strings are less desirable and should be assigned low fitness values. As in (Lu et al., 2004b), we define the fitness value $F(s_W)$ of a string s_W as

$$F(s_W) = \begin{cases} cL_{max} - L_0(s_W) & \text{if } s_W \text{ is legal,} \\ e(s_W)F_{min} & \text{otherwise,} \end{cases} \quad (10.11)$$

where c is a constant in the interval $(0, 3)$; L_{max} is the maximum loss of strings in the current population; F_{min} is the smallest fitness value of the legal strings in the current population if it exists, otherwise it is defined as one; and $e(s_W)$ is the legality ratio defined as the ratio of the number of nonempty clusters in s_W over k (so that $e(s_W) = 1$ if s_W is legal).

The selection operator randomly selects a string from the current population according to the distribution given by

$$P(s_i) = F(s_i) / \sum_{j=1}^N F(s_j),$$

where N is the population size. The population of the next generation is determined by N independent random experiments, i.e., the selection operator is applied N times.

10.9.2 The Mutation Operator

In GKMODE, mutation changes a string value based on the distance of the cluster mode from the corresponding data point. It performs the function of moving the algorithm out of a local minimum. The closer a data point is to a cluster mode, the higher the chance of changing the data point to that cluster.

Precisely, let s_W be a solution string and let $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k$ be the cluster modes corresponding to s_W . During mutation, the mutation operator replaces $s_W(i)$ with a cluster number randomly selected from $\{1, 2, \dots, k\}$ according to the distribution

$$p_j = [c_m d_{max}(\mathbf{x}_i) - d(\mathbf{x}_i, \mathbf{z}_j)] / \sum_{l=1}^k [c_m d_{max}(\mathbf{x}_i) - d(\mathbf{x}_i, \mathbf{z}_l)], \quad (10.12)$$

where $c_m > 1$ is a constant, $d(\mathbf{x}_i, \mathbf{z}_j)$ is the simple matching distance between \mathbf{x}_i and \mathbf{z}_j , and $d_{max}(\mathbf{x}_i) = \max_{1 \leq j \leq k} d(\mathbf{x}_i, \mathbf{z}_j)$. As in FGKA, (Lu et al., 2004b), $d(\mathbf{x}_i, \mathbf{z}_j)$ is defined as 0 if the j th cluster is empty. In general, mutation occurs with some mutation probability P_m specified by the user. The pseudocode of the mutation operator is given in Algorithm 10.6, where P_m is the mutation probability specified by the user. By applying the mutation operator, we may convert an illegal string to a legal one and a data point will move toward a closer cluster with a higher probability.

ALGORITHM 10.6. Mutation (s_W) in GKMODE.

```

for  $i = 1$  to  $n$  do
  if  $\text{drand}() < P_m$  then
    { $\text{drand}()$  returns a uniformly distributed random number in the range  $[0, 1]$ }
    Calculate modes  $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k$  corresponding to  $s_W$ ;
    Calculate  $d_j = d(\mathbf{x}_i, \mathbf{z}_j)$  for  $j = 1, 2, \dots, k$ ;
    Calculate  $p_1, p_2, \dots, p_k$  using equation (10.12);
     $s_W(i)$  = a number randomly selected from  $\{1, 2, \dots, k\}$  according to the distribution
     $\{p_1, p_2, \dots, p_k\}$ .
  end if
end for

```

10.9.3 The k -modes Operator

In GKA, KMO is used in place of the crossover operator in order to speed up the convergence process. In GKMODE, the k -modes operator, one step of the k -modes algorithm (Chaturvedi et al., 2001), is introduced for the same reason. Let s_W be a solution string. Then the k -modes operator on s_W yields $s_{\hat{W}}$ in the following two steps:

- 1. Estimate Z.** Given estimates of W , the mode matrix Z is determined as follows. The (j, l) entry z_{jl} of Z should be the mode of $(x_l : \mathbf{x} \in C_j)$, where x_l is the l -component of \mathbf{x} and $C_j = \{\mathbf{x}_i : s_W(i) = j, 1 \leq i \leq n\}$. The mode matrix Z formed above optimizes the L_0 loss function (Chaturvedi et al. (2001)).
- 2. Estimate W.** Given estimates of Z , the binary membership matrix W is determined as follows. The loss function $L_0(W)$ can be written as $L_0(W) = \sum_{i=1}^n f_i$, where $f_i (1 \leq i \leq n)$ is defined as $f_i = \sum_{j=1}^d \delta(x_{ij}, z_{s_W(i)j})$ ($\delta(x, y) = 0$ if $x = y$, 1 otherwise). Note that f_i is a function only of $s_W(i)$. Thus, to minimize L_0 , one can separately minimize f_i with respect to the parameter $s_W(i)$ for $i = 1, 2, \dots, n$. Since $s_W(i)$ has only k possible values, i.e., $\{1, 2, \dots, k\}$, we can try all these k values and select the value that minimizes f_i , i.e., $s_W(i) = \arg \min_{1 \leq l \leq k} \sum_{j=1}^d \delta(x_{ij}, z_{lj})$. To account for illegal strings, we define $\delta(x_{ij}, z_{lj}) = +\infty$ if the l th cluster is empty (Lu et al., 2004b). This new definition is introduced in order to avoid reassigning all data points to empty clusters. Thus, illegal strings remain illegal after the application of the k -modes operator.

10.10 The Genetic Fuzzy k -modes Algorithm

The genetic fuzzy k -modes algorithm is the fuzzy version of the genetic k -modes algorithm. In order to speed up the convergence process, a one-step fuzzy k -modes algorithm is used in place of the crossover process. In this section, we will introduce these five elements of the GA for fuzzy k -modes clustering.

Recall that a fuzzy clustering problem can be represented as an optimization problem (Dunn, 1974a)

$$\min_{W, Z} F(W, Z) = \sum_{l=1}^k \sum_{i=1}^n w_{li}^\alpha d(\mathbf{z}_l, \mathbf{x}_i)$$

such that

$$0 \leq w_{li} \leq 1, \quad 1 \leq l \leq k, \quad 1 \leq i \leq n, \quad (10.13a)$$

$$\sum_{l=1}^k w_{li} = 1, \quad 1 \leq i \leq n, \quad (10.13b)$$

$$0 < \sum_{i=1}^n w_{li} < n, \quad 1 \leq l \leq k, \quad (10.13c)$$

where n is the number of objects in the data set under consideration; k is the number of clusters; $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ is a set of n objects, each of which is described by d attributes; $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ is a set of k cluster centers; $W = (w_{li})$ is a $k \times n$ fuzzy membership matrix; $\alpha \in [1, \infty]$ is a weighting exponent; and $d(\mathbf{z}_l, \mathbf{x}_i)$ is a certain distance measure between the cluster center \mathbf{z}_l and the object \mathbf{x}_i .

10.10.1 String Representation

A natural coding approach is to represent the $n \times k$ fuzzy membership matrix W in a chromosome, where n is the number of objects in the data set and k is the number of clusters. That is, the length of a chromosome is $n \times k$, where the first k positions (or genes) represent the k fuzzy memberships of the first data point, the next k positions represent those of the second data point, and so on. For example, if $n = 4$ and $k = 3$, then the chromosome $(a_1, a_2, \dots, a_{12})$ represents the 3×4 fuzzy membership matrix

$$W = \begin{pmatrix} a_1 & a_4 & a_7 & a_{10} \\ a_2 & a_5 & a_8 & a_{11} \\ a_3 & a_6 & a_9 & a_{12} \end{pmatrix},$$

where W satisfies conditions (10.13a), (10.13b), and (10.13c). We call a chromosome representing a fuzzy membership matrix that satisfies conditions (10.13a), (10.13b), and (10.13c) a legal chromosome.

10.10.2 Initialization Process

In the initialization phase, a population of N legal chromosomes is generated, where N is the size of the population. To generate a chromosome $(a_1, a_2, \dots, a_{n \cdot k})$, we employ the method introduced by Zhao et al. (1996), which is described as follows:

for $i = 1$ to n **do**

Generate k random numbers $v_{i1}, v_{i2}, \dots, v_{ik}$ from $[0, 1]$ for the i th point of the chromosome;

Calculate $a_{(j-1)*n+i} = v_{ij}/\sum_{l=1}^k v_{il}$ for $j = 1, 2, \dots, k$;

end for

If the produced chromosome satisfies conditions (10.13a), (10.13b), and (10.13c), then generate the next chromosome; otherwise repeat the above process.

The process described above is repeated N times to generate an initial population.

10.10.3 Selection Process

To describe the selection process, let us first introduce how to calculate the fitness of a chromosome. In our algorithm, we use the well-known rank-based evaluation function

$$F(s_i) = \beta(1 - \beta)^{r_i-1}, \quad (10.14)$$

where s_i ($1 \leq i \leq N$) is the i th chromosome in the population, r_i is the rank of s_i , and $\beta \in [0, 1]$ is a parameter indicating the selective pressure of the algorithm. Note that the chromosome with rank 1 is the best one and the chromosome with rank N is the worst one.

In our algorithm, the selection process is based on spinning the roulette wheel (Zhao et al., 1996) N times, where each time a chromosome is selected for the next population. Let P_j ($0 \leq j \leq N$) be the cumulative probabilities defined as

$$P_j = \begin{cases} 0 & \text{for } j = 0, \\ \frac{\sum_{i=1}^j F(s_i)}{\sum_{i=1}^N F(s_i)} & \text{for } j = 1, 2, \dots, N. \end{cases}$$

Then the new population is generated as follows.

for $i = 1$ to N **do**

 Generate a random real number v from $[0, 1]$;

if $P_{j-1} < v < P_j$ **then**

 Select s_j ;

end if

end for

10.10.4 Crossover Process

After the selection process, the population will go through a crossover process. As in GKA (Krishna and Narasimha, 1999), in our algorithm we employ a one-step fuzzy k -modes algorithm as the crossover operator. We update each chromosome in the population as follows.

for $t = 1$ to N **do**

 Let W_t be the fuzzy membership matrix represented by s_t ;

 Obtain the new set of cluster centers \hat{Z}_t given W_t according to Theorem 8.1;

 Obtain the fuzzy membership matrix \hat{W}_t given \hat{Z}_t according to Theorem 8.2;

 Replace s_t with the chromosome representing \hat{W}_t .

end for

10.10.5 Mutation Process

In the mutation process, each gene has a small probability P_m (say, 0.01) of mutating, decided by generating a random number (the gene will mutate if the random number is less than 0.01; otherwise it won't). In our algorithm, a change of one gene of a chromosome will trigger a series of changes of genes in order to satisfy (10.13b). Thus, in the mutation process, the fuzzy memberships of a point in a chromosome will be selected to mutate together with probability P_m . The mutation process is described as follows.

```

for  $t = 1$  to  $N$  do
    Let  $(a_1, a_2, \dots, a_{n \cdot k})$  denote the chromosome  $s_t$ ;
    for  $i = 1$  to  $n$  do
        Generate a random real number  $v \in [0, 1]$ ;
        if  $v \leq P_m$  then
            Generate  $k$  random numbers  $v_{i1}, v_{i2}, \dots, v_{ik}$  from  $[0, 1]$  for the  $i$ th point of the
            chromosome;
            Replace  $a_{(j-1)*n+i}$  with  $v_{ij} / \sum_{l=1}^k v_{il}$  for  $j = 1, 2, \dots, k$ ;
        end if
    end for
end for

```

10.10.6 Termination Criterion

In our algorithm, the processes of selection, one-step fuzzy k -modes, and mutation are executed for G_{max} , the maximum number of iterations (or generations). The best chromosome up to the last generation provides the solution to the clustering problem. We have also implemented the elitist strategy (Cowgill et al., 1999) at each generation by creating $N - 1$ children and retaining the best parent of the current generation for the next generation.

10.11 SARS

SARS (Simulated Annealing using Random Sampling) (Hua et al., 1994) is a decomposition-based simulated annealing (SA) approach for data clustering that addresses the issue of excessive disk accesses during annealing. In this algorithm, the clustering problem is formulated as a graph partition problem (GPP). In order to reduce the costly disk I/O activities while obtaining optimal results, the statistical sampling technique is employed to select subgraphs of the GPP into memory for annealing.

Given a data set D with n records, the data set D can be represented as a graph $G = (N, A)$ in which

- N denotes the set of nodes representing the records;
- A denotes the set of edges that connect related records;
- a weight $w_j \geq 1$ is assigned to each arc $a_j \in A$;

- a partitioning divides N approximately evenly into k disjoint subgraphs N_1, N_2, \dots, N_v , where v is a given parameter;
- a cost function C is defined as $\sum w_j$, where the summation ranges over those arcs $a_j \in A$ that connect nodes in two different subgraphs N_i and $N_{i'}$ ($1 \leq i, i' \leq v$, and $i \neq i'$).

The goal of the algorithm is to find a partitioning of the graph G such that the cost function $C = \sum w_j$ is minimized. Since finding an optimal partitioning for GPP is NP-hard, heuristics (e.g., SA) have been proposed to find suboptimal solutions.

Since almost all SA schemes are memory-based algorithms, they are not suited for solving large problems. One possible solution is to use various buffer replacement strategies, but this phenomenon will incur excessive disk accesses and force the machine to run at the speed of the I/O subsystem. SARS, a decomposition-based approach to solving the GPP, is proposed to address the excessive disk access problem.

The conventional SA algorithm begins with an initial partitioning of the graph. The partitioned graph is perturbed by swapping two randomly selected nodes from two different subgraphs. The cost function C is reevaluated and the difference, ΔC , between the costs of the new and the current states, is computed. If $\Delta C < 0$ (downhill), the proposed perturbation is accepted and the new state is used as the starting point for the next move evaluation. If $\Delta C \geq 0$ (uphill), the perturbation is accepted with a probability of $e^{-\frac{\Delta C}{T}}$, where T is the current temperature. This will prevent the searches from becoming trapped in a local minimum by accepting a new state with a higher cost. This process of propose-evaluate-decide (P-E-D) is repeated until an equilibrium condition is reached. Then the temperature is decreased by some factor α and the P-E-D process is resumed with the new temperature. The whole SA process is terminated when a prescribed frozen temperature is met.

There are four parameters in the SA algorithm: the initial temperature T_0 , the equilibrium criterion, the temperature decrement factor α , and the frozen condition. The initial temperature T_0 should be high enough such that virtually all uphill moves are accepted. In SARS, the initial temperature T_0 is determined by running a procedure that works as follows. Initially, T_0 is set to zero, and then a sequence of moves are generated. After each move, a new value of T_0 is computed according to

$$T_0 = \frac{\overline{\Delta C}^{(+)}}{\ln \left(\frac{m_2}{m_2 \chi_0 - m_1 (1 - \chi_0)} \right)},$$

where χ_0 is an initial acceptance ratio defined as the ratio of the number of accepted moves over the number of attempted moves at the initial temperature; m_1 and m_2 denote the number of downhill and uphill moves, respectively, obtained so far; and $\overline{\Delta C}^{(+)}$ is the average difference in cost over m_2 uphill moves. The final value of T_0 is used as the initial temperature for the SA algorithm. In SARS, χ_0 is set to 0.95. Typically, α is chosen in $[0.8, 0.99]$. In SARS, α is set to 0.9. The equilibrium criterion determines the number M of iterations of the inner loop, which is called the length of the Markov chains. In SARS, $M = \beta n$, where β ranges from 5 to 8 and n is the number of records in the data set. The frozen criterion is T decreased below 0.001 and the cost values of the last states of two consecutive Markov chains are identical.

ALGORITHM 10.7. The SARS algorithm.

```

1: Start with an initial state,  $S$ , by randomly partitioning the graph  $G$  into  $v$  subgraphs;
2:  $T \leftarrow T_0$  ( $T_0$  is determined by a separate procedure);
3: while not frozen do
4:   while not equilibrium do
5:     if a nodes-perturbing move then
6:       Perturb  $S$  by swapping nodes to get a new state  $S'$ ;
7:        $\Delta C \leftarrow C(S') - C(S)$ ;
8:       if  $\Delta C < 0$  then
9:          $S \leftarrow S'$ ;
10:      else
11:         $S \leftarrow S'$  with probability of  $e^{-\frac{\Delta C}{T}}$ ;
12:      end if
13:    else
14:      Perform a buffer-perturbing move;
15:    end if
16:   end while
17:    $T \leftarrow \alpha T$ ;
18: end while

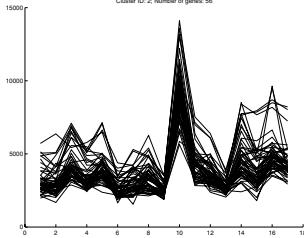
```

The SARS algorithm is described in Algorithm 10.7. In this algorithm, a decomposition approach is used to perform SA. Initially, the graph is arbitrarily partitioned into v subgraphs of approximately equal size, and then k subgraphs are randomly selected and brought into memory. Thus there are two operations, one, called the nodes-perturbing move, swaps two nodes of the k subgraphs in the memory, while the other called the buffer-perturbing move, randomly selects k subgraphs. To make a choice between the two moves, a random number is generated at each trial. If the number is less than a constant c ($0 < c < 1$), a buffer-perturbing move is performed; otherwise, a nodes-perturbing move is performed. It has been shown that the decomposition-based approach for SA preserves the convergence property of the SA process (Hua et al., 1994).

10.12 Summary

Some search-based clustering techniques have been presented and discussed in this chapter. To conduct a clustering based on search-based techniques, one first needs to define an evaluation function (or objective function) for the partitions of the data, and then a search-based technique is employed to find the globally optimal partition of the data. Although search-based clustering algorithms cannot guarantee a globally optimal solution, they do work toward the globally optimal solution.

The SA algorithm (McErlean et al., 1990; Bell et al., 1990; Klein and Dubes, 1989; Selim and Al-Sultan, 1991; Hua et al., 1994) is also used for the purpose of clustering.



Chapter 11

Graph-based Clustering Algorithms

A graph-based clustering algorithm will first construct a graph or hypergraph and then apply a clustering algorithm to partition the graph or hypergraph. A link-based clustering algorithm can also be considered as a graph-based one, because we can think of the links between data points as links between the graph nodes.

11.1 Chameleon

The chameleon (Karypis et al., 1999) algorithm is a graph-based clustering algorithm. Given a similarity matrix of the database, construct a sparse graph representation of the data items based on the commonly used k -nearest neighbor graph approach. Finally, use the agglomerative hierarchical clustering algorithm to merge the most similar subclusters by taking into account the relative interconnectivity and closeness of the clusters.

The *relative interconnectivity* between any pair of clusters C_i and C_j is given by

$$RI(C_i, C_j) = \frac{|EC_{\{C_i, C_j\}}|}{\frac{|EC_{C_i}| + |EC_{C_j}|}{2}},$$

where EC_{C_i} is the min-cut bisector of the cluster C_i , which characterizes the interconnectivity of the cluster C_i . $EC_{\{C_i, C_j\}}$ is the absolute interconnectivity between a pair of clusters C_i and C_j that can be defined to be the sum of the weights of the edges that connect vertices in C_i to vertices in C_j .

The *relative closeness* between a pair of clusters C_i and C_j is defined to be the absolute closeness between C_i and C_j normalized with respect to the internal closeness of the two clusters C_i and C_j , i.e.,

$$RC(C_i, C_j) = \frac{\bar{S}_{EC_{\{C_i, C_j\}}}}{\frac{|C_i|}{|C_i| + |C_j|} \bar{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i| + |C_j|} \bar{S}_{EC_{C_j}}},$$

Table 11.1. Description of the chameleon algorithm, where n is the number of data in the database and m is the number of initial subclusters.

Clustering Data Type	Categorical data
Time Complexity	$O(nm + n \log n + m^2 \log m)$
Algorithm Type	hierarchical

where $\bar{S}_{EC_{C_i}}$ is the average weight of the edges that belong in the min-cut bisector of the cluster C_i , and $\bar{S}_{EC_{[C_i, C_j]}}$ is the average weight of the edges that connect vertices in C_i to vertices in C_j .

ALGORITHM 11.1. The procedure of the chameleon algorithm.

Require: the similarity matrix of the data set;

- 1: Find initial subclusters;
- 2: Use a graph-partitioning algorithm to partition the k -nearest neighbor graph of the database into a large number of subclusters such that the edge-cut is minimized;
- 3: Merge the subclusters generated in step 2 using a dynamic framework.

The general procedure of the chameleon algorithm is listed in Algorithm 11.1. The general properties of the chameleon algorithm are shown in Table 11.1.

11.2 CACTUS

In the algorithm CACTUS (Categorical Clustering Using Summaries) developed by Ganti et al. (1999), the interattribute and intra-attribute summaries of the database are constructed, and then a graph called the similarity graph is defined according to those summaries. Finally, the clusters are found with respect to those graphs.

CACTUS uses support (cf. Section 6.7.3) to measure the similarity between two categorical attributes. Let A_1, \dots, A_n be a set of categorical attributes with domains D_1, \dots, D_n , respectively. Let D be a set of tuples with each tuple $t \in D_1 \times \dots \times D_n$. Let $a_i \in D_i$ and $a_j \in D_j$, $i \neq j$. Now let $C_i \subseteq D_i$ for $i = 1, 2, \dots, n$ and $|C_i| > 1$, $\alpha > 1$. $C = \{C_1, C_2, \dots, C_n\}$ is said to be a cluster over $\{A_1, A_2, \dots, A_n\}$ if the following conditions are satisfied:

1. $\forall i, j \in \{1, 2, \dots, n\}, i \neq j$, C_i and C_j are strongly connected.
2. $\forall i, j \in \{1, 2, \dots, n\}, i \neq j$, there is no $C'_i \supset C_i$ such that $\forall j \neq i$, C'_i and C_j are strongly connected.
3. The support satisfies $\sigma_D(C) \geq \alpha \cdot ES(C)$, where $ES(C)$ is the expected support of C under the assumption that the attributes are independent.

Let $a_1, a_2 \in D_i$ and $x \in D_j$. If (a_1, x) and (a_2, x) are strongly connected, then (a_1, a_2) are similar to each other with respect to A_j at some level of similarity (Ganti et al., 1999).

In this algorithm, the interattribute summary Σ_{IJ} is defined by

$$\Sigma_{IJ} = \{\Sigma_{ij} : i, j \in \{1, 2, \dots, n\}, i \neq j\},$$

where

$$\Sigma_{ij} = \{(a_i, a_j, \sigma_D^*(a_i, a_j)) : a_i \in D_i, a_j \in D_j, \text{ and } \sigma_D^*(a_i, a_j) > 0\},$$

and $\sigma_D^*(a_i, a_j)$ is defined as

$$\sigma_D^*(a_i, a_j) = \begin{cases} \sigma_D(a_i, a_j) & \text{if } a_i \text{ and } a_j \text{ are strongly connected,} \\ 0 & \text{otherwise.} \end{cases}$$

The intra-attribute summary Σ_{II} is defined by

$$\Sigma_{II} = \{\Sigma_{ii}^j : i, j \in \{1, 2, \dots, n\} \text{ and } i \neq j\},$$

where

$$\Sigma_{ii}^j = \{(a_{i1}, a_{i2}, \gamma^j(a_{i1}, a_{i2})) : a_{i1}, a_{i2} \in D_i, \text{ and } \gamma^j(a_{i1}, a_{i2}) > 0\},$$

and $\gamma^j(a_{i1}, a_{i2})$ is defined as

$$\gamma^j(a_{i1}, a_{i2}) = |\{x \in D_j : \sigma^*(a_{i1}, x) > 0 \text{ and } \sigma^*(a_{i2}, x) > 0\}|.$$

The CACTUS algorithm is described in Algorithm 11.2.

ALGORITHM 11.2. The CACTUS algorithm.

- 1: Compute the interattribute summaries and intra-attribute summaries from the database {summarization phase};
- 2: Analyze each attribute to compute all cluster projections on it, and then synthesize candidate clusters on sets of attributes from the cluster projections on individual attributes {clustering phase};
- 3: Compute the actual clusters from the candidate clusters {validation phase}.

11.3 A Dynamic System-based Approach

The clustering algorithm for a categorical data set proposed by Gibson et al. (2000) first constructs a hypergraph according to the database and then clusters the hypergraph using a discrete dynamic system. To construct the hypergraph, the algorithm links up attributes or columns via their common values in rows (horizontal co-occurrence) and links up the rows via their common values for the same attribute (vertical co-occurrence). A weight is assigned to each node of the hypergraph and is normalized so that the sum of the squares of the weights of the nodes associated with each attribute is 1. Then, it repeats updating the weights until a fixed point is obtained.

The algorithm developed by Gibson et al. (2000) may not converge. Zhang et al. (2000b) proposed a revised dynamical system to guarantee the convergence. The normalization functions in (Gibson et al., 2000) and (Zhang et al., 2000b) are different: in the former, the normalization function is associated with each field, while in the latter, the normalization normalizes the weights of all attribute values.

The algorithm proceeds as follows. Firstly, select an item of interest, A , and assign a small weight to A . Then propagate this weight to items with which A co-occurs frequently. Those items that have acquired the weight propagate further. Then, items highly related to A acquire the weight, even without direct co-occurrence. The nodes of large weights in the basin of attraction of this iteration tend to represent natural groups of data, and the positive and negative weights in the nonprincipal basins tend to represent good partitions of the data.

If we view the database as a table of relational data and let the table consist of a set of k fields (columns), then each field denotes an attribute. Each field can take one of many values. Each possible value in each possible field is represented by a node. Then the database is a set of tuples with each tuple consisting of one node from each field. The concept configuration was defined as an assignment of a weight w_v to each node v . A normalization function $N(w)$ (w is the entire configuration) is used to rescale the weights of the nodes associated with each field so that their squares add up to 1. The algorithm is described simply in Algorithm 11.3.

ALGORITHM 11.3. The dynamic system–based clustering algorithm.

- 1: Construct the hypergraph according to the database;
- 2: Choose an initial configuration (there are many methods of doing this; see (Gibson et al., 2000) for details);
- 3: Define a function f and repeat applying f to the configuration until a fixed point of the dynamical system is obtained. The function f can be defined as follows (Gibson et al., 2000):

To update the weight w_v : for each tuple $\tau = \{v, u_1, u_2, \dots, u_{k-1}\}$ containing v do

$$\begin{aligned} x_\tau &\leftarrow \oplus(u_1, u_2, \dots, u_{k-1}), \\ w_v &\leftarrow \sum_\tau x_\tau, \end{aligned}$$

where \oplus is a *combiner* function that can be chosen from among many operators, such as product and addition operators.

In Zhang et al. (2000b), the function f is defined as follows:

To update the configuration w : create a temporary configuration w' with weights w'_1, w'_2, \dots, w'_m for each weight $w_{u_i} \in w$ as follows: for each tuple $\tau = \{u_1, u_2, \dots, u_k\}$ containing u_i do

$$\begin{aligned} x_\tau &\leftarrow w_{u_1} + \dots + c w_{u_i} + \dots + w_{u_k}, \\ w'_{u_i} &\leftarrow \sum_\tau x_\tau. \end{aligned}$$

Then update $w \leftarrow w'$ and normalize the configuration w .

- 4: Group the data whose attribute values have the greatest weights and let the remaining data do the same process in steps 1 to 4 until more clusters are obtained if necessary.

11.4 ROCK

ROCK (RObust Clustering using linKs) (Guha et al., 2000b) is an agglomerative hierarchical clustering algorithm that employs links to merge clusters. ROCK uses the link-based similarity measure (cf. Section 6.7.2) to measure the similarity between two data points and between two clusters.

In order to characterize the best clusters, a criterion function is defined to be

$$E_l = \sum_{i=1}^k n_i \cdot \sum_{p_q, p_r \in C_i} \frac{\text{link}(p_q, p_r)}{n_i^{1+2f(\theta)}},$$

where k is the number of clusters, n_i is the size of cluster i , C_i denotes the cluster i , and $f(\theta)$ is a function of the parameter θ . To determine the function $f(\theta)$ is a difficult problem.

Also, in order to decide whether or not to merge clusters C_i and C_j , the goodness measure (i.e., the similarity measure between two clusters) $g(C_i, C_j)$ for merging clusters C_i and C_j is defined as

$$g(C_i, C_j) = \frac{\text{link}[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}},$$

where $\text{link}[C_i, C_j]$ is the number of cross-links between clusters C_i and C_j , i.e.,

$$\text{link}[C_i, C_j] = \sum_{p_q \in C_i, p_r \in C_j} \text{link}(p_q, p_r).$$

The clustering algorithm for ROCK is described in Algorithm 11.4. The properties of the ROCK algorithm are listed in Table 11.2.

ALGORITHM 11.4. The ROCK algorithm.

Require: n : number of data points; D : data set;

- 1: **for** $i = 1$ to $n - 1$ **do**
- 2: **for** $j = i + 1$ to n **do**
- 3: Compute $\text{link}(p_i, p_j)$;
- 4: **end for**
- 5: **end for**
- 6: **for** $i = 1$ to n **do**
- 7: Build a local heap $q[i]$ that contains every cluster C_j such that $\text{link}[C_i, C_j]$ is non-zero
 {at first, each data point forms a cluster};
- 8: **end for**
- 9: Build a global heap Q that is ordered in decreasing order of the best goodness measure and contains all the clusters;
- 10: **repeat**

Table 11.2. The properties of the ROCK algorithm, where n is the number of data points in the data set, m_m is the maximum number of neighbors for a point, and m_a is the average number of neighbors.

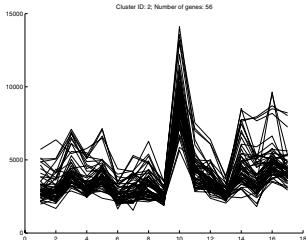
Clustering Data Type	Categorical data
Time Complexity	$O(n^2 + nm_m m_a + n^2 \log n)$
Space Complexity	$O(\min\{n^2, nm_m m_a\})$
Algorithm Type	agglomerative hierarchical

- 11: Merge the max cluster $C_j \in Q$ and the max cluster in $q[j]$ into a new cluster W , delete the two clusters from Q , and update $link$;
- 12: **until** the number of links between every pair of the remaining clusters becomes zero or $size(Q) = k$.

11.5 Summary

Hubert (1974) reviewed the relationship between graph theory and the clustering of a set of objects. Other discussions are provided in Jardine and Sibson (1968), Jardine (1971), Wirth et al. (1966), Gotlieb and Kumar (1968), Zahn (1971), Gower and Ross (1969), Augustson and Minker (1970), Wu and Leahy (1993), Estabrook (1966), Baker and Hubert (1976), and Legendre and Rogers (1972).

A branch of graph-based clustering that is not discussed in this chapter is spectral clustering. Spectral methods have recently emerged as effective methods for data clustering, image segmentation, Web-ranking analysis, and dimension reduction. Interested readers are referred to (Alpert and Yao, 1995), (Fowlkes et al., 2004), (Kannan et al., 2004), (Ng et al., 2002), (Spielmat and Teng, 1996), (Zien et al., 1997), (Zien et al., 1999), and (Chan et al., 1994).



Chapter 12

Grid-based Clustering Algorithms

Density-based and/or grid-based approaches are popular for mining clusters in a large multidimensional space wherein clusters are regarded as denser regions than their surroundings. In this chapter, we present some grid-based clustering algorithms.

The computational complexity of most clustering algorithms is at least linearly proportional to the size of the data set. The great advantage of grid-based clustering is its significant reduction of the computational complexity, especially for clustering very large data sets.

The grid-based clustering approach differs from the conventional clustering algorithms in that it is concerned not with the data points but with the value space that surrounds the data points. In general, a typical grid-based clustering algorithm consists of the following five basic steps (Grabusts and Borisov, 2002):

1. Creating the grid structure, i.e., partitioning the data space into a finite number of cells.
2. Calculating the cell density for each cell.
3. Sorting of the cells according to their densities.
4. Identifying cluster centers.
5. Traversal of neighbor cells.

12.1 STING

Wang et al. (1997) proposed a STatistical INformation Grid-based clustering method (STING) to cluster spatial databases. The algorithm can be used to facilitate several kinds of spatial queries. The spatial area is divided into rectangle cells, which are represented by a hierarchical structure. Let the root of the hierarchy be at level 1, its children at level 2, etc. The number of layers could be obtained by changing the number of cells that form a higher-level cell. A cell in level i corresponds to the union of the areas of its children in level $i + 1$. In the algorithm STING, each cell has 4 children and each child corresponds to one quadrant of the parent cell. Only two-dimensional spatial space is considered in this algorithm. Some related work can be found in (Wang et al., 1999b).

For each cell, the attribute-dependent and attribute-independent parameters are defined as follows:

- n — number of objects (points) in the cell;
- m — mean of all values in this cell;
- s — standard deviation of all values of the attribute in this cell;
- min — the minimum value of the attribute in this cell;
- max — the maximum value of the attribute in this cell;
- $dist$ — the type of distribution that the attribute value in this cell follows.

Given the hierarchical structure of grid cells, we can start with the root to calculate the likelihood that this cell is relevant to the query at some confidence level using the parameters of this cell. The algorithm is summarized in Algorithm 12.1.

ALGORITHM 12.1. The STING algorithm.

- 1: Construct the grid hierarchical structure according to the database and generate the parameters of each cell;
- 2: Determine a layer to begin with;
- 3: For each cell in this layer, compute the confidence interval of the probability that this cell is relevant to the query;
- 4: **if** this layer is not the bottom layer **then**
- 5: Go to the next level in the hierarchy structure and go to step 3 for the relevant cells of the higher-level layer;
- 6: **else if** the specification of the query is met **then**
- 7: Find the regions of relevant cells and return those regions that meet the requirements of the query;
- 8: **else**
- 9: Reprocess the data in the relevant cells and return the results that meet the requirements of the query;
- 10: **end if**

The algorithm cannot be scaled to high-dimensional databases because the computational complexity is $O(K)$, where K is the number of cells in the bottom layer. In high-dimensional data, if each cell has four children, then the number of cells in the second layer will be 2^d , where d is the number of dimensions of the database.

STING+ (Wang et al., 1999b) is an active spatial data-mining algorithm that based on active database systems and the algorithm STING. This algorithm enables users to specify complicated conditions. STING and STING+ have only been designed for low-dimensional data, and there is no straightforward extension to the high-dimensional case (Keim and Hinneburg, 1999).

12.2 OptiGrid

We introduce the OptiGrid clustering algorithm (Keim and Hinneburg, 1999), a very efficient algorithm for clustering high-dimensional databases with noise, in this section.

A cutting plane is a $(d - 1)$ -dimensional hyperplane consisting of all points y satisfying the equation $\sum_{i=1}^d w_i y_i = 1$. The cutting plane partitions the d -dimensional space \mathbb{R}^d into two half-spaces. The decision function $H(x)$, which determines where a point x is located, is defined to be

$$H(x) = \begin{cases} 1 & \sum_{i=1}^d w_i x_i \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

According to the cutting plane, a multidimensional grid G for the data space S is defined to be a set $H = \{H_1, H_2, \dots, H_k\}$ of $(d - 1)$ -dimensional cutting planes. The coding function $c^G : S \rightarrow \mathbb{N}$ is defined as

$$x \in S, c(x) = \sum_{i=1}^k 2^i \cdot H_i(x).$$

The algorithm uses cutting planes to determine the grids. First, the algorithm OptiGrid chooses a set of contracting projections and then uses the projections of the data space to determine the cutting planes efficiently. The algorithm then finds the clusters using the density function \hat{f}^D given by

$$\hat{f}^D = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right),$$

where D is a set of N d -dimensional points, h is the smoothness level, and K is the kernel density estimator.

A center-based cluster for a maximum x^* of the density function \hat{f}^D is the subset $C \subseteq D$, with $x \in C$ being density-attracted by x^* and $\hat{f}^D(x^*) \geq \xi$. Points $x \in D$ are called outliers if they are density-attracted by a local maximum x_0^* with $\hat{f}^D(x_0^*) < \xi$. The notion of density-attraction is defined by the gradient of the density function.

According to the definition of a center-based cluster, the algorithm OptiGrid is described in Algorithm 12.2.

ALGORITHM 12.2. The OptiGrid algorithm.

Require: data set D, q, min_cut_score

- 1: Determine a set of contracting projections $P = \{P_0, P_1, \dots, P_k\}$ and calculate all the projections of the data set D : $P_i(D), i = 1, 2, \dots, k$;
- 2: Initialize a list of cutting planes $BEST_CUT \Leftarrow \Phi, CUT \Leftarrow \Phi$;
- 3: **for** $i = 0$ to k **do**
- 4: $CUT \Leftarrow$ best local cuts $P_i(D)$;
- 5: $CUT_SCORE \Leftarrow$ Score best local cuts $P_i(D)$;
- 6: Insert all the cutting planes with a score $\geq min_cut_score$ into $BEST_CUT$;
- 7: **if** $BEST_CUT = \Phi$ **then**
- 8: Return D as a cluster;

```

9:   else
10:    Select the  $q$  cutting planes of the highest score from  $BEST\_CUT$  and construct
     a multidimensional grid  $G$  using the  $q$  cutting planes;
11:    Insert all data points in  $D$  into  $G$  and determine the highly populated grid cells in
      $G$ ; add these cells to the set of clusters  $C$ ;
12:    Refine  $C$ ;
13:    for all clusters  $C_i$  in  $C$  do
14:       Do the same process with data set  $C_i$ ;
15:    end for
16:   end if
17: end for

```

OptiGrid is efficient for clustering large databases. However, it may be very slow if the clusters are embedded in a low-dimensional subspace of a very high-dimensional database, because it uses a recursive method to reduce the dimension by one at every step. The clusters produced by OptiGrid are multicentered, i.e., a cluster may have more than one center. The algorithm is hierarchical, but the time complexity is between $O(N \cdot d)$ and $O(d \cdot N \cdot \log N)$.

12.3 GRIDCLUS

Schikuta (1996) proposed a hierarchical algorithm named GRIDCLUS for clustering very large data sets. The main idea of this algorithm is to use a multidimensional grid data structure to organize the value space surrounding the pattern values rather than to organize the patterns themselves. The patterns are grouped into blocks, which are clustered by a topological neighbor search algorithm.

The algorithm first establishes a grid structure that partitions the value space and administers the points by a set of surrounding rectangular blocks. Given a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ in a d -dimensional space, a block is defined to be a d -dimensional hyperrectangle containing up to a maximum of bs points, where bs denotes the block size. Let B_1, B_2, \dots, B_m be blocks of the data set D . They have the following properties:

1. $\forall \mathbf{x}_i \in D, \mathbf{x}_i \in B_j$ for some j ;
2. $B_j \cap B_k = \emptyset$ if $j \neq k$;
3. $B_j \neq \emptyset$;
4. $\bigcup_{j=1}^m B_j = D$.

The grid structure consists of three parts: d scales (one for each dimension), the grid directory (a d -dimensional array), and b data blocks. For a detailed description of the grid structure, we refer to Nievergelt et al. (1984).

Then the GRIDCLUS algorithm clusters the blocks B_i ($i = 1, 2, \dots, m$) into a nested sequence of nonempty and disjoint clusterings. To do this, a density index of each block is

defined via the number of points and the spatial volume of the block. Precisely, the density index of a block B is defined as

$$D_B = \frac{p_B}{V_B},$$

where p_B is the number of data points contained in the block B and V_B is the spatial volume of the block B , i.e.,

$$V_B = \prod_{j=1}^d e_{B_j},$$

with d being the dimensionality of the data space and e_{B_j} the extent of the block B_j in the j th dimension.

Blocks with the highest density index become clustering centers. The remaining blocks are then clustered iteratively in order of their density indices. Through this process, new cluster centers are built and existing clusters are merged. The algorithm only allows blocks adjacent to a cluster to be merged. This is done by a neighbor search procedure starting at the cluster center and inspecting adjacent blocks. If a neighbor block is found, the search proceeds recursively with the block.

ALGORITHM 12.3. The GRIDCLUS algorithm.

```

1: Set  $u := 0$ ,  $W[] := \{\}$ ,  $C[] := \{\}$  {initialization};
2: Create the grid structure and calculate the block density indices;
3: Generate a sorted block sequence  $B_{1'}$ ,  $B_{2'}$ ,  $\dots$ ,  $B_{b'}$  and mark all blocks “not active” and
   “not clustered”;
4: while a “not active” block exists do
5:    $u \Leftarrow u + 1$ ;
6:   mark first  $B_{1'}$ ,  $B_{2'}$ ,  $\dots$ ,  $B_{j'}$  with equal density index “active”;
7:   for each “not clustered” block  $B_{l'} := B_{1'}, B_{2'}, \dots, B_{j'}$  do
8:     Create a new cluster set  $C[u]$ ;
9:      $W[u] \Leftarrow W[u] + 1$ ,  $C[u, W[u]] \leftarrow B_{l'}$ ;
10:    Mark block  $B_{l'}$  clustered;
11:     $NEIGHBOR\_SEARCH(B_{l'}, C[u, W[u]])$ ;
12:   end for
13:   for each “not active” block  $B$  do
14:      $W[u] \Leftarrow W[u] + 1$ ,  $C[u, W[u]] \leftarrow B$ ;
15:   end for
16:   Mark all blocks “not clustered”;
17: end while
```

The GRIDCLUS algorithm consists of five main parts: (a) insertion of points into the grid structure, (b) calculation of density indices, (c) sorting of the blocks with respect to their density indices, (d) identification of cluster centers, and (e) traversal of neighbor blocks. The GRIDCLUS algorithm is described in Algorithm 12.3. The function $NEIGHBOR_SEARCH$ is the recursive procedure described in Algorithm 12.4.

ALGORITHM 12.4. Procedure *NEIGHBOR_SEARCH(B,C)*.

```

1: for each “active” and “not clustered” neighbor  $B'$  of  $B$  do
2:    $C \leftarrow B'$ ;
3:   Mark block  $B'$  “clustered”;
4:   NEIGHBOR_SEARCH( $B', C$ );
5: end for

```

The BANG-clustering algorithm (Schikuta and Erhart, 1997) is a hierarchical clustering algorithm based on GRIDCLUS. It is an extension of the grid clustering algorithm presented in (Schikuta, 1996). In the BANG-clustering system, the value space is partitioned binarily into a hierarchical set of grid regions and each grid region is assigned a unique identity (r, l) , where r is the region number and l is the level number. The blocks are sorted according to their density indices. Blocks with the highest density index become clustering centers and the remaining blocks are clustered iteratively in order of their density indices; the remaining blocks either build new cluster centers or merge with existing clusters. The main procedures of the BANG-clustering algorithm are described as follows:

1. Partition the value space binarily into rectangular grids such that each grid contains up to a maximum of p_{max} data points.
2. Construct a binary tree that contains the populated grids according to the level.
3. Calculate the dendrogram in which the density indices of all regions are sorted in decreasing order.
4. Starting with the highest-density region (i.e., the first region), determine all the neighbors and classify them in decreasing order; place the found regions to the right of the original regions in the dendrogram.
5. Repeat step 4 for the remaining regions of the dendrogram.

12.4 GDILC

GDILC is a Grid-based Density-IsoLine Clustering algorithm proposed by Zhao and Song (2001). The algorithm is capable of eliminating outliers and finding clusters of various shapes. The main idea behind this algorithm is that the distribution of data samples can be depicted very well by the so-called density-isoline figure. A grid-based method is employed to calculate the density of each data sample and find relatively dense regions. Before this algorithm is used, all data samples are assumed to be normalized, i.e., all attribute values are in the range of $[0, 1]$.

First, a density-isoline figure (i.e., the contour figure of density) is formed from the distribution densities of data samples. Then clusters are discovered from the density-isoline figure. By varying isolines, clusters of different densities can be obtained. Calculating density and getting a proper density-isoline figure are key steps in the algorithm GDILC.

In GDILC, the density of a data sample is computed as the count of samples in its neighbor region. Let \mathbf{x} be a data sample. Then the density of \mathbf{x} is calculated by the formula

$$\text{Density}(\mathbf{x}) = |\{\mathbf{y} : f(\mathbf{y}, \mathbf{x}) \leq T\}|,$$

where T is a given threshold and $f(\mathbf{y}, \mathbf{x})$ is a function used to measure the dissimilarity between samples \mathbf{y} and \mathbf{x} ; for example, $f(\cdot, \cdot)$ can be the Minkowsky distance.

The density-isoline figure is obtained from the density vectors. The density vectors are computed by counting the elements of each row of the distance matrix that are less than RT , the radius of the neighbor region. The density-isoline figure is not explicitly drawn in the clustering procedure.

Since the distances between each pair of data samples have to be computed in order to compute the density vector, a grid-based method has been employed. Before computing distances, GDILC partitions each dimension into several intervals such that the data space is partitioned into many hyperrectangular cells. Let D be a d -dimensional data set and suppose each dimension is divided into m_i intervals that are numbered from 0 to $m_i - 1$. Then each cell can be labeled with the sequence numbers of intervals. For example, $C_{i_1 i_2 \dots i_d}$ is the cell that is in interval i_l of dimension l for $l = 1, 2, \dots, d$.

In GDILC, computing the density of a sample \mathbf{x} involves only distances between those samples in $C_{\mathbf{x}}$ and its neighbor cells, where $C_{\mathbf{x}}$ is the cell that contains the sample \mathbf{x} . In GDILC, two cells $C_{i_1 i_2 \dots i_d}$ and $C_{j_1 j_2 \dots j_d}$ are said to be neighbor cells if the following condition is satisfied:

$$|i_l - j_l| \leq 1 \text{ for } l = 1, 2, \dots, d.$$

ALGORITHM 12.5. The GDILC algorithm.

S_1 (*Initializing cells*) Divide each dimension into m intervals;

S_2 (*Computing distance threshold RT*) For each point \mathbf{x} , compute the distances between \mathbf{x} and every point in the neighbor cells of $C_{\mathbf{x}}$. Then compute the distance threshold RT by calculating the average distance from those distances;

S_3 (*Computing density vector and density threshold DT*) For each point \mathbf{x} , compute the density of \mathbf{x} by counting the number of points within RT of \mathbf{x} . Then compute the density threshold DT by calculating the average density from the density vector;

S_4 (*Clustering*) At first, take each point whose density is more than DT as a cluster. Then, for each point \mathbf{x} , check whether the distance from \mathbf{x} of each point whose density is more than DT in the neighbor cells of $C_{\mathbf{x}}$ is less than RT . If so, then merge the two clusters containing those two points. Continue the above merging process until all point pairs have been checked;

S_5 (*Removing outliers*) Remove those clusters whose sizes are less than a certain number.

The algorithm GDILC consists of five steps and the procedure is described in Algorithm 12.5 in detail. In GDILC, there are two important parameters: the distance threshold

RT and the density threshold DT . In order to obtain an optimal density-isoline figure, the values of RT and DT are calculated dynamically according to the number and the distribution of data samples. Precisely, the distance threshold RT is computed as

$$RT = \frac{mean(Dist)}{d \times coefRT},$$

where $Dist$ is the array of all interesting distances, $mean(Dist)$ is the average of $Dist$, d is the number of dimensions, and $coefRT$ is an adjustable coefficient that can be adjusted to get good clustering results.

In GDILC, the density threshold DT is computed from the formula

$$DT = \begin{cases} 2, & n < 1000, \\ \frac{mean(Density)}{\log_{10}(n)} \times coefDT, & n \geq 2000, \end{cases}$$

where $mean(Density)$ is the average of the densities of all data samples, and $coefDT$ is an adjustable coefficient in the range $[0.7, 1]$.

In this algorithm, the number of intervals m is computed by the formula:

$$m = \sqrt[m]{\frac{n}{coefM}},$$

where $coefM$ is an adjustable coefficient that can be treated as the average number of data samples in a cell.

Since a grid-based method is employed to reduce lots of unnecessary computation, the time complexity of GDILC is nearly $O(n)$.

12.5 WaveCluster

WaveCluster (Sheikholeslami et al., 2000) is an algorithm for clustering spatial data based on wavelet transforms. WaveCluster is insensitive to noise, capable of detecting clusters of arbitrary shape at different degrees of detail, and efficient for large databases. The key idea of WaveCluster is to apply wavelet transforms on the feature space, instead of the objects themselves, to find the dense regions in the feature space.

The algorithm first partitions the original data space into nonoverlapping hyperrectangles, i.e., cells. The j th dimension is segmented into m_j of intervals. Each cell \mathbf{c}_i is the intersection of one interval from each dimension and has the form $(c_{i1}, c_{i2}, \dots, c_{id})$, where $c_{ij} = [l_{ij}, h_{ij})$ is the right open interval in the partitioning of the j th dimension and d is the number of dimensions.

A point $\mathbf{x} = (x_1, x_2, \dots, x_d)$ is said to be contained in a cell \mathbf{x}_i if $l_{ij} \leq x_j < h_{ij}$ for $j = 1, 2, \dots, d$. Let $\mathbf{c}_i \cdot count$ denote the number of points contained in the cell \mathbf{c}_i . Then the algorithm applies a wavelet transform to $\mathbf{c}_i \cdot count$ values. Then the transformed space is defined as the set of cells after the wavelet transformation on the count values of the cells in the quantized space.

WaveCluster is capable of identifying clusters of arbitrary shapes. In this algorithm, a cluster is defined to be a set of significant cells $\{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m\}$ that are k -connected in

the transformed space. A cell is called a significant cell if its count in the transformed space is above a certain threshold τ . A cell \mathbf{c}_1 is an ϵ -neighbor of a cell \mathbf{c}_2 if both are either significant cells (in transformed space) or nonempty cells (in quantized space) and $D(\mathbf{c}_1, \mathbf{c}_2) \leq \epsilon$, where $D(\cdot, \cdot)$ is an appropriate distance metric and $\epsilon > 0$. A cell \mathbf{c}_1 is a $k - \epsilon$ -neighbor of a cell \mathbf{c}_2 if both are significant cells or both are nonempty cells and if \mathbf{c}_1 is one of the k prespecified ϵ -neighbors of \mathbf{c}_2 . Two cells \mathbf{c}_1 and \mathbf{c}_2 are said to be k -connected if there is a sequence of cells $\mathbf{c}_1 = \mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_j = \mathbf{c}_2$ such that \mathbf{p}_{i+1} is a $k - \epsilon$ -neighbor of \mathbf{p}_i for $i = 1, 2, \dots, j$.

WaveCluster is very efficient, especially for very large databases. The computational complexity of WaveCluster is $O(n)$, where n is the number of objects in the data set. The clustering results produced by WaveCluster are not affected by noise and, in addition, not sensitive to the order of input objects to be processed. WaveCluster is capable of finding arbitrarily shaped clusters and the number of clusters is not required in advance.

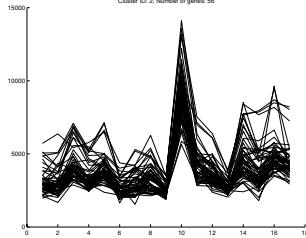
12.6 Summary

The main ideas behind the grid-based clustering algorithms are to use multiresolution grid data structures and to use dense grid cells to form clusters. In other words, the algorithms first quantize the original data space into a finite number of cells and then do all operations on the quantized space. The main characteristic of these grid-based clustering algorithms is their fast processing time, since similar data points will fall into the same cell and will be treated as a single point. This makes the algorithms independent of the number of data points in the original data set.

Most grid-based algorithms require users to specify a grid size or density thresholds. One difficulty of grid-based clustering algorithms is how to choose the grid size or density thresholds. Fine grid sizes lead to a huge amount of computation, while coarse grid sizes result in low quality of clusters. To alleviate this problem, Nagesh et al. (2001) proposed a technique of adaptive grids that automatically determines the size of grids based on the data distribution and does not require the user to specify any parameters like the grid size or the density thresholds.

Some grid-based clustering algorithms, such as STING (Wang et al., 1997) and WaveCluster (Sheikholeslami et al., 2000), are efficient only for low-dimensional data sets. Consider a data set in a 100-dimensional space, for example. If one splits each dimension only once at the center, then the resulting space is cut into $2^{100} \approx 10^{30}$ cells. Among the huge number of cells, most are empty and some may be filled with only one data point. It is impossible to determine a data distribution with such a coarse grid structure (Keim and Hinneburg, 1999). The algorithm OptiGrid due to Keim and Hinneburg (1999) is designed for the purpose of clustering high-dimensional data.

Unlike center-based clustering algorithms, grid-based clustering algorithms have received little attention. Further examples of grid-based clustering can be found in (Park and Lee, 2004), (Chang and Jin, 2002), (Chen et al., 2002), and (Ma and Chow, 2004).



Chapter 13

Density-based Clustering Algorithms

The density-based clustering approach is a methodology that is capable of finding arbitrarily shaped clusters, where clusters are defined as dense regions separated by low-density regions. A density-based algorithm needs only one scan of the original data set and can handle noise. The number of clusters is not required, since density-based clustering algorithms can automatically detect the clusters, along with the natural number of clusters (El-Sonbaty et al., 2004).

13.1 DBSCAN

Ester et al. (1996) proposed a density-based clustering algorithm called DBSCAN (Density-Based Spatial Clustering of Applications with Noise) to discover arbitrarily shaped clusters. Only one input parameter is required, and the algorithm also supports the user in determining an appropriate value for this input parameter.

To describe the algorithm, we start with some definitions and notation. An important concept in density-based algorithms is the ϵ -neighborhood of a point. Let \mathbf{x} be a point. Then the ϵ -neighborhood of \mathbf{x} is denoted by $N_\epsilon(\mathbf{x})$ and is defined as follows.

Definition 13.1 (ϵ -neighborhood of a point). *The ϵ -neighborhood of a point \mathbf{x} is defined as*

$$N_\epsilon(\mathbf{x}) = \{\mathbf{y} \in D : d(\mathbf{x}, \mathbf{y}) \leq \epsilon\},$$

where D is the data set and $d(\cdot, \cdot)$ is a certain distance function.

Definition 13.2 (Directly density-reachable). *A point \mathbf{x} is said to be directly density-reachable from a point \mathbf{y} (with respect to ϵ and N_{min}) if*

1. $\mathbf{x} \in N_\epsilon(\mathbf{y})$;
2. $|N_\epsilon(\mathbf{y})| \geq N_{min}$, where $|N_\epsilon(\mathbf{y})|$ denotes the number of points in $N_\epsilon(\mathbf{y})$.

Directly density-reachable is symmetric for pairs of core points (points inside a cluster), but it is in general not symmetric in case one core point and one border point (a point on the border of a cluster) are involved. As an extension of directly density-reachable, density-reachable, defined below, is also not symmetric in general. But density-connected is a symmetric relation.

Definition 13.3 (Density-reachable). A point \mathbf{x} is said to be density-reachable from a point \mathbf{y} if there is a sequence of points $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_i = \mathbf{y}$ such that \mathbf{x}_l is directly density-reachable from \mathbf{x}_{l+1} for $l = 1, 2, \dots, i - 1$.

Definition 13.4 (Density-connected). Two points \mathbf{x} and \mathbf{y} are said to be density-connected with respect to ϵ and N_{min} if there exists a point \mathbf{z} such that both \mathbf{x} and \mathbf{y} are density-reachable from \mathbf{z} with respect to ϵ and N_{min} .

A cluster is then very intuitively defined as a set of density-connected points that is maximal with respect to density-reachability. Mathematically, we have the following definition.

Definition 13.5 (Cluster). Let D be a data set. A cluster C with respect to ϵ and N_{min} is a nonempty subset of D satisfying the following conditions:

1. $\forall \mathbf{x}, \mathbf{y} \in D$, if $\mathbf{x} \in C$ and \mathbf{y} is density-reachable from \mathbf{x} with respect to ϵ and N_{min} , then $\mathbf{y} \in C$ (maximality).
2. $\forall \mathbf{x}, \mathbf{y} \in C$, \mathbf{x} and \mathbf{y} are density-connected with respect to ϵ and N_{min} (connectivity).

The noise is a set of points in the data set that do not belong to any cluster. We see from Definition 13.5 that a cluster contains at least N_{min} points. DBSCAN starts with an arbitrary point \mathbf{x} and finds all points that are density-reachable from \mathbf{x} with respect to ϵ and N_{min} . If \mathbf{x} is a core point, then a cluster with respect to ϵ and N_{min} is formed. If \mathbf{x} is a border point, then no points are density-reachable from \mathbf{x} and DBSCAN visits the next unclassified point. DBSCAN may merge two clusters if the two clusters are close to each other. In DBSCAN, the distance between two clusters C_1 and C_2 is defined as $d(C_1, C_2) = \min_{\mathbf{x} \in C_1, \mathbf{y} \in C_2} d(\mathbf{x}, \mathbf{y})$. As pointed out by Su et al. (2002), DBSCAN tends to merge many slightly connected clusters together.

DBSCAN requires two parameters, ϵ and N_{min} . These two parameters are used globally in the algorithm, i.e., the two parameters are the same for all clusters, so to choose the two parameters in advance is not easy. However, a heuristic is developed in DBSCAN to determine the parameters ϵ and N_{min} of the “thinnest” cluster in the database. This heuristic is called the sorted k -dist graph. Let $F_k : D \rightarrow \mathbb{R}$ be a function defined as

$$F_k(\mathbf{x}) = \text{distance between } \mathbf{x} \text{ and its } k \text{th nearest neighbor.}$$

Then $F_k(D)$ is sorted in descending order and plotted in a two-dimensional graph. ϵ is set to $F_4(\mathbf{z}_0)$, where \mathbf{z}_0 is the first point in the first “valley” of the graph of the sorted 4-dist graph, since the k -dist graph for $k > 4$ does not significantly differ from the 4-dist graph (Ester et al., 1996). Hence N_{min} is set to 4 for all two-dimensional data.

DBSCAN has several variations and extensions. Sander et al. (1998) generalized DBSCAN to GDBSCAN, which can cluster point objects as well as spatially extended objects with spatial and nonspatial attributes. GDBSCAN generalizes DBSCAN in two ways: the notion of a neighborhood and the measure in a neighborhood. For details, readers are referred to (Sander et al., 1998) and (Ester et al., 1997). PDBSCAN is a parallel version of DBSCAN proposed by Xu et al. (1999). DBSCAN is also extended by Ester et al. (1998) to incremental clustering. Zaiane and Lee (2002) proposed an algorithm called DBCluC (Density-Based Clustering with Constraints) based on DBSCAN to cluster spatial data in the presence of obstacles.

13.2 BRIDGE

Dash et al. (2001) proposed a hybrid clustering algorithm, called BRIDGE, which integrates the popular k -means algorithm and the density-based algorithm DBSCAN. BRIDGE enables DBSCAN to handle very large databases and at the same time improves the quality of k -means clusters by removing noise.

In BRIDGE, the k -means algorithm is first performed to partition the data set into k clusters, and then DBSCAN is applied to each partition to find dense clusters. The result of DBSCAN is then finally used to improve the k -means clusters, i.e., the k -means clusters are refined by removing the noise found by the density-based clustering.

To describe the algorithm, we first introduce some definitions. The core distance with respect to a cluster C is defined as half the distance between its center and its closest cluster center. A point is called a core point if it is not farther from its center than $\text{CoreDistance} - \epsilon$. The core region of a cluster is one inside which each data point is a core point. A point is called a $(+\epsilon)$ -core point if its distance from its cluster center is between CoreDistance and $\text{CoreDistance} + \epsilon$, whereas a point is called a $(-\epsilon)$ -core point if its distance from its cluster center is between CoreDistance and $\text{CoreDistance} - \epsilon$. A point is noncore if it is neither a core nor an ϵ -core ($(+\epsilon)$ -core or $(-\epsilon)$ -core) point. The noncore region is one inside which each point is noncore.

ALGORITHM 13.1. The BRIDGE algorithm.

Require: k : number of clusters;

- 1: Run the k -means algorithm and label each data point with the k -means cluster ID and core/ ϵ -core/noncore;
- 2: Determine ϵ and N_{min} for DBSCAN;
- 3: Run DBSCAN for core and ϵ -core points of each k -means cluster;
- 4: Run DBSCAN for all core and noncore points;
- 5: Resolve multiple cluster IDs;
- 6: Run the k -means algorithm without the noise found in DBSCAN, taking earlier centers as initial points.

The parameter ϵ is dependent on the specific data and is set by experiments. The parameter N_{min} is set according to the formula

$$N_{min} = \min_{1 \leq l \leq k} N_l,$$

where N_l is defined as

$$N_l = \frac{V_\epsilon(l)}{V_\tau(l)} \cdot |C_l|$$

with $V_\epsilon(l)$ for cluster C_l calculated as 2ϵ for one dimension, $\pi\epsilon^2$ for two dimensions, and $\frac{4}{3}\pi\epsilon^3$ for three dimensions and $V_\tau(l)$ the volume of the smallest hyperrectangle that circumscribes all points inside the cluster.

The integration of the k -means algorithm and DBSCAN overcomes some disadvantages of the two algorithms. In addition, Dash et al. (2001) also exploited the integration of BIRCH and DBSCAN.

13.3 DBCLASD

Xu et al. (1998) proposed an incremental clustering algorithm called DBCLASD (Distribution-Based Clustering of LArge Spatial Databases) based on the assumption that points inside a cluster are uniformly distributed. Like DBSCAN, DBCLASD is capable of finding arbitrarily shaped clusters, but DBCLASD does not require input parameters. DBCLASD dynamically determines the appropriate number and shape of clusters for a database, and the algorithm is efficient for larger databases.

The cluster produced by DBCLASD is defined in terms of the distribution of the nearest neighbor distances. Let S be a data set. Then the nearest neighbor of $\mathbf{x} \in S$, denoted by $NN_S(\mathbf{x})$, is the point in $S - \{\mathbf{x}\}$ that has the minimum distance to \mathbf{x} . This distance is called the nearest neighbor distance of \mathbf{x} , denoted by $NNdist_S(\mathbf{x})$. The multiset of all values of $NNdist_S(\mathbf{x})$ for $\mathbf{x} \in S$ is called the nearest neighbor distance set of S or distance set of S and is denoted by $NNdistSet(S)$. Based on the notion of nearest neighbor distances, a cluster can be defined as follows.

Definition 13.6 (Cluster). *Let D be a data set. Then a cluster C is a nonempty subset of D that satisfies the following conditions:*

1. *$NNdistSet(C)$ has the expected distribution with a required confidence level.*
2. *Each extension of C by neighboring points will fail the first condition (maximality).*
3. *Each pair of points in the cluster is connected by a path of occupied grid cells (connectivity).*

The probability distribution of the nearest neighbor distances is determined based on the assumption that the points inside a cluster are uniformly distributed. Let R be a data space with volume $Vol(R)$. Then the probability that the nearest neighbor distance X_d from any point \mathbf{y} to its nearest neighbor in R is greater than $x > 0$ is

$$P(X_d > x) = \left(1 - \frac{Vol(SP(\mathbf{y}, x))}{Vol(R)}\right)^N,$$

where N is the number of points inside R and $SP(\mathbf{y}, x)$ is a hypersphere around \mathbf{y} with radius x . Hence, in the two-dimensional case, the distribution function is given by

$$F(x) = 1 - P(X_d > x) = \left(1 - \frac{\pi x^2}{Vol(R)}\right)^N.$$

The parameter N can be determined straightforwardly, while $Vol(R)$ is approximated by the volume of the grid cells occupied by the points in R .

DBCLASD starts with an initial cluster and then adds neighboring points to the cluster incrementally as long as the nearest neighbor distance set of the resulting cluster still fits the expected distance distribution. For each new member \mathbf{x} of the current cluster C , new candidates are retrieved using a circle query with a suitable radius m , where m satisfies the condition

$$P(N N dist_C(\mathbf{x}) > m) < \frac{1}{N},$$

or

$$m > \left(\frac{A}{\pi \cdot (1 - N^{-\frac{1}{N}})}\right)^{\frac{1}{2}},$$

where N is the number of elements in C and A is the area of C .

A χ^2 -test is used to derive that the observed distance distribution fits the expected distance distribution. The new candidates are tested using the χ^2 -test. Since the order of testing is crucial, unsuccessful candidates are not discarded but tried again later and points may switch from one cluster to another.

13.4 DENCLUE

DENCLUE (DENsity based CLUstEring) (Hinneburg and Keim, 1998) is a density-based algorithm for clustering large multimedia data. It can find arbitrarily shaped clusters and deal with data sets with large amounts of noise. Before describing this algorithm, we introduce some concepts.

The density function of a data set D is defined as

$$f_B^D(\mathbf{x}) = \sum_{\mathbf{y} \in D} f_B(\mathbf{x}, \mathbf{y}),$$

where $f_B : F^d \times F^d \rightarrow R_0^+$ is a basic influence function. The square wave influence function (13.1a) and the Gaussian influence function (13.1b) are examples of basic influence functions:

$$f_{Square}(\mathbf{x}, \mathbf{y}) = \begin{cases} 0 & \text{if } d(\mathbf{x}, \mathbf{y}) > \sigma, \\ 1 & \text{otherwise,} \end{cases} \quad (13.1a)$$

$$f_{Gauss}(\mathbf{x}, \mathbf{y}) = e^{-\frac{d(\mathbf{x}, \mathbf{y})^2}{2\sigma^2}}. \quad (13.1b)$$

A density-attractor for a given influence function is a data point $\mathbf{x}^* \in F^d$ that is a local maximum of the density function f_B^D . A data point $\mathbf{x} \in F^d$ is density-attracted to a

density-attractor \mathbf{x}^* if and only if there exists $t \in \mathbb{N}$ such that $d(\mathbf{x}^t, \mathbf{x}^*) \leq \epsilon$ with $\mathbf{x}^0 = \mathbf{x}$ and

$$\mathbf{x}^i = \mathbf{x}^{i-1} + \delta \frac{\nabla f_B^D(\mathbf{x}^{i-1})}{\|\nabla f_B^D(\mathbf{x}^{i-1})\|}, \quad i = 1, 2, \dots, t, \quad (13.2)$$

where $\nabla f_B^D(\cdot)$ is the gradient of a function $f_B^D(\cdot)$, which is given by

$$\nabla f_B^D(\mathbf{x}) = \sum_{\mathbf{y} \in D} (\mathbf{y} - \mathbf{x}) f_B(\mathbf{x}, \mathbf{y}).$$

The local density function $\hat{f}^D(\mathbf{x})$, used to approximate the overall density function, is defined as

$$\hat{f}^D(\mathbf{x}) = \sum_{\mathbf{y} \in \text{near}(\mathbf{x})} f_B(\mathbf{x}, \mathbf{y}),$$

where $\text{near}(\mathbf{x}) = \{\mathbf{y} : d(\mathbf{y}, \mathbf{x}) \leq \sigma_{\text{near}}\}$.

Suppose $\sigma_{\text{near}} = s\sigma$. Then the upper bound of the error by using the local density function instead of the overall density function is

$$\sum_{\mathbf{y} \in D, d(\mathbf{y}, \mathbf{x}) > s\sigma} e^{-\frac{d(\mathbf{y}, \mathbf{x})^2}{2\sigma^2}} \leq |\{\mathbf{y} \in D : d(\mathbf{y}, \mathbf{x}) > s\sigma\}| \cdot e^{-\frac{s^2}{2}}.$$

Given two parameters σ and ξ , a subset $C \subseteq D$ is called a center-defined cluster for a density-attractor \mathbf{x}^* if \mathbf{x} is density-attracted by \mathbf{x}^* with $f_B^D(\mathbf{x}^*) \geq \xi$ for all $\mathbf{x} \in C$. Points $\mathbf{x} \in D$ are called outliers if they are density-attracted by a density-attractor \mathbf{x}^* with $f_B^D(\mathbf{x}^*) < \xi$. A arbitrarily shaped cluster for a set of density-attractors X is a subset $C \subseteq D$ satisfying (a) $\forall \mathbf{x} \in C, \exists \mathbf{x}^* \in X$ such that \mathbf{x} is density-attracted to \mathbf{x}^* and $f_B^D(\mathbf{x}^*) \geq \xi$, and (b) $\forall \mathbf{x}_1^*, \mathbf{x}_2^* \in X$, there exists a path $P \in F^d$ from \mathbf{x}_1^* to \mathbf{x}_2^* with $f_B^D(\mathbf{y}) \geq \xi$ for all $\mathbf{y} \in P$.

Now we can describe the DENCLUE algorithm, which consists of two main steps. The first step is the preclustering step, in which a map of the relevant portion of the data space is constructed. In this step, the data space is divided into hypercubes with an edge length of 2σ , and only populated hypercubes are determined and labeled. Also, the linear sum $\sum_{\mathbf{x} \in H} \mathbf{x}$ is stored for each populated hypercube H . A hypercube H is said to be a highly populated hypercube if $|H| \geq \xi_c$, where ξ_c is a outlier bound. Two hypercubes H_1 and H_2 are said to be connected if $d(\text{mean}(H_1), \text{mean}(H_2)) \leq 4\sigma$.

The second step is the actual clustering step, in which the density-attractors and the corresponding density-attracted points are identified by considering only the hypercubes in \mathcal{C}_r , which is the set of the highly populated hypercubes and hypercubes that are connected to a highly populated hypercube. Let $H \in \mathcal{C}_r$ and $\mathbf{x} \in H$. Then the resulting local density function is

$$\hat{f}_{\text{Gauss}}^D(\mathbf{x}) = \sum_{\mathbf{y} \in \text{near}(\mathbf{x})} e^{-\frac{d(\mathbf{x}, \mathbf{y})^2}{2\sigma^2}},$$

where $\text{near}(\mathbf{x}) = \{\mathbf{y} \in H : d(\text{mean}(H), \mathbf{x}) \leq s\sigma, H \in \mathcal{C}_r, \text{ and } H \text{ is connected to } H\}$.

Then the density-attractor for a data point \mathbf{x} is computed as in (13.2) by using $\hat{f}_{\text{Gauss}}^D(\cdot)$ instead of $f_B^D(\cdot)$. The calculation stops at $t \in \mathbb{N}$ if $\hat{f}^D(\mathbf{x}^{t+1}) < \hat{f}^D(\mathbf{x}^t)$ and takes $\mathbf{x}^* = \mathbf{x}^t$ as a new density-attractor. If $\hat{f}_{\text{Gauss}}^D(\mathbf{x}^*) \geq \xi$, then \mathbf{x} is assigned to the cluster belonging to \mathbf{x}^* .

In this algorithm, there are two important parameters: σ and ξ . Hinneburg and Keim (1998) suggest a way to choose good σ and ξ .

13.5 CUBN

Wang and Wang (2003) proposed the clustering algorithm CUBN, which integrates density-based and distance-based clustering. CUBN first finds border points using the erosion operation and then clusters border points and inner points according to the nearest distance. The algorithm is capable of finding nonspherical shapes and wide variances in size, and its complexity is $O(n)$ with n being the size of the data set.

To describe this algorithm, we need to introduce the erosion operation—one of the seven basic operations in mathematical morphology. The erosion operation has visual meaning for geometry. In the two-dimensional case, for example, applying the erosion operation to an area will eliminate the roughness of the border and retain the basic shape of the area.

Mathematically, let A be a set of objects and B be a single vector. Then the erosion operation is defined as

$$A \ominus B = \{a : a \in A, |\{w : w \in O(a + B, r) \cap A\}| > t\},$$

where t is a threshold and $O(a + B, r)$ is a supersphere with center $a + B$ and radius r . When B is a set of vectors with elements B_1, B_2, \dots, B_g , the erosion operation is defined as

$$A \ominus B = \bigcap_{i=1}^g (A \ominus B_i).$$

The border points of a cluster A are a set S defined as

$$S = A - \overline{A \ominus B} = A \ominus B = \bigcup_{i=1}^g \overline{A \ominus B_i},$$

where $\overline{A \ominus B_i} = \{a : a \in A, |\{w : w \in O(a + B_i, r) \cap A\}| \leq t\}$.

In CUBN, the set of vectors B is the set of row vectors of the matrix

$$\begin{pmatrix} p & 0 & \cdots & 0 \\ -p & 0 & \cdots & 0 \\ 0 & p & \cdots & 0 \\ 0 & -p & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p \\ 0 & 0 & \cdots & -p \end{pmatrix}_{2d \times d},$$

where d is the dimension of the data set and p satisfies the condition

$$\max d(\mathbf{x}, \mathbf{y}) < p + r < \min d(C_i, C_j), \quad (13.3)$$

where $d(\mathbf{x}, \mathbf{y})$ is the distance between two points in the same cluster and $d(C_i, C_j)$ is the distance between two clusters, e.g., the nearest neighbor distance.

To detect different cluster borders, the radius r of the supersphere is set to be $\frac{p}{\sqrt{2}}$ in CUBN. p is an input parameter specified by users and it must satisfy the condition given in

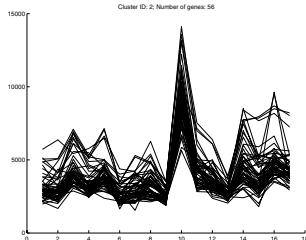
inequality (13.3). If $p + r > \min d(C_i, C_j)$, then CUBN cannot detect all border points; if $p + r < \max d(\mathbf{x}, \mathbf{y})$, then CUBN may regard an inner point as a border point.

The CUBN algorithm consists of three phases. At the first phase, the erosion operation is used to find border points. Then the nearest neighbor method is used to cluster the border points. Finally, the nearest neighbor method is employed to cluster the inner points.

13.6 Summary

In density-based clustering algorithms, clusters are regarded as regions in the data space where the data points are dense that are separated by regions of low density. Therefore, a cluster may have an arbitrary shape and the points inside a cluster may be arbitrarily distributed. An attractive feature of density-based approaches is that they can identify clusters of arbitrary shapes. In addition, density-based approaches can handle noise (or outliers) very efficiently.

One difficulty of most density-based approaches is that it is hard to choose the parameter values, such as the density threshold. Also, most density-based approaches were developed for clustering spatial databases.



Chapter 14

Model-based Clustering Algorithms

Model-based clustering is a major approach to clustering analysis. This chapter introduces model-based clustering algorithms. First, we present an overview of model-based clustering. Then, we introduce Gaussian mixture models, model-based agglomerative hierarchical clustering, and the expectation-maximization (EM) algorithm. Finally, we introduce model-based clustering and two model-based clustering algorithms.

14.1 Introduction

Clustering algorithms can also be developed based on probability models, such as the finite mixture model for probability densities. The word *model* is usually used to represent the type of constraints and geometric properties of the covariance matrices (Martinez and Martinez, 2005). In the family of model-based clustering algorithms, one uses certain models for clusters and tries to optimize the fit between the data and the models. In the model-based clustering approach, the data are viewed as coming from a mixture of probability distributions, each of which represents a different cluster. In other words, in model-based clustering, it is assumed that the data are generated by a mixture of probability distributions in which each component represents a different cluster. Thus a particular clustering method can be expected to work well when the data conform to the model.

Model-based clustering has a long history. A survey of cluster analysis in a probabilistic and inferential framework was presented by Bock (1996). Early work on model-based clustering can be found in (Edwards and Cavalli-Sforza, 1965), (Day, 1969), (Wolfe, 1970), (Scott and Symons, 1971b), and (Binder, 1978). Some issues in cluster analysis, such as the number of clusters, are discussed in (McLachlan and Basford, 1988), (Banfield and Raftery, 1993), (McLachlan and Peel, 2000), (Everitt et al., 2001), and (Fraley and Raftery, 2002).

Usually, there are two approaches to formulating the model for the composite of the clusters, the classification likelihood approach and the mixture likelihood approach (Fraley and Raftery, 1998). Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a set of observations; let $f_j(\mathbf{x}_i | \Theta_j)$ be the density of an observation \mathbf{x}_i from the j th component, where Θ_j are the corresponding parameters; and let k be the number of components in the mixture. For example, if we assume that the data come from a mixture of Gaussian distributions, then the parameters

Θ_j consist of a mean vector μ_j and a covariance matrix Σ_j , and the density has the form

$$f_j(\mathbf{x}_i | \mu_j, \Sigma_j) = \frac{\exp\left[-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j)\right]}{(2\pi)^{\frac{d}{2}} |\Sigma_j|^{\frac{1}{2}}},$$

where d is the dimension of the data.

More precisely, the classification likelihood approach and the mixture likelihood approach can be described as follows.

Classification likelihood approach. This approach maximizes

$$\mathcal{L}_C(\Theta_1, \Theta_2, \dots, \Theta_k; \gamma_1, \gamma_2, \dots, \gamma_n | D) = \prod_{i=1}^n f_{\gamma_i}(\mathbf{x}_i | \Theta_{\gamma_i}), \quad (14.1)$$

where γ_i are integers labeling the classification, i.e., $\gamma_i = j$ if \mathbf{x}_i belongs to the j th component.

Mixture likelihood approach. This approach assumes that the probability function can be the sum of weighted component densities. If we use mixture likelihood for clustering, the clustering problem becomes the estimation of the parameters of the assumed mixture model. Mathematically, this approach maximizes

$$\mathcal{L}_M(\Theta_1, \Theta_2, \dots, \Theta_k; \tau_1, \tau_2, \dots, \tau_k | D) = \prod_{i=1}^n \sum_{j=1}^k \tau_j f_j(\mathbf{x}_i | \Theta_j), \quad (14.2)$$

where $\tau_j \geq 0$ is the probability that an observation belongs to the j th component and

$$\sum_{j=1}^k \tau_j = 1.$$

In the mixture likelihood approach, the EM algorithm is the most widely used method for estimating the parameters of a finite mixture probability density. The model-based clustering framework provides a principal way to deal with several problems in this approach, such as the number of component densities (or clusters), initial values of the parameters (the EM algorithm needs initial parameter values to get started), and distributions of the component densities (e.g., Gaussian) (Martinez and Martinez, 2005).

The number of clusters and the distribution of the component densities can be considered as producing different statistical models for the data. The final model can be determined by the Bayesian information criterion (BIC) (Schwarz, 1978; Kass and Raftery, 1995). The model with the highest BIC value is chosen as the best model.

The procedure of a model-based clustering algorithm is illustrated in Figure 14.1. In model-based clustering, model-based agglomerative clustering (Murtagh and Raftery, 1984; Banfield and Raftery, 1993) is first used to initialize the EM algorithm (Dasgupta and Raftery, 1998). Model-based agglomerative clustering, which uses the same general ideas as agglomerative hierarchical clustering (see Section 7.2), merges two clusters when

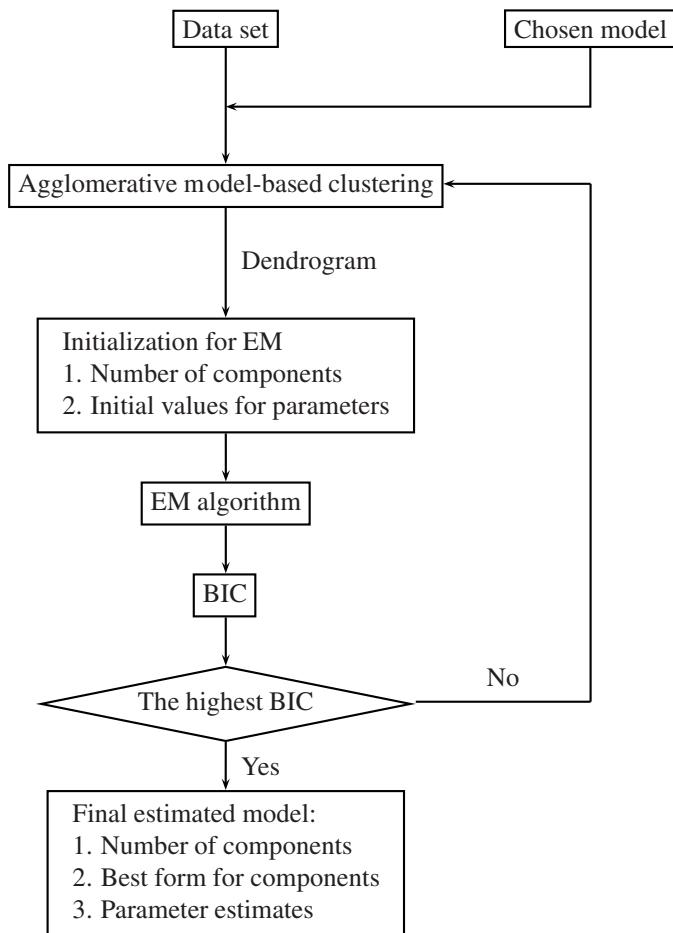


Figure 14.1. The flowchart of the model-based clustering procedure.

the classification likelihood (see Section 14.3) is maximized. Hierarchical clustering provides a complete nested partition from which initial estimates of component parameters can be obtained. Then a probability distribution is chosen for each component. A general framework for multivariate normal mixtures is proposed by Banfield and Raftery (1993). In this general framework, different constraints on the component covariance matrices yield different models.

In particular, some of the well-known clustering algorithms are approximate estimation methods for certain probability models (Fraley and Raftery, 2002). The conventional k -means algorithm (Macqueen, 1967) and Ward's method (Ward Jr., 1963), for example, are equivalent to the known procedures for approximately maximizing the multivariate normal classification likelihood under certain conditions (the covariance matrix is the same for each component and proportional to the identity matrix) (Fraley and Raftery, 2002).

Many clustering algorithms can be categorized into the family of model-based clustering methods, such as neural network approaches like self-organizing feature map (SOM), probability density-based approaches, and Gaussian mixture model, approaches based on the Bayesian clustering procedure. Some model-based clustering algorithms will be presented and discussed in later chapters.

14.2 Gaussian Clustering Models

Gaussian mixture models provide a classical and powerful approach to clustering analysis (Banfield and Raftery, 1993), and they are also useful for understanding and suggesting powerful clustering criteria. In Gaussian mixture models, the data $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ in a d -dimensional space are assumed to arise from a random vector with density (Celeux and Govaert, 1995)

$$f(\mathbf{x}) = \sum_{j=1}^k p_j \Phi(\mathbf{x}|\mu_j, \Sigma_j),$$

where p_j are the mixing proportions, i.e., $0 < p_j < 1$ and $\sum_{j=1}^k p_j = 1$, and $\Phi(\mathbf{x}|\mu, \Sigma)$ denotes the density of Gaussian distribution with mean vector μ and covariance matrix Σ , i.e.,

$$\Phi(\mathbf{x}|\mu, \Sigma) = \frac{\exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right]}{\sqrt{(2\pi)^d |\Sigma|}}. \quad (14.3)$$

Celeux and Govaert (1995) present four commonly used assumptions on the component variance matrices:

1. $\Sigma_1 = \Sigma_2 = \dots = \Sigma_k = \sigma^2 I$, where σ^2 is unknown.
2. $\Sigma_1 = \Sigma_2 = \dots = \Sigma_k = \text{Diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_d^2)$, where $(\sigma_1^2, \sigma_2^2, \dots, \sigma_d^2)$ is unknown and $\text{Diag}(a_1, a_2, \dots, a_d)$ denotes a diagonal matrix with diagonal vector a_1, a_2, \dots, a_d .
3. $\Sigma_1 = \Sigma_2 = \dots = \Sigma_k = \Sigma$, where Σ is an unknown symmetric matrix.
4. No restriction is imposed on the variance matrices $\Sigma_1, \Sigma_2, \dots, \Sigma_k$.

In order to embed the above-mentioned four assumptions in a framework that can lead to some clustering criteria, the variance matrix Σ_k is decomposed as

$$\Sigma_k = \lambda_k D_k A_k D_k^T,$$

where $\lambda_k = |\Sigma_k|^{\frac{1}{d}}$, D_k is the matrix of eigenvectors of Σ_k , and A_k is a diagonal matrix such that $|A_k| = 1$, with the normalized eigenvalues of Σ_k on the diagonal in decreasing order. λ_k , D_k , and A_k determine the volume, the orientation, and the shape of the k th cluster, respectively. Note that the volume of a cluster is different from the size of the cluster.

Two maximum likelihood approaches, the mixture approach and the classification approach, are proposed to estimate the parameters in a mixture. The mixture approach is aimed at maximizing the likelihood over the mixture parameters. In this approach, the

Table 14.1. Description of Gaussian mixture models in the general family.

Model	Distribution
$\Sigma_j = \lambda DAD^T$	Volume: Fixed Shape: Fixed Orientation: Fixed
$\Sigma_j = \lambda DAD^T$	Volume: Variable Shape: Fixed Orientation: Fixed
$\Sigma_j = \lambda DA_j D^T$	Volume: Fixed Shape: Variable Orientation: Fixed
$\Sigma_j = \lambda_j DA_j D^T$	Volume: Variable Shape: Variable Orientation: Variable
$\Sigma_j = \lambda_j D_j A D_j^T$	Volume: Fixed Shape: Fixed Orientation: Variable
$\Sigma_j = \lambda_j D_j A D_j^T$	Volume: Variable Shape: Fixed Orientation: Variable
$\Sigma_j = \lambda_j D_j A_j D_j^T$	Volume: Fixed Shape: Variable Orientation: Variable
$\Sigma_j = \lambda_j D_j A_j D_j^T$	Volume: Variable Shape: Variable Orientation: Variable

parameters $\theta = p_1, p_2, \dots, p_{k-1}, \mu_1, \mu_2, \dots, \mu_k, \Sigma_1, \Sigma_2, \dots, \Sigma_k$ are chosen to maximize the loglikelihood

$$L(\theta | \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) = \sum_{i=1}^n \ln \left[\sum_{j=1}^k p_j \Phi(\mathbf{x}_i | \mu_j, \Sigma_j) \right]. \quad (14.4)$$

The EM algorithm is used to find the parameters θ that maximize equation (14.4).

The classification approach is aimed at maximizing the likelihood over the mixture parameters and over the identifying labels of the mixture component origin for each point. In this approach, the indicator vectors $\mathbf{z}_i = (z_{i1}, z_{i2}, \dots, z_{ik})$, with $z_{ik} = 1$ or 0 according to whether \mathbf{x}_i has been drawn from the k th component or from another one, are treated as unknown parameters. Two different types of classification maximum likelihood (CML) criteria have been proposed according to the sampling scheme.

Table 14.2. Description of Gaussian mixture models in the diagonal family. \mathbf{B} is a diagonal matrix.

Model	Distribution
$\Sigma_j = \lambda B$	Volume: Fixed Shape: Fixed Orientation: Axes
$\Sigma_j = \lambda_j B$	Volume: Variable Shape: Fixed Orientation: Axes
$\Sigma_j = \lambda B_j$	Volume: Fixed Shape: Variable Orientation: Axes
$\Sigma_j = \lambda_j B_j$	Volume: Variable Shape: Variable Orientation: Axes

Table 14.3. Description of Gaussian mixture models in the diagonal family. \mathbf{I} is an identity matrix.

Model	Distribution
$\Sigma_j = \lambda I$	Volume: Fixed Shape: Fixed Orientation: NA
$\Sigma_j = \lambda_j I$	Volume: Variable Shape: Fixed Orientation: NA

Variations on assumptions on the parameters λ_k , D_k , and A_k lead to eight general clustering models. Fourteen models are discussed in Celeux and Govaert (1995). These models are classified into three categories: the general family (see Table 14.1), the diagonal family (see Table 14.2), and the spherical family (see Table 14.3). Some numerical experiments are presented in Celeux and Govaert (1995).

14.3 Model-based Agglomerative Hierarchical Clustering

Model-based agglomerative hierarchical clustering is used to find initial values of parameters for any given number of clusters in model-based clustering. As a major component of the model-based clustering process, model-based agglomerative hierarchical clustering works in a similar manner to the agglomerative hierarchical clustering described in Section 7.2. However, no distances are defined in model-based agglomerative hierarchical clustering.

Instead, the classification likelihood is defined as the objective function in model-based agglomerative hierarchical clustering.

The classification likelihood is defined as (Martinez and Martinez, 2005; Fraley, 1998)

$$\mathcal{L}_{CL}(\Theta_j, \gamma_i | \mathbf{x}_i) = \prod_{i=1}^{n_{\gamma_i}} f_{\gamma_i}(\mathbf{x}_i | \Theta_{\gamma_i}), \quad (14.5)$$

where γ_i is the index of the cluster to which \mathbf{x}_i belongs, and n_{γ_i} denotes the number of objects in the γ_i th component, i.e., the component to which \mathbf{x}_i belongs.

The goal of a model-based agglomerative hierarchical clustering algorithm is to maximize the classification likelihood. Such an algorithm starts with singleton clusters consisting of only one point, and then merges at each step two clusters producing the largest increase in the classification likelihood, i.e., a maximum likelihood pair of clusters. This process continues until all objects are in one cluster.

Fraley (1998) changed the notion of singleton clusters such that a model-based agglomerative hierarchical clustering algorithm can start with a given partition and proceed to form larger clusters. Fraley (1998) also proposed four methods for agglomerative hierarchical clustering based on Gaussian models (see Table 14.4). In these methods, the form of the objective function is adjusted as follows. When $f_j(\mathbf{x}_i | \Theta_j)$ is multivariate normal, the likelihood in equation (14.5) becomes

$$\begin{aligned} & \mathcal{L}_{CL}(\mu_1, \mu_2, \dots, \mu_k; \Sigma_1, \Sigma_2, \dots, \Sigma_k, \gamma | \mathbf{x}) \\ &= \prod_{j=1}^k \prod_{i \in \mathcal{I}_j} (2\pi)^{-\frac{d}{2}} |\Sigma_j|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j) \right\}, \end{aligned} \quad (14.6)$$

where $\mathcal{I}_j = \{i : \gamma_i = j\}$ is the set of indices corresponding to objects in the j th cluster. The maximum likelihood estimator of μ_j in equation (14.6) is the group average $\bar{\mathbf{x}}_j = \frac{\mathbf{s}_j}{n_j}$, where \mathbf{s}_j is the sum of objects in the j th group and n_j is the number of objects in the j th group. The concentrated loglikelihood is obtained by replacing μ_j by $\hat{\mu}_j = \bar{\mathbf{x}}_j$ as follows (Fraley, 1998):

$$\begin{aligned} & \mathcal{L}_{CL}(\Sigma_1, \Sigma_2, \dots, \Sigma_k, \gamma | \mathbf{x}; \hat{\mu}_1, \hat{\mu}_2, \dots, \hat{\mu}_k) \\ &= -\frac{pn \log(2\pi)}{2} - \frac{1}{2} \sum_{j=1}^k \left\{ \text{Tr}(W_j \Sigma_j^{-1}) + n_j \log |\Sigma_j| \right\}, \end{aligned} \quad (14.7)$$

where the covariance matrices Σ_j and γ remain to be determined, and the matrix W_j is defined as

$$W_j = \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \bar{\mathbf{x}}_j)(\mathbf{x} - \bar{\mathbf{x}}_j)^T,$$

with C_j being the j th cluster. To maximize the likelihood, one can minimize the corresponding criteria shown in Table 14.4.

Instead of maximizing the classification likelihood in equation (14.5), Fraley (1998) proposed the following objective functions to merge two clusters.

Table 14.4. Four parameterizations of the covariance matrix in the Gaussian model and their corresponding criteria to be minimized.

Σ_j	Criterion
$\sigma^2 I$	$\text{Tr}(\sum_{j=1}^k W_j)$
$\sigma_j^2 I$	$\sum_{j=1}^k n_j \log \left[\text{Tr}\left(\frac{W_j}{n_j}\right) \right]$
Σ	$\left \sum_{j=1}^k W_j \right $
Σ_j	$\sum_{j=1}^k n_j \log \left \frac{W_j}{n_j} \right $

Model $\Sigma_j = \sigma^2 I$. In this model, the covariance matrix is constrained to be diagonal and uniform for all clusters. The criterion to be minimized at each stage is

$$\text{Tr} \left(\sum_{j=1}^k W_j \right) = \sum_{j=1}^k \text{Tr}(W_j),$$

and the cost $\Delta(i, j)$ to merge the clusters i and j is

$$\begin{aligned} \Delta(i, j) &= \text{Tr}(W_{\langle i, j \rangle}) - \text{Tr}(W_i) - \text{Tr}(W_j) \\ &= \text{Tr}(W_{\langle i, j \rangle}) - W_i - W_j \\ &= \mathbf{w}_{ij}^T \mathbf{w}_{ij}, \end{aligned}$$

where

$$\mathbf{w}_{ij} = \eta_{ji} \mathbf{s}_i - \eta_{ij} \mathbf{s}_j, \quad \eta_{ij} = \sqrt{\frac{n_i}{n_j(n_i + n_j)}}.$$

Two clusters producing the minimum cost will be merged.

Model $\Sigma_j = \sigma_j^2 I$. In this model, the covariance matrix of each cluster is constrained to be diagonal but is allowed to vary between clusters. The criterion to be minimized at each stage is

$$\sum_{j=1}^k n_j \log \left[\text{Tr} \left(\frac{W_j}{n_j} \right) \right],$$

and the cost $\Delta(i, j)$ is defined as

$$\begin{aligned} \Delta(i, j) &= (n_i + n_j) \log \left(\frac{\text{Tr}(W_i) + \text{Tr}(W_j) + \mathbf{w}_{ij}^T \mathbf{w}_{ij}}{n_i + n_j} \right) \\ &\quad - n_i \log \left(\frac{\text{Tr}(W_i)}{n_i} \right) - n_j \log \left(\frac{\text{Tr}(W_j)}{n_j} \right), \end{aligned}$$

which is complicated. To avoid $\text{Tr}(W_j) = 0$ arising from singleton clusters, Fraley (1998) introduced a modified criterion defined as

$$\sum_{j=1}^k n_j \log \left(\frac{\text{Tr}(W_j) + \alpha \cdot \frac{\text{Tr}(\mathcal{W})}{nd}}{n_j} \right),$$

where \mathcal{W} is the sample cross-product matrix for the data set.

Model $\sum_j = \sum$. In this model, the covariance matrix is uniform across all clusters but has no structural constraints. The criterion to be minimized at each stage is

$$\left| \sum_{j=1}^k W_j \right|.$$

The cost $\Delta(i, j)$ is not needed.

Unconstrained model. In this model, the covariance matrix is allowed to vary arbitrarily across clusters. The criterion to be minimized at each stage is

$$\sum_{j=1}^k n_j \log \left| \frac{W_j}{n_j} \right|.$$

In order to handle singleton clusters, the criterion to be minimized is adjusted to

$$\sum_{j=1}^k n_j \log \left(\left| \frac{W_j}{n_j} \right| + \beta \cdot \frac{\text{Tr}(W_j) + \alpha \cdot \frac{\mathcal{W}}{nd}}{n_j} \right).$$

14.4 The EM Algorithm

The EM algorithm is a general statistical method of maximum likelihood estimation in the presence of incomplete data that can be used for the purpose of clustering. According to (Meng and van Dyk, 1997), more than 400 articles were devoted to the development and improvement of the EM algorithm within the 20 years since EM was first formulated by Dempster et al. (1977). The EM algorithm has many good features: simplicity, stability, and robustness to noise. In this section, we will introduce the basic EM algorithm.

Let \mathcal{X} and \mathcal{Y} be two sample spaces and assume that there is a many-one mapping from \mathcal{X} to \mathcal{Y} . The observed data \mathbf{y} are a realization from \mathcal{Y} and the corresponding \mathbf{x} in \mathcal{X} is not observed directly, but only indirectly through \mathbf{y} . \mathbf{x} is referred to as the complete data and \mathbf{y} is referred to as the incomplete data. Let $f(\mathbf{x}|\Phi)$ be a family of sampling densities for complete data, depending on parameters Φ , and $g(\mathbf{y}|\Phi)$ be its corresponding derived family of sampling densities. Then $f(\cdots | \cdots)$ is related to $g(\cdots | \cdots)$ by

$$g(\mathbf{y}|\Phi) = \int_{\mathcal{X}(\mathbf{y})} f(\mathbf{x}|\Phi) d\mathbf{x}, \quad (14.8)$$

where $\mathcal{X}(\mathbf{y})$ is the subset of \mathcal{X} determined by the equation $\mathbf{y} = \mathbf{y}(\mathbf{x})$, $\mathbf{x} \rightarrow \mathbf{y}(\mathbf{x})$, which is a many-one mapping from \mathcal{X} to \mathcal{Y} .

The goal of the EM algorithm is to find the value of Φ that maximizes $g(\mathbf{y}|\Phi)$ given an observed \mathbf{y} . The EM algorithm proceeds iteratively. Each iteration involves two steps: the expectation step (E-step) and the maximization step (M-step). Dempster et al. (1977) first defined the EM algorithm in special cases and then extended it gradually to more general cases.

Let Q be a function defined as

$$Q(\Phi'|\Phi) = E(\log f(\mathbf{x}|\Phi')|\mathbf{y}, \Phi), \quad (14.9)$$

which is assumed to exist for all pairs (Φ', Φ) , and $f(\mathbf{x}, \Phi)$ is assumed to be positive almost everywhere in \mathcal{X} for all $\Phi \in \Omega$, where Ω is an r -dimensional convex region. Then the EM iteration $\Phi^{(p)}$ to $\Phi^{(p+1)}$ is defined as follows:

- E-step: Compute $Q(\Phi|\Phi^{(p)})$.
- M-step: Choose $\Phi^{(p+1)}$ to be a value of $\Phi \in \Omega$ that maximizes $Q(\Phi|\Phi^{(p)})$.

In particular, if $f(\mathbf{x}|\Phi)$ is of the regular exponential family form, i.e.,

$$f(\mathbf{x}|\Phi) = b(\mathbf{x}) \exp(\Phi \mathbf{t}(\mathbf{x})^T) / a(\Phi), \quad (14.10)$$

where Φ is an r -dimensional parameter, $\mathbf{t}(\mathbf{x})$ is an r -dimensional vector of complete data sufficient statistics, and $\mathbf{t}(\mathbf{x})^T$ denotes the transpose of $\mathbf{t}(\mathbf{x})$, then the corresponding $Q(\Phi|\Phi^{(p)})$ is given by

$$Q(\Phi|\Phi^{(p)}) = -\log a(\Phi) + E(\log b(\mathbf{x})|\mathbf{y}, \Phi^{(p)}) + \Phi \mathbf{t}^{(p)T},$$

which implies that maximizing $Q(\Phi|\Phi^{(p)})$ in the M-step is equivalent to maximizing $-\log a(\Phi) + \Phi \mathbf{t}^{(p)T}$.

The algorithm defined above is a generalized EM algorithm (a GEM algorithm). The EM algorithm is a special case of GEM. The convergence of the GEM algorithm is theoretically guaranteed. Proofs can be found in (Wu, 1983), (Boyles, 1983), and (Dempster et al., 1977). Several techniques have been developed to accelerate the EM algorithm, including conjugate gradient acceleration (Jamshidian and Jennrich, 1993), quasi-Newton methods (Jamshidian and Jennrich, 1997), and proximal point iterations (Chrétien and Hero III, 1998).

When the EM algorithm is used for the purpose of clustering, the “complete” data are considered to be (Fraley and Raftery, 1998)

$$\{(\mathbf{x}_i, \mathbf{z}_i), i = 1, 2, \dots, n\},$$

where $D = \{\mathbf{x}_i, i = 1, 2, \dots, n\}$ is the original data set, and $\mathbf{z}_i = (z_{i1}, z_{i2}, \dots, z_{ik})$ are defined as

$$z_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ belongs to the } j\text{th cluster,} \\ 0 & \text{otherwise.} \end{cases}$$

The \mathbf{z}_i constitute the “missing” data. The relevant assumptions are that each \mathbf{z}_i is independent and identically distributed according to a multinomial distribution of one draw on k categories with probabilities $\tau_1, \tau_2, \dots, \tau_k$ and that the density of \mathbf{x}_i given \mathbf{z}_i is given by

$$\prod_{j=1}^k f_j(\mathbf{x}_i | \Theta_j)^{z_{ij}}.$$

Then the complete-data loglikelihood has the form

$$l(\Theta_j, \tau_j, z_{ij} | D) = \sum_{i=1}^n \sum_{j=1}^k z_{ij} \log[\tau_j f_j(\mathbf{x}_i | \Theta_j)], \quad (14.11)$$

and the conditional expectation of z_{ij} given the observation \mathbf{x}_i and the parameter values is given by

$$\hat{z}_{ij} = E[z_{ij} | \mathbf{x}_i, \Theta_1, \Theta_2, \dots, \Theta_k].$$

The value z_{ij}^* of \hat{z}_{ij} at a maximum of (14.2) is the conditional probability that \mathbf{x}_i belongs to cluster j . In a hard partition, an observation \mathbf{x}_i is assigned to the cluster with the highest membership, i.e., the classification of \mathbf{x}_i is taken to be

$$\{j_0 : z_{ij_0}^* = \max_{1 \leq j \leq k} z_{ij}^*\}.$$

The EM algorithm iterates between an E-step and an M-step. In an E-step, the values of \hat{z}_{ij} are computed from the data with current parameter estimates, while in an M-step, the complete-data loglikelihood (14.11), with each z_{ij} replaced by its current conditional expectation \hat{z}_{ij} , is maximized with respect to the parameters. For a comprehensive discussion of the EM algorithm, readers are referred to (McLachlan and Krishnan, 1997). The EM algorithm for clustering is also presented in (McLachlan and Basford, 1988).

14.5 Model-based Clustering

We have introduced Gaussian mixture models, model-based agglomerative hierarchical clustering, and the EM algorithm in previous sections. This section proceeds to model-based clustering. The model-based clustering framework consists of three major steps (Martinez and Martinez, 2005):

- (a) Initialize the EM algorithm using the partitions from model-based agglomerative hierarchical clustering.
- (b) Estimate the parameters using the EM algorithm;
- (c) Choose the model and the number of clusters according to the BIC.

The Bayesian approach to model selection originated from the work by Jeffreys (1935, 1961) in which a framework for calculating the evidence in favor of a null hypothesis using a quantity called the Bayes factor was developed. Kass and Raftery (1995) presented the use of Bayes factors in the context of several scientific applications.

In the case of two models, the data D are assumed to have arisen according to either model M_1 or model M_2 . Given the prior probabilities $p(M_1)$ and $p(M_2)$, according to Bayes's theorem, the posterior probability of hypothesis M_g given data D is calculated as (Martinez and Martinez, 2005)

$$p(M_g|D) = \frac{p(M_g)p(D|M_g)}{p(M_1)p(D|M_1) + p(M_2)p(D|M_2)}, \quad g = 1, 2. \quad (14.12)$$

Since the two probabilities have the same denominator, the ratio of the two posterior probabilities is

$$\frac{p(M_1|D)}{p(M_2|D)} = \frac{p(D|M_1)}{p(D|M_2)} \cdot \frac{p(M_1)}{p(M_2)}. \quad (14.13)$$

The first factor in the above equation is the well-known Bayes factor

$$B_{12} = \frac{p(D|M_1)}{p(D|M_2)}.$$

If the model M_g contains unknown parameters, the posterior probability $p(D|M_g)$ can be obtained by

$$p(D|M_g) = \int p(X|\Theta_g, M_g)p(\Theta_g|M_g)d\Theta_g.$$

The resulting $p(D|M_g)$ is called the integrated likelihood of model M_g . The model that is most likely given the data is selected. If the prior probabilities $p(M_1)$ and $p(M_2)$ are equal, from equation (14.13) we have

$$\frac{p(M_1|D)}{p(M_2|D)} = \frac{p(D|M_1)}{p(D|M_2)}.$$

In this case, the most likely model is the model with the highest integrated likelihood. In the case of more than two models, a Bayesian solution is also possible (Dasgupta and Raftery, 1998; Fraley and Raftery, 1998, 2002).

Since the integrated likelihood depends on the prior probabilities, the BIC is used to approximate the logarithm of the integrated likelihood for models satisfying certain regularity conditions. That is,

$$p(D|M_g) = BIC_g = 2 \log p(D|\hat{\Theta}_g, M_g) - m_g \log(n), \quad (14.14)$$

where m_g is the number of independent parameters that must be estimated in the model. However, the number of clusters is not considered an independent parameter (Fraley and Raftery, 1998). Although the above approximation is not valid for the finite-mixture model, the use of the BIC produces reasonable results (Dasgupta and Raftery, 1998; Fraley and Raftery, 1998).

ALGORITHM 14.1. Model-based clustering procedure.

Require: D - the data set;

- 1: Apply the model-based agglomerative hierarchical clustering to D {the unconstrained model is used here};
- 2: **repeat**
- 3: Find a partition with c clusters using the results of the model-based agglomerative hierarchical clustering;

- 4: Choose a model M from Table 14.1, Table 14.2, and Table 14.3;
- 5: Use the partition found in the previous step to calculate the mixing coefficients, means, and covariances for each cluster {the covariances are constrained according to the model selected};
- 6: Initialize the parameters of the EM algorithm using the values calculated in the previous step and apply EM to obtain the final estimates;
- 7: Calculate the BIC value for this c and M according to equation (14.14);
- 8: **until** the highest BIC is obtained
- 9: Output the best configuration corresponding to the highest BIC.

The procedure described in Algorithm 14.1 illustrates how to use model-based clustering based on mixture models. Once the best model (the number of clusters and form of the covariance matrix) is obtained, the objects are grouped according to their posterior probability. More precisely, let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set and the parameters to be estimated be $\Theta = p_1, p_2, \dots, p_{c-1}, \mu_1, \mu_2, \dots, \mu_c, \Sigma_1, \Sigma_2, \dots, \Sigma_c$. Then the loglikelihood to be maximized is defined as

$$\mathcal{L}(\Theta|D) = \sum_{i=1}^n \ln \left[\sum_{j=1}^c p_j \phi(\mathbf{x}_i | \mu_j, \Sigma_j) \right],$$

where $\phi(\mathbf{x}|\mu, \Sigma)$ is defined in equation (14.3). The posterior probability that an object \mathbf{x}_i belongs to the j th cluster is given by

$$\hat{u}_{ji} = \frac{\hat{p}_j \phi(\mathbf{x}_i | \hat{\mu}_j, \hat{\Sigma}_j)}{\sum_{s=1}^c \hat{p}_s \phi(\mathbf{x}_i | \hat{\mu}_s, \hat{\Sigma}_s)}, \quad j = 1, 2, \dots, c, \quad i = 1, 2, \dots, n.$$

The posterior probabilities defined above are unknown. The EM algorithm is used to obtain these posterior probabilities. The clustering based on posterior probabilities is given by

$$C_j = \left\{ \mathbf{x}_j : \hat{u}_{ji} = \max_{1 \leq s \leq c} \hat{u}_{si} \right\}, \quad j = 1, 2, \dots, c.$$

For the unconstrained model $\Sigma_j = \lambda_j D_j A_j D_j^T$, one can use the following equations in the EM algorithm (Everitt and Hand, 1981; Martinez and Martinez, 2005):

$$\begin{aligned} \hat{p}_j &= \frac{1}{n} \sum_{i=1}^n \hat{u}_{ji}, \\ \hat{\mu}_j &= \frac{1}{n} \sum_{i=1}^n \frac{\hat{u}_{ji} \mathbf{x}_i}{\hat{p}_j}, \\ \hat{\Sigma}_j &= \frac{1}{n} \sum_{i=1}^n \frac{\hat{u}_{ji} (\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^T}{\hat{p}_j}. \end{aligned}$$

14.6 COOLCAT

An algorithm called COOLCAT (Barbará et al., 2002) is proposed to cluster categorical attributes using entropy. Given a data set D of N data points $\hat{p}_1, \hat{p}_2, \dots, \hat{p}_N$, where each point is a multidimensional vector of d categorical attributes, i.e., $\hat{p}_j = \{p_j^1, p_j^2, \dots, p_j^d\}$, the goal of this algorithm is to minimize the entropy of the whole arrangement.

For a given integer k , suppose we want to separate the data set into k clusters C_1, C_2, \dots, C_k . The expected entropy is given by

$$\bar{E}(\check{C}) = \sum_k \left(\frac{|P(C_k)|}{|D|} (E(P(C_k))) \right),$$

where $E(P(C_1)), \dots, E(P(C_k))$ represent the entropies of the clusters and $P(C_i)$ denotes the points assigned to cluster C_i , with the property that $P(C_i) \cap P(C_j) = \emptyset$ for all $i, j = 1, 2, \dots, k, i \neq j$.

Assume that the attributes of the record are independent. Then the joint probability of the combined attribute values becomes the product of the probability of each attribute, and hence the entropy can be computed as the sum of the entropies of the attributes:

$$\begin{aligned} E(\hat{x}) &= - \sum_{x_1 \in S(X_1)} \cdots \sum_{x_n \in S(X_n)} (p(x_1) \cdots p(x_n)) \log(p(x_1) \cdots p(x_n)) \\ &= E(X_1) + E(X_2) + \cdots + E(X_n). \end{aligned}$$

The COOLCAT clustering algorithm consists of two steps: the first step is initialization, which finds a suitable set of clusters from a small sample of the entire data set, and the second step is an incremental step, which assigns the remaining records to a suitable cluster. The algorithm is described in Algorithm 14.2.

ALGORITHM 14.2. The COOLCAT clustering algorithm.

- 1: Draw a sample data set S ($|S| << N$) from the entire data set, where N is the size of the entire data set;
 - {1. Initialization phase}
- 2: Find the k most “dissimilar” records from the sample data set by maximizing the minimum pairwise entropy of the chosen data points;
 - {To do this, it first finds the two data points p_{s_1}, p_{s_2} such that the entropy $E(p_{s_1}, p_{s_2})$ is maximized, i.e., $E(p_{s_1}, p_{s_2}) \geq E(p_1, p_2) \forall p_1, p_2 \in S$, and then puts them in two separate clusters C_1, C_2 and marks the two data points; after it selects $j - 1$ points, it will find the j th point such that

$$\min_{i=1,2,\dots,j-1} (E(p_{s_i}, p_{s_j}))$$

is maximized and then put this point in the j th cluster C_j and mark this point.}

{2. Incremental phase}

- 3: Process the unmarked $|S| - k$ data points in the sample data set S and the remaining data points (i.e., data points outside the sample). Given the k initial sets of clusters

found in the first step, $\check{C} = C_1, C_2, \dots, C_k$, bring a batch of data points to the memory from disk and, for each data point p in this batch of data points, place p in cluster C_i and compute $\bar{E}(\check{C}^i)$, where \check{C}^i denotes the cluster obtained by placing p in C_i , and then find the index j such that

$$\bar{E}(\check{C}^j) \leq \bar{E}(\check{C}^i) \quad \forall i = 1, 2, \dots, k$$

and place p in cluster C_j . The above procedure is kept executing until all points have been assigned to some cluster.

One of the disadvantages of this algorithm is that the order of the processing data points has a definite impact on the clustering quality; a fraction m of the data points in the batch are reprocessed in order to reduce this effect.

14.7 STUCCO

In the algorithm STUCCO (Bay and Pazzani, 1999), a new concept of contrast-sets is defined in order to find the contrast-sets whose supports differ meaningfully among different groups. To do that, a method of tree searching is used to calculate all possible combinations of attribute values. One then retains the significant contrast-sets, postprocesses those contrast-sets, and then selects a subset.

The preprocessing phase is as follows:

$$\begin{aligned} \exists ij \quad & P(\text{cset} = \text{True}|G_i) \neq P(\text{cset} = \text{True}|G_j), \\ & \max_{ij} |\text{support}(\text{cset}, G_i) - \text{support}(\text{cset}, G_j)| \geq \text{mindev}, \end{aligned}$$

where mindev is a user-defined threshold.

The postprocessing phase is as follows:

$$\chi^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(O_{ij} - E_{ij})^2}{E_{ij}},$$

where O_{ij} is the observed frequency count in cell ij and E_{ij} is the expected frequency count in cell ij calculated as $E_{ij} = \sum_j O_{ij} \sum_i O_{ij} / N$ under the condition that the row and column variables are independent. To determine if the differences in proportions are significant, pick a test α level. The procedure of this algorithm is described in Algorithm 14.3.

ALGORITHM 14.3. The STUCCO clustering algorithm procedure.

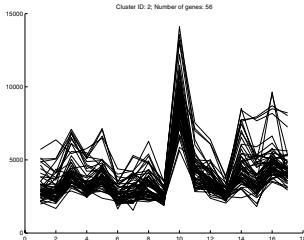
- 1: Using canonical ordering of attributes to construct a search tree, scan the database, count the support for each group, and retain the significant contrast-sets;
- 2: Test the null hypothesis that contrast-set support is equal across all groups or contrast-set support is independent of group membership to check whether a contrast-set is significant;

- 3: Prune the nodes that can never be significant contrast-sets;
- 4: Find surprising contrast-sets.

14.8 Summary

Many clustering algorithms presented in other chapters are heuristic clustering algorithms, different from the model-based clustering algorithms introduced here. Model-based clustering algorithms offer a principal alternative to heuristic algorithms. In particular, model-based approaches assume that the data are generated by a finite mixture of underlying probability distributions, such as multivariate normal distribution. To cluster a data set by heuristic algorithms, we encounter the problem of selecting a “good” clustering method and determining the “correct” number of clusters. Clustering a data set by model-based algorithms is reduced to the model selection problem in the probability framework.

Other model-based clustering algorithms are not presented in this chapter. For example, model-based algorithms for clustering gene expression data are studied by Yeung et al. (2001) and McLachlan et al. (2002). Fraley (1998) proposed an agglomerative hierarchical clustering algorithm based on the Gaussian probability model, where a maximum likelihood pair of clusters is chosen to merge at each stage. Bensmail et al. (1997) suggested a fully Bayesian analysis of the model-based method proposed by Banfield and Raftery (1993). Two variations of model-based clustering, balanced model-based clustering and hybrid model-based clustering, are proposed by Zhong and Ghosh (2003b).



Chapter 15

Subspace Clustering

Recently, subspace clustering has aroused great interest in researchers in the database community due to the new challenges associated with the high dimensionality of data sets in modern science and technology.

Many clustering algorithms have been developed to identify clusters in the whole data space; we refer to these clustering algorithms as conventional clustering algorithms. Unfortunately, most of these conventional clustering algorithms do not scale well to cluster high-dimensional data sets in terms of effectiveness and efficiency because of their inherent sparsity. In high-dimensional data sets, we encounter several problems. First, the distance between any two data points becomes almost the same (Beyer et al., 1999), so it is difficult to differentiate similar data points from dissimilar ones. Secondly, clusters are embedded in the subspaces of the high-dimensional data space, and different clusters may exist in different subspaces (Agrawal et al., 1998). Because of these problems, almost all conventional clustering algorithms fail to work well for high-dimensional data sets.

One possible solution is to use dimension reduction techniques such as principal component analysis (PCA) and the Karhunen-Loëve transformation (Agrawal et al., 1998) or feature selection techniques. In dimension reduction approaches, one first reduces the dimensionality of the original data set by removing less important variables or by transforming the original data set into a low-dimensional space and then applies conventional clustering algorithms to the new data set. In feature selection approaches, one finds the dimensions on which data points are correlated. In both dimension reduction and feature selection approaches, it is necessary to prune off some variables, which may lead to significant loss of information. This can be illustrated by considering a three-dimensional data set that has three clusters: one embedded in the (x, y) -plane, another embedded in the (y, z) -plane, and the third embedded in the (z, x) -plane. For such a data set, application of a dimension reduction or a feature selection method is unable to recover all the clustering structures, because the three clusters are formed in different subspaces. In general, clustering algorithms based on dimension reduction or feature selection techniques generate clusters that may not fully reflect the original structure of a given data set.

This difficulty that conventional clustering algorithms encounter in dealing with high-dimensional data sets motivates the concept of subspace clustering or projected clustering

Table 15.1. List of some subspace clustering algorithms. Num refers to numerical and Mix refers to mixed-type.

Algorithms	Data Type	H/P
CLIQUE (Agrawal et al., 1998)	Num	Other
ENCLUS (Cheng et al., 1999)	Num	Other
MAFIA (Goil et al., 1999)	Num	Other
PROCLUS (Aggarwal et al., 1999)	Num	Partitioning
ORCLUS (Aggarwal and Yu, 2000)	Num	Partitioning
FINDIT (Woo and Lee, 2002)	Num	Partitioning
FLOC (Yang et al., 2002a)	Num	Partitioning
DOC (Procopiuc et al., 2002)	Num	Partitioning
PART (Cao and Wu, 2002)	Num	Hierarchical
CLTree (Liu et al., 2000)	Num	Other
COSA	Mix	Other

(Agrawal et al., 1998), whose goal is to find clusters embedded in subspaces of the original data space with their own associated dimensions. In other words, subspace clustering finds clusters and their relevant attributes from a data set. Table 15.1 lists a few popular subspace clustering algorithms that will be introduced in this chapter.

15.1 CLIQUE

CLIQUE (Agrawal et al., 1998) is a clustering algorithm that is able to identify dense clusters in subspaces of maximum dimensionality. It is also the first subspace clustering algorithm. This algorithm takes two parameters: ξ , which specifies the number of intervals in each dimension, and τ , which is the density threshold. The output is clusters, each of which is represented by a minimal description in the form of a disjunct normal form (DNF) expression. One disadvantage of this algorithm is that it can only find clusters embedded in the same subspace. The clustering model can be described as follows.

Let $\mathcal{A} = \{A_1, A_2, \dots, A_d\}$ be a set of bounded, totally ordered numerical domains (for categorical data, each A_i should be a finite set of categorical values). Let $\mathcal{S} = A_1 \times A_2 \times \dots \times A_d$ be a d -dimensional numerical space. Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a database, where $\mathbf{x}_i = (\mathbf{x}_{i1}, \mathbf{x}_{i2}, \dots, \mathbf{x}_{id})$, and the j th component of \mathbf{x}_i is drawn from A_j .

The data space \mathcal{S} is partitioned into nonoverlapping rectangular units that are obtained by partitioning each dimension into ξ (an input parameter) intervals of equal length. Each unit u is the intersection of one interval from each dimension, and it has the form $\{u_1, u_2, \dots, u_d\}$, where $u_i = [l_i, h_i)$ is a right open interval in the partitioning of dimension A_i . A data point $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d)$ is said to be contained in a unit $u = \{u_1, u_2, \dots, u_d\}$ if the following condition is satisfied:

$$l_i \leq \mathbf{x}_i < h_i \text{ for all } u_i.$$

A unit u is said to be dense if $\text{selectivity}(u) > \tau$, where $\text{selectivity}(u)$ is the selectivity of unit u , which is defined to be the fraction of total data points contained in the unit u ; the density threshold τ is another input parameter. The units in all subspaces of the original d -dimensional space can be defined similarly. Let $\mathcal{S}' = \{A_{t_1}, A_{t_2}, \dots, A_{t_r}\}$ be any subspace of \mathcal{S} , where $l < d$ and $t_i < t_j$ if $i < j$. The units in \mathcal{S}' are the intersection of an interval from each of the r dimensions of the subspace \mathcal{S}' .

In the CLIQUE algorithm, a cluster is defined to be a maximal set of connected dense units in r dimensions. Two r -dimensional units are said to be connected if they have a common face or if there exists another r -dimensional unit such that the unit connects the two units. Two units $u_1 = \{r_{t_1}, r_{t_2}, \dots, r_{t_r}\}$ and $u_2 = \{r'_{t_1}, r'_{t_2}, \dots, r'_{t_r}\}$ are said to have a common face if there are $l - 1$ dimensions, assumed to be $A_{t_1}, A_{t_2}, \dots, A_{t_r}$, such that $r_{t_j} = r'_{t_j}$ ($1 \leq j \leq l - 1$) and either $h_{t_r} = l'_{t_r}$ or $h'_{t_r} = l_{t_r}$.

A region in r dimensions is an axis-parallel rectangular r -dimensional set that can be expressed as unions of units. A region R is said to be contained in a cluster C if $R \cap C = R$. A region R contained in a cluster C is said to be maximal if no proper superset of R is contained in C . A minimal description of a cluster C is a set \mathcal{R} of maximal regions such that their union equals C but the union of any proper subset of \mathcal{R} does not equal C .

The CLIQUE algorithm consists of three steps. In the first step, the subspaces that contain clusters are identified. In the second step, the clusters embedded in the subspaces identified in step 1 are found. Finally, a minimal description of each cluster is generated.

In the first step, a bottom-up algorithm is employed to find the dense units. The essential observation is that if a set of points S is a cluster in an r -dimensional space, then S is also part of a cluster in any $(r - 1)$ -dimensional projections of this space. Based on this fact, the algorithm proceeds level by level. It first determines one-dimensional dense units, and then continues iteratively: when the $(r - 1)$ -dimensional dense units are determined, the r -dimensional dense units are determined as follows. Let D_{r-1} be the set of all $(r - 1)$ -dimensional dense units, and let C_r be the set of r -dimensional units generated from D_{r-1} as follows. For two units u_1 and u_2 in D_{r-1} , if $u_1.a_j = u_2.a_j$, $u_1.l_j = u_2.l_j$, and $u_1.h_j = u_2.h_j$ for $j = 1, 2, \dots, r - 2$, and $u_1.a_{r-1} < u_2.a_{r-1}$, then insert $u = u_1.[l_1, h_1] \times u_1.[l_2, h_2] \times \dots \times u_1.[l_{r-1}, h_{r-1}] \times u_2.[l_{r-1}, h_{r-1}]$ into C_r , where $u.a_i$ is the i th dimension of unit u and $u.[l_i, h_j]$ is the interval in the i th dimension of u , and the relation $<$ represents lexicographic ordering on attributes. Then D_r is obtained by discarding those dense units from C_r that have a projection in $(r - 1)$ dimensions that is not included in C_{r-1} .

In order to make the first step faster, the minimal description length (MDL) principle is applied to decide which subspaces and corresponding dense units are interesting. Assume there are S_1, S_2, \dots, S_m subspaces found in the r th level (then S_1, S_2, \dots, S_m are r -dimensional subspaces). Let $c(S_j)$ be the coverage of subspace S_j defined as

$$c(S_j) = \sum_{u \in S_j} |u|,$$

where $|u|$ is the number of points that fall inside u . Subspaces with large coverage are selected to form candidate units in the next level of the dense unit generation algorithm, and the rest are pruned. Let the subspaces be sorted in decreasing order of their coverage. The sorted list is divided into the selected set I and the pruned set P . Assume $c(S_1) > c(S_2) >$

$\dots > c(S_m)$. Then $I = \{S_1, S_2, \dots, S_{i_0}\}$ and $P = \{S_{i_0+1}, S_{i_0+2}, \dots, S_m\}$ for some i . In CLIQUE, i_0 is selected such that the total length of encoding $CL(i)$ is minimized, where $CL(i)$ is defined as

$$\begin{aligned} CL(i) = & \log_2(\mu_I(i)) + \sum_{j=1}^i \log_2(|c(S_j) - \mu_I(i)|) \\ & + \log_2(\mu_P(i)) + \sum_{j=i+1}^m \log_2(|c(S_j) - \mu_P(i)|), \end{aligned}$$

where $\mu_I(i)$ and $\mu_P(i)$ are defined as

$$\begin{aligned} \mu_I(i) &= \left\lceil \sum_{j=1}^i \frac{c(S_j)}{i} \right\rceil, \\ \mu_P(i) &= \left\lceil \sum_{j=i+1}^m \frac{|c(S_j)|}{n-i} \right\rceil, \end{aligned}$$

where $\lceil \cdot \rceil$ is the ceiling function, i.e., $\lceil x \rceil$ is the least integer not less than x .

The second step of the CLIQUE algorithm is to find the clusters based on a set of dense units \mathcal{C} found in the first step. All units in \mathcal{C} are in the same r -dimensional space S . In this step, \mathcal{C} will be partitioned into C_1, C_2, \dots, C_k such that all units in C_i are connected and units in different groups are not connected. This process is converted to finding connected components in a graph defined as follows: graph vertices correspond to dense units, and two vertices have an edge between them if and only if the corresponding dense units have a common face. The depth-first search algorithm (Hopcroft and Tarjan, 1973; Aho et al., 1974) is used to find the connected components of the graph.

The third step is to generate minimal cluster descriptions for each cluster. The input to this step is disjoint sets of connected r -dimensional units in the same subspace. Given a cluster C in an r -dimensional subspace S , a set \mathcal{R} of regions in the same subspace S is said to be a cover of C if every region in \mathcal{R} is contained in C and each unit in C is contained in at least one of the regions in \mathcal{R} . In this step, a minimal cover is found for each cluster by first greedily covering the cluster by a number of maximal regions and then discarding the redundant regions.

The time complexity in step 1 is $O(c^k + mk)$ for a constant c . In step 2 and subsequent steps, the dense units are assumed to be stored in memory. The total number of data structure accesses is $2kn$ in step 2.

15.2 PROCLUS

PROCLUS (PROjected CLUSting) (Aggarwal et al., 1999) is a variation of the k -medoid algorithm (Kaufman and Rousseeuw, 1990) in subspace clustering. The input parameters of the algorithm are the number of clusters k and the average number of dimensions l ; the output is a partition $\{C_1, C_2, \dots, C_k, O\}$ together with a possibly different subset of dimensions P_i for each cluster C_i , with O denoting the outlier cluster.

The algorithm consists of three phases: the initialization phase, the iteration phase, and the refinement phase. In the initialization phase, a random set of k medoids is chosen by applying the greedy technique (Gonzalez, 1985; Ibaraki and Katoh, 1988) to samples of the original data set. In the iteration phase, the algorithm progressively improves the quality of medoids by iteratively replacing the bad medoids with new ones. The last phase computes new dimensions associated with each medoid and reassigns the points to the medoids relative to the new sets of dimensions. The PROCLUS algorithm finds out the subspace dimensions of each cluster via a process of evaluating the locality of the space near it.

ALGORITHM 15.1. The PROCLUS algorithm.

Require: D -Data set, k -Number of clusters, l -Average dimensions of cluster;

- 1: Let A, B be constant integers;
- 2: Draw a sample S of size $A \cdot k$ randomly;
- 3: Let medoids set $M \Leftarrow Greedy(S, B \cdot k)$;
- 4: Let $Best\ Objective \Leftarrow \infty$ and $M_{current} \Leftarrow$ random set of medoids $\{m_1, m_2, \dots, m_k\} \subset M$;
- 5: **repeat**
- 6: Let δ_i be the distance to the nearest medoid from m_i for $i = 1, 2, \dots, k$;
- 7: Let L_i be the set of points in a sphere centered at m_i with radius δ_i for $i = 1, 2, \dots, k$;
- 8: $(P_1, P_2, \dots, P_k) \Leftarrow FindDimensions(k, l, L_1, L_2, \dots, L_k)$;
- 9: $(C_1, C_2, \dots, C_k) \Leftarrow AssignPoints(P_1, P_2, \dots, P_k)$;
- 10: $Objective \Leftarrow EvaluateClusters(C_1, \dots, C_k, P_1, \dots, P_k)$;
- 11: **if** $Objective < Best\ Objective$ **then**
- 12: Let $Best\ Objective \Leftarrow Objective$, $M_{best} \Leftarrow M_{current}$;
- 13: Compute the bad medoids in M_{best} ;
- 14: **end if**
- 15: Compute $M_{current}$ by replacing the bad medoids in M_{best} with random points from M ;
- 16: **until** Stop criterion
- 17: $(P_1, P_2, \dots, P_k) \Leftarrow FindDimensions(k, l, L_1, L_2, \dots, L_k)$;
- 18: $(C_1, C_2, \dots, C_k) \Leftarrow AssignPoints(P_1, P_2, \dots, P_k)$;
- 19: Return $M_{best}, P_1, P_2, \dots, P_k$;

The pseudocode of the algorithm is described in Algorithm 15.1. In the initialization phase, a medoid candidate set M is selected using the greedy technique. In order to reduce the running time of the initialization phase, a sample S is drawn randomly from the original data set. The procedure of $Greedy(S, t)$ can be described as follows:

- 1: $M \Leftarrow \{m_1\}$ $\{m_1$ is a random point of $S\}$;
- 2: Let $dist(x) \Leftarrow d(x, m_1)$ for each $x \in S \setminus M$;
- 3: **for** $i = 2$ to t **do**
- 4: Let $m_i \in S \setminus M$ be such that $dist(m_i) = \max\{dist(x) : x \in S \setminus M\}$;
- 5: $M \Leftarrow M \cup \{m_i\}$;
- 6: Let $dist(x) \Leftarrow \min\{dist(x), d(x, m_i)\}$;

- 7: **end for**
 8: Return M .

In the iteration phase, the quality of medoids is improved progressively by iteratively replacing the bad medoids. This phase consists of three major procedures: *FindDimensions*, *AssignPoints*, and *EvaluateClusters*. In order to find the subspace dimensions, PROCLUS evaluates the locality of the space near the medoids. Let L_i be the set of points in a sphere centered at m_i with radius δ_i , where $\delta_i = \min_{1 \leq j \leq k, j \neq i} d(m_i, m_j)$. Let X_{ij} , Y_i , and σ_i be defined as

$$\begin{aligned} X_{ij} &= \frac{1}{|L_i|} \sum_{\mathbf{x} \in L_i} d(\mathbf{x}_j, m_{ij}), \\ Y_i &= \sum_{j=1}^d \frac{X_{ij}}{d}, \\ \sigma_i &= \left(\frac{1}{d-1} \sum_{j=1}^d (X_{ij} - Y_i)^2 \right)^{\frac{1}{2}} \end{aligned}$$

for $i = 1, 2, \dots, k$ and $j = 1, 2, \dots, d$, where \mathbf{x}_j and m_{ij} are the values of the j th dimension of \mathbf{x} and m_i , respectively, and $d(\cdot, \cdot)$ is the Manhattan segmental distance.

Let $Z_{ij} = \frac{X_{ij} - Y_i}{\sigma_i}$ for $i = 1, 2, \dots, k$ and $j = 1, 2, \dots, d$. The goal of the procedure *FindDimensions* is to pick up the $k \cdot l$ numbers with the least values of Z_{ij} subject to the constraint that there are at least two dimensions for each cluster and to put dimension j to P_i if Z_{ij} is picked. This can be achieved by the greedy technique.

The *Assign Points* procedure assigns each point \mathbf{x} in the data set to the cluster C_i such that $d_{P_i}(\mathbf{x}, m_i)$ has the lowest values among $d_{P_1}(\mathbf{x}, m_1), \dots, d_{P_k}(\mathbf{x}, m_k)$, where $d_{P_i}(\mathbf{x}, m_i)$ is the Manhattan segmental distance from \mathbf{x} to the medoid m_i relative to dimensions P_i , i.e.,

$$d_{P_i}(\mathbf{x}, m_i) = \frac{1}{|P_i|} \sum_{j \in P_i} |\mathbf{x}_j - m_{ij}|.$$

The *EvaluateClusters* procedure evaluates the quality of a set of medoids as the average Manhattan segmental distance from the points to the centroids of the clusters to which they belong. The quantity that indicates the goodness of a clustering is defined as

$$\frac{1}{n} \sum_{i=1}^k |C_i| \cdot w_i, \quad (15.1)$$

where n is the number of points in the data set and w_i ($i = 1, 2, \dots, k$) are weights defined as

$$w_i = \frac{1}{|P_i| \cdot |C_i|} \sum_{j \in P_i} \sum_{\mathbf{x} \in C_i} |\mathbf{x}_j - z_{ij}|,$$

where P_i is the dimension associated with C_i , and z_i is the centroid of C_i , i.e.,

$$z_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}.$$

The optimal clustering should have a minimum quantity, which is defined in (15.1). The goal of the iteration phase is to find an approximation of such an optimal clustering. Also in the iteration phase, bad medoids are determined. The medoid of any cluster that has less than $\frac{n}{k} \cdot \sigma_{min}$ points is bad, where σ_{min} is a constant smaller than 1. In PROCLUS, σ_{min} is set to be 0.1. Bad medoids may be outliers.

After the iteration phase, the best set of medoids is found. The refinement phase does one more pass over the data to improve the quality of the clustering. Outliers are also handled in the refinement phase. The running time for computing the segmental distances is $O(nkl)$ for each iteration.

15.3 ORCLUS

ORCLUS (arbitrarily ORiented projected CLUSter generation) (Aggarwal and Yu, 2000) is an extension of PROCLUS. It diagonalizes the covariance matrix of each cluster and finds information about projection subspaces from the diagonalization of the covariance matrix. In order to make the algorithm scale to large databases, ORCLUS also uses extended cluster feature (ECF) vectors (Zhang et al., 1996) and a progressive random sampling approach in the algorithm.

In the algorithm, a generalized projected cluster is defined for a pair (C, P) , where C is a set of data points and P is a set of vectors, such that the data points in C are closely clustered in the subspace defined by the vectors in P . Unlike CLIQUE (Agrawal et al., 1998), the subspaces for different clusters found by the ORCLUS algorithm may be different.

The ORCLUS algorithm takes two input parameters: the number of clusters k and the dimensionality l of the subspace in which each cluster is reported. The output of the algorithm consists of two parts: a $(k + 1)$ -way partition $\{C_1, C_2, \dots, C_k, O\}$ of the data set and a possibly different orthogonal set P_i of vectors for each cluster C_i ($1 \leq i \leq k$), where O is the outlier cluster, which can be assumed to be empty. For each cluster C_i , the cardinality of the corresponding set P_i is l , which is the user-specified parameter.

ALGORITHM 15.2. The pseudocode of the ORCLUS algorithm.

Require: D -Data set, k -Number of clusters, and l -Number of dimensions;

- 1: Pick $k_0 > k$ initial data points from D and denote them by $S = \{s_1, s_2, \dots, s_{k_0}\}$;
- 2: Set $k_c \leftarrow k_0$ and $l_c \leftarrow d$;
- 3: For each i , set P_i to be the original axis system;
- 4: Set $\alpha \leftarrow 0.5$ and compute β according to equation (15.3);
- 5: **while** $k_c > k$ **do**
- 6: $(s_1, \dots, s_{k_c}, C_1, \dots, C_{k_c}) = Assign(s_1, \dots, s_{k_c}, P_1, \dots, P_{k_c})$ {find the partitioning induced by the seeds};
- 7: Set $k_{new} \leftarrow \max\{k, k_c \cdot \alpha\}$ and $l_{new} \leftarrow \max\{l, l_c \cdot \beta\}$;
- 8: $(s_1, \dots, s_{k_{new}}, C_1, \dots, C_{k_{new}}, P_1, \dots, P_{k_{new}}) = Merge(C_1, \dots, C_{k_c}, k_{new}, l_{new})$;
- 9: Set $k_c \leftarrow k_{new}$ and $l_c \leftarrow l_{new}$;
- 10: **end while**
- 11: **for** $i = 1$ to k **do**
- 12: $P_i = FindVectors(C_i, l)$;

```

13: end for
14:  $(s_1, \dots, s_k, C_1, \dots, C_k) = Assign(s_1, \dots, s_k, P_1, \dots, P_k);$ 
15: Output  $(C_1, \dots, C_k);$ 

```

The ORCLUS algorithm consists of a number of iterations, in each of which a variant of the hierarchical merging method is applied in order to reduce the number of current clusters by the factor $\alpha < 1$. Initially, the dimensionality of each cluster is equal to the full dimensionality d . In each iteration, the dimensionality of the current clusters is reduced by the factor $\beta < 1$. Finally, the dimensionality of the clusters will be reduced gradually to the user-specified dimensionality l . In order to make the algorithm fast, a small number k_0 of initial points are picked as seeds. At each stage of the algorithm, there is a projected cluster (C_i, P_i) associated with each seed s_i . At each stage of the algorithm, k_c and l_c denote the number of current clusters and the dimensionality of the current clusters, respectively.

In order to make the reduction from k_0 to k clusters occur in the same number of iterations as the reduction from $l_0 = d$ to l dimensions, the values of α and β must satisfy the relationship

$$k_0\alpha^N = k \text{ and } l_0\beta^N = l,$$

where N is the number of iterations, which is equivalent to

$$\log_{\frac{1}{\alpha}}\left(\frac{k_0}{k}\right) = \log_{\frac{1}{\beta}}\left(\frac{d}{l}\right). \quad (15.2)$$

In the ORCLUS algorithm, the value of α is set to 0.5 and the value of β is calculated according to equation (15.2), i.e.,

$$\beta = -\exp\left(\frac{(\ln \frac{d}{l}) \cdot (\ln \frac{1}{\alpha})}{\ln \frac{k_0}{k}}\right). \quad (15.3)$$

The ORCLUS algorithm is briefly described in Algorithm 15.2. In the algorithm, there are three subprocedures: $Assign(s_1, \dots, s_{k_c}, P_1, \dots, P_{k_c})$, $Merge(s_1, \dots, s_{k_c}, k_{new}, l_{new})$, and $FindVectors(C_i, q)$. Like most partitional clustering algorithms, the $Assign$ procedure assigns each data point to its nearest seed according to the projected distance. Let \mathbf{x} and \mathbf{y} be two data points in the original d -dimensional space, and let $P = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_p\}$ be a set of orthonormal vectors. Then the projected distance between \mathbf{x} and \mathbf{y} in the subspace defined by the vectors in P is defined to be the Euclidean distance between their projections in the p -dimensional space represented by P , i.e.,

$$d_P(\mathbf{x}, \mathbf{y}, P) = \left(\sum_{i=1}^p (\mathbf{x} \cdot \mathbf{e}_i - \mathbf{y} \cdot \mathbf{e}_i)^2 \right)^{\frac{1}{2}}, \quad (15.4)$$

where $\mathbf{x} \cdot \mathbf{e}_i$ and $\mathbf{y} \cdot \mathbf{e}_i$ denote the dot-products of \mathbf{x} and \mathbf{e}_i and \mathbf{y} and \mathbf{e}_i , respectively.

ALGORITHM 15.3. $Assign(s_1, \dots, s_{k_c}, P_1, \dots, P_{k_c}).$

Require: D -Data set;

- 1: Set $C_i \Leftarrow \Phi$ for $i = 1, 2, \dots, k_c$;
- 2: **for all** data points $\mathbf{x} \in D$ **do**
- 3: Let $d_{min} \Leftarrow d_P(\mathbf{x}, s_1, P_1)$ and $i_{min} \Leftarrow 1$;

```

4:   for  $i = 2$  to  $k_c$  do
5:     if  $d_{min} > d_P(\mathbf{x}, s_i, P_i)$  then
6:        $d_{min} \Leftarrow d_P(\mathbf{x}, s_i, P_i);$ 
7:        $i_{min} \Leftarrow i;$ 
8:     end if
9:   end for
10:  Assign  $\mathbf{x}$  to  $C_{i_{min}};$ 
11: end for
12: Set  $s_i \Leftarrow \bar{X}(C_i)$  for  $i = 1, 2, \dots, k_c$  { $\bar{X}(C_i)$  is the centroid of cluster  $C_i$ };
13: Return  $(s_1, \dots, s_{k_c}, C_1, \dots, C_{k_c});$ 

```

The *Assign* procedure is described in Algorithm 15.3. In this procedure, the vector sets P_i ($i = 1, 2, \dots, k_c$) are fixed, and each data point is assigned to the nearest cluster according to the projected distance defined in (15.4). Inside the *Assign* procedure, $\bar{X}(C)$ denotes the centroid of C , which is defined as

$$\bar{X}(C) = \sum_{\mathbf{x} \in C} \frac{\mathbf{x}}{|C|}. \quad (15.5)$$

ALGORITHM 15.4. *Merge*($C_1, \dots, C_{k_c}, K_{new}, l_{new}$).

```

1: if  $k_{new} \leq k$  then
2:   Exit;
3: end if
4: for  $i = 1$  to  $k_c$  do
5:   for  $j = i + 1$  to  $k_c$  do
6:      $P_{ij} = FindVectors(C_i \cup C_j, l_{new});$ 
7:      $s_{ij} = \bar{X}(C_i \cup C_j)$  {centroid of  $C_i \cup C_j$ };
8:      $r_{ij} = R(C_i \cup C_j, P_{ij});$ 
9:   end for
10: end for
11: while  $k_c > k_{new}$  do
12:   Find  $i', j'$  such that  $r_{i'j'} = \min_{1 \leq i < j \leq k_c} r_{ij};$ 
13:   Set  $s_{i'} \Leftarrow s_{i'j'}, C_{i'} \Leftarrow C_{i'} \cup C_{j'}, P_{i'} \Leftarrow P_{i'j'} \{merge C_{i'} and C_{j'}\};$ 
14:   Discard seed  $s_{j'}$  and  $C_{j'}$ , renumber the seeds and clusters indexed larger than  $j'$  by
      subtracting 1, and renumber  $s_{ij}, P_{ij}, r_{ij}$  correspondingly for any  $i, j \geq j'$ ;
15:   for  $j = 1, j \neq i' to k_c - 1$  do
16:     Update  $P_{i'j}, s_{ij}$  and  $r_{ij}$  if  $i' < j$ ; otherwise update  $P_{ji'}, s_{ji'},$  and  $r_{ji'};$ 
17:   end for
18:    $k_c \Leftarrow k_c - 1;$ 
19: end while
20: Return  $(s_1, \dots, s_{k_{new}}, C_1, \dots, C_{k_{new}}, P_1, \dots, P_{k_{new}});$ 

```

The *Merge* procedure is used to reduce the number of clusters from k_c to $k_{new} = \alpha \cdot k_c$ during a given iteration. The pseudocode of the *Merge* procedure is described in

Algorithm 15.4. In this procedure, the two closest pairs of current clusters are merged successively. The closeness between two clusters is measured by projected energy. Let (C, P) be a projected cluster. Then the projected energy of cluster C in subspace P is defined as

$$R(C, P) = \sum_{\mathbf{x} \in C} \frac{1}{|C|} d_P^2(\mathbf{x}, \bar{X}(C), P), \quad (15.6)$$

where $d_P(\cdot, \cdot, \cdot)$ is the projected distance defined in (15.4), and $\bar{X}(C)$ is the centroid of C , which is defined in (15.5).

ALGORITHM 15.5. *FindVectors(C, q).*

- 1: Compute the $d \times d$ covariance matrix M for C ;
- 2: Find the eigenvectors and eigenvalues of matrix M ;
- 3: Let P be the set of eigenvectors corresponding to the smallest q eigenvalues;
- 4: Return P ;

The *FindVectors* procedure (Algorithm 15.5) is used to determine the optimal subspace associated with each cluster given the dimensionality. This is achieved by finding the eigenvalues and eigenvectors of the covariance matrix for the cluster. For a given cluster C , the subspace associated with C is defined by the orthonormal eigenvectors with the least spread (eigenvalues).

The ORCLUS algorithm can also handle outliers by measuring the projected distance between each data point and the seed of the cluster to which it belongs. More specifically, for each $i \in \{1, \dots, k_c\}$, let δ_i be the projected distance between the seed s_i and its nearest other seed in subspace P_i , let \mathbf{x} be any data point, and let s_r be the seed to which \mathbf{x} is assigned. If the projected distance between \mathbf{x} and s_r in subspace P_r is larger than δ_r , then \mathbf{x} is treated as an outlier.

In order to make the algorithm scalable to very large databases, the concept of ECF-vectors is used, as in BIRCH (Zhang et al., 1996). For each cluster, there is an ECF-vector associated with it. The ECF-vector for each cluster in the ORCLUS algorithm consists of $d^2 + d + 1$ entries. Let C be a cluster. Then the ECF-vector $\mathbf{v}(C) = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{d^2+d+1})$ for C is defined as

$$\begin{aligned} \mathbf{v}_{(i-1)d+j} &= \sum_{\mathbf{x} \in C} \mathbf{x}_i \cdot \mathbf{x}_j, \quad i, j = 1, 2, \dots, d, \\ \mathbf{v}_{d^2+i} &= \sum_{\mathbf{x} \in C} \mathbf{x}_i, \quad i = 1, 2, \dots, d, \\ \mathbf{v}_{d^2+d+1} &= |C|, \end{aligned}$$

where \mathbf{x}_i and \mathbf{x}_j denote the i th and j th components of the data point \mathbf{x} , respectively.

From the definition of the ECF-vectors, one can see that an ECF-vector can be divided into three parts. The ECF-vectors defined above have some good properties. For example, the ECF-vector satisfies the additive property, i.e., the ECF-vector for the union of two clusters is equal to the sum of the corresponding ECF-vectors. Also, the covariance matrix for a cluster can be derived directly from the ECF-vector of the cluster. Namely, let C be a

cluster and $\mathbf{v} = \mathbf{v}(C)$. Then the covariance matrix $M = (m_{ij})_{d \times d}$ for C is

$$m_{ij} = \frac{\mathbf{v}_{(i-1)d+j}}{\mathbf{x}_{d^2+d+1}} - \frac{\mathbf{v}_{d^2+i} \cdot \mathbf{v}_{d^2+j}}{\mathbf{v}_{d^2+d+1}^2}.$$

The running time for the *Merge* procedure is $O(k_0^3 + k_0^2 d)$, and the running time for the *Assign* procedure is $O(\frac{d}{1-\alpha} k_0 n)$. Since the running time for subspace determinations is strictly dominated by the subspace determinations during the *Merge* phase, this running time can be ignored. The total running time of the ORCLUS algorithm is therefore $O(k_0^3 + k_0 n d + k_0^2 d)$.

15.4 ENCLUS

ENCLUS (ENtropy-based CLUStering) (Cheng et al., 1999) is an entropy-based subspace clustering algorithm for clustering numerical data. It can find arbitrarily shaped clusters embedded in the subspaces of the original data space. It follows similar approaches suggested by CLIQUE, but does not make any assumptions about the cluster shapes and hence is capable of finding arbitrarily shaped clusters embedded in subspaces. The ENCLUS algorithm searches for subspace clusters based on the fact that a subspace with clusters typically has lower entropy than a subspace without clusters. In ENCLUS, the entropy of a data set is defined as

$$H(X) = - \sum_{x \in \chi} d(x) \log d(x), \quad (15.7)$$

where X is a set of variables, χ is the set of all units in the space formed by X , and $d(x)$ is the density of a unit x in terms of the percentage of data points contained in unit x . ENCLUS makes use of the relationship between an entropy and a clustering criterion to identify the subspaces with good clustering, and then applies methods developed in CLIQUE or other algorithms to identify all clusters in the subspaces identified. Entropy and clustering criteria are related. For example, the entropy decreases as the density of the dense units increases, and we also have the following correspondence between entropy and dimension correlations:

$$H(V_1, V_2, \dots, V_i) = \sum_{j=1}^i H(V_j) \quad (15.8)$$

if and only if V_1, V_2, \dots, V_i are independent;

$$H(V_1, \dots, V_i, Y) = H(V_1, V_2, \dots, V_i)$$

if and only if Y is a function of V_1, V_2, \dots, V_i .

To discover the subspaces with good clusters, the algorithm uses the fact of downward closure, i.e., if an l -dimensional subspace V_1, V_2, \dots, V_l has good clustering, so do all $(l-1)$ -dimensional projections of this space. To discover the correlations between variables, the algorithm uses the fact of upward closure, i.e., if a set of dimensions S is correlated, so is every superset of S . Upward closure allows us to find the minimally correlated subspaces.

A set of variables V_1, V_2, \dots, V_i are correlated if equation (15.8) is not satisfied. Define the *interest* of a set of variables by

$$\text{interest}(\{V_1, V_2, \dots, V_i\}) = \sum_{j=1}^i H(V_j) - H(V_1, V_2, \dots, V_i). \quad (15.9)$$

Let S_l be the set of l -dimensional significant subspaces and NS_l be the set of l -dimensional subspaces with good clustering but not minimal correlation. The procedure for mining significant subspaces is described in Algorithm 15.6. In this procedure, one first builds grids in the subspace c and calculates the density of each cell. Then the entropy for this subspace is computed based on the density information using the formula in (15.7). The candidate set C_{l+1} is generated as follows: select $p, q \in NS_l$ that satisfy $p.dim_i = q.dim_i$ for $i = 1, 2, \dots, l-1$ and $p.dim_l < q.dim_l$ and then insert p and $q.dim_l$ into C_{l+1} .

ALGORITHM 15.6. ENCLUS procedure for mining significant subspaces.

Require: D -Data set, ω -Entropy threshold, ϵ -Interest threshold;

```

1: Let  $l \Leftarrow 1$  and  $C_1$  be one-dimensional subspaces;
2: while  $C_l \neq \Phi$  do
3:   for all  $c \in C_l$  do
4:     Calculate the density  $f_c(\cdot)$ ;
5:     Calculate the entropy  $H(c)$  from  $f_c(\cdot)$ ;
6:     if  $H(c) < \omega$  then
7:       if  $\text{interest}(c) > \epsilon$  then
8:          $S_l \Leftarrow S_l \cup c$ ;
9:       else
10:         $NS_l \Leftarrow NS_l \cup c$ ;
11:      end if
12:    end if
13:   end for
14:   Generate  $C_{l+1}$  from  $NS_l$ ;
15:    $l \Leftarrow l + 1$ ;
16: end while
17: Output  $\bigcup_l S_l$  as significant subspaces.
```

Since the mining of high-dimensional clusters is often avoided by the above procedure, another procedure used to mine interesting subspaces has been proposed. Define the *interest_gain* of a set of variables by

$$\begin{aligned} & \text{interest_gain}(\{V_1, V_2, \dots, V_i\}) \\ &= \text{interest}(\{V_1, V_2, \dots, V_i\}) - \max_{1 \leq l \leq i} \{\text{interest}(\{V_1, V_2, \dots, V_i\} \setminus \{V_l\})\}, \end{aligned}$$

where $\text{interest}(\cdot)$ is defined in (15.9). The procedure for mining interesting subspaces is similar to the one for mining significant subspaces. The modified procedure for mining interesting subspaces is described in Algorithm 15.7. The procedure for mining interesting

subspaces has more candidates and a longer running time than the one for mining significant subspaces.

ALGORITHM 15.7. ENCLUS procedure for mining interesting subspaces.

Require: D -Data set, ω -Entropy threshold, ϵ' -Interest threshold;

```

1: Let  $l \Leftarrow 1$  and  $C_1$  be one-dimensional subspaces;
2: while  $C_l \neq \Phi$  do
3:   for all  $c \in C_l$  do
4:     Calculate the density  $f_c(\cdot)$ ;
5:     Calculate the entropy  $H(c)$  from  $f_c(\cdot)$ ;
6:     if  $H(c) < \omega$  then
7:       if  $interest\_gain(c) > \epsilon'$  then
8:          $I_l \Leftarrow I_l \cup c$ ;
9:       else
10:       $NI_l \Leftarrow NI_l \cup c$ ;
11:    end if
12:   end if
13:   end for
14:   Generate  $C_{l+1}$  from  $I_l \cup NI_l$ ;
15:    $l \Leftarrow l + 1$ ;
16: end while
17: Output  $\bigcup_l S_l$  as interesting subspaces.
```

15.5 FINDIT

FINDIT (a Fast and INtelligent subspace clustering algorithm using DImension voTing) (Woo and Lee, 2002) is a subspace clustering algorithm that uses a dimension-oriented distance measure and a dimension voting policy to determine the correlated dimensions for each cluster.

The dimension-oriented distance dod in FINDIT is defined as

$$dod_\epsilon(\mathbf{x}, \mathbf{y}) = \max\{dod_\epsilon(\mathbf{x} \rightarrow \mathbf{y}), dod_\epsilon(\mathbf{y} \rightarrow \mathbf{x})\}, \quad (15.10)$$

where $dod_\epsilon(\mathbf{x} \rightarrow \mathbf{y})$ is the directed dod defined as

$$dod_\epsilon(\mathbf{x} \rightarrow \mathbf{y}) = |Q_x| - |\{j : |\mathbf{x}_j - \mathbf{y}_j| \leq \epsilon, j \in Q_x \cap Q_y\}|, \quad (15.11)$$

where Q_x is the subspace dimension in which the dimensional values of the data point \mathbf{x} are meaningful.

The FINDIT algorithm consists of three phases: the sampling phase, the cluster-forming phase, and the data-assigning phase. In the sampling phase, two samples S and M are made for the original data set by Chernoff bounds (Motwani and Raghavan, 1995; Guha et al., 1998) according to the data set size and a parameter $C_{minsize}$. Let $C_b(k, n)$ be the minimum size of sample S such that every cluster has more than ξ data points in the

sample with probability $1 - \delta$. Then the $C_b(k, n)$ can be computed from the formula (Woo and Lee, 2002; Guha et al., 1998)

$$C_b(k, n) = \xi k\rho + k\rho \log \left(\frac{1}{\delta} \right) + k\rho \sqrt{\left(2\xi + \log \frac{1}{\delta} \right) \log \frac{1}{\delta}}, \quad (15.12)$$

where k is the number of clusters, ρ is a value satisfying the equation

$$\rho = \frac{n}{k \cdot |C_{min}|},$$

and C_{min} is the smallest cluster in the partition. FINDIT takes $\xi = 30$, $\delta = 0.01$, and $C_{min} = C_{minsize}$ in (15.12) for the sample S , and $\xi = 1$, $\delta = 0.01$, and $C_{min} = C_{minsize}$ for the sample M .

In the cluster-forming phase, the subspace dimensions of each cluster are determined by a method called dimension voting and the best medoid cluster set is formed in order to be used in the following data assignment phase. In order to find an appropriate ϵ , the cluster-forming phase is iterated several times with increasing ϵ value in $[\frac{1}{100}R, \frac{25}{100}R]$, where R is the normalized value range. For a fixed ϵ , the V nearest neighbors are sequentially searched from S for each medoid in M in order to determine the correlated dimensions of the medoid. The distance between a point $\mathbf{x} \in S$ and a medoid $\mathbf{z} \in M$ is measured as

$$dod_\epsilon(\mathbf{z}, \mathbf{x}) = d - |\{j : |\mathbf{z}_j - \mathbf{y}_j| \leq \epsilon, j \in Q\}|,$$

where $Q = \{1, 2, \dots, d\}$ and d is the number of dimensions, since no correlated dimensions have been selected for both points yet, i.e., $Q_x = Q_z = Q$. After selecting the V nearest neighbors, the algorithm calculates the number of selected neighbors that vote “yes” for each dimension, i.e., the number of selected neighbors from \mathbf{z} is less than or equal to ϵ for each dimension, i.e.,

$$V_j = |\{j : |\mathbf{x}_j - \mathbf{z}_j| \leq \epsilon, j \in Q\}|, \quad j = 1, 2, \dots, d.$$

If $V_j \geq V_s$, then a certain correlation exists in the j th dimension, where V_s is the decision threshold. For example, if $V = 20$, then we could take $V_s = 12$.

ALGORITHM 15.8. The FINDIT algorithm.

Require: D -The Data set, $C_{minsize}$ -Minimum size of clusters, $D_{mindist}$ -Minimum difference between two resultant clusters;

- 1: Draw two samples S and M from the data set according to the Chernoff bounds {sampling phase};
- 2: Let $\epsilon_i \Leftarrow \frac{i}{100}R$ for $i = 1, 2, \dots, 25$;
- 3: Let MC_i be the set of medoid clusters for ϵ_i ;
- 4: Set the optimal $\epsilon \Leftarrow 0$, $MC \Leftarrow \Phi$;
- 5: **for** $i = 1$ to 25 **do**
- 6: Determine correlated dimensions for every medoid in M ;
- 7: Assign every point in S to the nearest medoid in M ;

```

8: Merge similar medoids in  $M$  to make medoid cluster set  $MC_i$ ;
9: Refine medoid clusters in  $MC_i$  {cluster-forming phase};
10: end for
11: Let  $i_0$  be the subscript such that  $Soundness(MC_{i_0}) = \max_{1 \leq i \leq 25} Soundness(MC_i)$ ,
    and then set  $\epsilon \Leftarrow \epsilon_{i_0}$ ,  $MC \Leftarrow MC_{i_0}$ ;
12: Set  $min\_dod \Leftarrow d$ ,  $nearest\_mc \Leftarrow \Phi$ ;
13: for all  $\mathbf{x} \in D$  do
14:   for all  $A \in MC_{i_0}$  do
15:     for all  $\mathbf{z} \in A$  do
16:       if  $dod_{\epsilon_{i_0}}(\mathbf{z} \rightarrow \mathbf{x}) = 0$  and  $dod_{\epsilon_{i_0}}(\mathbf{z}, \mathbf{x}) < min\_dod$  then
17:          $min\_dod \Leftarrow dod_{\epsilon_{i_0}}(\mathbf{z}, \mathbf{x})$ ,  $nearest\_mc \Leftarrow A$ ;
18:       end if
19:     end for
20:   end for
21:   if  $nearest\_mc \neq \Phi$  then
22:     Assign  $\mathbf{x}$  to  $nearest\_mc$ ;
23:   else
24:     Assign  $\mathbf{x}$  to outlier cluster;
25:   end if
26: end for

```

A point \mathbf{x} in S is assigned to the nearest \mathbf{z} in M satisfying

$$dod_{\epsilon}(\mathbf{z} \rightarrow \mathbf{x}) = |Q_{\mathbf{z}}| - |\{j : |\mathbf{z}_j - \mathbf{x}_j| \leq \epsilon, j \in Q\}| = 0,$$

where $Q_{\mathbf{z}}$ is the set of correlated dimensions of \mathbf{z} . If there is more than one such medoid, the point \mathbf{x} will be assigned to the medoid which has the largest number of correlated dimensions.

In FINDIT, a hierarchical clustering algorithm is employed to cluster the medoids. The distance between two medoid clusters A and B is defined to be the weighted average between the medoids belonging to them, i.e.,

$$dod_{\epsilon}(A, B) = \frac{\sum_{\mathbf{z}_i \in A, \mathbf{z}_j \in B} |\mathbf{z}_i| \cdot |\mathbf{z}_j| \cdot dod_{\epsilon}(\mathbf{z}_i, \mathbf{z}_j)}{\left(\sum_{\mathbf{z}_i \in A} |\mathbf{z}_i| \right) \cdot \left(\sum_{\mathbf{z}_j \in B} |\mathbf{z}_j| \right)},$$

where $|\mathbf{z}_i|$ is the number of points assigned to \mathbf{z}_i , $|\mathbf{z}_j|$ is defined similarly, $dod_{\epsilon}(\mathbf{z}_i, \mathbf{z}_j)$ is defined as

$$dod_{\epsilon}(\mathbf{z}_i, \mathbf{z}_j) = \max\{|Q_{\mathbf{z}_i}|, |Q_{\mathbf{z}_j}|\} - |\{l : |\mathbf{z}_{il} - \mathbf{z}_{jl}| \leq \epsilon, l \in Q_{\mathbf{z}_i} \cap Q_{\mathbf{z}_j}\}|,$$

and \mathbf{z}_{il} and \mathbf{z}_{jl} are the values of the l th dimension of \mathbf{z}_i and \mathbf{z}_j , respectively.

To evaluate a medoid clustering, a criterion called soundness is defined for each MC_i , the set of medoid clusters for ϵ_i , by the formula

$$Soundness(MC_i) = \sum_{A \in MC_i} |A| \cdot |KDA|,$$

where KD_A is the set of key dimensions or correlated dimensions of the medoid cluster A , which is defined by

$$KD_A = \left\{ j : \frac{\sum_{\mathbf{z} \in A} \delta_j(\mathbf{z}) \cdot |\mathbf{z}|}{\sum_{\mathbf{z} \in A} |\mathbf{z}|} > \delta_0 \right\},$$

where δ_0 is a threshold that could be taken from 0.9 to 1, $|\mathbf{z}|$ is the number of points assigned to the medoid \mathbf{z} , and $\delta_j(\mathbf{z}) = 1$ if j is a correlated dimension of \mathbf{z} ; otherwise it is 0.

In the data-assigning phase, the data points in the original data set are assigned to the nearest medoid cluster in the optimal medoid clustering found in the cluster-forming phase or put into the outlier cluster. The FINDIT algorithm is sketched in Algorithm 15.8.

15.6 MAFIA

MAFIA (Merging of Adaptive Finite Intervals) (Nagesh et al., 2000; Goil et al., 1999) is a parallel subspace clustering algorithm using adaptive computation of the finite intervals in each dimension that are merged to explore clusters embedded in subspaces of a high-dimensional data set. It is also a density- and grid-based clustering algorithm.

The MAFIA algorithm uses adaptive grids to partition the data space into small units and then merges the dense units to form clusters. To compute the adaptive grids, it first constructs a histogram by partitioning the data space into a number of nonoverlapping regions and mapping the data points to each cell. The procedure of computing adaptive grids is listed in Algorithm 15.9.

ALGORITHM 15.9. Procedure of adaptive grids computation in MAFIA the algorithm.

Require: D_i – Domain of i th attribute; N – Total number of data points in the data set; a – Size of the generic interval;

- 1: **for** $i = 1$ to d **do**
- 2: Divide D_i into intervals of some small size x ;
- 3: Compute the histogram for each interval in the i th dimension, and set the value of the window to the maximum in the window;
- 4: From left to right, merge two adjacent intervals if they are within a threshold β ;
- 5: **if** number of bins == 1 **then**
- 6: Divide the i th dimension into a fixed number of equal intervals and set a threshold β' for it;
- 7: **end if**
- 8: **end for**

MAFIA is a parallel clustering algorithm. In its implementation, MAFIA introduces both data parallelism and task parallelism. The main properties of this algorithm are listed in Table 15.2.

Table 15.2. Description of the MAFIA algorithm, where c is a constant, k is the highest dimensionality of any dense cells in the data set, p is the number of processors, N is the number of data points in the data set, B is the number of records in the memory buffer on each processor, and γ is the I/O access time for a block of B records from the local disk.

Clustering Data Type	Numerical data
Cluster Shape	Centered cluster
Time Complexity	$O(c^k + \frac{N}{pB}k\gamma + \alpha Spk)$
Algorithm Type	Parallel, hierarchical

15.7 DOC

DOC (Density-based Optimal projective Clustering) (Procopiuc et al., 2002) is a Monte Carlo-based algorithm that computes a good approximation of an optimal projective cluster. The DOC algorithm can identify arbitrarily shaped clusters from the data sets, for example, image data.

In the DOC algorithm, a projective cluster of width ω is defined as a pair (C, P) , where C is a subset of data points of the original data set, and P is a subset of dimensions associated with C such that

1. C is α -dense, i.e., $|C| \geq \alpha|S|$, where $\alpha \in [0, 1]$ and S is the original data set;
2. $\forall j \in P$, $\max_{x \in C} x_j - \min_{y \in C} y_j \leq \omega$, where $\omega \geq 0$;
3. $\forall j \in P^c = \{1, 2, \dots, d\} \setminus P$, $\max_{x \in C} x_j - \min_{y \in C} y_j > \omega$, where $\omega \geq 0$.

Let $\mu : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ be a function such that $\mu(0, 0) = 0$ and μ is monotonically increasing in each argument. Then the quality of a projective cluster (C, P) is defined to be $\mu(|C|, |P|)$. For any $0 \leq \beta < 1$, the function μ is said to be β -balanced if $\mu(x, y) = \mu(\beta x, y + 1)$ for all $x > 0, y \geq 0$. One choice of such a function is $\mu(x, y) = x(\frac{1}{\beta})^y (0 < \beta < 1)$.

A projective cluster (C, P) is said to be μ -optimal if it maximizes μ over \mathcal{P}_α , where \mathcal{P}_α is the set of all α -dense projective clusters of width at most ω from D . Monotonicity is required since we want the projective cluster (C, P) so that C and P are maximal. The β -balanced condition is used to specify the tradeoff between the number of data points and the number of dimensions in a cluster.

ALGORITHM 15.10. The DOC algorithm for approximating an optimal projective cluster.

Require: D -Data set; α , β , and ω ;

- 1: Let $r \leftarrow \frac{\log(2d)}{-\log(2\beta)}$, $m \leftarrow (\frac{2}{\alpha})^r \ln 4$;
- 2: **for** $i = 1$ to $\frac{2}{\alpha}$ **do**
- 3: Choose $x \in D$ uniformly at random;
- 4: **for** $j = 1$ to m **do**
- 5: Choose $X \subseteq D$ of size r uniformly at random;

```

6:       $P \Leftarrow \{j : |\mathbf{y}_j - \mathbf{x}_j| \leq \omega, \forall \mathbf{y} \in X\};$ 
7:       $C \Leftarrow D \cap B_{\mathbf{x}, P};$ 
8:      if  $|C| < \alpha |D|$  then
9:           $(C, P) \Leftarrow (\Phi, \Phi);$ 
10:         end if
11:     end for
12: end for
13: Return cluster  $(C_0, P_0)$  that maximizes  $\mu(|C|, |P|)$  over all computed clusters  $(C, P)$ .

```

Let (C, P) be a projective cluster. An intuitive geometric objective is an axis-parallelled box given as

$$B_{C, P} = [l_1, h_1] \times [l_2, h_2] \times \cdots \times [l_d, h_d],$$

where l_j and h_j ($j = 1, 2, \dots, d$) are defined as

$$\begin{aligned} l_j &= \begin{cases} -\infty & \text{if } j \notin P, \\ \min_{\mathbf{x} \in C} \mathbf{x}_j & \text{if } j \in P, \end{cases} \\ h_j &= \begin{cases} \infty & \text{if } j \notin P, \\ \max_{\mathbf{x} \in C} \mathbf{x}_j & \text{if } j \in P. \end{cases} \end{aligned}$$

From the definition of $B_{C, P}$, a point $\mathbf{x} \in C$ if and only if \mathbf{x} is contained in $B_{C, P}$. Let \mathbf{x} be a data point and let $B_{\mathbf{x}, P}$ be defined as

$$B_{\mathbf{x}, P} = [l_1, h_1] \times [l_2, h_2] \times \cdots \times [l_d, h_d],$$

where l_j and h_j ($j = 1, 2, \dots, d$) are defined as

$$\begin{aligned} l_j &= \begin{cases} -\infty & \text{if } j \notin P, \\ \mathbf{x}_j - \omega & \text{if } j \in P, \end{cases} \\ h_j &= \begin{cases} \infty & \text{if } j \notin P, \\ \mathbf{x}_j + \omega & \text{if } j \in P. \end{cases} \end{aligned}$$

Then for any $\mathbf{x} \in C$, we have $B_{C, P} \subseteq B_{\mathbf{x}, P}$. The Monte Carlo algorithm for approximating a projective cluster is described in Algorithm 15.10. It has been shown that the Monte Carlo algorithm works correctly (Procopiuc et al., 2002). The total running time of the Monte Carlo algorithm described above is $O(nd^c)$, where $c = \frac{\log \alpha - \log 2}{\log(2\beta)}$.

There are three parameters in the DOC algorithm, i.e., α , β , and ω . In most experiments reported in (Procopiuc et al., 2002), these parameters are set as $\alpha = 0.1$, $\beta = 0.25$, and $\omega = 15$. Since the parameter ω controls the width of the box, it is hard to choose the right value for ω if we do not have much information about the data set. In order to choose a good value for ω , a heuristic method has been proposed in (Procopiuc et al., 2002) as follows. For a data point \mathbf{x}_i , let \mathbf{y}_i be its nearest neighbor. Then let ω be defined as

$$\omega = r \sum_{i=1}^n \frac{w_i}{n},$$

where r is a small constant, and w_i ($i = 1, 2, \dots, n$) are given by

$$w_i = \sum_{j=1}^d \frac{|\mathbf{x}_{ij} - \mathbf{y}_{ij}|}{d}.$$

15.8 CLTree

CLTree (CLustering based on decision Trees) (Liu et al., 2000) is a clustering algorithm for numerical data based on a supervised learning technique called decision tree construction. The CLTree algorithm is able to find clusters in the full dimension space as well as in subspaces. The resulting clusters found by CLTree are described in terms of hyperrectangle regions. The CLTree algorithm is able to separate outliers from real clusters effectively, since it naturally identifies sparse and dense regions.

The basic idea behind CLTree is to regard each data point in the original data set as having a class Y and to assume that the data space is uniformly distributed with another type of points given class N . The main goal of CLTree is to partition the data space into data (dense) regions and empty (sparse) regions. The CLTree algorithm consists of two steps. The first step is to construct a cluster tree using a modified decision tree algorithm; the second step is to prune the cluster tree interactively. The final clusters are described as a list of hyperrectangle regions.

Decision tree construction is a technique for classification. A decision tree has two types of nodes: decision nodes, which specify some test on a single attribute, and leaf nodes, which indicate the class. Since the CLTree algorithm is designed for numerical data, the tree algorithm performs a binary split, i.e., the current space is cut into two parts. For details about the decision tree construction algorithm and its improvements, readers are referred to (Quinlan, 1993), (Shafer et al., 1996), (Gehrke et al., 1998), and (Gehrke et al., 1999).

During the decision tree construction, a different number of N points is added to each node, but not physically. In order to separate cluster regions and noncluster regions, the number of N points is added according to some rules. That is, if the number of N points inherited from the parent node of E is less than the number of Y points in E , then the number of N points for E is increased to the number of Y points in E ; otherwise, the number of inherited N points is used for E .

In the CLTree algorithm, the decision tree algorithm is modified, since the N points are not physically added to the data. The first modification is to compute the number of N points in each region based on the area of the region when a cut is made, since the N points are assumed to be distributed uniformly in the current space. For example, a node E has 25 Y points and 25 N points and a cut P cuts E into two regions E_1 and E_2 in the ratio 2:3, i.e., the area of E_1 is 40% of the area of E and the area of E_2 is 60% of the area of E . Then the number of N points in E_1 is $0.4 \times 25 = 10$ and the number of N points in E_2 is $25 - 10 = 15$. The second modification is to evaluate on both sides of data points in order to find the best cuts.

Also, a new criterion is used in order to find the best cuts, since the gain criterion has problems in clustering, such as the cut with best information gain tends to cut into clusters. To do this, the relative density of a region is defined. Let E be a region. Then the relative density of E is $\frac{E.Y}{E.N}$, where $E.Y$ is the number of Y points in E and $E.N$ is defined similarly.

The basic idea of the new criterion is described as follows. For each dimension j of a node E , let d_{j_cut1} be the first cut found by the gain criterion, and let the two regions separated by d_{j_cut1} along dimension j be denoted by L_j and H_j , where L_j has a lower relative density than H_j ; then we have $E = L_j \cup H_j$. Let d_{j_cut2} be the cut of L_j found using the gain criterion. Suppose the region L_j is cut into L_j^1 and H_j^1 by d_{j_cut2} , where L_j^1 has a lower relative density than H_j^1 ; then we have $L_j = L_j^1 \cup H_j^1$. If H_j^1 is the region between d_{j_cut1} and d_{j_cut2} , then we stop and choose d_{j_cut2} as a best cut for dimension j of E . If the region between d_{j_cut1} and d_{j_cut2} is L_j^1 , then we let d_{j_cut3} be the cut of L_j^1 found using the gain criterion and choose d_{j_cut3} as the best cut for dimension j of E .

After the decision tree has been constructed, it is pruned in order to produce meaningful clusters. CLTree provides two pruning methods: browsing and user-oriented pruning. In the browsing method, a user interface is built in order to let the user simply explore the tree to find meaningful clusters. In the user-oriented pruning method, the decision tree is pruned using two user-specified parameters min_y and min_rd : min_y is the percentage of the total number of data points in the data set that a region must contain, and min_rd specifies whether an N region (node) E should join an adjacent region F to form a bigger cluster region.

15.9 PART

PART (Projective Adaptive Resonance Theory) (Cao and Wu, 2002) is a new neural network architecture that was proposed to find projected clusters for data sets in high-dimensional spaces. The neural network architecture in PART is based on the well known ART (Adaptive Resonance Theory) developed by Carpenter and Grossberg (1987b,a, 1990). In PART, a so-called selective output signaling mechanism is provided in order to deal with the inherent sparsity in the full space of the high-dimensional data points. Under this selective output signaling mechanism, signals generated in a neural node in the input layer can be transmitted to a neural node in the clustering layer only when the signal is similar to the top-down weight between the two neural nodes; hence, with this selective output signaling mechanism, PART is able to find dimensions where subspace clusters can be found.

The basic PART architecture consists of three components: the input layer (comparison layer) F_1 , the clustering layer F_2 , and a reset mechanism (Cao and Wu, 2002). Let the nodes in the F_1 layer be denoted by v_i , $i = 1, 2, \dots, m$; the nodes in the F_2 layer be denoted by v_j , $j = m + 1, \dots, m + n$; the activation of an F_1 node v_i be denoted by x_i ; and the activation of an F_2 node v_j be denoted by x_j . Let the bottom-up weight from v_i to v_j be denoted by z_{ij} and the top-down weight from v_j to v_i be denoted by z_{ji} . In PART, the selective output signal of an F_1 node v_i to a committed F_2 node v_j is defined by

$$h_{ij} = h(x_i, z_{ij}, z_{ji}) = h_\sigma(f_1(x_i), z_{ji})l(z_{ij}), \quad (15.13)$$

where f_1 is a signal function; $h_\sigma(\cdot, \cdot)$ is defined as

$$h_\sigma(a, b) = \begin{cases} 1 & \text{if } d(a, b) \leq \sigma, \\ 0 & \text{if } d(a, b) > \sigma, \end{cases} \quad (15.14)$$

with $d(a, b)$ being a quasi-distance function; and $l(\cdot)$ is defined as

$$l(z_{ij}) = \begin{cases} 1 & \text{if } z_{ij} > \theta, \\ 0 & \text{if } z_{ij} \leq \theta, \end{cases} \quad (15.15)$$

with θ being 0 or a small number to be specified as a threshold; σ is a distance parameter.

An F_1 node v_i is said to be active to v_j if $h_{ij} = 1$ and inactive to v_j if $h_{ij} = 0$.

In PART, an F_2 node v_j is said to be a winner either if $\Gamma \neq \phi$ and $T_j = \max \Gamma$ or if $\Gamma = \phi$ and node v_j is the next noncommitted node in the F_2 layer, where Γ is a set defined as $\Gamma = \{T_k : F_2$ node v_k is committed and has not been reset on the current trial}, with T_k defined as

$$T_k = \sum_{v_i \in F_1} z_{ik} h_{ik} = \sum_{v_i \in F_1} z_{ik} h(x_i, z_{ik}, z_{ki}). \quad (15.16)$$

A winning F_2 node will become active and all other F_2 nodes will become inactive, since the F_2 layer makes a choice by a winner-take-all paradigm:

$$f_2(x_j) = \begin{cases} 1 & \text{if node } v_j \text{ is a winner,} \\ 0 & \text{otherwise.} \end{cases} \quad (15.17)$$

For the vigilance and reset mechanism of PART, if a winning (active) F_2 node v_j does not satisfy some vigilance conditions, it will be reset so that the node v_j will always be inactive during the rest of the current trial. The vigilance conditions in PART also control the degree of similarity of patterns grouped in the same cluster. A winning F_2 node v_j will be reset if and only if

$$r_j < \rho, \quad (15.18)$$

where $\rho \in \{1, 2, \dots, m\}$ is a vigilance parameter and r_j is defined as

$$r_j = \sum_i h_{ij}. \quad (15.19)$$

Therefore, the vigilance parameter ρ controls the size of the subspace dimensions, and the distance parameter σ controls the degree of similarity in the specific dimension involved. For real-world data, it is a challenge to choose the distance parameter σ .

In PART, the learning is determined by the following formula. For a committed F_2 node v_j that has passed the vigilance test, the new bottom-up weight is defined as

$$z_{ij}^{new} = \begin{cases} \frac{L}{L-1+|X|} & \text{if } F_1 \text{ node } v_i \text{ is active to } v_j, \\ 0 & \text{if } F_1 \text{ node } v_i \text{ is inactive to } v_j, \end{cases} \quad (15.20)$$

where L is a constant and $|X|$ denotes the number of elements in the set $X = \{i : h_{ij} = 1\}$, and the new top-down weight is defined as

$$z_{ji}^{new} = (1 - \alpha)z_{ji}^{old} + \alpha I_i, \quad (15.21)$$

where $0 \leq \alpha \leq 1$ is the learning rate.

For a noncommitted winner v_j , and for every F_1 node v_i , the new weights are defined as

$$z_{ij}^{new} = \frac{L}{L - 1 + m}, \quad (15.22)$$

$$z_{ji}^{new} = I_i. \quad (15.23)$$

In PART, each committed F_2 node v_j represents a subspace cluster C_j . Let D_j be the set of subspace dimensions associated with C_j . Then $i \in D_j$ if and only if $l(z_{ij}) = 1$, i.e., the set D_j is determined by $l(z_{ij})$.

PART is very effective at finding the subspaces in which clusters are embedded, but the difficulty of choosing some of its parameters restricts its application. For example, it is very difficult for users to choose an appropriate value for the distance parameter σ . If we choose a relatively small value the algorithm may not capture the similarities of two similar data points; however, if we choose a relatively large value, the algorithm may not differentiate two dissimilar data points.

15.10 SUBCAD

All the subspace clustering algorithms mentioned before are designed for clustering numerical data. SUBCAD (SUBspace clustering for high-dimensional CAategorical Data) (Gan and Wu, 2004; Gan, 2003) is a subspace clustering algorithm designed for clustering high-dimensional categorical data. In the SUBCAD algorithm, the clustering process is treated as an optimization problem. An objective function is defined in SUBCAD to measure the quality of the clustering. The main goal of SUBCAD is to find an approximation of the optimal solution based on the objective function.

Given a data set D , let Q be the set of dimensions of D , i.e., $Q = \{1, 2, \dots, d\}$, where d is the number of dimensions of D , and let $\text{Span}(Q)$ denote the full space of the data set. Then by a subspace cluster we mean a cluster C associated with a set of dimensions P such that

1. the data points in C are “similar” to each other in the subspace $\text{Span}(P)$ of $\text{Span}(Q)$ (i.e., the data points in C are compact in this subspace);
2. the data points in C are sparse in the subspace $\text{Span}(R)$, where $R = Q \setminus P$ (i.e., the data points in C are spread in this subspace).

In SUBCAD, a subspace cluster is denoted by a pair (C, P) ($P \neq \emptyset$), where P is the nonempty set of dimensions associated with C . In particular, if $P = Q$, then this cluster is formed in the whole space of the data set. Therefore, if C is a cluster with the associated set of dimensions P , then C is also a cluster in every subspace of $\text{Span}(P)$. Hence, a good subspace clustering algorithm should be able to find clusters and the maximum associated set of dimensions. Consider, for example, the data set with five six-dimensional data points given in Table 2.1. In this data set, it is obvious that $C = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ is a cluster and the maximum set of dimensions should be $P = \{1, 2, 3, 4\}$. A good subspace clustering algorithm should be able to find this cluster and the maximum set of associated dimensions P .

The objective function is defined in terms of the compactness and separation of each subspace cluster. Let (C, P) be a subspace cluster. Then the compactness $Cp(C, P)$ and the separation $Sp(C, R)$ of cluster (C, P) are defined as

$$Cp(C, P) = \frac{2 \sum_{x,y \in C} \|x - y\|_P^2}{|P||C|^2}, \quad (15.24)$$

$$Sp(C, P) = \begin{cases} \frac{2 \sum_{x,y \in C} \|x - y\|_R^2}{|R||C|^2} & \text{if } R \neq \Phi, \\ 1 & \text{if } R = \Phi, \end{cases} \quad (15.25)$$

where $|P|$ and $|R|$ denote the number of elements in the sets P and R , respectively; $\|x - y\|_P$ denotes the distance between x and y in the subspace spanned by the dimensions in P ; and $\|x - y\|_R$ denotes the distance in the subspace spanned by the dimensions in R .

Let C_1, C_2, \dots, C_k be a partition of the data set, where k is the number of clusters. Let P_j be the set of dimensions associated with cluster C_j . The objective function is defined as

$$F_{obj} = \sum_{j=1}^k (Cp(C_j, P_j) + 1 - Sp(C_j, R_j)). \quad (15.26)$$

Given the number of clusters k , the goal of SUBCAD is to partition the data set into k nonoverlapping groups such that the objective function F_{obj} defined in equation (15.26) is minimized. The algorithm finds an approximation of the optimal partition.

Let C be a cluster associated with the set P of dimensions and $T_f(C)$ be its frequency table (cf. Section 2.1). Since the square of the simple matching distance is equal to itself, we have

$$\begin{aligned} \sum_{\mathbf{x}, \mathbf{y} \in C} \|\mathbf{x} - \mathbf{y}\|_P^2 &= \sum_{\mathbf{x}, \mathbf{y} \in C} \sum_{j \in P} \delta(\mathbf{x}_j, \mathbf{y}_j)^2 \\ &= \sum_{j \in P} \sum_{\mathbf{x}, \mathbf{y} \in C} \delta(\mathbf{x}_j, \mathbf{y}_j) \\ &= \sum_{j \in P} \sum_{1 \leq r < s \leq n_j} f_{jr}(C) \cdot f_{js}(C) \\ &= \sum_{j \in P} \frac{1}{2} \left[\left(\sum_{r=1}^{n_j} f_{jr}(C) \right)^2 - \sum_{r=1}^{n_j} f_{jr}^2(C) \right] \\ &= \frac{1}{2} |P| \cdot |C|^2 - \frac{1}{2} \sum_{j \in P} \sum_{r=1}^{n_j} f_{jr}^2(C) \\ &= \frac{1}{2} |P| \cdot |C|^2 - \frac{1}{2} \sum_{j \in P} \|\mathbf{f}_j(C)\|^2, \end{aligned} \quad (15.27)$$

where $\delta(\cdot, \cdot)$ is the simple matching distance, and $\mathbf{f}_j(C)$ is defined in equation (2.3).

Thus from equation (15.27), we obtain the following simplified formulas of compactness and separation:

$$\text{Cp}(C, P) = 1 - \frac{\sum_{j \in P} \|\mathbf{f}_j(C)\|^2}{|P||C|^2}, \quad (15.28)$$

$$\text{Sp}(C, R) = \begin{cases} 1 - \frac{\sum_{j \in R} \|\mathbf{f}_j(C)\|^2}{|R||C|^2} & \text{if } R \neq \Phi, \\ 1 & \text{if } R = \Phi. \end{cases} \quad (15.29)$$

The SUBCAD algorithm consists of two phases: the initialization phase and the optimization phase. In the initialization phase, an initial partition is made. Good initial partition leads to fast convergence of the algorithm, while bad initial partition may make the algorithm converge very slowly. Some initialization methods have been proposed in the literature of clustering, such as the cluster-based method (Bradley and Fayyad, 1998) and the kd -tree-based method (Pelleg and Moore, 1999). A sketch of the algorithm is described in Algorithm 15.11.

ALGORITHM 15.11. The SUBCAD algorithm.

Require: D - Data Set, k - Number of Clusters;

Ensure: $2 \leq k \leq |D|$;

```

1: if  $D$  is a large data set then
2:   Draw a sample from  $D$ ;
3: end if
4: Compute the proximity matrix from the whole data set or the sample;
5: Pick  $k$  most dissimilar data points as seeds;
6: Assign the remaining data points to the nearest seed;
7: repeat
8:   for  $i = 1$  to  $|D|$  do
9:     Let  $(C_l, P_l)$  be the subspace cluster that contains  $\mathbf{x}_i$ ;
10:    for  $m = 1, m \neq l$  to  $k$  do
11:      if inequality (15.35) is true then
12:        Move  $\mathbf{x}$  from  $C_l$  to  $C_m$ ;
13:        Update subspaces  $P_l$  and  $P_m$ ;
14:      end if
15:    end for
16:  end for
17: until No further change in the cluster memberships;
18: Output results.

```

Initialization

In the initialization phase, the k most dissimilar data points are picked as seeds, where k is the number of clusters, and then the remaining data points are assigned to the nearest

seed. Let D be a data set and k be a positive integer such that $k \leq |D|$. We say that $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k \in D$ are the k most dissimilar data points of the data set D if the following condition is satisfied:

$$\min_{1 \leq r < s \leq k} \|\mathbf{x}_r - \mathbf{x}_s\| = \max_{E \in \mathcal{F}} \min_{\mathbf{x}, \mathbf{y} \in E} \|\mathbf{x} - \mathbf{y}\|, \quad (15.30)$$

where \mathcal{F} is the class that contains all subsets E of D such that $|E| = k$, i.e.,

$$\mathcal{F} = \{E : E \subseteq D, |E| = k\},$$

and $\|\mathbf{x} - \mathbf{y}\|$ is the distance between \mathbf{x} and \mathbf{y} .

Let $X(k, D)$ ($k \leq |D|$) denote the set of the k most dissimilar data points of the data set D . Since there is a total of $\binom{n}{k}$ elements in \mathcal{F} , when n is large, it is impractical to enumerate all sets in \mathcal{F} to find the set of the k most dissimilar data points. Thus an approximation algorithm is employed to find a set of k data points that is near the set of the k most dissimilar data points. The basic idea is to choose k initial data points and then continuously replace the bad data points with good ones until no further changes are necessary.

More specifically, let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set or a sample from a data set. First, let $X(k, D)$ be $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$, and let \mathbf{x}_r and \mathbf{x}_s ($1 \leq r < s \leq k$) be such that

$$\|\mathbf{x}_r - \mathbf{x}_s\| = \min_{\mathbf{x}, \mathbf{y} \in X(k, D)} \|\mathbf{x} - \mathbf{y}\|. \quad (15.31)$$

Secondly, for each of the data points $\mathbf{x} \in D \setminus X(k, D)$, let

$$S_r = \min_{\mathbf{y} \in (X(k, D) \setminus \{\mathbf{x}_s\})} \|\mathbf{x} - \mathbf{y}\|, \quad (15.32)$$

$$S_s = \min_{\mathbf{y} \in (X(k, D) \setminus \{\mathbf{x}_r\})} \|\mathbf{x} - \mathbf{y}\|. \quad (15.33)$$

Then if $S_r > \|\mathbf{x}_r - \mathbf{x}_s\|$, we let $X(k, D) \cup \{\mathbf{x}\} \setminus \{\mathbf{x}_s\}$ replace $X(k, D)$; if $S_s > \|\mathbf{x}_r - \mathbf{x}_s\|$, we let $X(k, D) \cup \{\mathbf{x}\} \setminus \{\mathbf{x}_r\}$ replace $X(k, D)$.

Optimization

In the optimization phase, the data points are reassigned in order to minimize the objective function (15.26) and update the subspaces associated with those clusters whose membership has changed. Let $(C_1, P_1), (C_2, P_2), \dots, (C_k, P_k)$ be a partition of the data set D , and let \mathbf{x} be a data point in the subspace cluster (C_l, P_l) . To achieve the membership changing rules, the “exact assignment test” (Wishart, 2002) technique is used in the optimization phase. \mathbf{x} will be moved from subspace cluster (C_l, P_l) to another subspace cluster (C_m, P_m) ($m \neq l$) if the resulting cost function decreases, i.e., if the following inequality is true:

$$\begin{aligned} \sum_{i=1}^k (\text{Cp}(C_i) + 1 - \text{Sp}(C_i)) &> \sum_{i=1, i \neq l, m}^k \text{Cp}(C_i) \\ &+ \text{Cp}(C_l - \mathbf{x}) + 1 - \text{Sp}(C_l - \mathbf{x}) + \text{Cp}(C_m + \mathbf{x}) \\ &+ 1 - \text{Sp}(C_m + \mathbf{x}), \end{aligned} \quad (15.34)$$

where $C_l - \mathbf{x}$ means $C_l \setminus \{\mathbf{x}\}$ $C_m + \mathbf{x}$ means $C_m \cup \{\mathbf{x}\}$.

The inequality (15.34) is equivalent to

$$\begin{aligned} & \text{Cp}(C_l) - \text{Cp}(C_l - \mathbf{x}) - \text{Sp}(C_l) + \text{Sp}(C_l - \mathbf{x}) \\ & > -\text{Cp}(C_m) + \text{Cp}(C_m + \mathbf{x}) + \text{Sp}(C_m) - \text{Sp}(C_m + \mathbf{x}). \end{aligned} \quad (15.35)$$

Determining Subspace Dimensions

During the optimization phase of the algorithm, if a data point is moved from its current cluster to another one, then the subspace dimensions associated with the two clusters should be updated. In the SUBCAD algorithm, the set of subspace dimensions associated with each cluster is determined by an objective function.

Let (C, E) be a subspace cluster of a d -dimensional data set D . In order to determine the set P of subspace dimensions associated with C , an objective function whose domain is all the subsets of $Q = \{1, 2, \dots, d\}$ is defined as

$$F(C, E) = \text{Cp}(C, E) + 1 - \text{Sp}(C, Q \setminus E), \quad \Phi \neq E \subseteq Q, \quad (15.36)$$

where $\text{Cp}(C, E)$ and $\text{Sp}(C, Q \setminus E)$ are the compactness and separation of cluster C under the subspace dimensions set E .

Our general idea to determine the set P associated with the cluster C is to find a P such that the objective function defined in equation (15.36) is minimized. Also, from equation (15.27), if $P \neq \Phi$ or $P \neq Q$, the objective function in equation (15.36) can be written as

$$F(C, E) = 1 - \frac{\sum_{j \in E} \|\mathbf{f}_j(C)\|^2}{|E||C|^2} + \frac{\sum_{j \in Q \setminus E} \|\mathbf{f}_j(C)\|^2}{|Q \setminus E||C|^2}, \quad E \subseteq Q, \quad (15.37)$$

where $R = Q \setminus P$.

The objective function defined in (15.37) has the following properties.

Theorem 15.1 (Condition of constancy). *The objective function defined in equation (15.36) is constant for any subset E of Q if and only if*

$$\|\mathbf{f}_1(C)\| = \|\mathbf{f}_2(C)\| = \dots = \|\mathbf{f}_d(C)\|. \quad (15.38)$$

In addition, if the objective function is a constant, then it is equal to 1.

From the definition of the objective function $F(C, E)$, the proof of the above theorem is straightforward and is thus omitted. Thus, from Theorem 15.1, if the objective function is constant, then it is minimized at any subset of Q . In this case, the set of subspace dimensions associated with C is defined to be Q . If the objective function is not constant, then the set of subspace dimensions associated with C is defined to be the set $P \subseteq Q$ that minimizes the objective function. In fact, it can be shown that such a set P is unique if the objective is not constant (Gan, 2003). Hence, the following definition of subspace associated with each cluster is well defined.

Definition 15.2. Let C be a cluster. Then the set P of subspace dimensions associated with C is defined as follows:

1. If the objective function $F(C, E)$ is constant for any $E \subseteq Q$, then let $P = Q$.
2. If the objective function $F(C, E)$ is not constant, then P is defined as

$$P = \arg \max_{E \in \mathcal{E}} |E|, \quad (15.39)$$

where \mathcal{E} is defined as

$$\mathcal{E} = \{O : F(C, O) = \min_{E \in \aleph} F(C, E), O \in \aleph\} \quad (15.40)$$

and is defined as

$$\aleph = \{E : E \subset Q, E \neq \Phi, E \neq Q\}. \quad (15.41)$$

From Definition 15.2, the set P defined in equation (15.39) is nonempty. Moreover, if the objective function $F(C, E)$ is not constant, then the set P is a true subset of Q , i.e., $P \subsetneq Q$. The objective function in (15.37) has many good properties, which will be given as theorems or corollaries as follows. Readers may refer to Gan (2003) for detailed proofs.

Theorem 15.3. Let (C, P) be a subspace cluster of a d -dimensional data set D ($d > 2$), and let P be defined in equation (15.39). Then we have the following:

1. Let $r \in P$. If there exists an s ($1 \leq s \leq d$) such that $\|\mathbf{f}_s(C)\| > \|\mathbf{f}_r(C)\|$, then $s \in P$.
2. Let $r \in R$. If there exists an s ($1 \leq s \leq d$) such that $\|\mathbf{f}_s(C)\| < \|\mathbf{f}_r(C)\|$, then $s \in R$, where $R = Q \setminus P$.

Corollary 15.4 (Monotonicity). Let (C, P) be a subspace cluster of D , where P is the set of subspace dimensions defined in equation (15.39) and let $T_f(C)$ be the frequency table of C . Then for any $r \in P$ and $s \in R$ ($R = Q \setminus P$), we have

$$\|\mathbf{f}_r(C)\| \geq \|\mathbf{f}_s(C)\|. \quad (15.42)$$

Theorem 15.5. Let (C, P) be a subspace cluster of a d -dimensional data set D ($d > 2$), where P is defined in equation (15.39). Let $T_f(C)$ be the frequency table and let r, s ($1 \leq r, s \leq d$) be given so that $\|\mathbf{f}_s(C)\| = \|\mathbf{f}_r(C)\|$. Then either $r, s \in P$ or $r, s \in R$, i.e., r, s must be in the same set P or R , where $R = Q \setminus P$.

Corollary 15.6. Let (C, P) be a subspace cluster of D , where P is the set of subspace dimensions defined in equation (15.39), and let $T_f(C)$ be the frequency table of C . If the objective function $F(C, E)$ is not constant, then for any $r \in P$ and $s \in R$ ($R = Q \setminus P$), we have

$$\|\mathbf{f}_r(C)\| > \|\mathbf{f}_s(C)\|. \quad (15.43)$$

Theorem 15.7 (Uniqueness of subspace). Let $F(C, E)$ be the objective function defined in equation (15.36) and let P be the set defined in equation (15.39). If the objective function $F(C, E)$ is not constant, then the set P is unique.

Theorem 15.8 (Contiguity). Let (C, P) be a subspace cluster of D , where P is the set of subspace dimensions defined in equation (15.39). Let $T_f(C)$ be the frequency table of C , and let i_1, i_2, \dots, i_d be a combination of $1, 2, \dots, d$ such that

$$\|\mathbf{f}_{i_1}(C)\| \geq \|\mathbf{f}_{i_2}(C)\| \geq \cdots \geq \|\mathbf{f}_{i_d}(C)\|.$$

Finally, let G_s be the set of subscripts defined as

$$G_s = \{t : \|\mathbf{f}_{i_t}(C)\| \neq \|\mathbf{f}_{i_{t+1}}(C)\|, 1 \leq t \leq d - 1\}. \quad (15.44)$$

If the objective function defined in equation (15.36) is not constant, then the set of subspace dimensions P defined in equation (15.39) must be one of the P_k 's ($k = 1, 2, \dots, |G_s|$) defined as

$$P_k = \{i_t : t = 1, 2, \dots, g_k\}, \quad k = 1, 2, \dots, |G_s|, \quad (15.45)$$

where $g_1 < g_2 < \cdots < g_{|G_s|}$ are elements of G_s .

Based on Theorem 15.8, the set of subspace dimensions P for a cluster C can be found very quickly. There are $2^d - 1$ nonempty subsets of Q , so it is impractical to find an optimal P by enumerating these $2^d - 1$ subsets. Based on Theorem 15.8, a fast algorithm is possible for determining the set of subspace dimensions.

15.11 Fuzzy Subspace Clustering

In fuzzy clustering algorithms, each object has a fuzzy membership associated with each cluster indicating the degree of association of the object to the cluster. In this section we present a fuzzy subspace clustering (FSC) algorithm (Gan et al., 2006; Gan, 2006) for clustering high-dimensional data sets. In FSC each dimension has a fuzzy membership associated with each cluster indicating the degree of importance of the dimension to the cluster (see Definition 15.9). Using fuzzy techniques for subspace clustering, FSC avoids the difficulty of choosing appropriate subspace dimensions for each cluster during the iterations.

Definition 15.9 (Fuzzy dimension weight matrix). A $k \times d$ matrix $W = (w_{jh})$ is said to be a fuzzy dimension weight matrix if W satisfies the following conditions:

$$w_{jh} \in [0, 1], \quad 1 \leq j \leq k, \quad 1 \leq h \leq d, \quad (15.46a)$$

$$\sum_{h=1}^d w_{jh} = 1, \quad 1 \leq j \leq k. \quad (15.46b)$$

The set of all such fuzzy dimension weight matrices is denoted by M_{fk} , i.e.,

$$M_{fk} = \{W \in \mathbb{R}^{kd} | W \text{ satisfies equation (15.46)}\}. \quad (15.47)$$

The main idea behind the FSC algorithm is to impose weights on the distance measure of the k -means algorithm (Hartigan, 1975; Hartigan and Wong, 1979) in order to capture appropriate subspace information. To describe FSC, we first briefly describe the k -means algorithm. Let the set of k centers be denoted by $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$. Then the objective function of the k -means algorithm is

$$E = \sum_{j=1}^k \sum_{\mathbf{x} \in C_j} \|\mathbf{x} - \mathbf{z}_j\|^2, \quad (15.48)$$

where $\|\cdot\|$ is the Euclidean norm and C_j is the j th cluster. Alternatively, the objective function of k -means in equation (15.48) can be formulated as

$$E = \sum_{j=1}^k \sum_{i=1}^n u_{ji} \|\mathbf{x} - \mathbf{z}_j\|^2,$$

where $U = (u_{ji})$ is a hard k -partition (see Subsection 1.2.4) of D .

Like locally adaptive clustering (LAC) (Domeniconi et al., 2004), FSC associates with each cluster a weight vector in order to capture the subspace information of that cluster. For example, the h th dimension is associated with the j th cluster to a degree of w_{jh} or the j th cluster has fuzzy dimension weights specified by $\mathbf{w}_j = (w_{j1}, w_{j2}, \dots, w_{jd})^T$. The fuzzy dimension weights of the k clusters are represented by a fuzzy dimension weight matrix $W = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k)^T$.

Mathematically, the objective function of the algorithm is formed by imposing weights on the distance measure in equation (15.48) as

$$E_{\alpha, \epsilon}(W, Z, U) = \sum_{j=1}^k \sum_{i=1}^n u_{ji} \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh})^2 + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha, \quad (15.49)$$

where W , Z , and U are the fuzzy dimension weight matrix, the centers, and the hard k -partition of D , respectively; $\alpha \in (1, \infty)$ is a weight component or fuzzifier; and ϵ is a very small positive real number. Note that any one of W , Z , and U can be determined from the other two. The following three theorems (Theorems 15.10–15.12) show how to estimate one of W , Z , and U from the other two such that the objective function $E_{\alpha, \epsilon}(W, Z, U)$ is minimized.

From the definition of the objective function in equation (15.49), we see that there is a term $\epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha$. As you will see while analyzing the FSC algorithm, this term is introduced in the objective function in order to avoid the divide-by-zero error when calculating W given the estimates of Z and U . The parameter ϵ is specified to be a very small positive real number so that the term has very little impact on the objective function.

The following theorem is used to find a hard k -partition U given the estimates of W and Z such that the objective function $E_{\alpha, \epsilon}(W, Z, U)$ defined in equation (15.49) is minimized.

Theorem 15.10. *Given an estimate W^* of W and an estimate Z^* of Z , then a hard k -partition $U = (u_{ji})$ minimizes the objective function $E_{\alpha, \epsilon}(W^*, Z^*, U)$ if it satisfies the*

condition

$$u_{ji} = 1 \text{ implies } j \in \left\{ r \in \{1, 2, \dots, k\} \mid r = \arg \min_{1 \leq l \leq k} d_{li} \right\}, \quad (15.50)$$

where $d_{li} = \sum_{h=1}^d (w_{lh}^*)^\alpha (x_{ih} - z_{jh}^*)^2$. The corresponding clusters C_j ($j = 1, 2, \dots, k$) are formulated as

$$C_j = \{\mathbf{x}_i \in D \mid u_{ij} = 1, 1 \leq i \leq n\}. \quad (15.51)$$

Proof. We rearrange the objective function $E_{\alpha,\epsilon}(W^*, Z^*, U)$ as

$$\begin{aligned} E_{\alpha,\epsilon}(W^*, Z^*, U) &= \sum_{i=1}^n \sum_{j=1}^k u_{ji} \sum_{h=1}^d (w_{jh}^*)^\alpha (x_{ih} - z_{jh}^*)^2 + \epsilon \sum_{j=1}^k \sum_{h=1}^d (w_{jh}^*)^\alpha \\ &= \sum_{i=1}^n \sum_{j=1}^k u_{ji} d_{ji} + \epsilon \sum_{j=1}^k \sum_{h=1}^d (w_{jh}^*)^\alpha, \end{aligned}$$

where $d_{ji} = \sum_{h=1}^d (w_{jh}^*)^\alpha (x_{ih} - z_{jh}^*)^2$. Since $\sum_{j=1}^k u_{ji} d_{ji}$ ($i = 1, 2, \dots, n$) are mutually uncorrelated and $\epsilon \sum_{j=1}^k \sum_{h=1}^d (w_{jh}^*)^\alpha$ is fixed, $E_{\alpha,\epsilon}(W^*, Z^*, U)$ is minimized if each term $\sum_{j=1}^k u_{ji} d_{ji}$ is minimized for $i = 1, 2, \dots, n$. Note that only one of $u_{1i}, u_{2i}, \dots, u_{ki}$ is 1 and all others are zero. From this it follows immediately that $\sum_{j=1}^k u_{ji} d_{ji}$ is minimized if u_{ji} satisfies equation (15.50).

Equation (15.51) follows from the fact that every point is assigned to its nearest center in terms of the weighted distance. This proves the theorem. \square

From Theorem 15.10, we see that the objective function $E_{\alpha,\epsilon}(W, Z, U)$ of the FSC algorithm can be formulated as

$$\begin{aligned} &E_{\alpha,\epsilon}(W, Z, U) \\ &= \sum_{i=1}^n \min \left\{ \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh})^2 \mid j = 1, 2, \dots, k \right\} + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha, \quad (15.52) \end{aligned}$$

since only one of $u_{1i}, u_{2i}, \dots, u_{ki}$ is equal to 1 and the others are equal to zero.

Note that the hard k -partition U calculated from equation (15.50) is not unique, since a data point can be assigned to any center to which it has the nearest distance. A particular choice of U must be made when implementing the FSC algorithm.

The following theorem is used to find the cluster centers Z given the estimates of W and U such that the objective function $E_{\alpha,\epsilon}(W, Z, U)$ defined in equation (15.49) is minimized.

Theorem 15.11. Given an estimate W^* of W and an estimate U^* of U , then the objective function $E_{\alpha,\epsilon}(W^*, Z, U^*)$ is minimized if $Z = (z_{jh})$ is calculated as

$$z_{jh} = \frac{\sum_{i=1}^n u_{ji}^* x_{ih}}{\sum_{i=1}^n u_{ji}^*}, \quad 1 \leq j \leq k, 1 \leq h \leq d. \quad (15.53)$$

Proof. To prove this theorem, we take partial derivatives of $E_{\alpha,\epsilon}(W^*, Z, U^*)$ with respect to z_{jh} s, set them to zero, and solve the resulting system. That is,

$$\frac{\partial E_{\alpha,\epsilon}(W^*, Z, U^*)}{\partial z_{jh}} = \sum_{i=1}^n -2u_{ji}^*(w_{jh}^*)^\alpha(x_{ih} - z_{jh}) = 0, \quad 1 \leq j \leq k, 1 \leq h \leq d.$$

This gives

$$z_{jh} = \frac{\sum_{i=1}^n u_{ji}^*(w_{jh}^*)^\alpha x_{ih}}{\sum_{i=1}^n u_{ji}^*(w_{jh}^*)^\alpha} = \frac{\sum_{i=1}^n u_{ji}^* x_{ih}}{\sum_{i=1}^n u_{ji}^*}, \quad 1 \leq j \leq k, 1 \leq h \leq d. \quad (15.54)$$

This proves the theorem. \square

The following theorem is used to find the fuzzy dimension weight matrix W given the estimates of Z and U such that the objective function $E_{\alpha,\epsilon}(W, Z, U)$ is minimized.

Theorem 15.12. *Given an estimate Z^* of Z and an estimate U^* of U , then the objective function $E_{\alpha,\epsilon}(W, Z^*, U^*)$ is minimized if $W = (w_{jh})$ is calculated as*

$$w_{jh} = \frac{1}{\sum_{l=1}^d \left[\frac{\sum_{i=1}^n u_{ji}^*(x_{ih} - z_{jh}^*)^2 + \epsilon}{\sum_{i=1}^n u_{ji}^*(x_{il} - z_{jl}^*)^2 + \epsilon} \right]^{\frac{1}{\alpha-1}}}, \quad 1 \leq j \leq k, 1 \leq h \leq d. \quad (15.55)$$

Proof. To prove this theorem, we use the method of Lagrange multipliers. To do this, let us first write the Lagrangian function as

$$F(W, Z^*, U^*, \Lambda) = E_{\alpha,\epsilon}(W, Z^*, U^*) - \sum_{j=1}^k \lambda_j \left(\sum_{h=1}^d w_{jh} - 1 \right).$$

By taking partial derivatives with respect to W_{jh} , we have

$$\begin{aligned} \frac{\partial F(W, Z^*, U^*, \Lambda)}{\partial w_{jh}} &= \sum_{i=1}^n \alpha u_{ji}^* w_{jh}^{\alpha-1} (x_{ih} - z_{jh}^*)^2 + \epsilon \alpha w_{jh}^{\alpha-1} - \lambda_j \\ &= \alpha w_{jh}^{\alpha-1} \left(\sum_{i=1}^n u_{ji}^* (x_{ih} - z_{jh}^*)^2 + \epsilon \right) - \lambda_j \\ &= 0 \end{aligned} \quad (15.56)$$

for $1 \leq j \leq k, 1 \leq h \leq d$ and

$$\frac{\partial F(W, Z^*, U^*, \Lambda)}{\partial \lambda_j} = \sum_{h=1}^d w_{jh} - 1 = 0 \quad (15.57)$$

for $1 \leq j \leq k$.

From equation (15.56), we have

$$w_{jh} = \left[\frac{\lambda_j}{\alpha \left(\sum_{i=1}^n u_{ji}^* (x_{ih} - z_{jh}^*)^2 + \epsilon \right)} \right]^{\frac{1}{\alpha-1}}, \quad 1 \leq j \leq k, 1 \leq h \leq d. \quad (15.58)$$

Combining equation (15.57) and equation (15.58) gives

$$\sum_{h=1}^d \left[\frac{\lambda_j}{\alpha \left(\sum_{i=1}^n u_{ji}^* (x_{ih} - z_{jh}^*)^2 + \epsilon \right)} \right]^{\frac{1}{\alpha-1}} = 1, \quad 1 \leq j \leq k,$$

or

$$\lambda_j^{\frac{1}{\alpha-1}} = \frac{1}{\sum_{h=1}^d \left[\frac{1}{\alpha \left(\sum_{i=1}^n u_{ji}^* (x_{ih} - z_{jh}^*)^2 + \epsilon \right)} \right]^{\frac{1}{\alpha-1}}}, \quad 1 \leq j \leq k. \quad (15.59)$$

Plugging λ_j from equation (15.59) into equation (15.58), we have

$$w_{jh} = \frac{1}{\sum_{l=1}^d \left[\frac{\sum_{i=1}^n u_{ji}^* (x_{ih} - z_{jh}^*)^2 + \epsilon}{\sum_{i=1}^n u_{jl}^* (x_{il} - z_{jl}^*)^2 + \epsilon} \right]^{\frac{1}{\alpha-1}}}, \quad 1 \leq j \leq k, 1 \leq h \leq d.$$

This proves the theorem. \square

ALGORITHM 15.12. The pseudocode of the FSC algorithm.

Require: D - the data set, k - the number of clusters, and α - the fuzzifier;

- 1: Initialize Z by choosing k points from D randomly;
- 2: Initialize W with $w_{jh} = \frac{1}{d}$ ($1 \leq j \leq k, 1 \leq h \leq d$);
- 3: Estimate U from initial values of W and Z according to equation (15.50);
- 4: Let $error = 1$ and $Obj = E_{\alpha, \epsilon}(W, Z)$;
- 5: **while** $error > 0$ **do**
- 6: Update Z according to Theorem 15.11;

-
- 7: Update W according to Theorem 15.12;
 8: Update U according to Theorem 15.10;
 9: Calculate $NewObj = E_{\alpha,\epsilon}(W, Z)$;
 10: Let $error = |NewObj - Obj|$, and then $Obj \Leftarrow NewObj$;
11: end while
 12: Output W, Z , and U .

We see from equation (15.53) and equation (15.55) that FSC is very similar to the fuzzy k -means algorithm (Huang and Ng, 1999) in terms of the way they update centers and fuzzy memberships. Like the fuzzy k -means algorithm, FSC is implemented recursively (see Algorithm 15.12). FSC starts with initial estimates of Z , W , and the partition U calculated from Z and W , and then repeats estimating the centers Z given the estimates of W and U , estimating the fuzzy dimension weight matrix W given the estimates of Z and U , and estimating the partition U given the estimates of W and Z , until it converges.

15.12 Mean Shift for Subspace Clustering

This section presents a novel approach, the mean shift for subspace clustering (MSSC) algorithm (Gan, 2006), to identifying subspace clusters. First, we introduce the relationship among several clustering algorithms: the mean shift algorithm, the maximum-entropy clustering (MEC) algorithm, the FSC algorithm, the k -means algorithm, and the MSSC algorithm. Then we briefly introduce the MSSC algorithm. Finally, we examine some special cases of the MSSC algorithm and compute the critical value for β , a key parameter required by the algorithm, when the first phase transition occurs.

Figure 15.1 shows the relationship between the mean shift algorithm and its derivatives. The k -means algorithm is a limit case of the MEC algorithm (Rose et al., 1990) which, in turn, is a special case of the mean shift algorithm (Cheng, 1995). The MSSC algorithm is an extension of the MEC algorithm for subspace clustering.

The main idea behind the MSSC algorithm is to impose weights on the distance measure of the MEC algorithm (Rose et al., 1990) in order to capture appropriate subspace information. Readers are referred to Section 9.7 for a brief review of the MEC algorithm. Given a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ and a set of centers $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$, recall that the free energy defined in the MEC algorithm is defined as

$$F = -\frac{1}{\beta} \sum_{\mathbf{x} \in D} \ln \left(\sum_{j=1}^k e^{-\beta \|\mathbf{x} - \mathbf{z}_j\|^2} \right),$$

where β is a parameter called the Lagrange multiplier.

Like the FSC algorithm (see Section 15.11), the MSSC algorithm associates with each cluster a weight vector in order to capture the subspace information of that cluster. In particular, the h th dimension is associated with the j th cluster C_j to a degree of w_{jh}

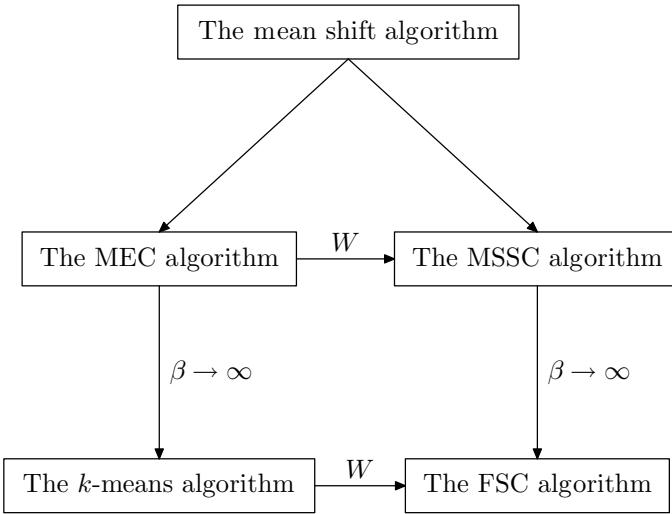


Figure 15.1. The relationship between the mean shift algorithm and its derivatives.

or the j th cluster has fuzzy dimension weights specified by $\mathbf{w}_j = (w_{j1}, w_{j2}, \dots, w_{jd})^T$. Mathematically, the objective function of the MSSC algorithm is formulated by imposing weights on the distance measure in the free energy as

$$\begin{aligned}
 & F_{\alpha, \beta, \epsilon}(W, Z) \\
 &= -\frac{1}{\beta} \sum_{i=1}^n \ln \left[\sum_{j=1}^k \exp \left(-\beta \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh})^2 \right) \right] + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha \\
 &= -\frac{1}{\beta} \sum_{i=1}^n \ln \left(\sum_{j=1}^k e^{-\beta d_{ji}(W, Z)} \right) + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha, \tag{15.60}
 \end{aligned}$$

where $\alpha \in (1, \infty)$ is the fuzzifier controlling the fuzzy dimension weights; β is a parameter ranging from zero to infinity, i.e., $\beta \in (0, \infty)$; ϵ is a very small positive number; $W = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k)^T$ is the fuzzy dimension weight matrix; $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ is the set of centers; and $d_{ji}(W, Z)$ is defined as

$$d_{ji}(W, Z) = \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh})^2, \quad 1 \leq i \leq n, 1 \leq j \leq k. \tag{15.61}$$

The following theorem is used to find the set of centers Z given the estimate of W such that the objective function $F_{\alpha, \beta, \epsilon}(W, Z)$ is minimized.

Theorem 15.13. Let $\gamma : \mathbb{R}^{kd} \rightarrow \mathbb{R}$ be defined as

$$\gamma(Z) = F_{\alpha, \beta, \epsilon}(W, Z), \quad (15.62)$$

where $W \in M_{fk}$ is fixed. Assume $w_{jh} > 0$ for all $1 \leq j \leq k$ and $1 \leq h \leq d$. Then the set of centers Z^* optimizes the function γ if Z^* satisfies the implicit equation

$$z_{jh}^* = \frac{\sum_{i=1}^n x_{ih} P(\mathbf{x}_i \in C_j)}{\sum_{i=1}^n P(\mathbf{x}_i \in C_j)}, \quad 1 \leq j \leq k, \quad 1 \leq h \leq d, \quad (15.63a)$$

or

$$\mathbf{z}_j^* = \frac{\sum_{i=1}^n \mathbf{x}_i P(\mathbf{x}_i \in C_j)}{\sum_{i=1}^n P(\mathbf{x}_i \in C_j)} = \frac{\sum_{i=1}^n \mathbf{x}_i u_{ji}}{\sum_{i=1}^n u_{ji}}, \quad 1 \leq j \leq k, \quad (15.63b)$$

where $P(\mathbf{x}_i \in C_j) = u_{ji}$ is the fuzzy membership of \mathbf{x}_i associated with cluster C_j and is defined as

$$P(\mathbf{x}_i \in C_j) = u_{ji} = \frac{e^{-\beta d_{ji}(W, Z^*)}}{\sum_{l=1}^k e^{-\beta d_{li}(W, Z^*)}}, \quad 1 \leq i \leq n, \quad 1 \leq j \leq k, \quad (15.64)$$

with $d_{ji}(\cdot, \cdot)$ defined as in equation (15.61). In addition, for $\beta = 0$, the set of centers Z^* satisfying equation (15.63) is the global minimum for $F_{\alpha, 0, \epsilon}$.

Proof. Since minimization of γ over \mathbb{R}^{kd} is an unconstrained problem, the necessity of equation (15.63) follows by requiring $\nabla_Z \gamma$ to vanish. Equivalently, the directional derivative $\frac{\partial \gamma(Z^* + tZ)}{\partial Z}$ vanishes at Z^* in arbitrary directions $Z \in \mathbb{R}^{kd}$, $Z \neq 0$. Let $t \in \mathbb{R}$ and define

$$h(t) = \gamma(Z^* + tZ) = -\frac{1}{\beta} \sum_{i=1}^n \ln \left(\sum_{j=1}^k e^{-\beta d_{ji}(W, Z^* + tZ)} \right) + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha,$$

where $d_{ji}(\cdot, \cdot)$ is defined in equation (15.61). Rearranging the terms in d_{ji} in the above equation leads to

$$h(t) = -\frac{1}{\beta} \sum_{i=1}^n \ln \left(\sum_{j=1}^k e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})} \right) + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha,$$

where a_{ji} , b_{ji} , and c_{ji} are defined as

$$\begin{aligned} a_{ji} &= \sum_{h=1}^d w_{jh}^\alpha z_{jh}^2, \\ b_{ji} &= -2 \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh}^*) z_{jh}, \\ c_{ji} &= \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh}^*)^2. \end{aligned}$$

Taking the derivative of $h(t)$, we have

$$h'(t) = \frac{dh(t)}{dt} = \sum_{i=1}^n \frac{\sum_{j=1}^k (2a_{ji}t + b_{ji}) e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})}}{\sum_{j=1}^k e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})}}, \quad (15.65)$$

and

$$\begin{aligned} h'(0) &= \sum_{i=1}^n \frac{\sum_{j=1}^k b_{ji} e^{-\beta c_{ji}}}{\sum_{j=1}^k e^{-\beta c_{ji}}} = -2 \sum_{i=1}^n \frac{\sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh}^*) z_{jh} e^{-\beta c_{ji}}}{\sum_{j=1}^k e^{-\beta c_{ji}}} \\ &= -2 \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha z_{jh} \sum_{i=1}^n \frac{(x_{ih} - z_{jh}^*) e^{-\beta c_{ji}}}{\sum_{l=1}^k e^{-\beta c_{li}}} \\ &= 0. \end{aligned}$$

Since $h'(0) = 0$ holds for arbitrary directions $Z = (z_{jh})$, noting that $w_{jh} > 0$ for all j, h , it is necessary for every j, h that

$$\sum_{i=1}^n \frac{(x_{ih} - z_{jh}^*) e^{-\beta c_{ji}}}{\sum_{l=1}^k e^{-\beta c_{li}}} = 0,$$

from which equation (15.63) follows and the necessity is established.

To prove that Z^* is the global minimum of $F_{\alpha,\beta,\epsilon}$ for $\beta = 0$, we examine the $kd \times kd$ Hessian matrix $H_\gamma(Z^*)$ of γ evaluated at Z^* . In fact, from equation (15.65), we have

$$h''(t) = \sum_{i=1}^n \frac{\sum_{j=1}^k [2a_{ji} - \beta(2a_{ji}t + b_{ji})^2] e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})} \sum_{j=1}^k e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})}}{\left(\sum_{j=1}^k e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})}\right)^2} - \sum_{i=1}^n \frac{-\beta \left(\sum_{j=1}^k (2a_{ji}t + b_{ji}) e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})}\right)^2}{\left(\sum_{j=1}^k e^{-\beta(a_{ji}t^2 + b_{ji}t + c_{ji})}\right)^2}.$$

Thus, at $t = 0$, noting that $\beta = 0$ we have

$$\begin{aligned} h''(0) &= (z_{11}, z_{12}, \dots, z_{kd}) H_\gamma(Z^*) (z_{11}, z_{12}, \dots, z_{kd})^T \\ &= \sum_{i=1}^n \frac{\sum_{j=1}^k [2a_{ji} - \beta b_{ji}^2] e^{-\beta c_{ji}} \sum_{j=1}^k e^{-\beta c_{ji}} + \beta \left(\sum_{j=1}^k b_{ji} e^{-\beta c_{ji}}\right)^2}{\left(\sum_{l=1}^k e^{-\beta c_{li}}\right)^2} \\ &= \sum_{i=1}^n \frac{\sum_{j=1}^k 2a_{ji} e^{-\beta c_{ji}} \sum_{j=1}^k e^{-\beta c_{ji}} + \beta \left[\left(\sum_{j=1}^k b_{ji} e^{-\beta c_{ji}}\right)^2 - \sum_{j=1}^k b_{ji}^2 e^{-\beta c_{ji}} \sum_{j=1}^k e^{-\beta c_{ji}}\right]}{\left(\sum_{l=1}^k e^{-\beta c_{li}}\right)^2} \\ &= \frac{2}{k} \sum_{i=1}^n \sum_{j=1}^k a_{ji} \\ &> 0. \end{aligned}$$

Hence, the Hessian matrix is positive definite, so Z^* is the global minimum of $F_{\alpha,0,\epsilon}$. This proves the theorem. \square

From Theorem 15.13 we see that at $\beta = 0$ there is only one cluster, since $u_{ji} = \frac{1}{k}$ for all j, i and \mathbf{z}_j , $j = 1, 2, \dots, k$, are identical to the center of the data set. At higher β , the objective function $F_{\alpha,\beta,\epsilon}$ may have many local minima and the cluster will split into smaller clusters. Once we obtain a new set of centers, we need to update the fuzzy dimension weight for the clusters. The following theorem tells how to do this.

Theorem 15.14. Let $\eta : M_{fk} \rightarrow \mathbb{R}$ be defined as

$$\eta(W) = F_{\alpha,\beta,\epsilon}(W, Z), \quad (15.66)$$

where $Z \in \mathbb{R}^{kd}$ is fixed. Then the fuzzy dimension weight W^* optimizes the function η if W^* satisfies the implicit function

$$w_{jh}^* = \frac{1}{\sum_{l=1}^d \left[\frac{\sum_{i=1}^n u_{ji} (x_{ih} - z_{jh})^2 + \epsilon}{\sum_{i=1}^n u_{ji} (x_{il} - z_{jl})^2 + \epsilon} \right]^{\frac{1}{\alpha-1}}}, \quad 1 \leq j \leq k, 1 \leq h \leq d, \quad (15.67)$$

where the fuzzy membership u_{ji} is defined as

$$u_{ji} = \frac{\exp \left(-\beta \sum_{h=1}^d (w_{jh}^*)^\alpha (x_{ih} - z_{jh})^2 \right)}{\sum_{l=1}^k \exp \left(-\beta \sum_{h=1}^d (w_{lh}^*)^\alpha (x_{ih} - z_{lh})^2 \right)}, \quad 1 \leq j \leq k, 1 \leq i \leq n. \quad (15.68)$$

Proof. Minimization of η over M_{fk} is a Kuhn-Tucker problem carrying the kd inequality constraints (15.46a) and k equality constraints (15.46b) (Bezdek, 1980). This can be done via the method of Lagrange multipliers. Let $\Lambda = (\lambda_1, \lambda_2, \dots, \lambda_k)$ be the multipliers and $\Gamma(W, \Lambda)$ be the Lagrangian

$$\Gamma(W, \Lambda) = -\frac{1}{\beta} \sum_{i=1}^n \ln \left(\sum_{j=1}^k e^{-\beta d_{ji}(W, Z)} \right) + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha - \sum_{j=1}^k \lambda_j \left(\sum_{h=1}^d w_{jh} - 1 \right).$$

If (W^*, Λ^*) is to minimize Γ , its gradient in both sets of variables must vanish, i.e.,

$$\begin{aligned} \frac{\partial \Gamma(W^*, \Lambda^*)}{\partial w_{jh}} &= \sum_{i=1}^n \frac{\alpha (w_{jh}^*)^{\alpha-1} (x_{ih} - z_{jh})^2 e^{-\beta d_{ji}(W^*, Z^*)}}{\sum_{j=1}^k e^{-\beta d_{ji}(W^*, Z^*)}} + \epsilon \alpha (w_{jh}^*)^{\alpha-1} - \lambda_j \\ &= \alpha (w_{jh}^*)^{\alpha-1} \left(\sum_{i=1}^n u_{ji} (x_{ih} - z_{jh})^2 + \epsilon \right) - \lambda_j \\ &= 0, \quad 1 \leq j \leq k, 1 \leq h \leq d, \end{aligned} \quad (15.69a)$$

where u_{ji} is defined in Equation (15.68), and

$$\frac{\partial \Gamma(W^*, \Lambda^*)}{\partial \lambda_j} = \sum_{h=1}^d w_{jh}^* - 1 = 0, \quad 1 \leq j \leq k. \quad (15.69b)$$

Equation (15.67) following immediately by solving the $k(d+1)$ equations in (15.69a) and (15.69b). This proves the theorem. \square

From equation (15.68), we see that $u_{ji} > 0$ for all $1 \leq j \leq k$ and $1 \leq i \leq n$. If the data set D is such that for each dimension h there exist two distinct values, then $\sum_{i=1}^n u_{li} (x_{ih} - z_{lh})^2 > 0$ for all $1 \leq l \leq k$ and $1 \leq h \leq d$. In this case, we set the parameter

$\epsilon = 0$, since the purpose of ϵ is to avoid divide-by-zero errors when implementing the MSSC algorithm. In all analysis below, we assume $\epsilon = 0$.

The FSC algorithm presented in Section 15.11 is a limit case of the MSSC algorithm presented above when $\beta \rightarrow \infty$ (see Figure 15.1). In fact, letting β approach infinity, we have

$$\begin{aligned} & \lim_{\beta \rightarrow \infty} F_{\alpha, \beta, \epsilon}(W, Z) \\ &= \lim_{\beta \rightarrow \infty} \frac{1}{-1} \sum_{i=1}^n \frac{\sum_{j=1}^k -d_{ji}(W, Z) e^{-\beta d_{ji}(W, Z)}}{\sum_{j=1}^k e^{-\beta d_{ji}(W, Z)}} + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha \\ &= \sum_{i=1}^n \min \{d_{ji}(W, Z) | j = 1, 2, \dots, k\} + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha \\ &= \sum_{i=1}^n \min \left\{ \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh})^2 | j = 1, 2, \dots, k \right\} + \epsilon \sum_{j=1}^k \sum_{h=1}^d w_{jh}^\alpha, \end{aligned}$$

which is exactly the objective function of the FSC algorithm defined in equation (15.52).

Clearly equation (15.63b) is an implicit equation in \mathbf{z}_j through equation (15.64). It is natural to propose the following iterative algorithm.

Definition 15.15 (The MSSC algorithm). Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a finite data set. Let $G_\alpha^\beta(\mathbf{x}, \mathbf{w})$ be the Gaussian kernel defined as

$$G_\alpha^\beta(\mathbf{x}, \mathbf{w}) = e^{-\beta \sum_{h=1}^d w_h^\alpha x_h^2}. \quad (15.70)$$

Let $v : D \rightarrow (0, \infty)$ be a weight function defined as

$$v(\mathbf{x}) = \frac{1}{\sum_{j=1}^k G_\alpha^\beta(\mathbf{x} - \mathbf{z}_j, \mathbf{w}_j)}. \quad (15.71)$$

The sample mean with the kernel G_α^β at $\mathbf{z}_j \in D$ is defined as

$$m(\mathbf{z}_j) = \frac{\sum_{\mathbf{x} \in D} G_\alpha^\beta(\mathbf{x} - \mathbf{z}_j, \mathbf{w}_j) v(\mathbf{x}) \mathbf{x}}{\sum_{\mathbf{x} \in D} G_\alpha^\beta(\mathbf{x} - \mathbf{z}_j, \mathbf{w}_j) v(\mathbf{x})}. \quad (15.72)$$

Let $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\} \subset D$ be a finite set of cluster centers. The evolution of Z in the form of iterations

$$Z^{(r)} = \left\{ \mathbf{z}_j^{(r)} = m(\mathbf{z}_j^{(r-1)}) | j = 1, 2, \dots, k \right\}, \quad r = 1, 2, \dots, \quad (15.73)$$

is called the MSSC algorithm, where $\mathbf{z}_j^{(0)} = \mathbf{z}_j$, $j = 1, 2, \dots, k$. The weights $v(\mathbf{x})$ and the fuzzy memberships $U = (u_{ji})$ are reevaluated after each iteration. The sequence $\mathbf{z}, m(\mathbf{z}), m^2(\mathbf{z}), \dots$ is called a trajectory of \mathbf{z} .

From Definition 9.5, we know that the above fixed-point iteration is the mean shift process with $v(\mathbf{x})$ being the weight of the point \mathbf{x} . If we choose $Z = D$, then this algorithm is also called a blurring process (Cheng, 1995).

ALGORITHM 15.13. The pseudocode of the MSSC algorithm.

Require: D - the data set, k - the number of starting points, α - the fuzzifier, and β - the Lagrange multiplier;

- 1: Initialize $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ by choosing k points from D randomly;
- 2: Initialize $W = \{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k\}$ with $w_{jh} = \frac{1}{d}$ ($1 \leq j \leq k$, $1 \leq h \leq d$);
- 3: Calculate fuzzy memberships U according to equation (15.64) or (15.68);
- 4: $Z^{new} \Leftarrow m(Z) = \{m(\mathbf{z}_1), m(\mathbf{z}_2), \dots, m(\mathbf{z}_k)\}$ according to equation (15.63);
- 5: Update W according to equation (15.67);
- 6: **while** $\|Z^{new} - Z\| > 0$ **do**
- 7: $Z \Leftarrow Z^{new}$;
- 8: $Z^{new} \Leftarrow m(Z)$ according to equation (15.63);
- 9: Update W according to equation (15.67);
- 10: Calculate fuzzy memberships U according to Equation (15.64) or (15.68);
- 11: **end while**
- 12: Output W , Z , and U .

Let $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ be an initial set of centers. Then the procedure defined in equation (15.73) determines a set of equilibrium points $Z^* = \{\mathbf{z}_1^*, \mathbf{z}_2^*, \dots, \mathbf{z}_k^*\}$ for fixed β , α , and ϵ , where $\mathbf{z}_j^* = \lim_{r \rightarrow \infty} m^r(\mathbf{z}_j)$. In principle, changing k will modify the resulting set of equilibrium points. However, there exists some k_c such that for all $k > k_c$, one gets only k_c distinct limit points for the simple reason that some points converge to the same equilibrium point.

The pseudocode of the MSSC algorithm is described in Algorithm 15.13. Once we obtain the fuzzy dimension weight matrix W , the equilibrium set Z , and the fuzzy memberships U , we can use the procedure in Algorithm 15.14 to get the final subspace clusters.

ALGORITHM 15.14. The postprocessing procedure to get the final subspace clusters.

Require: η , W , Z , and U ;

- 1: Let $Q = \{1, 2, \dots, k\}$;
- 2: **for** $j = 2$ to k **do**
- 3: **for** $i = 1$ to $j - 1$ **do**
- 4: **if** $\|\mathbf{z}_j - \mathbf{z}_i\|_\infty < \eta$ and $i \in Q$ **then**

```

5:       $Q = Q - \{j\}$   $\{\mathbf{z}_j$  is identical to  $\mathbf{z}_i\}$ ;
6:      Break;
7:  end if
8:  end for
9: end for
10: Let  $Q = \{j_1, j_2, \dots, j_{k_c}\}$   $\{k_c$  is the number of distinct equilibrium points};
11: for  $i = 1$  to  $n$  do
12:   if  $l_0 = \arg \max_{1 \leq l \leq k_c} u_{j_i l}$  then
13:     The point  $\mathbf{x}_i$  is assigned to the  $l_0$ th cluster  $C_{l_0}$ ;
14:   end if
15: end for
16: Get set of cluster dimensions  $Q_l$  for  $C_l$  by applying  $k$ -means to  $\mathbf{w}_{j_i}$ ;
17: Output  $C_l$  and  $Q_l$  for  $l = 1, 2, \dots, k_c$ .

```

If $\beta = 0$, each data point is uniformly associated with all clusters, and thus all the centers $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k$ are identical (Gan, 2006). Let k_c be the number of distinct centers in $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$. Then, at $\beta = 0$ we have $k_c = 1$, but at some positive β we shall have $k_c > 1$. In other words, the single cluster will split into smaller clusters. The new clusters will split at higher β . At $\beta = \infty$, $k_c = k$, i.e., we shall get k clusters, where $k (< n)$ is the number of initial centers.

Clearly, when $k = 1$ the objective function is

$$\begin{aligned}
F_{\alpha, \beta, 0}(W, Z) &= -\frac{1}{\beta} \sum_{i=1}^n \ln(e^{-\beta d_{1i}(W, Z)}) = \sum_{i=1}^n d_{1i}(W, Z) \\
&= \sum_{i=1}^n \sum_{h=1}^d w_{1h}^\alpha (x_{ih} - z_{1h})^2,
\end{aligned}$$

which has a single minimum or the global minimum; when $k \geq 2$ and $\beta = 0$ the objective function is

$$F_{\alpha, 0, 0}(W, Z) = -\frac{1}{\beta} \sum_{i=1}^n \ln(k) = -\infty.$$

Thus, at $\beta = 0$ the objective function has a single minimum or the global minimum. At higher β , the objective function may have many local minima.

Since the objective function has only one minimum when $k = 1$, to derive the critical value β_α at which the first phase transition occurs, we assume that $k > 1$ in the following discussion. We know that at $\beta = \beta_\alpha$ there is one cluster centered at the mean of the data set. Without loss of generality, we move the data set such that the mean of the new data set is located at the origin. Then the first phase transition occurs when the objective function

has nonzero minimum, i.e., nonzero solution to the equations

$$\begin{aligned}\frac{\partial F_{\alpha,\beta,0}(W, Z)}{\partial z_{jh}} &= -\frac{1}{\beta} \sum_{i=1}^n \frac{2\beta w_{jh}^\alpha (x_{ih} - z_{jh}) e^{-\beta d_{ji}(W, Z)}}{\sum_{l=1}^k e^{-\beta d_{li}(W, Z)}} \\ &= -2w_{jh}^\alpha \sum_{i=1}^n \frac{(x_{ih} - z_{jh}) e^{-\beta d_{ji}(W, Z)}}{\sum_{l=1}^k e^{-\beta d_{li}(W, Z)}} \\ &= 0, \quad 1 \leq j \leq k, \quad 1 \leq h \leq d,\end{aligned}$$

and

$$\sum_{i=1}^n \frac{(\mathbf{x}_i - \mathbf{z}_j) e^{-\beta d_{ji}(W, Z)}}{\sum_{l=1}^k e^{-\beta d_{li}(W, Z)}} = \mathbf{0}, \quad 1 \leq j \leq k, \quad (15.74)$$

where $\mathbf{0}$ is the zero vector and $d_{ji}(W, Z)$ is defined as

$$d_{ji}(W, Z) = \sum_{h=1}^d w_{jh}^\alpha (x_{ih} - z_{jh})^2, \quad 1 \leq i \leq n, \quad 1 \leq j \leq k.$$

Note that in a small neighborhood of the origin, we have

$$w_{jh} = w_h \approx \frac{1}{\sum_{l=1}^d \left[\frac{\sum_{i=1}^n x_{ih}^2}{\sum_{i=1}^n x_{il}^2} \right]^{\frac{1}{\alpha-1}}}, \quad 1 \leq j \leq k, \quad 1 \leq h \leq d,$$

and

$$z_{jh} = z_h \approx 0, \quad 1 \leq j \leq k, \quad 1 \leq h \leq d.$$

Now expanding equation (15.74) on a small neighborhood of the origin by Taylor's series in $\mathbf{z}_j = (z_{j1}, z_{j2}, \dots, z_{jd})^T$ and ignoring higher-order terms of z_{jh} , we have

$$\begin{aligned}\mathbf{0} &= \sum_{i=1}^n \frac{(\mathbf{x}_i - \mathbf{z}_j) e^{-\beta d_{ji}(W, Z)}}{\sum_{l=1}^k e^{-\beta d_{li}(W, Z)}} \\ &\approx \sum_{i=1}^n \frac{(\mathbf{x}_i - \mathbf{z}_j) \exp \left(-\beta \sum_{t=1}^d w_t^\alpha (x_{it} - z_{jt})^2 \right)}{k \exp \left(-\beta \sum_{t=1}^d w_t^\alpha x_{it}^2 \right)} \\ &\approx \frac{1}{k} \sum_{i=1}^n (\mathbf{x}_i - \mathbf{z}_j) \exp \left(2\beta \sum_{t=1}^d w_t^\alpha x_{it} z_{jt} - \beta \sum_{t=1}^d w_t^\alpha z_{jt}^2 \right) \\ &\approx \frac{1}{k} \sum_{i=1}^n (\mathbf{x}_i - \mathbf{z}_j) \left(1 + 2\beta \sum_{t=1}^d w_t^\alpha x_{it} z_{jt} \right). \quad (15.75)\end{aligned}$$

Since $\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$, rearranging equation (15.75) and ignoring higher-order terms of z_{jh} s give

$$\sum_{i=1}^n \left(\mathbf{x}_i \cdot 2\beta \sum_{t=1}^d w_i^\alpha x_{it} z_{jt} - \mathbf{z}_j \right) = \mathbf{0},$$

or

$$\sum_{i=1}^n \left[2\beta \begin{pmatrix} w_1^\alpha x_{i1}^2 & w_2^\alpha x_{i1}x_{i2} & \cdots & w_d^\alpha x_{i1}x_{id} \\ w_1^\alpha x_{i2}x_{i1} & w_2^\alpha x_{i2}^2 & \cdots & w_d^\alpha x_{i2}x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ w_1^\alpha x_{id}x_{i1} & w_2^\alpha x_{id}x_{i2} & \cdots & w_d^\alpha x_{id}^2 \end{pmatrix} \begin{pmatrix} z_{j1} \\ z_{j2} \\ \vdots \\ z_{jd} \end{pmatrix} - \begin{pmatrix} z_{j1} \\ z_{j2} \\ \vdots \\ z_{jd} \end{pmatrix} \right] = \mathbf{0}. \quad (15.76)$$

Let $\mathbf{v}_j = (x_{1j}, x_{2j}, \dots, x_{nj})$ for $j = 1, 2, \dots, k$, and let the variance and covariance be defined as

$$\text{var}(\mathbf{v}_j) = \frac{1}{n} \sum_{i=1}^n (x_{ij} - \mu_j)^2 = \frac{1}{n} \sum_{i=1}^n x_{ij}^2,$$

$$\text{Cor}(\mathbf{v}_j, \mathbf{v}_l) = \frac{1}{n} \sum_{i=1}^n (x_{ij} - \mu_j)(x_{il} - \mu_l),$$

where

$$\mu_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, \quad 1 \leq j \leq k.$$

From equation (15.76), we have

$$(I - 2\beta C_\alpha) \mathbf{z}_j = \mathbf{0}, \quad (15.77)$$

where I is the $d \times d$ identity matrix and C_α is the $d \times d$ matrix defined as

$$C_\alpha = \begin{pmatrix} w_1^\alpha \text{var}(\mathbf{v}_1) & w_2^\alpha \text{Cor}(\mathbf{v}_1, \mathbf{v}_2) & \cdots & w_d^\alpha \text{Cor}(\mathbf{v}_1, \mathbf{v}_d) \\ w_1^\alpha \text{Cor}(\mathbf{v}_2, \mathbf{v}_1) & w_2^\alpha \text{var}(\mathbf{v}_2) & \cdots & w_d^\alpha \text{Cor}(\mathbf{v}_2, \mathbf{v}_d) \\ \vdots & \vdots & \ddots & \vdots \\ w_1^\alpha \text{Cor}(\mathbf{v}_d, \mathbf{v}_1) & w_2^\alpha \text{Cor}(\mathbf{v}_d, \mathbf{v}_2) & \cdots & w_d^\alpha \text{var}(\mathbf{v}_d) \end{pmatrix}.$$

Thus the critical value for β is

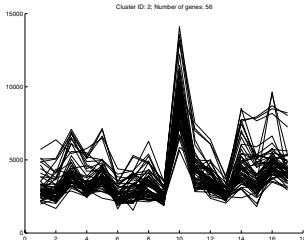
$$\beta_\alpha = \frac{1}{2\lambda_{max}}, \quad (15.78)$$

where λ_{max} is the largest eigenvalue of C_α . Moreover, the center of the new cluster is the eigenvector of C_α .

15.13 Summary

The subspace clustering introduced in this chapter is an extension of traditional clustering. In general, subspace clustering algorithms can be classified into two major categories (Parsons

et al., 2004b): top-down algorithms and bottom-up algorithms. A top-down algorithm finds an initial clustering in the full set of the dimensions and evaluates the subspaces of each cluster, whereas a bottom-up algorithm finds dense regions in low-dimensional spaces and then combines them to form clusters. CLIQUE, ENCLUS, MAFIA, CLTree, DOC, and CBF (Cell-Based Clustering method) (Chang and Jin, 2002) are examples of bottom-up subspace clustering algorithms, PART, PROCLUS, ORCLUS, FINDIT, and δ -cluster (Yang et al., 2002a) are examples of top-down subspace clustering algorithms. A comparison of these subspace clustering algorithms can be found in Parsons et al. (2004a). Other discussions related to subspace clustering can be found in Aggarwal and Yu (2002), Amir et al. (2003), Agarwal and Mustafa (2004), Domeniconi et al. (2004), Har-Peled and Varadarajan (2002), Ke and Kanade (2004), Krishnapuram and Freg (1991), Kroeger et al. (2004), Sarafis et al. (2003), Wang et al. (2004), and Narahashi and Suzuki (2002).



Chapter 16

Miscellaneous Algorithms

This chapter introduces some miscellaneous clustering algorithms, including algorithms for clustering time series data, data streams, and transaction data.

16.1 Time Series Clustering Algorithms

As an independent exploratory technique or a subroutine in more complex data mining algorithms such as indexing (Hetland, 2004; Keogh et al., 2001; Li et al., 1998), rule discovery (Das et al., 1998; Tsumoto, 1999; Caraça-Valente and López-Chavarrías, 2000; Chiu et al., 2003), and classification (Cotofrei and Stoffel, 2002), time series clustering has attracted much attention. In general, the time series data to be clustered can be classified into two categories: many individual time series and a single time series. This leads to two categories of clusterings: whole clustering and subsequence clustering (Keogh et al., 2003).

The notion of whole clustering is similar to the conventional clustering discussed in previous chapters. Precisely, given a set of individual time series, the objective is to group these time series into clusters such that time series from the same cluster are more similar to each other than time series from different clusters. Subsequence clustering means clustering a single time series. Given a single time series, subsequences are extracted with a sliding window and then subsequence clustering is performed on the extracted subsequences. Subsequence clustering is commonly used as a subroutine in algorithms such as indexing, rule discovery, prediction, anomaly detection, and classification (Keogh et al., 2003).

Keogh et al. (2003) and Lin et al. (2003) showed that subsequence clustering of time series is meaningless, where “meaningless” means that the clustering output is independent of the input. This invalidates the contributions of dozens of previously published papers that use subsequence clustering. The authors also gave several conditions that must be satisfied for subsequence clustering to be meaningful. Assume that a time series contains k approximately or exactly repeated patterns of length w with k and w known in advance. Then necessary conditions for a clustering algorithm to discover the k patterns are as follows:

1. The weighted mean of patterns must sum to a horizontal line.
2. The k patterns must have approximately equal numbers of trivial matches (Lin et al., 2003).

Since the chances of both conditions being met is essentially zero, subsequence clustering is meaningless.

In terms of models, time series clustering algorithms can be categorized as model based or model free (Bagnall and Janacek, 2004). Model-based algorithms, also called generative algorithms, assume some form of the underlying generating process, estimate the model from each data point, and then cluster based on similarity measures between model parameters. Examples of the most commonly assumed models are polynomial mixture models (Bagnall et al., 2003; Bar-Joseph et al., 2002), autoregressive moving average (ARMA) model (Kalpakis et al., 2001; Bagnall and Janacek, 2004; Xiong and Yeung, 2002), Markov chain models (MCs) (Ramoni et al., 2002), and hidden Markov chain models (HMMs) (Oates et al., 2001; Zhong and Ghosh, 2003a).

Model-free clustering algorithms use a specific distance measure on the transformed data or the original data. Some measures for time series are presented in Section 6.6. In what follows, we focus on whole clustering of time series. For subsequence clustering, readers are referred to (Keogh et al., 2003) and (Lin et al., 2003) and the references therein.

We now introduce an algorithm based on the ARMA model, which is a commonly assumed model for time series. An ARMA(p, q) model has the form (Bagnall and Janacek, 2004; Maharaj, 2000)

$$X(t) = \sum_{i=1}^p \phi_i \cdot X(t-i) + \epsilon(t) + \sum_{i=1}^q \theta_i \cdot \epsilon(t-i),$$

where $\epsilon(t)$ are $N(0, \sigma^2)$, i.e., normally distributed with variance σ^2 and mean 0, and σ, p, q, ϕ_i , and θ_i are constants in the model. Autocorrelations can be employed to estimate the parameters of an ARMA model (Bagnall and Janacek, 2004).

Bagnall and Janacek (2004) proposed a clustering algorithm for data derived from ARMA models using the k -means and k -medoids algorithms with Euclidean distance between estimated model parameters. In this algorithm, real-valued time series are first transformed into binary sequences through a process called *clipping* or *hard limiting*. Much of the underlying structure that characterizes the original time series is retained in the clipped series. For example, given a real-valued time series Y and letting μ be the population mean of series Y , then the corresponding binary series C is obtained as

$$C(t) = \begin{cases} 1 & \text{if } Y(t) > \mu, \\ 0 & \text{otherwise.} \end{cases}$$

In addition, the algorithm proposed by Bagnall and Janacek (2004) assumes that the series in each true underlying cluster are generated by an ARMA model. A model for cluster i without outliers, for example, has the form

$$X_i(t) = \sum_{j=1}^p \phi_j \cdot X_i(t-j) + \epsilon_i(t) + \sum_{j=1}^q \theta_j \cdot \epsilon_i(t-j),$$

where $\epsilon_i(t)$ are $N(0, \sigma)$, i.e., normally distributed with variance σ^2 and mean 0, and σ, p, q, ϕ_j , and θ_j are constants for the model. To model the effect of outliers, a further term is added to $X_i(t)$ as

$$Y_i(t) = X_i(t) + s \cdot a \cdot b,$$

where s is a constant and $a \in \{0, 1\}$ and $b \in \{-1, 1\}$ are observations of independent variables A and B with density functions $f(a) = p^a(1-p)^{1-a}$ and $f(b) = \frac{1}{2}$, respectively. p is the probability that a random shock effect occurs.

Since any invertible ARMA model can be represented as an infinite autoregressive (AR) model,

$$\epsilon(t) = \sum_{i=1}^{\infty} \pi_i X(t-i).$$

The ARMA model parameters are estimated via estimating the AR model for each series by using a standard three-stage procedure (Bagnall and Janacek, 2004). With the fitted parameters, the k -means and k -medoids algorithms are used to cluster the parameters using the Euclidean distance.

Clustering ARMA or autoregressive integrated moving average (ARIMA) time series has also been discussed by Maharaj (2000), Xiong and Yeung (2002), and Kalpakis et al. (2001). Maharaj (2000) proposed a test of a hypothesis to compare two stationary time series. The test of the hypothesis is performed for every pair of time series in a data set to determine the p -values associated with the data set, and then an agglomerative algorithm is used to cluster the p -values hierarchically.

Xiong and Yeung (2002) proposed a clustering algorithm using mixtures of ARMA models. An expectation-maximization (EM) algorithm for learning the mixing coefficients and the parameters of the component models is derived.

16.2 Streaming Algorithms

Data streams (Guha et al., 2003; Barbará, 2002; Gaber et al., 2005; Lu and Huang, 2005), such as customer click streams, multimedia data, and financial transactions, refer to ordered sequences of points that must be accessed in order and that can be read only once or a small number of times. Each reading of such a sequence is called a linear scan or a pass. Stream models (Henzinger et al., 1998) were motivated to deal with massive data sets that are far too large to fit in main memory and are typically stored in secondary devices. For such massive data sets, random accesses are extremely expensive and thus impractical, so linear scans are the only cost-effective access method.

A major feature of a data stream clustering algorithm is that it only stores a summary of the data scanned in the past, leaving enough memory space for the processing of future data. This is necessary because the size of massive data stream sets far exceeds the amount of main memory available to an algorithm and thus it is not possible for the algorithm to remember too much of the scanned data. In other words, a data stream algorithm makes decisions before all the data are available. Therefore, besides the running time and the memory usage, the number of linear scans is also a criterion to measure the performance of a data stream algorithm.

The data stream model has attracted many researchers' attention recently for its applicability to various types of data, such as click streams, financial transactions, and telephone records. In general, stream algorithms can be classified into the following six categories (Guha et al., 2003):

- (a) frequency estimation approaches (Haas et al., 1995; Charikar et al., 2000);
- (b) norm estimation approaches (Alon et al., 1996; Feigenbaum et al., 1999; Indyk, 2000);
- (c) order statistics approaches (Gilbert et al., 2002b; Greenwald and Khanna, 2001; Manku et al., 1998, 1999; Munro and Paterson, 1980);
- (d) synopsis structure approaches (Gibbons and Matias, 1999);
- (e) time-indexed approaches (Babcock et al., 2002; Datar et al., 2002; Ganti et al., 2000, 2001; Guha and Koudas, 2002);
- (f) signal reconstruction approaches (Achlioptas and McSherry, 2001; Drineas et al., 1999; Frieze et al., 1998, 2004; Gilbert et al., 2002a; Guha et al., 2001; Thaper et al., 2002).

16.2.1 LSEARCH

The LSEARCH algorithm is proposed by Guha et al. (2003, 2001), where computation takes place within a bounded space and data can only be accessed via linear scans. The LSEARCH algorithm solves the k -median problem (see Definition 16.1) in the stream context and guarantees theoretical performance. In fact, LSEARCH requires only $O(n^\epsilon)$ space and its running time is $\tilde{O}(nk)$, where $\epsilon < 1$ is a constant and $\tilde{O}(x) = O(x \log^2 x)$.

Definition 16.1 (The k -median problem). *Given an instance (D, k) of k -median, where k is an integer and D is a data set containing n points, the k -median cost or cost of a set of medians $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k$ is*

$$f(D, \mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k) = \sum_{\mathbf{x} \in D} \min_{1 \leq i \leq k} d(\mathbf{x}, \mathbf{z}_i),$$

where $d(\cdot, \cdot)$ is a distance function. $\text{cost}(D, Q)$ is the smallest possible cost if the medians are required to belong in the set Q . In the discrete case, the optimization problem is to find $\text{cost}(D, D)$; in the Euclidean case, the optimization problem is to find $\text{cost}(D, \mathbb{R}^d)$, where d is the dimensionality of the Euclidean space.

LSEARCH starts with an initial solution and then refines it by making local improvements. To speed up local search, LSEARCH reflexes the number of clusters in the intermediate steps and achieves exactly k clusters in the final step (see the facility location problem in Definition 16.2). In the intermediate steps, LSEARCH uses at least k but not too many medians since the best solution with $k - 1$ medians can be much more expensive than the best solution with k medians, and too many medians cost space. The algorithm to obtain a good initial solution is described in Algorithm 16.1.

Definition 16.2 (The facility location problem). *Given a set D of n data points in a metric space, a distance function $d : D \times D \rightarrow \mathbb{R}^+$, and a parameter λ , then for any choice*

of $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\} \subset D$ of k cluster centers, we define a partition of D into k clusters C_1, C_2, \dots, C_k such that C_i ($1 \leq i \leq k$) contains all data points in D that are closer to \mathbf{z}_i than to any other center. The optimization problem is to select a value of k and a set of centers Z such that the facility clustering cost function

$$FC(D, Z) = z|Z| + \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{z}_i) \quad (16.1)$$

is minimized, where z denotes the facility cost.

The facility location problem given in Definition 16.2 is the Lagrangian relaxation (Borodin et al., 1999; Feder and Greene, 1988; Guha et al., 2000a; Indyk, 1999; Mettu and Plaxton, 2004; Thorup, 2001) of the k -median problem. In the facility location problem, the number of centers is unrestricted, but each center included in the solution has an additional cost (see the facility cost defined in equation (16.1)).

ALGORITHM 16.1. The InitialSolution algorithm.

Require: D - the data set, z - the facility cost;

- 1: Reorder the data points in D randomly;
- 2: Create a cluster center at the first point;
- 3: **for** every point after the first **do**
- 4: Let d be the distance from the current point to the nearest existing cluster center;
- 5: Create a new cluster center at the current point with probability $\frac{d}{z}$; otherwise, add the current point to the best current cluster;
- 6: **end for**

The pseudocode of the LSEARCH algorithm is described in Algorithm 16.2. The function $FL(D, d(\cdot, \cdot), \epsilon, (F, g))$ called by the LSEARCH algorithm is described in Algorithm 16.2. The LSEARCH algorithm constructs a hierarchical clustering but operates at the same time on all layers of the dendrogram tree and maintains a front.

ALGORITHM 16.2. The LSEARCH algorithm.

Require: D - the data set, $d(\cdot, \cdot)$ - the distance function, k - the number of clusters, $\epsilon, \epsilon', \epsilon''$ - parameters;

- 1: $z_{min} \Leftarrow 0$;
- 2: $z_{max} \Leftarrow \sum_{\mathbf{x} \in D} d(\mathbf{x}, \mathbf{x}_0)$, where \mathbf{x}_0 is an arbitrary point in D ;
- 3: $z \Leftarrow \frac{z_{min} + z_{max}}{2}$;
- 4: $(I, a) \Leftarrow InitialSolution(D, z)\{I \subseteq D \text{ is a set of facilities and } a : D \rightarrow I \text{ is an assignment function}\}$;
- 5: Pick $\Theta\left(\frac{\log k}{p}\right)$ points randomly as feasible medians $\{f(x) \text{ is } \Theta(h(x)) \text{ if } f(x) \text{ is } O(h(x)) \text{ and } h(x) \text{ is } O(f(x))\}$;
- 6: **while** the number of medians $\neq k$ and $z_{min} < (1 - \epsilon'')z_{max}$ **do**

```

7: Let  $(F, g)$  be the current solution;
8:  $(F', g') \Leftarrow FL(D, d(\cdot, \cdot), \epsilon, (F, g))$ ;
9: if  $k \leq |F'| \leq 2k$  then
10:   Break;
11: else if  $|F'| > 2k$  then
12:    $z_{min} \Leftarrow z$ ;
13:    $z \Leftarrow \frac{z_{min} + z_{max}}{2}$ ;
14: else if  $|F'| < k$  then
15:    $z_{max} \Leftarrow z$ ;
16:    $z \Leftarrow \frac{z_{min} + z_{max}}{2}$ ;
17: end if
18: end while
19: Output solution  $(F', g')$ .

```

For the $FL(D, d(\cdot, \cdot), z, \epsilon, (I, a))$ function, an algorithm proposed by Charikar and Guha (1999) is modified to solve the facility location problem to get a new solution. For every $\mathbf{x} \in D$, the *gain* of \mathbf{x} is defined to be the cost saved or expended if a facility at \mathbf{x} were to be opened and all advantageous reassessments and facility closings were performed subject to the following constraints:

- (a) Points cannot be reassigned except to \mathbf{x} .
- (b) A facility can be closed only if its members are first reassigned to \mathbf{x} .

ALGORITHM 16.3. The $FL(D, d(\cdot, \cdot), z, \epsilon, (I, a))$ function.

Require: D - the data set, $d(\cdot, \cdot)$ - the distance function, z - the facility cost, ϵ - the parameter, (I, a) - a solution;

- 1: Begin with (I, a) as the current solution;
- 2: Let C be the cost of the current solution;
- 3: Obtain a new solution (I', a') by performing all advantageous closures and reassessments;
- 4: Let C' be the cost of the new solution;
- 5: **if** $C' \leq (1 - \epsilon)C$ **then**
- 6: Return to step 2.
- 7: **end if**

Like the CURE algorithm (see Subsection 7.4.5), the LSEARCH algorithm applies a partitioning approach and clusters the data set bottom up. The difference between LSEARCH and CURE is that LSEARCH is designed to produce provably good clustering while CURE is designed to produce robust and arbitrarily shaped clustering.

The performance of LSEARCH is compared to that of k -means and BIRCH (see Subsection 7.4.4) in terms of the sum of squared distances (SSD) measure. It seems that k -means is more sensitive than LSEARCH to dimensionality (Guha et al., 2003), but LSEARCH runs approximately three times as long as k -means. Babcock et al. (2003) used an exponential histogram (EH) data structure to improve the LSEARCH algorithm. Charikar et al. (2003)

proposed another k -median algorithm to overcome the problem of increasing approximation factors in LSEARCH with the increase in the number of levels.

16.2.2 Other Streaming Algorithms

The STREAM algorithm was proposed by O'Callaghan et al. (2002) to produce high-quality data stream clustering. The STREAM algorithm first determines the size of the sample and then applies the LSEARCH algorithm if the sample size is larger than a prespecified one. This process is repeated for each data chunk and the LSEARCH algorithm is applied to the cluster centers generated in the previous iterations. The STREAM algorithm is two or three times better than BIRCH in terms of the SSD measure (Guha et al., 2003) but runs two or three times slower than BIRCH.

The CluStream algorithm was proposed by Aggarwal et al. (2003) as a framework to cluster data streams. The CluStream algorithm divides the clustering process into two components: the online component and the offline component. The online component stores the summarized statistics of the data streams while the offline component clusters the summarized data according to user preferences such as the number of clusters.

The HPStream algorithm was proposed by Aggarwal et al. (2004) to cluster high-dimensional data streams. The HPStream algorithm introduces the concept of projected clustering or subspace clustering to data streams and achieves consistently high clustering quality for such projected clustering. The fading cluster structure (a data-structure designed to capture key statistical characteristics of the clusters generated during the course of a data stream) and the projection-based clustering methodology (a technique determines clusters for a specific subset of dimensions) are used in the HPStream algorithm.

Ordonez (2003) proposed a variant of the k -means algorithm, the incremental k -means algorithm, to cluster binary data streams. The proposed incremental k -means algorithm is faster than the scalable k -means algorithm (Bradley et al., 1998) and produces clusterings of comparable quality. The proposed incremental k -means algorithm also outperforms the online k -means algorithm (Zhang et al., 1996). The algorithm uses a mean-based initialization and incremental learning to obtain clusterings of high quality and achieves speedup through a simplified set of sufficient statistics and operations with sparse matrices.

16.3 Transaction Data Clustering Algorithms

Transaction data are also referred to as market basket data, which have been studied extensively in mining association rules (Hipp et al., 2000) for discovering the set of items that are frequently purchased (Yun et al., 2002). In this section, we introduce three algorithms, LargeItem, CLOPE, and OAK, to deal with transaction data from the perspective of cluster analysis. Other discussions related to transaction data clustering can be found in (Xu et al., 2003).

Transaction clustering refers to partitioning a set of transactions (see Section 2.3) into clusters such that similar transactions are in the same cluster and dissimilar transactions are in different clusters. Transaction clustering plays an important role in the recent developments of information retrieval (IR), Web technologies (Broder et al., 1997), and data mining.

Also, transaction clustering has many potential applications in e-commerce intelligence, retail industry, and Web applications.

Some algorithms targeted specifically to cluster transaction data have been proposed and studied, such as LargeItem (Wang et al., 1999a), OAK (Xiao and Dunham, 2001), and CLOPE (Yang et al., 2002b).

16.3.1 LargeItem

LargeItem (Wang et al., 1999a) is an optimization algorithm designed for clustering transaction data based on a criterion function. The LargeItem algorithm consists of two phases: the allocation phase and the refinement phase.

Given a user-specified minimum support θ ($0 < \theta \leq 1$), an item i is *large* in a cluster C if its support in C is at least $\theta|C|$, i.e., $|\{t : i \in t, t \in C\}| \geq \theta|C|$. Otherwise, item i is *small* in C . The objective function or cost function is defined in terms of the intracluster cost and the intercluster cost.

Given a clustering $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$, the intracluster cost is measured by the total number of small items, i.e.,

$$Intra(\mathcal{C}) = \left| \bigcup_{i=1}^k Small_i \right|,$$

where $Small_i$ is the set of small items in C_i .

The intercluster cost is defined in terms of the overlapping of large items of each cluster, i.e.,

$$Inter(\mathcal{C}) = \sum_{i=1}^k |Large_i| - \left| \bigcup_{i=1}^k Large_i \right|,$$

where $Large_i$ is the set of large items in C_i .

Then the overall cost function is defined as

$$Cost(\mathcal{C}) = w \cdot Intra(\mathcal{C}) + Inter(\mathcal{C}), \quad (16.2)$$

where $w > 0$ is a weight that balances between the intracluster similarity and the intercluster dissimilarity.

The best solution is the one that minimizes the criterion function (16.2). Since to find the exact best solution is not feasible, the goal of this algorithm is to find an approximate solution that is sufficient for practical applications. Unlike the k -means algorithm, the LargeItem algorithm allows k to vary, i.e., k is not required to be predetermined.

To avoid scanning all transactions in a cluster during clustering, some cluster features, such as $|Large_i|$, $|\bigcup_{i=1}^k Small_i|$, and $|\bigcup_{i=1}^k Large_i|$, are maintained after each allocation or move of a transaction. There are two cases of movement: moving a transaction to a cluster and moving a transaction out of a cluster. In either of the two cases, *small* items may become *large* and *large* items may become *small*.

The LargeItem algorithm takes advantage of some standard indexing techniques such as hash tables and B-tree in the maintenance and updating of the cluster features for each cluster.

16.3.2 CLOPE

CLOPE (Clustering with sLOPE) (Yang et al., 2002b) is an algorithm designed for clustering categorical data, especially transaction data. Like most partitional clustering approaches, CLOPE has a criterion function that will guide the algorithm to approximate the best partition by iteratively scanning the database.

For categorical data, the criterion function can be defined locally or globally. Locally, the criterion function is defined in terms of pairwise similarity between objects, such as Jaccard's coefficient. Globally, no pairwise similarity measures between individual objects are required. The global approach is very effective for clustering large categorical data sets. CLOPE uses a global criterion function.

Let $D = \{t_1, t_2, \dots, t_n\}$ be a transaction data set with n transactions, and let m be the number of items and k be the number of clusters. Given a cluster C , let $D(C)$ be the set of distinct items occurring in C , and let $O(i, C)$ be the number of occurrences of item i in C , i.e.,

$$D(C) = \left| \bigcup_{t \in C} t \right|,$$

$$O(i, C) = |\{t : i \in t, t \in C\}|.$$

The size $S(C)$ and width $W(C)$ of a cluster C are defined as

$$S(C) = \sum_{i \in D(C)} O(i, C) = \sum_{t \in C} |t|,$$

$$W(C) = |D(C)|.$$

Then the height of C is defined as $H(C) = \frac{S(C)}{W(C)}$, and the gradient of C is defined as $G(C) = \frac{H(C)}{W(C)} = \frac{S(C)}{W(C)^2}$.

Let $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ be a partition of D . Then the criterion function is defined as

$$F(\mathcal{C}) = \frac{\sum_{i=1}^k G(C_i)|C_i|}{\sum_{i=1}^k |C_i|} = \frac{\sum_{i=1}^k \frac{S(C_i)|C_i|}{W(C_i)^2}}{|D|}. \quad (16.3)$$

The criterion function (16.3) can also be generalized to

$$F_r(\mathcal{C}) = \frac{\sum_{i=1}^k \frac{S(C_i)|C_i|}{W(C_i)^r}}{|D|}, \quad (16.4)$$

where r is a positive real number called the repulsion. Adding a transaction t to a cluster C , the increase in the criterion function is proportional to

$$\Delta(C, t, r) = \frac{(S(C) + |t|)(|C| + 1)}{W(C \cup \{t\})^r} - \frac{S(C)|C|}{W(C)^r}, \quad (16.5)$$

which can be computed straightforwardly using cluster features, i.e., the number of transactions $|C|$, the number of distinct items $W(C)$, $O(i, C)$ for each item i , and the size $S(C)$. It can be shown that F_r is maximized by putting t to C_i if $\Delta(C_i, t, r)$ is the maximum.

ALGORITHM 16.4. The CLOPE algorithm.

Require: A data set D , number of clusters k ;

```

1: while not end of database file do
2:   Read the next transaction  $t$ ;
3:   Put  $t$  to an existing cluster or a new cluster  $C_i$  that maximizes the criterion function;
4:   Write  $\langle t, i \rangle$  back to the database;
5: end while
6: repeat
7:   Rewind the database file;
8:    $moved \Leftarrow false$ ;
9:   while Not end of the database file do
10:    Read  $\langle t, i \rangle$ ;
11:    Move  $t$  to an existing cluster or a new cluster  $C_j$  that maximizes the criterion
        function;
12:    if  $i \neq j$  then
13:      Write  $\langle t, j \rangle$  and set  $moved \Leftarrow true$ ;
14:    end if
15:   end while
16: until No further movements.

```

The pseudocode of the algorithm CLOPE is described in Algorithm 16.4. The time complexity for one iteration is $O(nkl)$, where n is the number of transactions, k is the number of clusters, and l is the average length of a transaction.

16.3.3 OAK

OAK (Online Adaptive Klustering) (Xiao and Dunham, 2001) is another algorithm designed for clustering transaction data. It is also an interactive, incremental, and scalable clustering algorithm.

OAK combines hierarchical and partitional clustering. The dendrogram created by OAK may not have one leaf node per point. It uses a representative for an individual point or a set of similar points. The dendrogram is maintained upon the representatives. For different clusters, the number of representatives may be different. The dendrogram is updated by using other incremental agglomerative hierarchical algorithms, such as SLINK (Sibson, 1973) and CLINK (Defays, 1977).

OAK always condenses the most similar points or subclusters such that the dendrogram can fit into main memory regardless of the memory size. The dendrogram is represented by the pointer representation (Sibson, 1973), which is defined by two vectors. To describe the algorithm, let us define some data structures used in OAK. Let $D = \{t_1, t_2, \dots, t_n\}$ be a set of n transactions, k the number of clusters, and M the number of representatives:

$rep[1, \dots, M]$	vectors of representatives;
$\lambda[1, \dots, M]$	dissimilarity at each level of the dendrogram;
$\pi[1, \dots, M]$	the pair to be merged at each level of the dendrogram;
d_{min}	threshold for condensation;
$profile[1, \dots, M]$	cluster profile for each cluster.

ALGORITHM 16.5. A sketch of the OAK algorithm.

Require: D -A transaction data set; k -Number of clusters; M -Number of representatives (may be determined by memory size);

```

1:  $numRep \Leftarrow 1, rep[1] \Leftarrow t_1, \pi[1] \Leftarrow 1, \lambda[1] \Leftarrow \infty;$ 
2: for  $i = 2$  to  $n$  do
3:   Read  $t_i$ ;
4:   if  $i \leq M$  then
5:     Update  $\lambda$  and  $\pi$  incrementally by SLINK;
6:      $numRep++$ ,  $rep[numRep] \Leftarrow t_i$ ;
7:      $d_{min} \Leftarrow \min\{\lambda[j] : j = 1, 2, \dots, numRep\}$ ;
8:     Update cluster profile;
9:   else
10:    Condense dendrogram;
11:   end if
12:   if Interrupted by user then
13:     Update  $k$  from user;
14:   end if
15: end for

```

The OAK algorithm is described in Algorithm 16.5. The total time complexity of OAK is $O(Mn)$, where M is the number of representatives and n is the number of objects in the data set. OAK provides two approaches for interactive clustering: ad hoc interaction and browsing interaction. In ad hoc interaction, users can indicate a new independent value for k , while in browsing interaction, users can change k to $k + 1$ or $k - 1$.

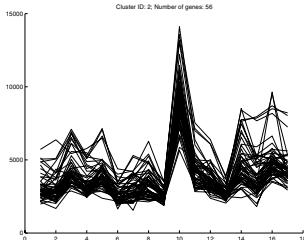
16.4 Summary

Time series clustering has become an important topic; various similarity measures, clustering and classification algorithms, and data-mining tools have been developed. In this chapter, we focused on clustering algorithms for time series from the perspective of databases. Due to a huge volume of literature in this field, we were forced to present a very brief summary of some commonly used methods. As an example of clustering time series, we also presented a clustering algorithm designed to cluster time series generated by a certain ARMA model. There are many published works related to time series clustering; interested readers may find the references at the end of this book helpful.

Streaming clustering algorithms, which are designed to cluster massive data sets such as network data, temperatures, and satellite imagery data, were also introduced in this chapter. Streaming clustering algorithms are required to process new data and identify

changes in evolving clustering models in a fast and incremental manner. More requirements for streaming algorithms are discussed by Barbará (2002).

Transaction data are also referred to as market basket data, which have been studied extensively in mining association rules (Hipp et al., 2000) for discovering the set of items that are frequently purchased (Yun et al., 2002). In this chapter, we introduced three algorithms, LargeItem, CLOPE, and OAK, to deal with transaction data from the perspective of cluster analysis. Other discussions related to transaction data clustering can be found in Xu et al. (2003).



Chapter 17

Evaluation of Clustering Algorithms

In the literature of data clustering, a lot of algorithms have been proposed for different applications and different sizes of data. But clustering a data set is an unsupervised process; there are no predefined classes and no examples that can show that the clusters found by the clustering algorithms are valid (Halkidi et al., 2002a). To compare the clustering results of different clustering algorithms, it is necessary to develop some validity criteria. Also, if the number of clusters is not given in the clustering algorithms, it is a highly nontrivial task to find the optimal number of clusters in the data set. To do this, we need some cluster validity methods. In this chapter, we will present various kinds of cluster validity methods appearing in the literature.

17.1 Introduction

In general, there are three fundamental criteria to investigate the cluster validity: external criteria, internal criteria, and relative criteria (Jain and Dubes, 1988; Theodoridis and Koutrouvas, 1999; Halkidi et al., 2002a). Some validity index criteria work well when the clusters are compact but do not work sufficiently (Halkidi et al., 2002a) if the clusters have arbitrary shape (applications in spatial data, biology data, etc.).

Figure 17.1 summarizes some popular criteria. The first two approaches involve statistical testing, which is computationally expensive. The third, i.e., relative criteria, does not involve statistical testing.

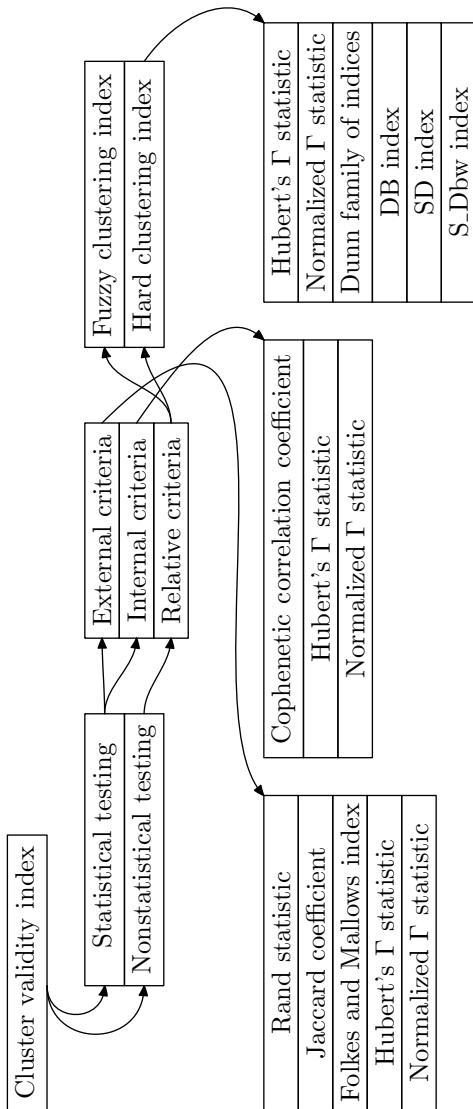


Figure 17.1. Diagram of the cluster validity indices.

17.1.1 Hypothesis Testing

In the general approach of cluster validity, the basic idea is to test whether the data points in the data set are randomly structured or not (Halkidi et al., 2002a). The test is based on the null hypothesis, H_0 , which is the hypothesis of random structure of the data set. If we accept the null hypothesis, then the data in the data set are randomly distributed.

Jain and Dubes (1988) suggested three forms of randomness hypotheses:

- (a) The random graph hypothesis, i.e., H_0 : All $n \times n$ random order proximity matrices are equally likely.
- (b) The random label hypothesis, i.e., H_0 : All permutations of the labels on n objects are equally likely.
- (c) The random position hypothesis, i.e., H_0 : All sets of n locations in some region of a d -dimensional space are equally likely.

The population for the random graph hypothesis is the set of all ordinal proximity matrices on n objects. There are $\frac{n(n-1)}{2}$ entries in an $n \times n$ ordinal proximity matrix, and these entries can be taken to be integers from 1 to $\frac{n(n-1)}{2}$. Since under the random graph hypothesis all ordinal proximity matrices are equally likely, each distinct ordinal proximity matrix is assigned probability $(\frac{n(n-1)}{2}!)^{-1}$.

The population for the random label hypothesis is the set of $n!$ permutations of the labels on the n objects. Since all the permutations are equally likely, each permutation is assigned probability $\frac{1}{n!}$. This hypothesis can be applied with all data types (Jain and Dubes, 1988).

The random position hypothesis is appropriate for ratio data, and it can be expressed in different ways. For example, each of the n points is inserted randomly into the region; the n points are samples of a Poisson process conditioned on the number of points. There are two major differences between the random graph hypothesis and the random position hypothesis. The first is that the former is for ordinal proximities and the latter is for proximities on a ratio scale, and the second is dimensionality (Jain and Dubes, 1988).

Hypothesis tests in cluster validity have been discussed in (Jain and Dubes, 1988). Monte Carlo and bootstrapping (Jain and Dubes, 1988) are two computer simulation tools used in statistical testing of hypotheses. The algorithm of using the Monte Carlo method to compute the probability density function of the indices is listed in Algorithm 17.1.

ALGORITHM 17.1. The Monte Carlo technique for computing the probability density function of the indices.

Require: n -number of data points, r -number of values of the index q ;

- 1: **for** $i = 1$ to r **do**
- 2: Generate a data set X_i with n data points of the same dimension as the data point in X using the normal distribution {generate phase};
- 3: Assign each data point $y_{j,i} \in X_i$ to the cluster of partition P to which $x_j \in X$ belongs;
- 4: Run the same clustering algorithm used to produce the cluster structure C for data set X_i and get the cluster structure C_i ;

-
- 5: Compute the defined index value $q_i = q(C_i)$ for P and C_i {various indices will be defined in Section 17.1.2};
- 6: **end for**
- 7: Create the scatter plot of the r validity index values, i.e., plot the approximation of the probability density function of the defined validity index.

17.1.2 External Criteria

In an external criteria approach, we evaluate the results of a clustering algorithm based on a prespecified structure which is imposed on a data set and reflects the intuitive structure of the data set. Based on the external criteria, there are two different approaches:

- Comparing the resulting clustering structure C to an independent partition of the data P , which was built to people's intuition about the clustering structure of the data set.
- Comparing the proximity matrix Q to the partition P .

Both approaches are computationally expensive; the Monte Carlo technique is employed to solve the problems in Halkidi et al. (2002a).

First Approach

To begin the first approach, i.e., comparing the resulting clustering structure $C = \{C_1, \dots, C_m\}$ to the partition $P = \{P_1, \dots, P_s\}$, we need to define a, b, c , and d as follows. a is the number of pairs of data points which are in the same cluster of C and in the same cluster of P , b is the number of pairs of data points which are in the same cluster of C but in different clusters of P , c is the number of pairs of data points which are in different clusters of C but in the same cluster of P , and d is the number of pairs of data points which are in different clusters of C and in different clusters of P . Let M be the total number of pairs of data points in the data set. Then we have

$$M = a + b + c + d = \frac{n(n - 1)}{2},$$

where n is the number of data points in the data set.

Some common indices to measure the degree of similarity between C and P are shown in Table 17.1.

The range for the first three indices is $[0, 1]$, i.e., $R, J, FM \in [0, 1]$. High values of these indices indicate great similarity between C and P . High values of the Hubert's Γ statistic index indicate a strong similarity between X and Y ; its range is also $[0, 1]$. The range of the normalized Γ statistic index $\hat{\Gamma}$ is $[-1, 1]$.

Second Approach

The second approach compares the proximity matrix Q to the partitioning P . The Γ statistic (or normalized Γ statistic) index can be computed using the proximity matrix Q and the

Table 17.1. Some indices that measure the degree of similarity between C and P based on the external criteria, where X_{ij} , Y_{ij} are the (i, j) elements of matrices X , Y , respectively, and μ_x , μ_y , $\sigma_x \sigma_y$ are the means and variances of X , Y , respectively.

Index Name	Formula
Rand statistic	$R = \frac{a+d}{M}$
Jaccard coefficient	$J = \frac{a}{a+b+c}$
Folkes and Mallows index	$FM = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}$
Hubert's Γ statistic	$\Gamma = \frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij} Y_{ij}$
Normalized Γ statistic	$\hat{\Gamma} = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n (X_{ij} - \mu_x)(Y_{ij} - \mu_y)}{M\sigma_x\sigma_y}$

matrix Y , where Y is defined as

$$Y_{ij} = \begin{cases} 1 & \text{if } g(x_i) \neq g(x_j), \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i, j = 1, 2, \dots, n,$$

with g being a mapping introduced by the partition P , i.e.,

$$g : X \rightarrow \{1, 2, \dots, n_c\},$$

where n_c is the number of clusters in the partition P .

In both approaches, the Monte Carlo method (see Algorithm 17.1) is employed to compute the probability density function. See P43 in Halkidi et al. (2002a) for a specific example.

In the Monte Carlo analysis, a computer experiment needs to be set up to simulate the generation of data under some hypothesis, such as randomness. For situations when such experiments cannot easily be arranged, bootstrap techniques (Efron, 1979; Efron and Gong, 1983; Jain and Dubes, 1988) are used instead of Monte Carlo analysis.

17.1.3 Internal Criteria

The goal of internal criteria is to evaluate the clustering structure produced by an algorithm using only quantities and features inherited from the data set. To apply the internal criteria, there are two situations: (a) hierarchy of clustering schemes (such as the hierarchical clustering algorithm) and (b) single clustering scheme (such as the partitional algorithm).

The idea to validate the hierarchy of clustering schemes is to use the so-called cophenetic matrix P_c and then to use the cophenetic correlation coefficient to measure the degree of similarity between P_c and the proximity matrix P . The cophenetic matrix P_c is defined in such a way that the element $P_c(i, j)$ represents the proximity level at which the two data points x_i and x_j are found in the same cluster for the first time (note the difference between

agglomerative hierarchy and divisive hierarchy). The cophenetic correlation coefficient index is defined as

$$CPCC = \frac{\frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij} c_{ij} - \mu_P \mu_C}{\sqrt{\left(\frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij}^2 - \mu_P^2 \right) \left(\frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ij}^2 - \mu_C^2 \right)}}, \quad (17.1)$$

where $M = \frac{n(n-1)}{2}$ and μ_P, μ_C are defined as

$$\mu_P = \frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij},$$

$$\mu_C = \frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ij},$$

where d_{ij} and c_{ij} are the (i, j) elements of matrices P and P_c , respectively.

The range of $CPCC$ is $[-1, 1]$; the high value indicates great similarity between P and P_c .

The idea to validate a single clustering scheme is to use Hubert's Γ statistic (or the normalized Γ statistic) index to measure the degree of similarity between a given clustering scheme C and the proximity matrix P . Another matrix in the Γ statistic index is defined as

$$Y_{ij} = \begin{cases} 1 & \text{if } x_i \text{ and } x_j \text{ are in different clusters,} \\ 0 & \text{otherwise.} \end{cases}$$

Since statistical testing approaches have a drawback that they demand high computation, it is necessary to seek other cluster validity criteria, which are described in the next subsection.

17.1.4 Relative Criteria

The basic idea of relative criteria is to choose the best clustering result out of a set of defined schemes according to a predefined criterion (Halkidi et al., 2002b). Let P_{alg} be the set of parameters of a specific clustering algorithm. Then the set of defined schemes is produced by this clustering algorithm using different parameters in P_{alg} . According to whether the parameter n_c (number of clusters) is in P_{alg} or not, we can divide the problem into two cases.

- $n_c \notin P_{alg}$: In this case, to choose the optimal parameter values, the clustering algorithm runs for a wide range of the parameter values and identifies the largest range for which n_c remains constant. Then the values that correspond to the middle of this range are picked as optimal parameter values.
- $n_c \in P_{alg}$: In this case, to choose the best clustering scheme, the following procedure is performed:

```

1: for  $n_c = n_{cmin}$  to  $n_{cmax}$  do
2:   for  $i = 1$  to  $r$  do
3:     Run the clustering algorithm using parameters which are different from in
       the previous running;
4:     Compute the value  $q_i$  of the validity index (selected before);
5:   end for
6:   Choose the best validity index in  $\{q_1, \dots, q_r\}$ ;
7: end for

```

In the following sections, various validity indices are presented. According to whether a data point can belong to two or more clusters, clustering algorithms are divided into two classes: hard clustering (or crisp clustering) and fuzzy clustering. In hard clustering, a data point can only belong to one cluster, while in fuzzy clustering, a data point can belong to two or more clusters.

17.2 Evaluation of Partitional Clustering

In this section, we will present a review of cluster validity indices for hard clustering. Some of the validity indices are defined in the same way as the validity indices that use statistical testing, although we do not need statistical testing but the method specified above.

17.2.1 Modified Hubert's Γ Statistic

We have defined the Hubert Γ statistic before. Here the modified Hubert Γ statistic (Theodoridis and Koutroubas, 1999; Halkidi et al., 2002b) is defined as

$$\Gamma = \frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n P_{ij} Q_{ij},$$

where n is the number of data points in the data set, $M = \frac{n(n-1)}{2}$, P is the proximity matrix of the data set, and Q is a matrix defined by

$$Q_{ij} = d(\mu_{c_i}, \mu_{c_j}), \quad 1 \leq i, j \leq n,$$

with $d(\cdot, \cdot)$ being a distance function (cf. Section 6.2), and μ_{c_i}, μ_{c_j} are the representative points of the clusters that data points i, j belong to.

The normalized Hubert Γ statistic $\hat{\Gamma}$ can be defined in the same way (Halkidi et al., 2002b). From the definition of the modified Hubert Γ statistic, we can see that a high value of Γ or $\hat{\Gamma}$ indicates that there exist compact clusters. Also note that the validity index Γ or $\hat{\Gamma}$ is not defined when the number of clusters is equal to 1 or n .

17.2.2 The Davies-Bouldin Index

The Davies-Bouldin (DB) index (Davies and Bouldin, 1979; Halkidi et al., 2002b) is a validity index that does not depend on the number of clusters and the clustering algorithms.

To define the DB index, we need to define the dispersion measure and the cluster similarity measure.

The dispersion measure S of a cluster C is defined in such a way that the following properties are satisfied:

1. $S \geq 0$;
2. $S = 0$ if and only if $x = y \forall x, y \in C$.

For example, we can define the dispersion measure of cluster C_i as

$$S_i = \left(\frac{1}{|C_i|} \sum_{x \in C_i} d^p(x, c_i) \right)^{\frac{1}{p}}, \quad p > 0, \quad (17.2)$$

where $|C_i|$ is the number of data points in cluster C_i , c_i is the center (or representative data point) of cluster C_i , and $d(x, c_i)$ is the distance between x and c_i .

The cluster similarity measure R_{ij} between clusters C_i and C_j is defined based on the dispersion measures of clusters C_i and C_j and satisfies the following conditions:

1. $R_{ij} \geq 0$;
2. $R_{ij} = R_{ji}$;
3. $R_{ij} = 0$ if and only if $S_i = S_j$;
4. if $S_j = S_k$ and $D_{ij} < D_{ik}$, then $R_{ij} > R_{ik}$;
5. if $S_j > S_k$ and $D_{ij} = D_{ik}$, then $R_{ij} > R_{ik}$.

Here, S_i, S_j, S_k are the dispersion measures of clusters C_i, C_j, C_k , respectively, and D_{ij} is the distance (dissimilarity measure) between the two clusters C_i and C_j , which can be defined as the distance between the centroids of the two clusters (Pal and Biswas, 1997), i.e.,

$$D_{ij} = \left(\sum_{l=1}^d |\mathbf{v}_{il} - \mathbf{v}_{jl}|^t \right)^{\frac{1}{t}}, \quad (17.3)$$

where $\mathbf{v}_i, \mathbf{v}_j$ are the centroids of clusters C_i and C_j , respectively, and $t > 1$.

A very simple choice for R_{ij} is (Davies and Bouldin, 1979)

$$R_{ij} = \frac{S_i + S_j}{D_{ij}}.$$

Then the DB index is defined as

$$V_{DB} = \frac{1}{k} \sum_{i=1}^k R_i,$$

where k is the number of clusters and R_i is defined as

$$R_i = \max_{j \neq i} R_{ij}.$$

Note that p in equation (17.2) and t in equation (17.3) can be selected independently of each other (Pal and Biswas, 1997).

17.2.3 Dunn's Index

The Dunn family of indices proposed in Dunn (1974a,b) was designed to find compact and well-separated (CWS) clusters (Halkidi et al., 2002b). The Dunn index is defined as

$$V_D = \min_{1 \leq i \leq k} \left\{ \min_{i+1 \leq j \leq k} \left(\frac{D(C_i, C_j)}{\max_{1 \leq l \leq k} diam(C_l)} \right) \right\},$$

where k is the number of clusters, $D(C_i, C_j)$ is the distance between clusters C_i and C_j , and $diam(C_l)$ is the diameter of the cluster C_l . Here we can define $D(C_i, C_j)$ and $diam(C_l)$ as

$$\begin{aligned} D(C_i, C_j) &= \min_{x \in C_i, y \in C_j} d(x, y), \\ diam(C_l) &= \max_{x, y \in C_l} d(x, y). \end{aligned}$$

From the definition of the Dunn index, we can conclude that a high value of the index indicates the existence of CWS clusters. If we plot Dunn's index against the number of clusters, the number of clusters that corresponds to the maximum value of the plot is the optimal number of clusters that fits the data set.

Dunn also proposed a second validity index for CWS clusters as (Dunn, 1977)

$$V'_D = \min_{1 \leq i \leq k} \left\{ \min_{1 \leq j \leq k, j \neq i} \left(\frac{D(C_i, conv(C_j))}{\max_{1 \leq l \leq k} diam(C_l)} \right) \right\},$$

where $conv(C)$ is the convex hull (Barber et al., 1996) of the cluster C . This index is not used generally, because $conv(C)$ involves high computation.

There are some disadvantages of Dunn's index, such as it is very sensitive to the presence of noise and it is also time-consuming.

17.2.4 The SD Validity Index

The SD validity index (Halkidi et al., 2000, 2002b) is defined based on average scattering of clusters (i.e., intracluster distance) and total separation between clusters (i.e., intercluster distance). The SD validity index is defined as

$$SD = \alpha S_a + S_t,$$

where α is a weighting factor, S_a is the average scattering of clusters, and S_t is the total separation between clusters.

The average scattering S_a of clusters is defined as

$$S_a = \frac{1}{k} \sum_{i=1}^k \frac{\|\sigma(v_i)\|}{\|\sigma(X)\|},$$

where k is the number of clusters.

The total separation between clusters S_t is defined as

$$S_t = \frac{D_{\max}}{D_{\min}} \sum_{i=1}^k \left(\sum_{j=1}^k \|v_i - v_j\| \right)^{-1},$$

where k is the number of clusters, $D_{\max} = \max_{1 \leq i, j \leq k} \|v_i - v_j\|$, and $D_{\min} = \min_{1 \leq i, j \leq k} \|v_i - v_j\|$.

From the definition of the SD validity index, we can see that the number k that minimizes the SD validity index is the optimal number of clusters that fits the data set.

17.2.5 The S_Dbw Validity Index

Similarly to the SD validity index, the definition of the S_Dbw validity index (Halkidi and Vazirgiannis, 2001) is also based on compactness and separation, i.e., intradistance and interdistance of clusters.

To define the S_Dbw validity index, we need to define the intercluster density and the intracluster variance (Halkidi and Vazirgiannis, 2001). The intercluster density is defined as

$$Dens_bw(k) = \frac{1}{k(k-1)} \sum_{i=1}^k \left(\sum_{j=1, j \neq i}^k \frac{density(C_i \cup C_j)}{\max\{density(C_i), density(C_j)\}} \right),$$

where k is the number of clusters; the function $density(C)$ is defined as

$$density(C) = \sum_{i=1}^{|C|} f(x_i, \mu),$$

where μ is the center of cluster C ; $|C|$ is the number of data points in cluster C ; and the function $f(x, u)$ is defined as

$$f(x, u) = \begin{cases} 0 & \text{if } d(x, u) > stdev, \\ 1 & \text{otherwise.} \end{cases}$$

Here $stdev$ is the average standard deviation of clusters:

$$stdev = \sqrt{\frac{1}{k} \sum_{i=1}^k \|\sigma(C_i)\|^2}.$$

If $C = C_i \cup C_j$, we can take μ as the middle point of the line segment defined by μ_i and μ_j , which are the centers of clusters C_i and C_j , respectively.

The intracluster variance measures the average scattering for clusters. It is defined as

$$Scat(k) = \frac{1}{k} \sum_{i=1}^k \frac{\|\sigma(C_i)\|}{\|\sigma(D)\|},$$

where $\sigma(S)$ is the variance of a data set D , its p th dimension is defined as

$$\sigma(S)_p = \frac{1}{n} \sum_{i=1}^n \left(x_{ip} - \frac{\sum_{j=1}^n x_{jp}}{n} \right)^2, \quad p = 1, 2, \dots, d,$$

where n is the number of data points in D , i.e., $D = \{x_1, \dots, x_n\}$, and d is the dimension of the data points.

Similarly, $\sigma(C_i)$ is the variance of cluster C_i , i.e.,

$$\sigma(C_i)_p = \frac{1}{|C_i|} \sum_{y \in C_i} (y_p - \mu_{ip})^2, \quad p = 1, 2, \dots, d,$$

where μ_{ip} is the p th dimension of the center of cluster C_i and y_p is the p th dimension of data point y in cluster C_i .

The S_Dbw validity index is defined as

$$S_Dbw(k) = Scat(k) + Dens_bw(k).$$

The number k that minimizes the S_Dbw validity index is the optimal number of clusters that fits the data set.

17.2.6 The RMSSTD Index

The root-mean-square standard deviation (RMSSTD) index is used to determine the number of clusters existing in a data set. It measures the homogeneity of the resulting clusters. To define the RMSSTD index, we need to define some notation first.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set in a d -dimensional space. Then the sum of squares of the data set D is defined as

$$SS = \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{y}})^2 = \sum_{i=1}^n \sum_{j=1}^d (\mathbf{x}_{ij} - \bar{\mathbf{y}}_j)^2, \quad (17.4)$$

where $\bar{\mathbf{y}}$ is the mean of the points in D .

Let $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ be a partition of the data set D , and let SS_w , SS_b , SS_t be defined as follows:

SS_w : The sum of squares within a group.

SS_b : The sum of squares between groups.

SS_t : The total sum of squares of the whole data set.

Then the RMSSTD index is defined as (Sharma, 1996; Halkidi et al., 2001a)

$$\begin{aligned} V_{RMSSTD} &= \left(\frac{SS_w}{d(n-k)} \right)^{\frac{1}{2}} \\ &= \left[\frac{\sum_{i=1}^k \sum_{\mathbf{x} \in C_i} \sum_{j=1}^d (\mathbf{x}_{ij} - \mu_{ij})^2}{d(n-k)} \right]^{\frac{1}{2}}, \end{aligned} \quad (17.5)$$

where μ_{ij} is the j th component of μ_i which is the mean of points in C_i , i.e.,

$$\mu_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}.$$

The RMSSTD index defined in (17.5) is a simplified version, in which each attribute is assumed to have the same number of values in the data set. A general RMSSTD index is presented in (Halkidi et al., 2002b).

To validate a nonhierarchical clustering such as k -means, let the algorithm run a number of times for a different number of clusters each time and then plot the RMSSTD validity indices. The optimal number of clusters is the number of clusters at which the “knee” is observed.

To validate a hierarchical clustering, the RMSSTD index has to be used with other indices simultaneously to determine the number of clusters inherent to the data set. The indices to be used with the RMSSTD index are semipartial R-squared (SPR), R-squared (RS), and distance between two clusters (CD)(Sharma, 1996; Halkidi et al., 2002b).

17.2.7 The RS Index

The RS index is defined to be the ratio of SS_b over SS_t , where SS_b and SS_t are defined as in the RMSSTD index.

Let $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ be a partition of a data set D with n data points. Then the RS index is defined as (Sharma, 1996; Halkidi et al., 2002b)

$$\begin{aligned} V_{RS} &= \frac{SS_b}{SS_t} \\ &= \frac{SS_t - SS_w}{SS_t} \\ &= \frac{\sum_{\mathbf{x} \in D} \sum_{j=1}^d (\mathbf{x}_j - \bar{\mathbf{y}}_j)^2 - \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} \sum_{j=1}^d (\mathbf{x}_{ij} - \mu_{ij})^2}{\sum_{i=1}^n \sum_{j=1}^d (\mathbf{x}_{ij} - \bar{\mathbf{y}}_j)^2}, \end{aligned} \quad (17.6)$$

where $\bar{\mathbf{y}}$ is the mean of all the data points in D and μ_{ij} is the j th component of μ_i which is the mean of the data points in C_i .

The RS index defined in (17.6) is also a simplified version of the general RS index which is presented in Halkidi et al. (2002b).

Since $SS_t = SS_b + SS_w$, the greater the differences between groups are, the more homogeneous each group is, and vice versa. Hence, the RS index can be considered as a measure of dissimilarity between clusters.

17.2.8 The Calinski-Harabasz Index

The Calinski-Harabasz (CH) index Calinski and Harabasz (1974) is a validity index defined in terms of the traces of the between-clusters and within-cluster scatter matrices. This index is also discussed in Maulik and Bandyopadhyay (2002).

Let n be the number of data points in the data set and k be the number of clusters. Then the CH index is defined as

$$V_{CH} = \frac{(n - k)\text{Tr}(B)}{(k - 1)\text{Tr}(W)}, \quad (17.7)$$

where $\text{Tr}(B)$ and $\text{Tr}(W)$ are the traces of matrices B and W , respectively, and B and W are the between-clusters and within-cluster scatter matrices (cf. Section 6.1).

The traces of B and W can be written as

$$\begin{aligned} \text{Tr}(B) &= \sum_{i=1}^k |C_i|(\mathbf{z}_i - \mathbf{z})^T(\mathbf{z}_i - \mathbf{z}), \\ \text{Tr}(W) &= \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{z}_i)^T(\mathbf{x} - \mathbf{z}_i), \end{aligned}$$

where \mathbf{z}, \mathbf{z}_i are the means of the entire data set and cluster C_i , respectively.

17.2.9 Rand's Index

Rand (1971) suggests an objective criterion for comparing two arbitrary clusterings based on how pairs of data points are clustered. Given two clusterings, then for any two data points, there are two cases: the first case is that the two points are placed together in a cluster in each of the two clusterings or they are assigned to different clusters in both clusterings; the second case is that the two points are placed together in a cluster in one clustering and they are assigned to different clusters in the other. Based on this, Rand's index is defined as follows.

Given a data set D with n data points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, let $\mathcal{P} = \{C_1, C_2, \dots, C_{k_1}\}$ and $\mathcal{P}' = \{C'_1, C'_2, \dots, C'_{k_2}\}$ be two clusterings of D . Then Rand's index is defined as

$$c(\mathcal{P}, \mathcal{P}') = \frac{1}{\binom{n}{2}} \sum_{1 \leq i < j \leq n} \gamma_{ij}, \quad (17.8)$$

where

$$\gamma_{ij} = \begin{cases} 1 & \text{if } \exists l, l' \text{ s.t. } \mathbf{x}_i \in C_l \cap C'_{l'} \text{ and } \mathbf{x}_j \in C_l \cap C'_{l'}, \\ 1 & \text{if } \exists l, l' \text{ s.t. } \mathbf{x}_i \in C_l \cap C'_{l'} \text{ and } \mathbf{x}_j \notin C_l \cup C'_{l'}, \\ 0 & \text{otherwise.} \end{cases} \quad (17.9)$$

Let n_{ij} be the number of points simultaneously in the i th cluster of \mathcal{P} and the j th cluster of \mathcal{P}' , i.e.,

$$n_{ij} = |C_i \cap C'_j|.$$

Then $c(\mathcal{P}, \mathcal{P}')$ is simplified to be

$$\begin{aligned} c(\mathcal{P}, \mathcal{P}') &= 1 + \frac{1}{\binom{n}{2}} \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} n_{ij}^2 \\ &\quad - \frac{1}{2 \binom{n}{2}} \left[\sum_{i=1}^{k_1} \left(\sum_{j=1}^{k_2} n_{ij} \right)^2 + \sum_{j=1}^{k_2} \left(\sum_{i=1}^{k_1} n_{ij} \right)^2 \right]. \end{aligned} \quad (17.10)$$

We can see that $c(\mathcal{P}, \mathcal{P}')$ is a measure of similarity, ranging from 0 to 1. When $c(\mathcal{P}, \mathcal{P}') = 0$, the two clusterings have no similarities; when $c(\mathcal{P}, \mathcal{P}') = 1$, the two clusterings are identical.

17.2.10 Average of Compactness

Zait and Messatfa (1997) proposed a measure of the quality of a clustering result based on the compactness of all clusters, i.e., the average of compactness.

Given a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ with n records, each of which is described by d attributes, let C_1, C_2, \dots, C_k be the found clusters. Then the average of compactness is defined as

$$c_{ave} = \sum_{i=1}^k \text{Diam}(C_i) \frac{|C_i|}{n}, \quad (17.11)$$

where

$$\text{Diam}(C_i) = \sum_{\mathbf{x}, \mathbf{y} \in C_i} \sum_{j=1}^d \frac{d(\mathbf{x}_j, \mathbf{y}_j)^2}{|C_i|(|C_i| - 1)},$$

and $d(\cdot, \cdot)$ is a general distance defined as

$$d(\mathbf{x}_j, \mathbf{y}_j) = \begin{cases} \frac{\mathbf{x}_j - \mathbf{y}_j}{R_j} & \text{if the } j\text{th attribute is continuous,} \\ \delta(\mathbf{x}_j, \mathbf{y}_j) & \text{if the } j\text{th attribute is categorical,} \end{cases}$$

with $\delta(\cdot, \cdot)$ being defined in (6.22), and R_j the range for the j th attribute if it is continuous, i.e.,

$$R_j = \max_{1 \leq i \leq n} \mathbf{x}_{ij} - \min_{1 \leq i \leq n} \mathbf{x}_{ij}.$$

The “best” clustering result is the one that minimizes the average of compactness c_{ave} . According to this property, the optimal clustering result is that each cluster has exactly one record. Thus to compare two clustering methods, the number of clusters in both clustering methods should be set equal.

17.2.11 Distances between Partitions

Mántaras (1991) proposed two distances between partitions based on information theory. The distances were originally designed to select attributes. It turns out, however, that they can be used to calculate the distances between two partitions (Fujikawa and Ho, 2002).

Let D be a data set and P_A and P_B be two partitions of D . Let the partition P_A have k_1 clusters A_1, A_2, \dots, A_{k_1} and the partition P_B have k_2 clusters B_1, B_2, \dots, B_{k_2} . Let P_i, P_j, P_{ij} , and $P_{j|i}$ be the probabilities defined as

$$\begin{aligned} P_i &= P(A_i) = \frac{|D \cap A_i|}{|D|}, \\ P_j &= P(B_j) = \frac{|D \cap B_j|}{|D|}, \\ P_{ij} &= P(A_i \cap B_j) = \frac{|D \cap A_i \cap B_j|}{|D|}, \\ P_{j|i} &= P(B_j | A_i) = \frac{P(B_j \cap A_i)}{P(A_i)} = \frac{P_{ij}}{P_i}, \\ P_{i|j} &= P(A_i | B_j) = \frac{P(A_i \cap B_j)}{P(B_j)} = \frac{P_{ij}}{P_j} \end{aligned}$$

for $i = 1, 2, \dots, k_1$ and $j = 1, 2, \dots, k_2$.

The average information of the partition P_A , which measures the randomness of the distribution of objects of D over the k_1 clusters of P_A , is defined to be

$$I(P_A) = - \sum_{i=1}^{k_1} P_i \log_2 P_i. \quad (17.12)$$

Similarly, the average information of the partition P_B is defined as

$$I(P_B) = - \sum_{j=1}^{k_2} P_j \log_2 P_j. \quad (17.13)$$

The mutual average information of the intersection of the two partitions $P_A \cap P_B$ is defined as

$$I(P_A \cap P_B) = - \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} P_{ij} \log_2 P_{ij}, \quad (17.14)$$

and the conditional information of P_B given P_A is defined as

$$\begin{aligned} I(P_B | P_A) &= I(P_B \cap P_A) - I(P_A) \\ &= - \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} P_{ij} \log_2 \frac{P_{ij}}{P_i} \\ &= - \sum_{i=1}^{k_1} P_i \sum_{j=1}^{k_2} P_{j|i} \log_2 P_{j|i}. \end{aligned} \quad (17.15)$$

Let $D(P_A, P_B)$ be defined as

$$D(P_A, P_B) = I(P_B | P_A) + I(P_A | P_B). \quad (17.16)$$

Mántaras (1991) has shown that the distance defined in equation (17.16) is a metric distance measure. Mántaras (1991) also proposed a normalized distance defined as

$$D_N(P_A, P_B) = \frac{D(P_A, P_B)}{I(P_A \cap P_B)}, \quad (17.17)$$

which is in $[0, 1]$.

17.3 Evaluation of Hierarchical Clustering

Given a data set, there may exist more than one clustering algorithm that can be used to analyze the data, and as such it is always difficult to choose appropriate clustering algorithms. One way is to evaluate the results of applying the clustering algorithms to the data set. Also, by evaluating the clustering results, one can establish whether they are a valid summary of the data or whether unwarranted or inappropriate structure is being imposed on the data set. Gordon (1996) presented four different types of evaluation for hierarchical clustering algorithms: tests for the absence of structure, and procedures for validating hierarchies, partitions, and individual clusters.

Validation of partitions and individual clusters was discussed in the previous sections of this chapter on evaluation of hard clustering, so it will not be discussed in this section. The other two types of evaluation will be presented in this section.

17.3.1 Testing Absence of Structure

If data do not possess any cluster structure, then analyzing the data by any clustering algorithm is meaningless. Therefore, it is useful to develop tests for the absence of structure in data. Two main classes of null models involving randomness of the pattern matrix and randomness of the dissimilarity matrix have been discussed by Sneath (1967), Bock (1985), Jain and Dubes (1988), and Gordon (1996). One null model for a pattern matrix is that the points are randomly distributed in some region of space; another one is that there is a single group in the data. The former can be referred to as the Poisson model, while the latter can be referred to as the unimodal model (Gordon, 1996).

The Poisson model assumes that the data points under consideration are uniformly distributed in some region A of a d -dimensional space. This assumption is also known as the uniformity hypothesis and the random position hypothesis (Gordon, 1999). The region A can be chosen to be either the unit d -dimensional hypercube or hypersphere (Zeng and Dubes, 1985; Hartigan and Mohanty, 1992) or the convex hull of the points in the data set (Bock, 1985).

Unlike the Poisson model, the unimodal model assumes that the data points under consideration are distributed near a center. There are many possible unimodal distributions, but multivariate normal distributions with identity covariance matrix are usually used (Gordon, 1999). In addition to the Poisson model and the unimodal model, the random permutation model, the random dissimilarity matrix model, and the random labels model are also discussed in Gordon (1999).

Comprehensive reviews of tests of the absence of cluster structure are provided by Bock (1996) and Gordon (1998).

17.3.2 Testing Hierarchical Structures

The methods for testing hierarchical structures of a data set can be classified into three categories: the so-called stability methods, in which the original clusterings are compared with clusterings of modified versions of the data; the analysis of the reduced data set, which is obtained by deleting one or more data points from the original data set via assessing the influence of the deleted objects; and modifications of the set of variables used in the clustering.

In the stability method, modified data can be obtained by adding random errors to each element of a pattern matrix or perturbing the elements of the dissimilarity matrix. Large differences between the clusterings indicate inaccuracy of the clustering results (Gordon, 1996).

As an example of the analysis of a reduced data set, Lanyon (1985) suggested a method in which n separate hierarchical clusterings are obtained by deleting a single, different data point each time from a data set and then synthesizing these results to obtain a single consensus clustering, where n is the number of data points in the data set. In this kind of method, one can also divide a data set into two and compare the clustering of a subset with the clustering of the relevant part of the entire data set.

Some methods involving modifications of the set of variables were motivated by phylogenetic concerns. For example, Felsenstein (1985) used the bootstrap procedure to obtain new data sets by resampling from the set of variables, and synthesized the hierarchical clusterings of these resulting data sets into a majority rule consensus tree from which a final clustering is identified.

17.4 Validity Indices for Fuzzy Clustering

Unlike hard clustering, fuzzy clustering allows each data point to belong to every cluster with a different degree of membership. As with the hard clustering case, we can define a validity index for fuzzy clustering. The objective is to seek clustering schemes where most of the data points in the data set exhibit a high degree of membership in one cluster.

Generally, there are two categories of fuzzy validity index: the category involving only the membership values and the category involving both the membership values and the data values (Halkidi et al., 2001b). In what follows, various fuzzy validity indices are defined and discussed.

17.4.1 The Partition Coefficient Index

Let $U = (u_{li})$ ($1 \leq l \leq c$, $1 \leq i \leq n$) be the membership matrix of a fuzzy c -partition of a data set D with n records, i.e., U satisfies the following conditions:

- $u_{li} \in [0, 1]$ for $1 \leq l \leq c$, $1 \leq i \leq n$;
- $\sum_{l=1}^c u_{li} = 1$, $1 \leq i \leq n$;
- $\sum_{i=1}^n u_{li} > 0$, $1 \leq l \leq c$.

Based on the membership matrix, the partition coefficient (PC) index is defined as (Bezdek, 1981; Trauwaert, 1988)

$$V_{PC} = \frac{1}{n} \sum_{l=1}^c \sum_{i=1}^n u_{li}^2. \quad (17.18)$$

From the definition of V_{PC} in (17.18), we see that the values of the PC index range in $[\frac{1}{c}, 1]$. The closer the value of V_{PC} to $\frac{1}{c}$, the fuzzier the clustering is. The lower value of V_{PC} is obtained when $u_{li} = \frac{1}{c}$ for all l, i .

Since the PC index involves only the membership values, it has some drawbacks. For example, it is sensitive to the fuzzifier (Halkidi et al., 2001b). As the fuzzifier tends to 1, the index gives the same values for all values of c , while as the fuzzifier tends to ∞ , the index exhibits a significant knee at $c = 2$ if one plots $V_{PC}(c)$ against c . Another drawback is the lack of direct connection to the geometry of the data set (Dave, 1996).

If we allow c to vary from 2 to $n - 1$, then an optimal number of clusters is a c^* that maximizes $V_{PC}(c)$ over c (Yang and Wu, 2001), i.e.,

$$V_{PC}(c^*) = \max_{2 \leq c \leq n-1} V_{PC}(c).$$

17.4.2 The Partition Entropy Index

The partition entropy (PE) index is another fuzzy validity index that involves only the membership values. It is defined as (Bezdek, 1974a, 1981)

$$V_{PE} = -\frac{1}{n} \sum_{l=1}^c \sum_{i=1}^n u_{li} \log_a(u_{li}), \quad (17.19)$$

where a is the base of the logarithm and $U = (u_{li})$ is the membership matrix of a fuzzy c -partition.

The values of the PE index V_{PE} range in $[0, \log_a c]$. The closer the value of V_{PE} to 0, the harder the clustering is. The values of V_{PE} close to the upper bound indicate the absence of any clustering structure inherent in the data set or the inability of the algorithm to extract it. The PE index has the same drawbacks as the PC index.

Also, if we let c vary from 2 to $n - 1$, then the optimal number of clusters is a c^* that minimizes $V_{PE}(c)$ over c (Yang and Wu, 2001), i.e.,

$$V_{PE}(c^*) = \min_{2 \leq c \leq n-1} V_{PE}(c).$$

17.4.3 The Fukuyama-Sugeno Index

Fukuyama and Sugeno (1989) proposed a fuzzy validity index based on both the membership values and the data values. Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set and $U = (u_{li})$ be the membership matrix of a fuzzy c -partition of D . Then the Fukuyama-Sugeno (FS) index is defined as

$$V_{FS} = \sum_{i=1}^n \sum_{l=1}^c u_{li}^m (\|\mathbf{x}_i - \mathbf{z}_l\|^2 - \|\mathbf{z}_l - \mathbf{z}\|^2), \quad (17.20)$$

where m is a fuzzifier and \mathbf{z} and \mathbf{z}_l are the mean of the entire data set and the mean of cluster C_l , respectively.

From the definition of the FS index, we see that small values of V_{FS} indicate that CWS clusters are embedded in the data set.

17.4.4 Validity Based on Fuzzy Similarity

Pei and Yang (2000) proposed a clustering validity index based on fuzzy similarity. Let U be a fuzzy c -partition of a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ containing n objects and let A_1, A_2, \dots, A_C represent the pattern prototype of the c -partition. The fuzzy set of the i th pattern is denoted as

$$\tilde{A}_i = \sum_{i=1}^n u_{li}/\mathbf{x}_i, \quad l = 1, 2, \dots, C,$$

where u_{li} is the degree of object \mathbf{x}_i belonging to \tilde{A}_l .

The degree to which \tilde{A}_l is a subset of \tilde{A}_p is defined as

$$S(\tilde{A}_l, \tilde{A}_p) = \frac{M(\tilde{A}_l \cap \tilde{A}_p)}{M(\tilde{A}_l)},$$

where $M(\tilde{A}_j)$ is defined as

$$M(\tilde{A}_j) = \sum_{i=1}^n u_{ji}.$$

Based on the fuzzy subsethood degree $S(\tilde{A}_l, \tilde{A}_p)$, the following three fuzzy similarities are well defined:

$$N_1(\tilde{A}_l, \tilde{A}_p) = \frac{S(\tilde{A}_l, \tilde{A}_p) + S(\tilde{A}_p, \tilde{A}_l)}{2}, \quad (17.21a)$$

$$N_2(\tilde{A}_l, \tilde{A}_p) = \min(S(\tilde{A}_l, \tilde{A}_p), S(\tilde{A}_p, \tilde{A}_l)), \quad (17.21b)$$

$$N_3(\tilde{A}_l, \tilde{A}_p) = S(\tilde{A}_l \cup \tilde{A}_p, \tilde{A}_p \cap \tilde{A}_l). \quad (17.21c)$$

Then clustering validity based on fuzzy similarity can be defined as

$$N_v(U; c) = \max_{1 \leq l \leq C} \max_{1 \leq p \leq C, p \neq l} N(\tilde{A}_l, \tilde{A}_p), \quad (17.22)$$

where the fuzzy similarity $N(\tilde{A}_l, \tilde{A}_p)$ can be chosen as any one of equations (17.21).

A better fuzzy c -partition is a partition that minimizes the fuzzy similarity of fuzzy subsets \tilde{A}_i , $i = 1, 2, \dots, C$. The best clustering minimizes $N_v(U; c)$ in equation (17.22), i.e.,

$$N_v^*(U; c^*) = \min_{1 \leq k \leq C} N_v(U; k).$$

17.4.5 A Compact and Separate Fuzzy Validity Criterion

Xie and Beni (1991) introduced a fuzzy validity criterion based on a validity function that identifies an overall compact and separate fuzzy c -partition without assumptions as to the number of substructures inherent in the data. The validity function depends on the data set, geometric distance measure, distance between cluster centroids, and fuzzy partition generated by any fuzzy clustering algorithms.

Consider a fuzzy c -partition of a data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. Let V_i ($i = 1, 2, \dots, c$) be the centroid of each cluster and μ_{ij} ($i = 1, 2, \dots, c$, $j = 1, 2, \dots, n$) be the fuzzy membership of object \mathbf{x}_j belonging to class i . Now the total variation of data set D with respect to the fuzzy c -partition is defined to be the summation of the variations of all classes, i.e.,

$$\sigma = \sum_{i=1}^c \sigma_i = \sum_{i=1}^c \sum_{j=1}^n d_{ij}^2,$$

where σ_i ($i = 1, 2, \dots, c$) is the variation of class i and d_{ij} is the fuzzy deviation of \mathbf{x}_j from class i , i.e.,

$$d_{ij} = \mu_{ij} \|\mathbf{x}_j - V_i\|,$$

where $\|\cdot\|$ is the usual Euclidean norm.

Then the compactness of the fuzzy c -partition, π , is defined to be the ratio of the total variation to the size of the data set, i.e.,

$$\pi = \frac{\sigma}{n}. \quad (17.23)$$

The separation of the fuzzy c -partition, s , is defined to be the square of the minimum distance between cluster centroids, i.e.,

$$s = d_{min}^2 = \left(\min_{1 \leq i < j \leq c} \|V_i - V_j\| \right)^2 = \min_{1 \leq i < j \leq c} \|V_i - V_j\|^2. \quad (17.24)$$

The compactness and separation validity criterion S is defined as the ratio of the compactness π to the separation s , i.e.,

$$S = \frac{\pi}{s} = \frac{\sum_{i=1}^c \sum_{j=1}^n \|V_i - \mathbf{x}_j\|^2}{n \min_{1 \leq i < j \leq c} \|V_i - V_j\|^2}. \quad (17.25)$$

From the definition of the validity criterion in equation (17.25), one can see that smaller S indicates a more compact and separate c -partition. For each $c = 2, 3, \dots, n-1$, let Ω_c denote the optimality candidates at c . The most valid fuzzy clustering of the data set D is thus the solution of

$$\min_{2 \leq c \leq n-1} \left(\min_{\Omega_c} S \right).$$

17.4.6 A Partition Separation Index

Yang and Wu (2001) proposed a validity index for fuzzy clustering called a partition separation (PS) index, which is based on a normalized partition coefficient and an exponential separation measure.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set in \mathbb{R}^d and μ_{ij} ($i = 1, 2, \dots, c$, $j = 1, 2, \dots, n$) be the fuzzy membership of object \mathbf{x}_j belonging to class i . Then the PS index for class i is defined as

$$PS_i = \sum_{j=1}^n \frac{\mu_{ij}^2}{\mu_M} - \exp\left(-\frac{\min_{l \neq i} \|V_i - V_l\|^2}{\beta_T}\right),$$

where V_i ($i = 1, 2, \dots, c$) is the centroid of cluster i and μ_M and β_T are defined as

$$\begin{aligned}\mu_M &= \max_{1 \leq i \leq c} \sum_{j=1}^n \mu_{ij}^2, \\ \beta_T &= \frac{1}{c} \sum_{i=1}^c \|V_i - \bar{V}\|^2,\end{aligned}$$

where \bar{V} is the arithmetic average of the centroids V_1, V_2, \dots, V_c , i.e., $\bar{V} = \frac{1}{c} \sum_{i=1}^c V_i$.

The PS validity index is defined as

$$PS(c) = \sum_{i=1}^c PS_i.$$

The PS validity index can detect each cluster with two measures of a normalized partition coefficient and an exponential separation. A large PS_i value indicates that the cluster is compact inside the cluster and separate between the other $c - 1$ clusters. An optimal clustering of the data set is

$$PS(c^*) = \max_{2 \leq c \leq n} PS(c).$$

17.4.7 An Index Based on the Mini-max Filter Concept and Fuzzy Theory

Rhee and Oh (1996) introduced a measure G of the quality of clustering that is based on the mini-max filter concept and the fuzzy theory. It measures the overall average compactness and separation of a fuzzy c -partition. Based on this measure, a more suitable measure I_G is introduced to compare the clustering results of one fuzzy c_1 -partition with another c_2 -partition of a data set.

Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a data set and μ_{ij} ($i = 1, 2, \dots, c$, $j = 1, 2, \dots, n$) be the membership of a fuzzy c -partition. Then the overall average intraclass distance, which is defined as the distance between the data points inside a cluster i , is defined as

$$C = \frac{2}{n(n-1)} \sum_{j_1=1}^{n-1} \sum_{j_2=j_1+1}^n \sum_{i=1}^c d^2(\mathbf{x}_{j_1}, \mathbf{x}_{j_2}) \omega_{j_1 j_2},$$

where $d(\cdot, \cdot)$ is the Euclidean distance and $\omega_{j_1 j_2} = \min\{\mu_{i_1 j_1}, \mu_{i_2 j_2}\}$.

The overall average interclass distance, which is defined as the distance between the objects in cluster i_1 and in cluster i_2 ($i_2 \neq i_1$), is defined as

$$D = \frac{1}{n^2} \sum_{j_1=1}^n \sum_{j_2=1}^n d^2(\mathbf{x}_{j_1}, \mathbf{x}_{j_2}) \xi_{j_1 j_2},$$

where $\xi_{j_1 j_2} = \min\{\max_{i_1} \mu_{i_1 j_1}, \max_{i_2 \neq i_1} \mu_{i_2 j_2}\}$.

The compactness and separation validity function G is defined to be the ratio of the separation index D to the compactness index C , i.e.,

$$G = \frac{D}{C}. \quad (17.26)$$

The measurement P which measures the similarity of the population in each cluster is defined as

$$P = 1 - \frac{\sigma}{\sigma_{max}}, \quad (17.27)$$

where σ_{max} is the standard deviation of $\bar{N} = (n, 0, \dots, 0)$ ($c - 1$ zeros); σ is the standard deviation of the fuzzy number vector $N = (n_1, n_2, \dots, n_c)$; and the fuzzy number of objects belonging to cluster i , n_i ($i = 1, 2, \dots, c$), is defined as

$$n_i = \sum_{j=1}^n \mu_{ij}.$$

From this definition, P is proportional to the similarity of the population in each cluster. Particularly, $P = 1$ means that all clusters have equal population, while $P = 0$ means that all objects belong to just one cluster.

Based on G and P , the compactness and separation validity measure, denoted by I_G , is defined as

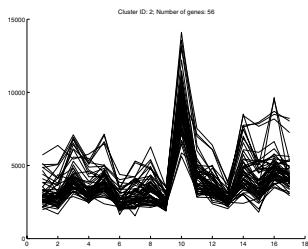
$$I_G = \frac{G}{P},$$

where G and P are defined in equations (17.26) and (17.27), respectively.

The validity measure I_G defined above can be used in selecting the optimal number of clusters. The clustering with the largest I_G is the optimal clustering of the data set.

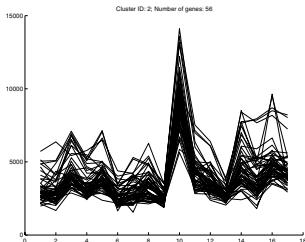
17.5 Summary

Many clustering algorithms have been designed, and thus it is important to decide how to choose a good clustering algorithm for a given data set and how to evaluate a clustering method. The validity indices introduced in this chapter may help with these aspects. Some other validity indices are also introduced for specific purposes, such as the significance test on external variables suggested by Barbará et al. (2002), the category utility (CU) function suggested by Gluck and Corter (1985) and Barbará et al. (2002), and the purity used by Gibson et al. (2000) and Zhang et al. (2000b). Further discussion of evaluation of clustering methods can be found in Sneath (1969), Bock (1996), and Cunningham and Ogilvie (1972).



Part III

Applications of Clustering



Chapter 18

Clustering Gene Expression Data

This chapter introduces the application of clustering to gene expression data. First, we introduce some background information on gene expression data. Then we introduce some applications of gene expression data clustering and two types of gene expression data clustering (gene-based and sample-based). Some guidelines for choosing clustering algorithms and similarity measures to cluster gene expression data are also presented in this chapter. Finally, we present a case study by clustering a real gene expression data set.

18.1 Background

In this section we briefly describe cDNA microarrays and oligonucleotide (oligo) microarrays that generate gene expression data. For more information about microarray technologies and applications, readers are referred to Marshall and Hodgson (1998), Ramsay (1998), and Eisen and Brown (1999).

In genomic research, unlike in the traditional approach which has focused on the local examination and collection of data on single genes, microarray technologies have focused on monitoring the expression levels for a large number of genes in parallel (Jiang et al., 2004). cDNA microarrays (Schena et al., 1995) and oligo microarrays (Lockhart et al., 1996) are two major types of microarray experiments which involve four common basic procedures (Jiang et al., 2004): chip manufacture, target preparation, labeling and hybridization, and the scanning process.

cDNA microarrays that contain large sets of cDNA sequences immobilized on a solid substrate are produced by spotting polymerase chain reaction (PCR) products representing specific genes onto a matrix. In oligo microarrays, each spot on the array contains a short synthetic oligo. In general, both cDNA microarrays and oligo microarrays measure the expression level for each DNA sequence or gene by the ratio of fluorescent intensities between the test sample and the control sample. The measurements collected by both technologies are called gene expression data.

In a microarray experiment, a large number of DNA sequences such as genes, cDNA clones, and expressed sequence tags (ESTs) are assessed under multiple conditions. Currently, the number of genes involved in a typical microarray experiment is about 1,000 to

10,000 and is expected to reach 1,000,000, while the number of samples involved in a typical microarray experiment is usually less than 100 (Jiang et al., 2004). For convenience, we refer to DNA sequences as genes and experimental conditions as samples if no confusion will be caused. A gene expression data set can be represented by a real-valued expression matrix $E = (e_{ij})$ ($1 \leq i \leq n$; $1 \leq j \leq d$) as

$$E = \begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1d} \\ e_{21} & e_{22} & \cdots & e_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1} & e_{n2} & \cdots & e_{nd} \end{pmatrix}, \quad (18.1)$$

where n is the number of genes and d is the number of samples. In the expression matrix E , the rows $\mathbf{g}_i = (e_{i1}, e_{i2}, \dots, e_{id})$ ($i = 1, 2, \dots, n$) form the expression patterns of genes, the columns $\mathbf{s}_j = (e_{1s}, e_{2s}, \dots, e_{ns})^T$ ($s = 1, 2, \dots, d$) form the expression profiles of samples, and the component e_{ij} represents the measured expression level of the i th gene in the j th sample.

Since the original gene expression matrix obtained via a scanning process contains noises, missing values, and systematic variations, data preprocessing is necessary before any cluster analysis can be performed. In general, there are five preprocessing procedures (Herrero et al., 2003): scale transformation (e.g., logtransformation), replicate handling, missing value handling (Kim et al., 2005; Troyanskaya et al., 2001), flat pattern filtering, and pattern standardization (Schuchhardt et al., 2000; Hill et al., 2001). These data preprocessing procedures are not addressed in this book.

18.2 Applications of Gene Expression Data Clustering

DNA microarray technologies have made it possible to monitor the expression levels of a large number of genes simultaneously. However, the large number of genes and the complexity of biological networks have made it difficult to understand and interpret the huge amount of gene expression data. Cluster analysis, which seeks to partition a given set of data points into homogeneous groups based on specified features so that the data points within a group are more similar to each other than to the data points in different groups, has proven to be very useful in understanding gene function and regulation, cellular processes, and subtypes of cells (Jiang et al., 2004). Coexpressed genes, i.e., genes with similar expression patterns, can be clustered together and manifest similar cellular functions (Jiang et al., 2004).

In general, as a powerful tool for studying functional relationships of genes in a biological process, gene expression data clustering has the following applications (Jiang et al., 2004):

- Gene expression data clustering is helpful to understand the functions of genes whose information has not been previously available (Tavazoie et al., 1999; Eisen et al., 1998).
- Gene expression data clustering is likely to identify genes that involve the same cellular processes.

- Gene expression data clustering is helpful to propose cis-regulatory elements (Tava-
zie et al., 1999; Brazma and Vilo, 2000).
- Gene expression data clustering helps to formulate hypotheses about the mechanism
of the transcriptional regulatory network.
- Subcell types, which are difficult to identify by traditional morphology-based ap-
proaches, can be revealed by clustering different samples based on their corresponding
expression profiles (Golub et al., 1999).

18.3 Types of Gene Expression Data Clustering

In gene expression data clustering, it is meaningful to cluster both genes and samples (Jiang et al., 2004). Therefore, there are two types of gene expression data clustering: gene-based clustering and sample-based clustering. In gene-based clustering, genes are treated as objects while samples are treated as features. In other words, in gene-based clustering, genes are partitioned into homogeneous groups. Conversely, in sample-based clustering, samples are treated as objects while genes are treated as features. In such sample-based clustering, samples are partitioned into homogeneous groups.

In gene-based clustering, the goal is to identify coexpressed genes that indicate co-function and coregulation. Gene-based clustering faces the following challenges:

- Determination of the true number of clusters in the data set.
- Capability of handling a high level of background noises arising from the complex
procedures of microarray experiments.
- Capability of handling overlapped clusters.
- Representation of cluster structures.

In sample-based clustering, the goal is to identify the phenotype structures or sub-structures of the samples. In sample-based clustering, some genes whose expression levels strongly correlate with the cluster distinction are referred to as informative genes, while other genes are treated as irrelevant genes or noises in the data set. In sample-based clustering, the signal-to-noise ratio, i.e., the ratio of the number of informative genes over the number of irrelevant genes, is usually smaller than $\frac{1}{10}$ and thus may seriously degrade the quality and reliability of the clustering results. Supervised analysis and unsupervised analysis are two major methods of selecting informative genes to cluster samples (Jiang et al., 2004).

18.4 Some Guidelines for Gene Expression Clustering

Many clustering algorithms have been developed and most of them can be applied to gene expression data. In order to obtain good clustering results, D'haeseleer (2005) summarized the following general guidelines for choosing clustering algorithms for gene expression data.

- Single-linkage clustering performs very badly on most real-world data, including gene
expression data (Handl et al., 2005; Gibbons and Roth, 2002; Costa et al., 2004).

- Complete linkage seems to outperform average linkage (Gibbons and Roth, 2002).
- k -means and self organizing map (SOM) outperform hierarchical clustering (Gibbons and Roth, 2002; Costa et al., 2004).
- SOM may outperform k -means, especially for larger numbers of clusters (Gibbons and Roth, 2002).
- Both Euclidean distance and Pearson's correlation coefficient seem to work well as distance measures (Gibbons and Roth, 2002; Costa et al., 2004). In addition, Euclidean distance may be more appropriate for log ratio data, while Pearson's correlation coefficient may be better for absolute-valued data (Gibbons and Roth, 2002).

Cluster analysis for gene expression data is performed after the original gene expression data is preprocessed. Therefore, the clustering results may be affected by preprocessing procedures such as standardization, missing value handling, and flat pattern filtering (D'haeseleer, 2005). The best way to cluster gene expression data is to use more than one clustering algorithm and compare the results with those on randomized data. Clustering algorithms that may give different results based on different initial conditions should be run several times to find the best solution.

18.5 Similarity Measures for Gene Expression Data

We have presented various similarity and dissimilarity measures in Chapter 6. In this section, we introduce some similarity measures for gene expression data. Some measures in this section were already presented in Chapter 6. To describe the similarity measures for gene expression data, we start with some notation. Let $\mathbf{x} = (x_1, x_2, \dots, x_p)^T$ and $\mathbf{y} = (y_1, y_2, \dots, y_p)^T$ be two numerical vectors that denote two gene expression data objects, where the objects can be either genes or samples and p is the number of features.

18.5.1 Euclidean Distance

Euclidean distance is a commonly used method to measure the dissimilarity or distance between two gene expression data objects. In particular, the Euclidean distance between \mathbf{x} and \mathbf{y} is calculated as

$$d_{euc}(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^p (x_j - y_j)^2 \right)^{\frac{1}{2}}. \quad (18.2)$$

Since Euclidean distance does not score well for shifting or scaled patterns (Wang et al., 2002), each object vector is standardized with zero mean and one variance before Euclidean distance is calculated (Sharan and Shamir, 2000; De Smet et al., 2002).

18.5.2 Pearson's Correlation Coefficient

Pearson's correlation coefficient (Jiang et al., 2003; Tang et al., 2001; Yang et al., 2002a; Tang and Zhang, 2002; D'haeseleer, 2005) is another commonly used tool to measure the

similarity between two gene expression data objects. Mathematically, Pearson's correlation coefficient between \mathbf{x} and \mathbf{y} is defined as

$$r_{pea}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{j=1}^p (x_j - \bar{x})(y_j - \bar{y})}{\sqrt{\sum_{j=1}^p (x_j - \bar{x})^2} \sqrt{\sum_{j=1}^p (y_j - \bar{y})^2}}, \quad (18.3)$$

where \bar{x} and \bar{y} are defined as

$$\begin{aligned}\bar{x} &= \frac{1}{p} \sum_{j=1}^p x_j, \\ \bar{y} &= \frac{1}{p} \sum_{j=1}^p y_j.\end{aligned}$$

The corresponding distance measure of Pearson's correlation coefficient can be defined as (D'haeseleer, 2005)

$$d_{pea}(\mathbf{x}, \mathbf{y}) = 1 - r_{pea}(\mathbf{x}, \mathbf{y})$$

or

$$d_{pea}(\mathbf{x}, \mathbf{y}) = 1 - |r_{pea}(\mathbf{x}, \mathbf{y})|$$

or

$$d_{pea}(\mathbf{x}, \mathbf{y}) = 1 - r_{pea}^2(\mathbf{x}, \mathbf{y}).$$

One drawback of Pearson's correlation coefficient is that it is not robust with respect to outliers (Heyer et al., 1999). For example, it may yield false positives and thus assign a high similarity score to a pair of dissimilar objects (Jiang et al., 2004). In addition, if two objects have a common peak or valley at a single feature, then the correlation will be dominated by this feature no matter how dissimilar the objects are at the remaining features. To deal with this problem, a so-called jackknife correlation (Heyer et al., 1999; Efron, 1982; Wu, 1986) has been proposed. The jackknife correlation between two objects \mathbf{x} and \mathbf{y} is defined as

$$r_{jac}(\mathbf{x}, \mathbf{y}) = \min \{r_{pea}(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}) : j = 1, 2, \dots, p\}, \quad (18.4)$$

where $r_{pea}(\cdot, \cdot)$ is defined in Equation (18.3), and $\mathbf{x}^{(j)}$ and $\mathbf{y}^{(j)}$ are defined as

$$\begin{aligned}\mathbf{x}^{(j)} &= (x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_p)^T, \\ \mathbf{y}^{(j)} &= (y_1, \dots, y_{j-1}, y_{j+1}, \dots, y_p)^T\end{aligned}$$

for $j = 1, 2, \dots, p$.

Another drawback of Pearson's correlation coefficient is that it assumes the object features are approximately Gaussian distributed and thus may not be robust for objects that are not Gaussian distributed (D'haeseleer et al., 1998). To deal with this problem, Spearman's rank-order correlation coefficient has been proposed as the similarity measure (Jiang et al., 2004). This rank-order correlation coefficient does not assume object features

are approximately Gaussian distributed but replaces the numerical expression levels by their corresponding ranks among all conditions. As a result, a significant amount of information is lost. On average, Spearman's rank-order correlation coefficient does not perform as well as Pearson's correlation coefficient (Jiang et al., 2004).

18.6 A Case Study

In this section, we apply the fuzzy subspace clustering (FSC) algorithm (see Section 15.11) to the gene expression data set gathered by Cho et al. (1998). The data set can be obtained from the website

http://genomics.stanford.edu/yeast_cell_cycle/cellcycle.html.

The data set contains 6,600 records, each of which is described by 17 attributes. We saved the data set in a comma-delimited file (i.e., a .csv file) with quotation marks ("") and (") in the gene name being removed. These gene expression data were also studied and discussed in (Tavazoie et al., 1999), (De Smet et al., 2002), (Eisen et al., 1998), and (Reymond et al., 2000).

18.6.1 C++ Code

The pseudocode of the FSC algorithm is described in Algorithm 15.12. Here we discuss some implementation issues of the FSC algorithm for clustering the expression data. The complete code is given in Appendix D.2.

The first thing to cluster a data set is to load the data into memory. We save a data set into a .csv file. The first row and the first column of a typical data file represent the attribute names and record names. An example .csv data file is shown in Example 18.1, where EOF denotes the end of file. Note that there are no commas at the end of any lines.

Example 18.1 The content of an example .csv data file.

```
Gene name, v1, v2, v3, v4
G1, 1, 2, 3, 4
G2, 4, 3, 2, 1
EOF ■
```

The C++ code given in Example 18.3 shows how to read such a .csv data file. How to use this code is shown in the program in Appendix D.2. The function `ReadData` requires four parameters: `filename`, `svLabel`, `svAtt`, and `dmData`. The parameter `filename` contains the name of the data file and its path. (If the data file is located in the same folder as the program file, the path can be ignored.) The parameter `svLabel` contains the record names, such as gene names. The third parameter, `svAtt`, contains attribute names, i.e., feature names. The last parameter, `dmData`, will contain the data matrix after the function is executed. The function needs two global or static variables `n` and `d`, which denote the number of records and the number of attributes, respectively.

Example 18.2 The content of an example .csv data file.

```
Gene name, v1, v2, v3, v4
G1, 1, 2, 3, 4
G2, 4, 3, 2, 1
EOF
```

■

The function given in Example 18.3 first counts the number of attributes and the number of records in the data file. It is very easy to get the wrong number of records in this part. For example, if the function is used to read the .csv file in Example 18.1, the number of records is 2, while if the function is used to read the .csv file in Example 18.2, the number of records will be 1.

After getting the size of the data set, the function then opens the data file again and reads the attribute names, record names, and data. Here we assume the data are numerical. This function can be easily modified to read other types of data such as categorical data.

Note that a .csv data file cannot contain quotation marks ("") and (") when the function is used to read the .csv data file. For example, if the record names or attribute names contain quotation marks, then the function will give wrong results. It is better to check whether quotation marks are present in a data file before reading the data.

Example 18.3 C++ code for reading data from a file.

```
1 int ReadData(
2     const char *filename ,
3     sVector &svLabel ,
4     sVector &svAtt ,
5     dMatrix &dmData
6 ){
7     int i ,j ;
8     ifstream ifsFile ;
9     char buff[MAX_FIELD_BUFFER] ;
10
11    try {
12        ifsFile .open(filename );
13        // count the number of attributes ;
14        i = 1 ;
15        do {
16            if (ifsFile .peek() == ',')
17                i++;
18        } while (ifsFile .get() != '\n');
19        d = i-1; // the first column is record name
20
21        // now count the number of records
22        i = 0 ;
23        do {
24            ifsFile .ignore(MAX_LINE_BUFFER, '\n');
25            i++;
26        } while (!ifsFile .eof());
27        n = i-1; // one return at the end of the last line of the file
28        // NO return after the last line of the file
29        ifsFile .close();
30
31        // open the file again
32        ifsFile .open(filename );
33
34        svAtt = sVector(d,"v");
35        ifsFile .getline(buff, MAX_FIELD_BUFFER, ',');
36        for (i=0; i<d; i++){
37            if (i == d-1){
38                ifsFile .getline(buff, MAX_FIELD_BUFFER, '\n');
```

```

39          // remove '\n'
40          for(j=0;j<sizeof(buff);j++){
41              if(buff[j] == '\n'){
42                  buff[j] = ' ';
43              }
44          svAtt[i] = buff;
45      } else {
46          ifsFile.getline(buff, MAX_FIELD_BUFFER, ' ', ' ');
47          svAtt[i] =buff;
48      }
49  }
50
51 dmData = dMatrix(n);
52 i = 0;
53 while(! ifsFile.eof() && i<n){
54     // get the record name
55     ifsFile.getline(buff, MAX_FIELD_BUFFER, ' ', ' ');
56     svLabel.push_back(buff);
57     for(j=0; j<d; j++){
58         if (j == d-1){ // denotes the end of the line
59             ifsFile.getline(buff, MAX_FIELD_BUFFER, '\n');
60             dmData[i].push_back(atof(buff));
61         } else {
62             ifsFile.getline(buff, MAX_FIELD_BUFFER, ' ', ' ');
63             dmData[i].push_back(atof(buff));
64         }
65     }
66     i++;
67 }
68
69 ifsFile.close();
70 return 0;
71 }
72 catch(...){
73     cout<<"reading -data -error "<<endl;
74     return -1;
75 }
76 } //ReadData()
77

```

The code given in Example 18.4 is the main clustering function of the FSC algorithm. The function `FuzzySub` requires four parameters: `dmData`, `dmW`, `dmZ`, and `nvShip`. The first parameter `dmData` contains the data matrix that will be clustered. The second parameter `dmW` and the third parameter `dmZ` will contain the fuzzy dimension weight matrix and the cluster centers, respectively, after the function is executed. The last parameter will contain the membership information of the clustering after the function is executed.

The function `FuzzySub` calls five other functions: `Initialization`, `GetShip`, `CalObj`, `EstZ`, and `EstW`. The function `FuzzySub` first initializes the fuzzy dimension weight matrix and the cluster centers and estimates the membership for all records. The function then repeats estimating the cluster centers, estimating the fuzzy dimension weight matrix, and estimating the membership until two consecutive objective function values are equal.

Example 18.4 The function `FuzzySub`.

```

1 double FuzzySub(
2     dMatrix &dmData,
3     dMatrix &dmW,

```

```

4     dMatrix &dmZ,
5     nVector &nvShip
6 {
7     int i, j;
8     double dObj, dNewObj, dError;
9
10    Initialization(dmZ, dmW, dmData);
11    GetShip(dmData, dmW, dmZ, nvShip);
12    dObj = CalObj(dmData, dmW, dmZ, nvShip);
13
14    dError = 1.0;
15    while (dError > 0){
16        EstZ(dmData, dmZ, nvShip);
17        EstW(dmData, dmZ, dmW, nvShip);
18        GetShip(dmData, dmW, dmZ, nvShip);
19        dNewObj = CalObj(dmData, dmW, dmZ, nvShip);
20        dError = fabs(dNewObj - dObj);
21        dObj = dNewObj;
22    }
23
24    return dObj;
25 } // FuzzySub()

```

The function `Initialization` (see Example 18.5) initializes the fuzzy dimension weight matrix and the cluster centers. Cluster centers are randomly selected from the original data set and the fuzzy dimension weights for each cluster are initialized to be equal.

Example 18.5 The function for initializing the fuzzy dimension weight matrix and the cluster centers.

```

1 int Initialization(
2     dMatrix &dmZ,
3     dMatrix &dmW,
4     dMatrix &dmData
5 ){
6     int i, j, nIndex;
7
8     for (i=0; i<k; i++){
9         nIndex = (int)(n*rand()/(RAND_MAX+1.0));
10        for (j=0; j<d; j++){
11            dmZ[i][j] = dmData[nIndex][j];
12            dmW[i][j] = 1.0/d;
13        }
14    }
15
16    return 0;
17 } // Initialization()

```

The function `GetShip` (see Example 18.6) returns the membership of each record based on the fuzzy dimension weight matrix and the cluster centers. In this function, each record is assigned to the cluster whose center is nearest to the record in terms of the weighted distance. The function `QuickSort` (see Example 20.5) is called here.

Example 18.6 The function for estimating the membership of each record.

```

1 int GetShip(
2     dMatrix &dmData,

```

```

3   dMatrix &dmW,
4   dMatrix &dmZ,
5   nVector &nvShip
6 ) {
7   int i, j;
8   double dTemp;
9   dVector dvDist;
10  nVector nvIndex;
11
12  dvDist = dVector(k, 0.0);
13  nvIndex = nVector(k, 0);
14
15  for (i=0; i<n; i++) {
16    for (j=0; j<k; j++) {
17      dvDist[j] = Distance(dmW[j], dmZ[j], dmData[i]);
18      nvIndex[j] = j;
19    }
20    dTemp = dvDist[nvShip[i]];
21    QuickSort(dvDist, nvIndex);
22    if (dTemp > dvDist[0])
23      nvShip[i] = nvIndex[0];
24  }
25  return 0;
} // GetShip()

```



The function `CalObj` (see Example 18.7) calculates the objective function value. The objective function value is just the sum of the weighted distances between each record and its nearest center.

Example 18.7 The function for calculating the objective function value.

```

1  double CalObj(
2   dMatrix &dmData,
3   dMatrix &dmW,
4   dMatrix &dmZ,
5   nVector &nvShip
6 ) {
7   int i, j;
8   double dTemp;
9
10  dTemp = 0.0;
11  for (i=0; i<n; i++) {
12    dTemp += Distance(dmW[nvShip[i]], dmZ[nvShip[i]], dmData[i]);
13  }
14
15  return dTemp;
16 } // CalObj()

```



The function `EstZ` (see Example 18.8) estimates the cluster center based on the old centers and the old membership. Since the estimation of new centers does not depend on the fuzzy dimension weight matrix, the fuzzy dimension weight matrix is not required for this function. In fact, the new center of a cluster is just the mean of the cluster.

Example 18.8 The function for estimating the cluster centers.

```

1  int EstZ(
2   dMatrix &dmData,
3   dMatrix &dmZ,

```

```

4     nVector &nvShip
5  ){
6      int i,j,h;
7      double dTemp;
8      nVector nvSize;
9
10     nvSize = nVector(k,0);
11    for(j=0;j<k;j++){
12        for(h=0;h<d;h++){
13            dmZ[j][h] = 0.0;
14        }
15    }
16    for(i=0;i<n;i++){
17        nvSize[nvShip[i]]++;
18        for(h=0;h<d;h++)
19            dmZ[nvShip[i]][h] += dmData[i][h];
20    }
21    for(j=0;j<k;j++){
22        for(h=0;h<d;h++){
23            dmZ[j][h] /= nvSize[j];
24        }
25    }
26
27    return 0;
28 } // EstZ()

```



The function `EstW` given in Example 18.9 estimates the fuzzy dimension weight matrix based on the cluster centers and the old fuzzy dimension weight matrix. The formula for estimating the fuzzy dimension weight matrix is presented in Section 15.11.

Example 18.9 The function for estimating the fuzzy dimension weight matrix.

```

1 int EstW(
2     dMatrix &dmData ,
3     dMatrix &dmZ,
4     dMatrix &dmW,
5     nVector &nvShip
6  ){
7     int j,h;
8     double dTemp;
9     dVector dvVar;
10
11    dvVar = dVector(d,0.0);
12    for(j=0;j<k;j++){
13        dTemp = 0.0;
14        for(h=0;h<d;h++){
15            dvVar[h] = Variance(j,h,dmData,dmZ,nvShip)+epsilon ;
16            dTemp += pow(1/dvVar[h],1/(alpha -1));
17        }
18        for(h=0;h<d;h++)
19            dmW[j][h] = pow(1/dvVar[h],1/(alpha -1))/dTemp;
20    }
21    return 0;
22 } // EstW()

```



Note that except for the parameters in the function `ReadData`, which are not initialized, the parameters in all other functions are initialized before these function are called. For example, the parameters `dmW`, `dmZ`, and `nvShip` in the function `FuzzySub` are initialized before the function `FuzzySub` is called.

18.6.2 Results

The program given in Appendix D.2 is complied by g++ with option -O on a Sun Unix workstation, and the resulting executable program works properly. It seems that the executable program produced by Bloodshed Dev-C++ Version 4.9.9.2 cannot work correctly.

We first run the program with 10 clusters. The resulting 10 clusters are depicted in Figures 18.1, 18.2, 18.3, 18.4, and 18.5. These figures are produced by the MATLAB program given in Example 18.10.

Example 18.10 MATLAB code for plotting clusters.

```

1 function PlotResult
2
3 clear all
4 close all
5 profile on
6
7 % Read data from file
8 dmData=csvread('DataResult.csv');
9
10 % Get the membership from the first column
11 nvShip = dmData(:,1);
12 dmData(:,1) = [];
13 nvX = 1:17;
14
15 for i=1:10
16     nvIndex = find(nvShip == i);
17     figure(i), clf, hold on
18     sTitle = 'Cluster ID:' ;
19     sTitle = [sTitle int2str(i) ';' ];
20     title([sTitle 'Number of genes:' int2str(length(nvIndex))]);
21     plot(nvX,dmData(nvIndex,:),'k');
22     hold off
23     filename = ['Cluster' int2str(i)];
24     fname = ['-f' int2str(i)];
25     print(fname,'-depsc',filename);
26     close;
27 end

```



Cluster 1 and cluster 2 are plotted in Figure 18.1. We can see from Figure 18.1 that all genes in cluster 2 are similar to each other. From Figures 18.1, 18.2, 18.3, and 18.4, we see that clusters 2, 4, 6, and 8 are similar.

The main purpose of this case study is to show how to implement a clustering algorithm and apply it to cluster a data set. Therefore, the program just outputs the membership of each record. Interested readers may modify the program given in Appendix D.2 to include many other functions such as calculation of the variance for each cluster.

18.7 Summary

In this chapter we introduced gene expression data clustering. Gene expression data clustering is a powerful tool for arranging genes according to similarity in their expression patterns. Cluster analysis is also the first step in analyzing gene expression data. Many traditional clustering algorithms such as k -means (see Section 9.1) and SOM (Kohonen, 1989) can be used to cluster gene expression data. For more details, readers are referred to a survey on gene expression data clustering by Jiang et al. (2004) and references therein.

In addition to the background on gene expression clustering, we introduced briefly how to use the FSC algorithm (see Section 15.11) to cluster a gene expression data set. The C++ code of the FSC algorithm is presented. Readers may try other gene expression clustering algorithms such as Adap_Cluster at <http://homes.esat.kuleuven.be/~thijs/Work/Clustering.html>.

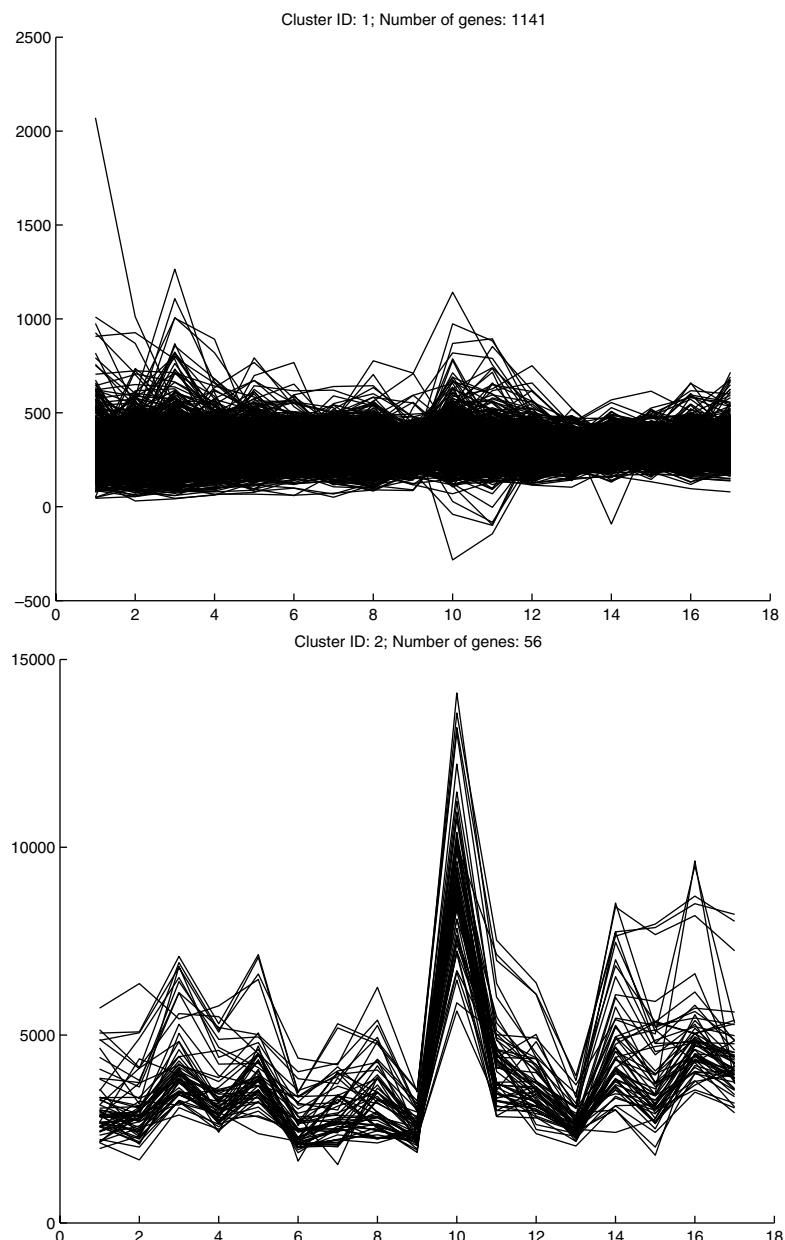


Figure 18.1. Cluster 1 and cluster 2.

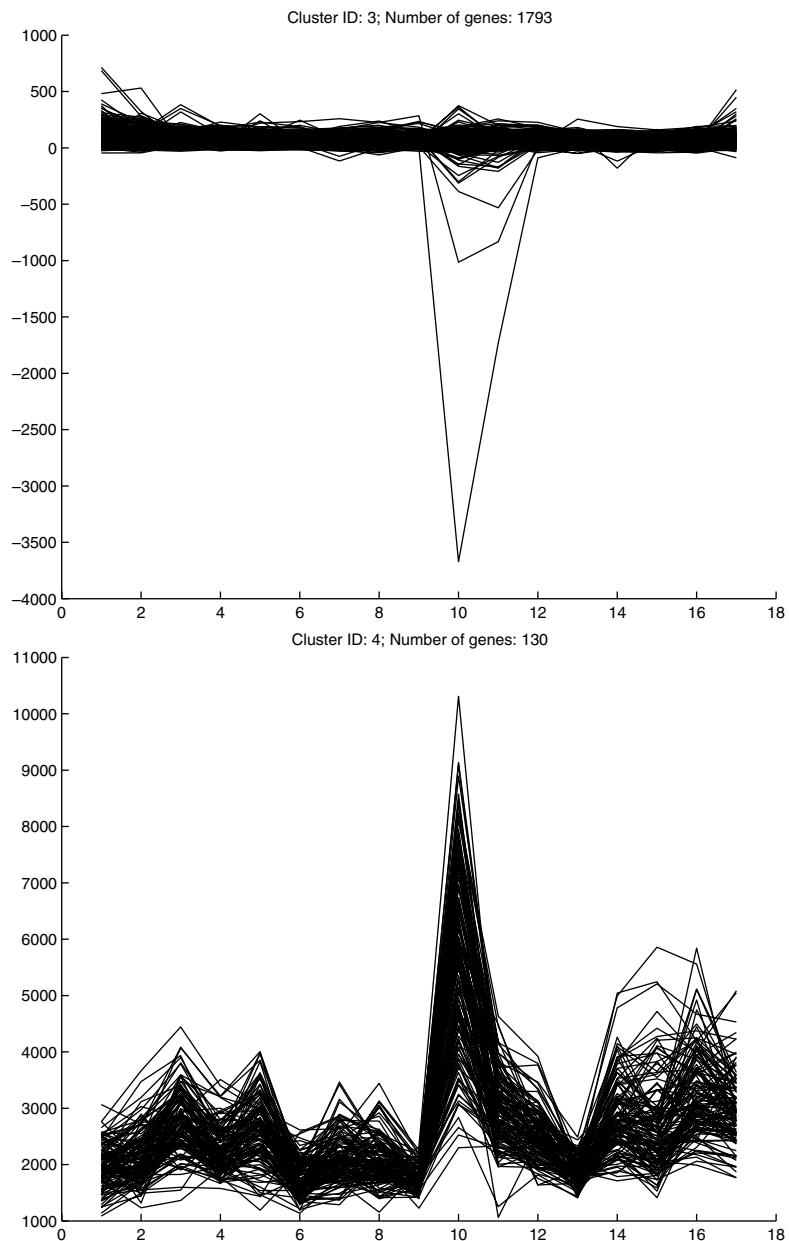


Figure 18.2. Cluster 3 and cluster 4.

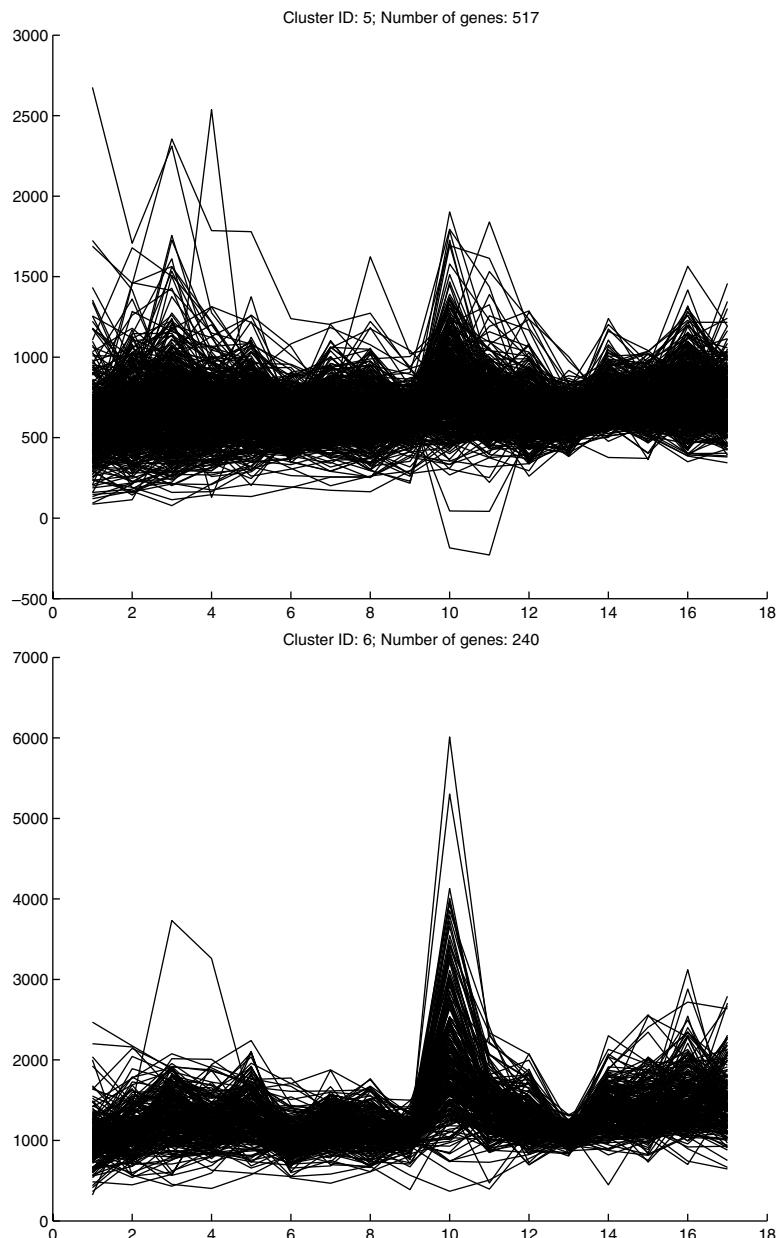


Figure 18.3. Cluster 5 and cluster 6.

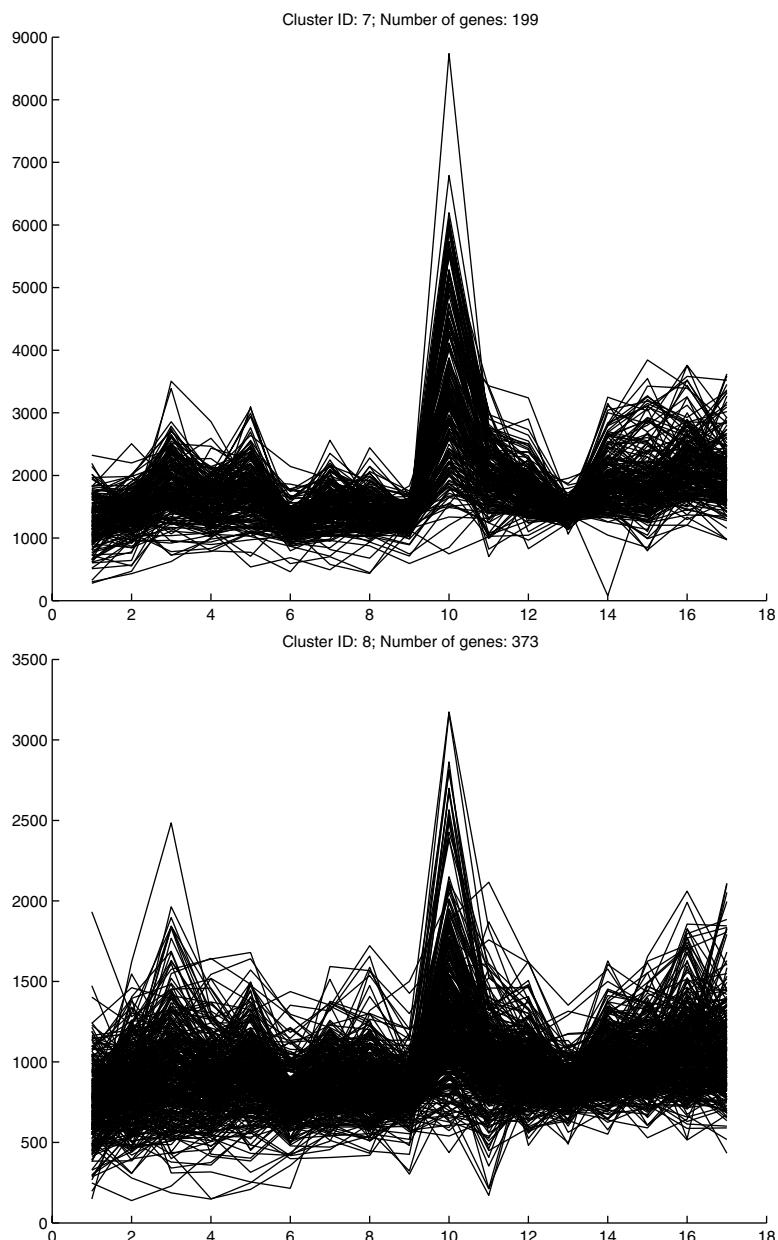


Figure 18.4. Cluster 7 and cluster 8.

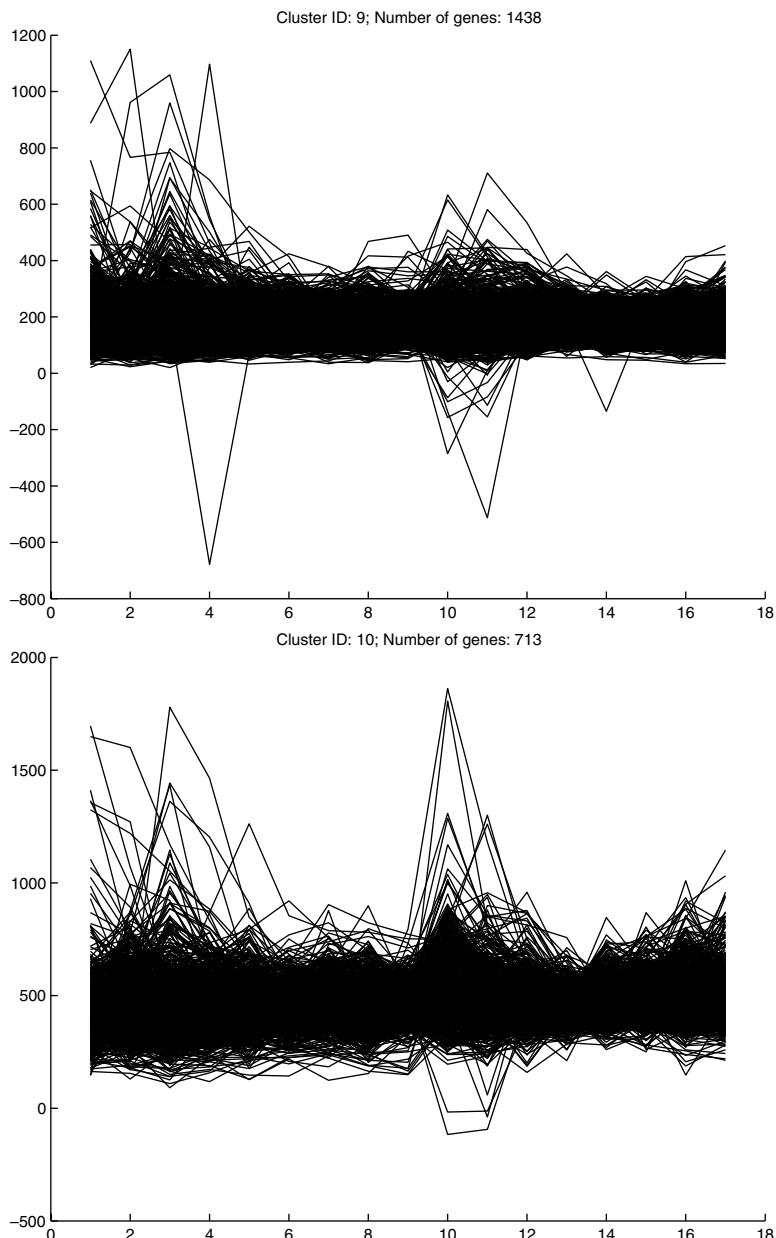
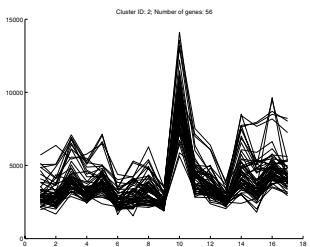
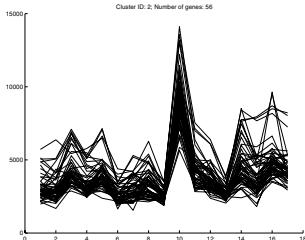


Figure 18.5. Cluster 9 and cluster 10.



Part IV

MATLAB and C++ for Clustering



Chapter 19

Data Clustering in MATLAB

Clustering is a primary tool in data mining. It has been implemented as packages in much software, such SAS, S-PLUS, MATLAB, and other data analysis software. In this chapter, we will focus on clustering in MATLAB. Readers are referred to (SAS Institute Inc., 1983), (Struyf et al., 1996), and (Struyf et al., 1997) for clustering in SAS and S-PLUS.

MATLAB (MATrix LABoratory) is an integrated environment that provides mathematical computing, visualization, and a powerful technical language. It was introduced by The MathWorks.

Using MATLAB to implement a clustering algorithm is convenient and easy. The MATLAB environment enables researchers to plot clustering results on the fly. Some clustering-related toolboxes for MATLAB have been developed, such as the SOM toolbox (Vesanto et al., 2000; Vesanto, 2000; Vesanto et al., 1999a,b), and the Netlab toolbox (Nabney, 2002).

In this chapter, we will introduce some basic MATLAB commands that will be commonly used in implementing a clustering algorithm. Also, we will give a complete example to illustrate how to implement a clustering algorithm in the MATLAB environment. Throughout this chapter, we assume that the reader has basic knowledge of the MATLAB environment. For introductory programming in MATLAB, we refer to (Herniter, 2001).

19.1 Read and Write Data Files

Usually, the data for testing a clustering algorithm are stored in a file or several files. It is not a good idea to input data from the keyboard when the data set is large. The functionalities of reading and writing data files in MATLAB give us many advantages. For example, the test data may be prepared by another program, such as Microsoft Excel, but we can still read the data in MATLAB. Also, we could create test data using MATLAB and write the data to a file that can be used by another program. This allows us to exchange data with others. Table 19.1 gives some commands commonly used in file operations.

Whenever we are reading or writing a file, the first thing we have to do is open the file. The command in MATLAB is `fopen`. The `fopen` command takes two parameters: *filename* and *permission*. The syntax for `fopen` is

Table 19.1. Some MATLAB commands related to reading and writing files.

Command	Description
fopen	Open a file for reading or writing
fclose	Close a file opened by fopen
fprintf	Write formatted text to the device
fscanf	Read formatted text from the device
fwrite	Write binary data to the device
fread	Read binary data from the device
csvwrite	Write a comma-separated value file
csvread	Read a comma-separated value file
dlmwrite	Write a matrix to an ASCII-delimited file
dlmread	Read an ASCII-delimited file into a matrix
textread	Read formatted data from a text file

```
fid = fopen(filename, permission)
```

The first parameter, *filename*, is a text string that contains the file name, while the second parameter, *permission*, is a code that specifies how the file is opened. Variable *fid* is a code that identifies the file just opened. Note that each file you open will be identified by a unique *fid* code. The *fid* codes returned by the *fopen* function start at 3, since -1 through 2 are reserved in MATLAB. Code -1 indicates an error in opening the file. For example, MATLAB will return a code of -1 if you try to open a nonexistent file. Code 0 stands for standard input, code 1 stands for standard output, and code 2 stands for standard error. Some permission codes available in MATLAB are described in Table 19.2.

When you are done with a file, you should always close the file before quitting MATLAB. The command in MATLAB is *fclose*. *fclose* takes one parameter: *fid*. The syntax of this command is

```
status = fclose(fid)
```

The parameter *fid* contains the code of the file to be closed. The code given in Example 19.1 illustrates how to open and close a file in MATLAB.

Example 19.1 File operation in MATLAB.

```

1 >> fid = fopen('data.test','r')
2 fid =
3      3
4 >> fclose(fid)
5 ans =
6      0
7 >> pwd
```

Table 19.2. Permission codes for opening a file in MATLAB.

Permission code	Action
r	Open the file for reading (default).
r+	Open the file for reading and writing.
w	Delete the contents of an existing file or create a new file, and open it for writing.
w+	Delete the contents of an existing file or create a new file, and open it for reading and writing.
a	Create and open a new file or open an existing file for writing, appending to the end of the file.
a+	Create and open a new file or open an existing file for reading and writing, appending to the end of the file.

```

8 | ans =
9 | /home/grads/gjgan/data
10| >> fid2 = fopen( '/home/grads/gjgan/Matlab/data.csv' , 'r' )
11| fid2 =
12|      3
13| >> fclose( fid2 )
14| ans =
15|      0

```



MATLAB provides two commands for writing and reading formatted text to files: `fprintf` and `fscanf`. The usage of the two functions is the same as in the C language. We will not go into detail about the two commands here. The `fprintf` command puts human-readable text in a file whose type is also called an ASCII text file. The disadvantage of this function is that it uses a lot of disk space. If we need to write a large data file that will only be read by the computer, we can use functions for reading and writing binary files.

The functions for writing and reading binary files in MATLAB are `fwrite` and `fread`. `fwrite` takes three parameters: *fid*, *d*, and *precision*. The syntax of `fwrite` is

```
count = fwrite(fid, d, precision)
```

The first parameter, *fid*, is the identifier of the file to which the data will be written. The second parameter, *d*, contains the data to be written. The third parameter, *precision*, is a text string that tells how you want to store the data. The variable *count* contains the number of elements successfully written to the file. Some values of *precision* for the `fwrite` function are given in Table 19.3. For a comprehensive description of this parameter, readers may refer to the MATLAB manual by typing the following command in MATLAB:

```
help fread
```

The `fread` command is used to retrieve the data stored in a binary file. It also takes three parameters: *fid*, *size*, and *precision*. The syntax of `fread` is

Table 19.3. Some values of precision for the fwrite function in MATLAB.

Precision	Description
'char'	8-bit signed or unsigned character
'short'	16-bit integer
'int'	32-bit integer
'long'	32- or 64-bit integer
'float'	32-bit floating point
'double'	64-bit floating point

```
[data, count] = fread(fid, size, precision)
```

The parameter *fid* is the identifier of the file from which we want to read data. The parameter *size* contains the dimensions of the array we want to read. The last parameter *precision* is the same as that in the fwrite function. Note that in order to read data correctly, you need to know exactly how it was written. The variable *data* contains the data retrieved from the file, and the variable *count* contains how many elements were read successfully. The code given in Example 19.2 illustrates how the two functions fwrite and fread work.

Example 19.2 Write and read files in MATLAB.

```

1 >> fid3 = fopen('test.dat', 'w')
2 fid3 =
3     3
4 >> B = rand(2,2)
5 B =
6     0.9501    0.6068
7     0.2311    0.4860
8 >> fwrite(fid3, B, 'double')
9 ans =
10    4
11 >> fclose(fid3)
12 ans =
13     0
14 >> fid4 = fopen('test.dat', 'r')
15 fid4 =
16     3
17 >> [A, count] = fread(fid4, [2,2], 'double')
18 A =
19     0.9501    0.6068
20     0.2311    0.4860
21 count =
22    4
23 >> fclose(fid4)
24 ans =
25     0

```

In many clustering applications, the data are stored in a text file that separates the data fields by commas. The comma is not part of the data. It is only used to separate the various

data fields. A character used in this manner is called a delimiter since it “delimits” the data. The only requirement of a character to be a delimiter is that the character is not used in the data. Some common characters used for delimiters are commas, spaces, semicolons, and tabs.

MATLAB provides two functions for writing and reading numerical data to .csv files. The two functions are `csvwrite` and `csvread`. The two functions are only used for writing and reading numerical data. The `csvwrite` function takes two parameters: *filename* and *m*. The syntax of `csvwrite` is

```
csvwrite(filename, m)
```

The parameter *filename* is the name of the file to which you want to write the matrix. The parameter *m* is the matrix to be written.

The `csvread` function is used to read .csv files. It takes only one parameter: *filename*. The syntax of the `csvread` function is

```
m = csvread(filename)
```

The parameter *filename* is the name of the file from which you want to read data. The variable *m* contains the matrix read from the file. Use of the two functions is shown in Example 19.3.

Example 19.3 Write and read a .csv file in MATLAB.

```
1 >> B = rand(2,2)
2 B =
3     0.8913    0.4565
4     0.7621    0.0185
5 >> csvwrite('test.csv',B)
6 >> A = csvread('test.csv')
7 A =
8     0.8913    0.4565
9     0.7621    0.0185
```



19.2 Handle Categorical Data

MATLAB deals best with purely numerical data. This means that when you have nonnumerical categorical data it is best to convert these data into numerical codes. One way to do this is to replace the categorical symbols by corresponding integers in software such as Microsoft Excel. This is feasible when the data set is not large. When the data set under consideration is very large, one can write a small program to convert the data into numerical codes before clustering in MATLAB. For example, the program in Example 19.4 can be used to convert a categorical data set to a data set of integers and preserve the inherent structures.

Example 19.4 Conversion of categorical values.

```
1 int Cat2Int(
2     sMatrix &smData, // Input categorical data matrix
3     nMatrix &nMData // Output integer matrix
```

```

4  ){
5      int n,d,i,j,s,nTemp;
6      sMatrix smSymbol;
7      sList slTemp;
8      sList sliTemp;
9
10     // Get the dimension of the data matrix
11     n = smData.size();
12     d = smData[0].size();
13
14     // Count distinct attribute values for each attribute
15     smSymbol = sMatrix(d);
16     for(j=0;j<d;j++){
17         for(i=0;i<n;i++){
18             slTemp.push_back(smData[i][j]);
19         }
20         slTemp.sort();
21         slTemp.unique();
22
23         sliTemp = slTemp.begin();
24         while( sliTemp != slTemp.end() ){
25             smSymbol[j].push_back(*sliTemp);
26             sliTemp++;
27         }
28         slTemp.clear();
29     }
30
31     // Convert categorical values to integers
32     nmData = nMatrix(n);
33     for(i=0;i<n;i++){
34         nmData[i] = nVector(d,0);
35     }
36     for(j=0;j<d;j++){
37         nTemp = smSymbol[j].size();
38         for(i=0;i<n;i++){
39             for(s=0;s<nTemp;s++){
40                 if(smSymbol[j][s] == smData[i][j]){
41                     nmData[i][j] = s+1;
42                     break;
43                 }
44             }
45         }
46     }
47
48     return 0;
} // Cat2Int

```



Example 19.5 A categorical data set.

Name,	v1,	v2,	v3
R1,	A,	B,	C
R2,	B,	B,	A
R3,	A,	C,	C
R4,	B,	B,	C
R5,	C,	A,	A



A complete program for converting categorical data sets is given in Appendix D.1. In particular, if one uses the program to convert the data set in Example 19.5, one will get the data set shown in Example 19.6. In this program, the attribute names and record names are not exported to the converted data.

Example 19.6 A converted data set.

```
1,2,2  
2,2,1  
1,3,2  
2,2,2  
3,1,1
```

 ■

19.3 M-files, MEX-files, and MAT-files

M-files, MAT-files, and MEX-files are three special files that one deals with in MATLAB. It is helpful to get to know what these files are and how they work in MATLAB programming.

19.3.1 M-files

M-files are ordinary text (ASCII) files to which one can save MATLAB commands. You can create M-files using any of your favorite text editors such as emacs, pico, vi, etc. from the UNIX shell. All M-files have the extension “.m.” Comments can be inserted with “%.” Only the output from the commands gets displayed when an M-file is run. One can type echo on at the MATLAB prompt before running an M-file script to see the input commands and comments as the script M-file is run. By typing what at the MATLAB prompt, one can see what M-files and MAT-files are available in the current directory.

M-files can be further classified into two categories: script M-files and function M-files. A script M-file consists of just a collection of MATLAB commands or scripts. For example, if you collect a sequence of commands that you input during an interactive session with MATLAB and save them in a file with extension “.m,” then that file becomes a script M-file. You could run a script M-file by just typing its name without the extension “.m” at the MATLAB command prompt.

Example 19.7 An example M-file.

```
1 function egmfile1  
2 % egmfile1.m  
3 % An example script M-file  
4  
5 % Define a matrix  
6 A = [1, 2; 3, 4]  
7  
8 % Calculate the determinant of matrix A  
9 det(A)
```

 ■

Example 19.7 gives a script M-file. If you save this file to your current directory, and then type what and the file name egmfile1 at the MATLAB command prompt, you will see the results as follows.

```
1 >> what  
2 M-files in the current directory /home/grads/gjgan/Matlab  
3 distance egmfile1 kmodes  
4 >> egmfile1
```

```

5 A =
6 1 2
7 3 4
8 ans =
9 -2

```

Function M-files are M-files that accept input arguments and give out output values after they are called. The internal variables in a function M-file are local to the function and invisible to the main MATLAB workspace. The function definition line in a function M-file consists of four parts: *keyword*, *output arguments*, *function name*, and *input arguments*. The function line can be described as follows:

```
function [o1,o2,...] = fname(i1,i2...)
```

where *function* is the *keyword*, *o1*, *o2*, ... are output arguments, *fname* is the name of the function, and *i1*, *i2*, ... are input arguments. Usually, the name of the function M-file is chosen as the name of the function. One can type `help fname` at the MATLAB prompt to get the comments in the function M-file “*fname.m*.”

Example 19.8 An example function M-file for computing the distance between two points.

```

1 function d=distance(A,B,f)
2 % function d=distance(A,B,f)– This function calculates the
3 % distance between two records A and B. f is a parameter
4 % which indicates which distance measure will be used.
5 %
6 % d = distance between A and B, if A,B,f are valid inputs
7 % d = -1 if A and B are of different dimensions
8 % d = -2 if f is not a valid parameter
9 %
10 % f = 1 Euclidean distance
11 % f = 2 squared Euclidean distance
12
13 dimA = size(A);
14 dimB = size(B);
15
16 if(dimA(2) ~= dimB(2))
17     d = -1;
18     return;
19 end
20
21 d = 0.0;
22 switch f
23 case 1
24     for j=1:dimA(2)
25         d = d + (A(j)-B(j))*(A(j)-B(j));
26     end
27     d = sqrt(d);
28 case 2
29     for j=1:dimA(2)
30         d = d + (A(j)-B(j))*(A(j)-B(j));
31     end
32 otherwise
33     fprintf('Invalid f parameter.\n');
34     d = -2;
35 end

```

The M-file given in Example 19.8 is a function M-file. The function *distance* has three input arguments and one output argument. If you type the sequence of commands

help distance, $x=[1, 2, 3]$, $y=[3, 2, 1]$, $\text{distance}(x, y, 1)$, $\text{distance}(x, y, 2)$, and $\text{distance}(x, y, 3)$ at the MATLAB command prompt, you will see the following results.

```

1 >> help distance
2
3 function d=distance(A,B,f) % This function calculates the
4 % distance between two records A and B. f is a parameter
5 % which indicates which distance measure will be used.
6
7 d = distance between A and B, if A,B,f are valid inputs
8 d = -1 if A and B are of different dimensions
9 d = -2 if f is not a valid parameter
10
11 f = 1 Euclidean distance
12 f = 2 squared Euclidean distance
13
14 >> x=[1,2,3];
15 >> y=[3,2,1];
16 >> distance(x,y,1)
17 ans =
18 2.8284
19 >> distance(x,y,2)
20 ans =
21 8
22 >> distance(x,y,3)
23 Invalid f parameter.
24 ans =
25 -2

```

It is your responsibility to handle all possible invalid input arguments in a function. If you try $\text{distance}(x, y, 1)$ for $x=[1, 2]$ and $y=[1, 2, 3]$, you will get output -1 .

19.3.2 MEX-files

MATLAB scripts are very slow when it comes to loops. Sometimes, however, the loops can be avoided by vectorizing the code. In the cases where one has to use loops (such as when a computation is not vectorizable) and speed is important, it is possible to rewrite the MATLAB script in C language through a MEX-file (MATLAB EXecutable file). MEX-files enable the programmer to write efficient code in C without losing the flexibility of the interpreted language. Everything you can do from an M-file, you can also do from a MEX-file. You can also write MEX-files in C++ and FORTRAN, which is very similar.

MEX-files have some disadvantages. One limitation of MEX-files is that they are operating system dependent (Kuncicky, 2004). Another limitation is that data structure conversions from MATLAB to C require an understanding of MATLAB data storage methods. For a different platform, the extensions for the MEX- files are different. Table 19.4 gives some MEX-file extensions for different platforms.

Example 19.9 A simple example MEX-file hello.c written in C.

```

1 /* A simple MEX-file: hello.c*/
2
3 #include "mex.h"
4
5 void mexFunction(int nlhs, mxArray *plhs[],
6                  int nrhs, const mxArray *prhs[]){
7     mexPrintf("Hello, MEX-files!\n");
8 }

```



Table 19.4. MEX-file extensions for various platforms.

<i>Platform</i>	<i>MEX-file Extension</i>
Alpha	mexaxp
HP, Version 10.20	mexhp7
HP, Version 11.x	mexhpux
IBM RS/6000	mexrs6
Linux	mexglx
SGI, SGI64	mexsg
Solaris	mexsol
Windows	dll

Each MEX-file must have a function called `mexFunction` with a predefined set of parameters. The code given in Example 19.9 is a very simple MEX-file. The `mexFunction` has four parameters: `nlhs`, the number of left-hand side parameters; `plhs[]`, the array of left-hand side parameters; `nrhs`, the number of right-hand side parameters; and `prhs[]`, the array of right-hand side parameters. `plhs[]` and `prhs[]` are arrays of pointers to `mxArray` structures that encapsulate MATLAB arrays. Readers may refer to the “MATLAB Application Program Interface Reference,” which lists many functions to deal with the `mxArray`. To compile the MEX-file *hello.c* given in Example 19.9, type

```
mex hello.c
```

at the MATLAB command prompt. This will create a MEX-file called *hello* with an appropriate extension that depends on the platform. The MEX-file can be executed in the same way you would execute an M-file. To execute the MEX-file *hello.c*, type *hello* at the MATLAB command prompt, and you will get “Hello, MEX-files!” displayed.

Example 19.10 An example function M-file and its corresponding MEX-file.

```

1  function s = msum(n)
2  % Function s=msum(n) - This function calculates the
3  % sum of 1,2,\ldots,n.
4  % The argument n must be a positive integer.
5  %
6  % s = -1 if n is an invalid input
7  % s = n*n*(n+1)*(n+1)/4 if n is valid
8  %
9
10 if n<= 0
11     s = -1;
12     return;
13 end
14
15 if n~=floor(n)
16     s = -1;
17     return;
18 end
19
20 s = 0;
21 for i=1:n

```

```

22     for j=1:n
23         s = s+i*j;
24     end
25 end

```

```

1 /*
2  * mxsum.c
3  */
4
5 #include "mex.h"
6
7 void mexFunction(int nlhs, mxArray *plhs[],
8                  int nrhs, const mxArray *prhs[])
9 {
10    double s=0;
11    double *y;
12    int n,i,j;
13    /*Check for proper number of input and output arguments */
14    if(nrhs!=1){
15        mexErrMsgTxt("One input argument is required.");
16    }
17    if(nlhs>1){
18        mexErrMsgTxt("Too many output arguments.");
19    }
20
21    n = *mxGetPr(prhs[0]);
22    for(i=1;i<n+1;i++){
23        for(j=1;j<n+1;j++){
24            s += i*j;
25        }
26    }
27
28    /*Create an output matrix and put the result in it*/
29    plhs[0] = mxCreateDoubleMatrix(1,1,mxREAL);
30    y = mxGetPr(plhs[0]);
31    *y = s;
32 }

```



The greatest advantage of MEX-files is that they run much faster than the corresponding M-files. Example 19.10 gives an example to compare the running time of a function written in an M-file and the running times of the same function written in a MEX-file. To compare the running times of the M-file and the MEX-file given in Example 19.10, one can use the following script:

```

tic
s1 = msum(500)
t1 = toc
tic
s2 = mxsum(500)
t2 = toc

```

Execution of the above MATLAB code gives the following results (the results are dependent on the computer on which you run MATLAB):

```
>>testmmex
```

```
s1 =
1.5688e+10
t1 =
11.9972
s2 =
1.5688e+10
t2 =
0.1534
```

The time complexity of the function computed in the M-file and the MEX-file is $O(n^2)$, where n is the input parameter. The M-file uses 11.9972 seconds, while the corresponding MEX-file uses only 0.1534 second.

19.3.3 MAT-files

MAT-files are machine-dependent binary data files that MATLAB uses for saving data and workspace to disk. A sort of machine independence is achieved with MAT-files, since they can be ported to different architectures, as MATLAB can identify the architecture information from the MAT-files.

To save the MATLAB workspace to disk, one can just type `save` at the MATLAB command prompt. Then the workspace is saved to “`matlab.mat`” by default. To save the workspace with a different name, one can use `save filename`. To load the workspace into MATLAB, one can use the MATLAB function `load`.

19.4 Speed up MATLAB

Since MATLAB is an interpreted language instead of a compiled language (such as C, C++, or FORTRAN), its speed will almost always lag behind that of a custom program written in a compiled language like C, especially for loops. However, this speed loss is made up for by the ease of code development. In this section, we introduce some ways to speed up MATLAB.

The speed of MATLAB can be increased significantly by careful construction of the scripts. One basic rule to follow when writing a MATLAB script is that one should try to avoid the use of loops in MATLAB. If loops cannot be avoided, one can vectorize the loop or write part of the code as a MEX function (see Section 19.3).

Let us consider an example of calculating the simple matching distance between two categorical objects `x` and `y`. It is natural to construct the following code using the `for` loop in MATLAB:

```
s=0;
for i=1:d
    if (x(i) != y(i))
        s = s+1;
    end
end
```

Table 19.5. Some MATLAB clustering functions.

Function	Description
clusterdata	Construct clusters from data
pdist	Pairwise distance between observations
linkage	Create hierarchical cluster tree
cluster	Construct clusters from linkage output
dendrogram	Plot dendrogram graphs
kmeans	k -means clustering
silhouette	Silhouette plot for clustered data
inconsistent	Calculate the inconsistency coefficient of a cluster tree

For $d = 100,000$, the above code takes about 0.48 second to execute. A vectorized approach is to use the MATLAB function `find`:

```
s = length(find(abs(x-y)>0));
```

This code takes about 0.02 second to execute. Therefore, the vectorized approach is approximately 24 times faster than the natural approach.

If loops in a clustering algorithm cannot be avoided and cannot be vectorized, then one can write the loops in a MEX function. We assume that one uses MATLAB as an assistant while developing an algorithm. After the algorithm is developed, one may write the whole algorithm in a compiled language such as C or C++ to test its performance.

19.5 Some Clustering Functions

MATLAB has some built-in functions for the purpose of clustering. In particular, hierarchical clustering and k -means are implemented in the Statistics Toolbox of MATLAB. Some clustering functions are listed in Table 19.5. This section introduces some functions and clustering examples in MATLAB.

19.5.1 Hierarchical Clustering

In MATLAB, hierarchical clustering of a data set consists of three steps:

- Calculate pairwise distances of objects in the data set. The Statistics Toolbox function `pdist` is used for this purpose.
- Construct a binary, hierarchical cluster tree. The function `linkage` is used to group objects into such a cluster tree based on the pairwise distances calculated in the previous step.
- Cut the hierarchical tree into clusters. In this step, the function `cluster` can be used to create clusters.

Example 19.11 illustrates how to calculate pairwise distances of objects in a data set using the function `pdist`. The function `squareform` transforms a vector of pairwise distances to matrix form. The many options of the function `pdist` are described in Table 19.6. In addition to calculating distances using the options in Table 19.6, the function `pdist` can calculate distances using a user-defined distance function. Example 19.12 illustrates how to customize the distance function to calculate pairwise distances.

Example 19.11 Calculate pairwise distance using `pdist`.

```

1 >> D=[1 1;2 2]
2
3 D =
4
5      1      1
6      2      2
7      2      1
8
9 >> Y = pdist(D)
10
11 Y =
12
13      1.4142    1.0000    1.0000
14
15 >> squareform(Y)
16
17 ans =
18
19      0      1.4142    1.0000
20      1.4142      0      1.0000
21      1.0000    1.0000      0
22
23 >> Y = pdist(D, 'mahalanobis')
24
25 Y =
26
27      2.0000    2.0000    2.0000
28
29 >> squareform(Y)
30
31 ans =
32
33      0      2.0000    2.0000
34      2.0000      0      2.0000
35      2.0000    2.0000      0
36
37 >> Y = pdist(D, 'minkowski',2)
38
39 Y =
40
41      1.4142    1.0000    1.0000

```



Example 19.12 A user-defined distance function.

```

1 function d = distfunction(v,U)
2 %
3 % File name: distfunction.m
4 % User-defined distance function
5 % Here we use Euclidean distance
6 %
7 %
8 % Check whether dimensions match

```

Table 19.6. Options of the function pdist.

Option	Description
euclidean	Euclidean distance (default)
seuclidean	Standardized Euclidean distance
mahalanobis	Mahalanobis distance
cityblock	City block metric
minkowski	Minkowski metric
cosine	One minus the cosine of the included angle between points
correlation	One minus the sample correlation between points
spearman	One minus the sample Spearman's rank correlation
hamming	Hamming distance
jaccard	One minus the Jaccard coefficient
chebychev	Chebychev distance

```

9 | if size(v,2) ~= size(U,2)
10|   fprintf('Dimensions of input parameters do not match !\n');
11| end
12|
13| [nObj nDim] = size(U);
14| d = zeros(nObj,1);
15|
16| % Calculate distances
17| for i=1:nObj
18|   d(i) = sqrt(sum((v-U(i,:)).^2));
19| end

```

If we save the above MATLAB code to an M-file named *distfunction.m*, we can call this distance function in the function pdist as follows.

```

1>> D=[1 1;2 2;1 2]
2
3 D =
4
5     1     1
6     2     2
7     1     2
8
9 >> Y = pdist(D, @distfunction)
10 Warning: The input arguments for caller-defined distance functions have
11 changed beginning in R14. See the help for details.
12 > In pdist at 302
13
14 Y =
15
16      1.4142    1.0000    1.0000

```



The function `linkage` creates a hierarchical tree and its input parameter is a distance vector of length $\frac{n(n-1)}{2}$, where n is the number of objects in the data set. The options to create

Table 19.7. Options of the function `linkage`.

<code>single</code>	Shortest distance (default)
<code>complete</code>	Farthest distance
<code>average</code>	Unweighted average distance (UPGMA)
<code>weighted</code>	Weighted average distance (WPGMA)
<code>centroid</code>	Centroid distance (UPGMC)
<code>median</code>	Weighted center of mass distance (WPGMC)
<code>ward</code>	Inner squared distance (minimum variance algorithm)

a hierarchical tree are described in Table 19.7. These hierarchical methods are discussed in Chapter 7. Example 19.13 shows how to use the function `linkage`.

Example 19.13 Create a hierarchical tree using the function `linkage`.

```

1 >> D=[1 1;2 2;4 4;2 1;3 4;4 3];
2 >> Y = pdist(D);
3 >> Z = linkage(Y)
4
5 Z =
6
7      3.0000    6.0000    1.0000
8      5.0000    7.0000    1.0000
9      1.0000    4.0000    1.0000
10     2.0000    9.0000    1.0000
11     8.0000   10.0000    2.2361
12
13 >> Z = linkage(Y, 'median')
14
15 Z =
16
17      3.0000    6.0000    1.0000
18      2.0000    4.0000    1.0000
19      5.0000    7.0000    1.1180
20      1.0000    8.0000    1.1180
21     9.0000   10.0000    3.2016

```



Once a hierarchical tree is created by the function `linkage`, a dendrogram can be obtained by the command `dendrogram`. For example, let Z be the hierarchical tree in Example 19.13. Then the corresponding dendrogram can be obtained by typing `dendrogram(Z)` in the MATLAB command prompt. The resulting dendrogram is shown in Figure 19.1.

The function `cluster` forms clusters by cutting a hierarchical tree created by the function `linkage`. Example 19.14 illustrates how to construct clusters using the function `cluster`. The result of the function `cluster` is an $n \times 1$ vector whose i th component represents the index of the cluster to which the i th object belongs, where n is the number of objects. In fact, a data set can be clustered by a single function, i.e., `clusterdata`. Example 19.15 shows how to cluster a data set in one step using the function `clusterdata`.

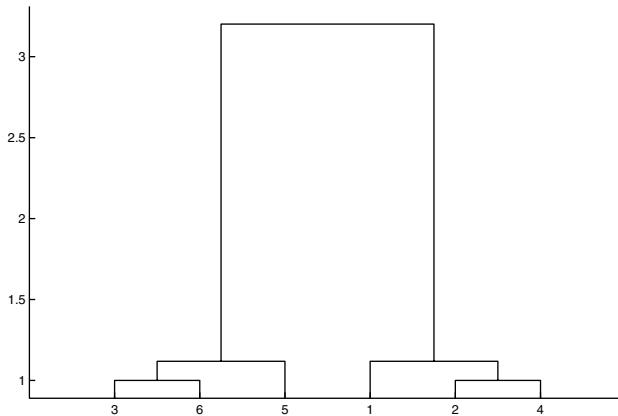


Figure 19.1. A dendrogram created by the function `dendrogram`.

Example 19.14 Construct clusters using the function `cluster`.

```

1 >> D=[1 1;2 2;4 4;2 1;3 4;4 3];
2 >> Y = pdist(D);
3 >> Z = linkage(Y, 'median');
4 >> cluster(Z, 'maxclust',2)
5
6 ans =
7
8     2
9     2
10    1
11    2
12    1
13    1

```



Example 19.15 Construct clusters using the function `clusterdata`.

```

1 >> D=[1 1;2 2;4 4;2 1;3 4;4 3];
2 >> clusterdata(D, 'maxclust',2)
3
4 ans =
5
6     2
7     2
8     1
9     2
10    1
11    1

```



19.5.2 *k*-means Clustering

k-means clustering in MATLAB can be done by the function `kmeans`. In MATLAB, `kmeans` uses a two-phase iterative algorithm to minimize the sum of point-to-centroid distances:

Table 19.8. Values of the parameter *distance* in the function *kmeans*.

Value	Description
sqEuclidean	Squared Euclidean distance (default)
cityblock	Sum of absolute differences
cosine	One minus the cosine of the included angle between points
correlation	One minus the sample correlation between points
Hamming	Percentage of bits that differ (only for binary data)

Table 19.9. Values of the parameter *start* in the function *kmeans*.

Value	Description
sample	Select k objects from D at random (default)
uniform	Select k points uniformly at random from the range of D
cluster	Perform a preliminary clustering phase on a random 10% subsample of D
Matrix	Form the $k \times d$ matrix of starting centers

- (a) In the first phase, points are reassigned to their nearest cluster centroid in each iteration and then cluster centroids are recalculated all at once. This phase provides a fast but potentially only approximate solution that is used as a starting point for the second phase.
- (b) In the second phase, points are individually reassigned if the reassignment will reduce the sum of distances, and cluster centroids are recalculated after each reassignment. In this phase, each iteration consists of one pass through all the points.

The function *kmeans* can be called in the following ways:

- (a) $A = \text{kmeans}(D, k);$
- (b) $[A, B] = \text{kmeans}(D, k);$
- (c) $[A, B, C] = \text{kmeans}(D, k);$
- (d) $[A, B, C, Y] = \text{kmeans}(D, k);$
- (e) $[...] = \text{kmeans}(..., 'param1', val1, 'param2', val2, ...),$

where D is an input (say $n \times d$) data matrix, k is the input number of clusters, A is an $n \times 1$ vector containing clustering indices of each point, B is an output $k \times d$ matrix containing the k cluster centers, C is an output $1 \times k$ vector containing the within-cluster sum of point-to-center distances, and Y is an output $n \times k$ matrix containing distances from each point to every center.

The function *kmeans* can take the following parameters:

- (a) *distance*, the distance measure used in *kmeans*, where the values that this parameter can take are described in Table 19.8;

Table 19.10. Values of the parameter `emptyaction` in the function `kmeans`.

Value	Description
<code>error</code>	Treat an empty cluster as an error (default)
<code>drop</code>	Remove any empty clusters
<code>singleton</code>	Create a new cluster consisting of the one point farthest from its center

Table 19.11. Values of the parameter `display` in the function `kmeans`.

Value	Description
<code>notify</code>	Display only warning and error messages (default)
<code>off</code>	Display no output
<code>iter</code>	Display information about each iteration
<code>final</code>	Display a summary of each run

- (b) `start`, the cluster centers' initialization in `kmeans`, where the available methods of initialization are described in Table 19.9;
- (c) `replicates`, the number of runs of the k -means algorithm;
- (d) `maxiter`, the maximum number of iterations, where the default value is 100;
- (e) `emptyaction`, actions to deal with empty clusters, where the available methods to do so are described in Table 19.10;
- (f) `display`, ways to display output, where the available options of this parameter are described in Table 19.11.

Example 19.16 illustrates how to use the function `kmeans`. In this example, the initial centers are supplied to the function `kmeans`. The algorithm converges in two iterations for this data set.

Example 19.16 k -means clustering using the function `kmeans`.

```

1 >> D=[1 1;2 2;4 4;2 1;3 4;4 3];
2 >> InitCenter=[0 1;5 3];
3 >> [A,B,C,Y] = kmeans(D,2, 'distance ', 'cityblock ', 'start ', InitCenter)
4 2 iterations , total sum of distances = 4
5
6 A =
7
8     1
9     1
10    2
11    1
12    2
13    2
14
15 B =
16
17     2      1
18

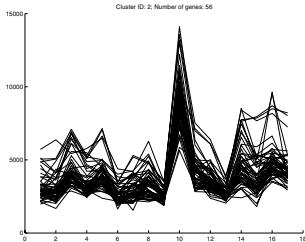
```

```
19      4      4
20
21
22 C =
23
24      2
25      2
26
27
28 Y =
29
30      1      6
31      1      4
32      5      0
33      0      5
34      4      1
35      4      1
```



19.6 Summary

Some popular clustering algorithms have been implemented as built-in functions in MATLAB. Some of these functions are introduced in this chapter with examples. For detailed descriptions of these functions, readers are referred to the help manuals accompanying the MATLAB software or on The MathWorks website. Some clustering-related MATLAB toolboxes and codes can be found at <http://www.fmt.vein.hu/softcomp/fclusttoolbox/>.



Chapter 20

Clustering in C/C++

The C programming language is widely used in both industry and academics. The C++ programming language is an extension of the C language for object-oriented programming. We shall of course focus on cluster analysis in this chapter rather than the programming languages.

This chapter introduces clustering using C/C++. First, we introduce SGI's Standard Template Library (STL) and some classes, including *vector* and *list*. Then we introduce how to compile a C++ program. Finally, we introduce some fundamental elements of implementing clustering algorithms in C/C++. We assume that readers have some basic knowledge of the C/C++ programming languages.

20.1 The STL

The STL is a C++ library of container classes, algorithms, and iterators, which provides many of the basic algorithms and data structures of computer science. The STL includes the classes *vector*, *list*, *deque*, *set*, *multiset*, *map*, *multimap*, *hash_set*, *hash_multiset*, *hash_map*, and *hash_multimap*. Each of these classes is a template and can be instantiated to contain any type of object. For more information, readers are referred to the site <http://www.sgi.com/tech/stl/>.

For the clustering algorithms in this chapter, only two STL classes are used: the *vector* class and the *list* class.

20.1.1 The *vector* Class

The *vector* class is the simplest class of the STL. It supports random access to elements, constant time insertion, and removal of elements at the end and linear time insertion and removal of elements at the beginning and in the middle. The number of elements in a vector may vary dynamically and the memory management is automatic.

To use the *vector* class, one first needs to add the line

```
#include <vector>
```

to the header of the program, and then instantiate the class to contain the desired type of object. For instance, one can use the code

```
typedef vector<int> nVector;
```

to instantiate the class to contain integers.

Example 20.1 The STL *vector* class.

```

1  /*
2   * usevec .cpp
3   */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <vector>
8
9  using namespace std;
10
11 // vectors
12 typedef vector<int> nVector;
13
14 int main(int argc, char *argv[]){
15     int i, nSize;
16     nVector nvExa;
17
18     nSize = 100;
19     nvExa = nVector(nSize, 0);
20     for(i=0;i<nSize;i++)
21         nvExa[i] = i+1;
22     printf("%d\n", nvExa.size());
23
24     for(i=0;i<nSize;i++)
25         nvExa.pop_back();
26     printf("%d\n", nvExa.size());
27
28     for(i=0;i<nSize;i++)
29         nvExa.push_back(i+1);
30     printf("%d\n", nvExa.size());
31
32     return 0;
33 }
```

Compiling and running the code given in Example 20.1 will give the following results:

```
100
0
100
```

Example 20.1 illustrates how to use the STL *vector* class. In this example, four members of the *vector* class are tested. Table 20.1 gives a list of some frequently used members of the *vector* class. A complete list of members of this class can be found at <http://www.sgi.com/tech/stl/Vector.html>.

20.1.2 The *list* Class

The STL *list* class supports both forward and backward traversal and constant time insertion and removal of elements at the beginning or the end or in the middle of the sequence. The important feature of the *list* class is that it can sort and remove its duplicate elements.

Table 20.1. Some members of the vector class.

Member	Description
<code>size()</code>	Returns the size of the vector
<code>vector()</code>	Creates an empty vector
<code>vector(n, & t)</code>	Creates a vector with n copies of t
<code>push_back(const T&)</code>	Inserts a new element at the end
<code>pop_back()</code>	Removes the last element

To use the *list* class, one first needs to add the line

```
#include <list>
```

to the header of the program, and then instantiate the class to contain the desired type of object. For instance, one can use the code

```
typedef list<int> nList;
```

to instantiate the class to contain integers.

Example 20.2 The STL *list* class.

```

1  /*
2   *  useslst .cpp
3   */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <list>
8
9 using namespace std;
10
11 // vectors
12 typedef list<int>          nList;
13 typedef nList::iterator      nIter;
14
15 int main(int argc, char *argv[]){
16     int i, nSize;
17     nList nlExa;
18     nIter ilTemp;
19
20     nSize = 5;
21
22     ilTemp = nlExa.begin();
23     for(i=0;i<nSize;i++){
24         nlExa.push_back(5-i);
25         nlExa.push_back(1+i);
26     }
27     for(ilTemp = nlExa.begin();ilTemp != nlExa.end();ilTemp++)
28         printf("%d", *ilTemp);
29     printf("\n");
30
31     nlExa.sort();
32     for(ilTemp = nlExa.begin();ilTemp != nlExa.end();ilTemp++)
33         printf("%d", *ilTemp);
34     printf("\n");
35
36     nlExa.unique();
37     for(ilTemp = nlExa.begin();ilTemp != nlExa.end();ilTemp++)

```

Table 20.2. Some members of the *list* class.

Member	Description
<code>size()</code>	Returns the size of the list
<code>list()</code>	Creates an empty list
<code>list(n, & t)</code>	Creates a list with n copies of t
<code>push_front(const T&)</code>	Inserts a new element at the beginning
<code>push_back(const T&)</code>	Inserts a new element at the end
<code>pop_front()</code>	Removes the first element
<code>pop_back()</code>	Removes the last element
<code>sort()</code>	Sorts according to operator <
<code>unique()</code>	Removes all but the first element in every consecutive group of equal elements

```

38     printf("%d", *ilTemp);
39     printf("\n");
40
41     return 0;
42 }
```

Compiling and running the code given in Example 20.2 will give the following results:

```

5 1 4 2 3 3 2 4 1 5
1 1 2 2 3 3 4 4 5 5
1 2 3 4 5
```

An example of how to use the STL *list* class is given in Example 20.2. In this small program, several members of the *list* class are tested. Table 20.2 gives a list of some commonly used members of the *list* class. A complete list of members of the *list* class can be found at <http://www.sgi.com/tech/stl/List.html>.

20.2 C/C++ Program Compilation

In this section, we go through the basic processes of compiling a C/C++ program. To create a program, one can use any text editor to create a file containing the complete program, such as the program in Example 20.1 or Example 20.2. The next step is to compile the program using a C++ compiler.

There are many C++ compilers, such as CC and g++ in the UNIX system. Suppose the program in Example 20.1 is saved in a file named *usevec.cpp*. Then under the UNIX system, one can type the command

```
g++ usevec.cpp
```

to compile the program.

To compile the program under the Windows system, one can use Bloodshed Dev-C++, which is a full-featured integrated development environment (IDE) for the C/C++

programming language. Bloodshed Dev-C++ can be downloaded from <http://www.bloodshed.net/>.

20.3 Data Structure and Implementation

This section introduces some basic data structures encountered in implementing a clustering algorithm. To implement a clustering algorithm based on neural networks in the C++ programming language, readers are referred to (Blum, 1992), (Rao and Rao, 1995), and (Pandya and Macy, 1996).

20.3.1 Data Matrices and Centers

The data matrix is the basic element of a clustering algorithm. In C++ programs, it is convenient to use the STL *vector* class to store the data matrix. Suppose the objects in a d -dimensional data set with n objects are arranged in an $n \times d$ matrix such that the first row contains the first object, the second row contains the second object, and so on, i.e.,

$$D = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix},$$

where D is the data matrix and $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})^T$ is the i th object.

To represent the matrix D , we use a vector of vectors using the STL *vector* class. If the data set under consideration is numerical, we use the *double* type. If the data set under consideration is categorical, we first represent categorical values by integers and then use the *int* type.

Example 20.3 (Numerical data matrices). In the following code, the data matrix is initialized to a zero matrix of *double* type.

```
typedef vector<double> dVector;
typedef vector<dVector> dMatrix;

int i;
dMatrix dmData;
dmData=dMatrix(n);

for(i=0;i<n;i++)
    dmData[i]=dVector(d,0.0);
```



Example 20.4 (Categorical data matrices). In the following code, the data matrix is initialized to a zero matrix of `int` type.

```
typedef vector<int> nVector;
typedef vector<nVector> nMatrix;

int i;
nMatrix nmData;
nmData=nMatrix(n);

for(i=0;i<n;i++)
    nmData[i]=nVector(d,0);
```

Example 20.3 and Example 20.4 give the code for initializing the data matrix. After initialization, one can access the data matrix by indices. For example, to get the j th component of the i th object, one can use `dmData [i] [j]`.

Centers of clusters can be represented in the same way. Consider a set of k centers of a numerical data set. We can use a $k \times d$ matrix to represent the k centers such that the first row of the matrix contains the first center, the second row of the matrix contains the second center, and so on. Mathematically,

$$Z = \begin{pmatrix} z_{11} & z_{12} & \cdots & z_{1d} \\ z_{21} & z_{22} & \cdots & z_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ z_{k1} & z_{k2} & \cdots & z_{kd} \end{pmatrix},$$

where Z is the center matrix and $\mathbf{z}_i = (z_{i1}, z_{i2}, \dots, z_{id})^T$ is the i th center.

20.3.2 Clustering Results

The clustering result of a clustering algorithm can be represented by a matrix, such as a fuzzy k -partition (Huang and Ng, 1999). To represent the clustering result of a hard clustering algorithm, an economic way is to use the string-of-group-members encoding (Jones and Beltramo, 1991). Precisely, the string-of-group-members encoding represents the hard k -partition of a set of n objects by a vector of length n with the i th component being the index of the cluster to which the i th object belongs. Mathematically,

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix},$$

where v_i is the index of the cluster to which the i th object belongs. For convenience, we use $0, 1, 2, \dots, k - 1$ to represent the k clusters.

20.3.3 The Quick Sort Algorithm

The quick sort algorithm (Hoare, 1961; Sedgewick, 1978) is very useful in implementing clustering algorithms. In some cases, we need to sort a sequence of real numbers and at the same time keep the positions of these values. To find the maximum element of a sequence of real numbers, for example, one can first sort the sequence in ascending order, and then the last element of the sorted sequence is the maximum element.

Example 20.5 (The quick sort program). The following algorithm is modified from the quick sort algorithm at <http://linux.wku.edu/~lamonml/algor/sort/quick.html>.

```

1  /*
2   * http://linux.wku.edu/~lamonml/algor/sort/quick.html
3   * sorted sequence a_1<=a_2<=...<=a_n
4   */
5  int QuickSort(dVector &dvData , nVector &nvIndex )
6  {
7      double l_hold , r_hold , dTemp;
8      int left , right , nSize , nTemp , nIndex ;
9      nSize = (int)dvData . size ();
10     for(left=0;left < nSize -1;left ++){
11         l_hold = dvData [ left ];
12         r_hold = nvIndex [ left ];
13         nIndex = left ;
14         for(right=left+1;right < nSize ;right ++){
15             if (dvData [ right ] < l_hold ){
16                 l_hold = dvData [ right ];
17                 r_hold = nvIndex [ right ];
18                 nIndex = right ;
19             }
20         }
21         if (left != nIndex ){
22             dTemp = dvData [ left ];
23             dvData [ left ] = l_hold ;
24             dvData [ nIndex ] = dTemp ;
25
26             nTemp = nvIndex [ left ];
27             nvIndex [ left ] = nvIndex [ nIndex ];
28             nvIndex [ nIndex ] = nTemp ;
29         }
30     }
31     return 0;
32 } //QuickSort()

```

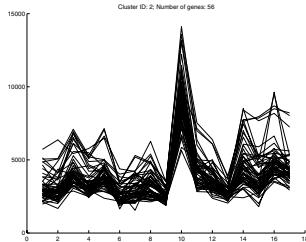


Example 20.5 gives a program for quick sorting a sequence and at the same time keeping the original positions of the real numbers. The sequence of real numbers is represented by a vector.

20.4 Summary

As a general-purpose programming language, C++ is commonly used in cluster analysis. This chapter introduces some basic elements related to implementing a clustering algorithm in C++. Unlike MATLAB programs, C++ programs are hard to debug. For neural networks

and fuzzy logic in C++, readers are referred to (Pandya and Macy, 1996), (Rao and Rao, 1995), and (Blum, 1992).



Appendix A Some Clustering Algorithms

In the following table, some clustering algorithms and their reference locations in the book are presented. The algorithms are listed in alphabetic order.

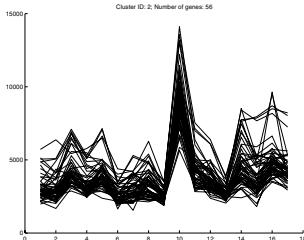
<i>Algorithm</i>	<i>Reference</i>
A dynamic system-based algorithm	Section 11.3 (p. 205)
Al-Sultan's method	Section 10.4 (p. 187)
BIRCH	Subsection 7.4.4 (p. 144)
BRIDGE	Section 13.2 (p. 221)
CACTUS	Section 11.2 (p. 204)
Chameleon	Section 11.1 (p. 203)
CLIQUE	Section 15.1 (p. 244)
CLOPE	Subsection 16.3.2 (p. 295)
CLTree	Section 15.8 (p. 261)
<i>c</i> -means	Section 8.5 (p. 158)
COOLCAT	Section 14.6 (p. 240)
CUBN	Section 13.5 (p. 225)
CURE	Subsection 7.4.5 (p. 144)
DBCLASD	Section 13.3 (p. 222)
DBSCAN	Section 13.1 (p. 219)
DENCLUE	Section 13.4 (p. 223)
DIANA	Subsection 7.4.6 (p. 145)
DISMEA	Subsection 7.4.7 (p. 147)
DOC	Section 15.7 (p. 259)

Continued on next page

Continued from previous page	
Algorithm	Reference
Edwards and Cavalli-Sforza's method	Subsection 7.4.8 (p. 147)
EM	Section 14.4 (p. 235)
ENCLUS	Section 15.4 (p. 253)
FINDIT	Section 15.5 (p. 255)
Fuzzy k -means	Section 8.3 (p. 154)
Fuzzy k -modes	Section 8.4 (p. 156)
Fuzzy subspace clustering	Section 15.11 (p. 270)
Gaussian clustering models	Section 14.2 (p. 230)
GDILC	Section 12.4 (p. 214)
Genetic k -means	Section 10.7 (p. 192)
Global k -means	Section 10.8 (p. 195)
GRIDCLUS	Section 12.3 (p. 212)
J -means	Section 10.6 (p. 190)
k -harmonic means	Section 9.5 (p. 171)
k -means	Section 9.1 (p. 161)
k -modes (Chaturvedi et al.)	Section 9.9 (p. 178)
k -modes (Huang)	Section 9.8 (p. 176)
k -probabilities	Section 9.10 (p. 179)
k -prototypes	Section 9.11 (p. 181)
LargeItem	Subsection 16.3.1 (p. 294)
LSEARCH	Subsection 16.2.1 (p. 290)
Maximum entropy clustering	Section 9.7 (p. 175)
Mean shift	Section 9.6 (p. 173)
Mean shift for subspace clustering	Section 15.12 (p. 275)
OAK	Subsection 16.3.3 (p. 296)
OptiGrid	Section 12.2 (p. 210)
ORCLUS	Section 15.3 (p. 249)
PART	Section 15.9 (p. 262)
PROCLUS	Section 15.2 (p. 246)
ROCK	Section 11.4 (p. 207)
SARS	Section 10.11 (p. 200)
SLINK	Subsection 7.4.1 (p. 138)
SLINK based on minimum spanning trees	Subsection 7.4.2 (p. 140)

Continued on next page

Continued from previous page	
<i>Algorithm</i>	<i>Reference</i>
STING	Section 12.1 (p. 209)
STUCCO	Section 14.7 (p. 241)
SUBCAD	Section 15.10 (p. 264)
Tabu search-based algorithms	Section 10.5 (p. 189)
WaveCluster	Section 12.5 (p. 216)
x -Means	Section 9.4 (p. 170)



Appendix B

The *kd*-tree Data Structure

A *kd*-tree is a data structure for storing a finite set of points from a finite-dimensional space (Bentley, 1975, 1980; Preparata and Shamos, 1985). A *kd*-tree is a binary tree that recursively splits the whole data set into partitions, in a manner similar to a decision tree acting on numerical data sets. Details about *kd*-trees are given in (Moore, 1990) and (Deng and Moore, 1995).

In a *kd*-tree, each node represents a partition of the data space and the children of this node denote the subsets of the partition. Hence the root node of the *kd*-tree is the whole data space, while the leaves are the smallest possible partitions of the data space. Usually, a *kd*-tree is built based on the local density of a data space and represents a recursive subdivision of the whole data space by means of $(d - 1)$ -dimensional hyperplanes which are iso-oriented and alternate among d possibilities. Each splitting hyperplane contains at least one data point which is used for its representation in the tree.

A typical *kd*-tree of a d -dimensional data set has the following properties (Pelleg and Moore, 1999):

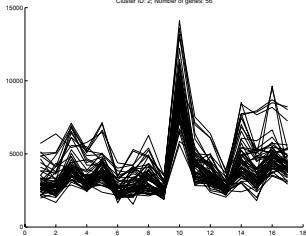
1. It is a binary tree.
2. Each node of the tree contains information about all data points contained in a hyperrectangle h , such as the number of points in h , the center of mass, and the sum of the Euclidean norms of all points contained in h . The hyperrectangle can be stored in the node as two d -dimensional vectors \mathbf{h}_{max} and \mathbf{h}_{min} . Children of the node represent hyperrectangles contained in h .
3. Each interior (nonleaf) node has a “split dimension” j_0 and a “split value” x_0 such that all the data points in h_l (its left child) have their j_0 -dimensional value smaller than x_0 and all the data points in h_r (its right child) have their j_0 -dimensional value at least x_0 .
4. The root node of the tree represents the hyperrectangle that contains all the data points in the data set.
5. Leaf nodes store the actual data points.

The *kd*-tree has some disadvantages. One disadvantage is that the structure is sensitive to the order in which the data points are inserted. Another disadvantage is that the data points are scattered all over the tree (Gaede and Günther, 1998). As a result, the adaptive *kd*-tree (Bentley and Friedman, 1979) was proposed. There is a common disadvantage for all *kd*-trees, that is, no hyperplane can be found that splits the data points evenly for certain distributions (Gaede and Günther, 1998). Other partitioning schemes, such as the *BSP*-tree, the *BD*-tree, and the Quadtree, which can avoid this common disadvantage, are presented in (Gaede and Günther, 1998).

There are some operations associated with *kd*-trees. Searching and insertion of new data points are straightforward operations, but deletion of an existing data point is somewhat more complicated and may cause a reorganization of the subtree below the data point. Algorithms for carrying out these operations are discussed and presented in (Bentley, 1975).

A *kd*-tree can be built in $O(n \log n)$ time and $O(n)$ storage. <http://www.rolemaker.dk/nonRoleMaker/uni/algogem/kdtree.htm>.

The *kd*-tree data structure has been applied to cluster analysis successfully. Examples are using the *kd*-tree in the expectation-maximization (EM) algorithm (Moore, 1999) and accelerating the *k*-means algorithm with the *kd*-tree (Pelleg and Moore, 1999).



Appendix C MATLAB Codes

This appendix presents the MATLAB code for generating subspace clusters, the MATLAB code for the k -modes algorithm (see Section 9.8 and Section 9.9), and the MATLAB code for the MSSC algorithm (see Section 15.12).

C.1 The MATLAB Code for Generating Subspace Clusters

The data generation method introduced by Aggarwal et al. (1999) uses the so-called anchor points to generate clusters embedded in subspaces of a high-dimensional space. To generate k clusters embedded in different subspaces of different dimensions, the method proceeds by first generating k uniformly distributed anchor points $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_k$ in the d -dimensional space. The method then generates the number of dimensions and the number of points associated with each cluster. Finally, it generates points for each cluster and outliers.

The number of dimensions associated with a cluster is generated by a Poisson process with mean μ , with the additional restriction that this number is in $[2, d]$. The dimensions for the first cluster are chosen randomly. Once the dimensions for the $(i - 1)$ th cluster are chosen, the dimensions for the i th cluster are generated inductively by choosing $\min\{d_{i-1}, \frac{d_i}{2}\}$ dimensions from the $(i - 1)$ th cluster and generating other dimensions randomly, where d_i is the number of dimensions for the i th cluster. Given the percentage of outliers $F_{outlier}$ and the size of the data set n , the number of points in the i th cluster is $N_c \cdot r_i / \sum_{j=1}^k r_j$, where r_1, r_2, \dots, r_k are generated randomly from an exponential distribution with mean 1, and $N_c = n(1 - F_{outlier})$.

In the final step, points in each cluster and outliers are generated as follows. For the i th cluster, the coordinates of the points in noncluster dimensions are generated uniformly at random from $[0, 100]$, while the coordinates of the points in a cluster dimension j are generated by a normal distribution with mean at the respective coordinate of the anchor point and variance $(s_{ij}r)^2$, where s_{ij} is a scale factor generated uniformly at random from $[1, s]$ and r is a fixed spread parameter. Outliers are generated uniformly at random from the entire space $[0, 100]^d$. The MATLAB implementation of this data generation method is presented below.

```

1 function GenData
2 %
3 % GeneData.m
4 % Generate subspace clusters
5 %
6
7 % Initialize parameters
8 n = 10000;
9 k = 5;
10 d = 100;
11 Foutlier = 0;
12 mu = 16;
13 r = 2;
14 s = 2;
15
16 Anchor = rand(k,d)*100;
17 NumDim = random('poiss',mu,1,k);
18 Dim = zeros(k,d);
19 index =GenRand(NumDim(1),d);
20 Dim(1,find(index==1))=1;
21 for i=2:k
22     dprev = min(NumDim(i-1),floor(NumDim(i)/2));
23     down = NumDim(i) - dprev;
24     dprevindex = find(GenRand(dprev,NumDim(i-1))==1);
25     dprevdim = find(Dim(i-1,:)==1);
26     Dim(i,dprevdim(dprevindex)) = 1;
27     dotherdim = find(Dim(i,:)==0);
28     ddim = find(GenRand(down,d-dprev)==1);
29     Dim(i , dotherdim(ddim)) = 1;
30 end
31
32 NumPoints = zeros(1,k);
33 Nc = floor(n*(1-Foutlier));
34 nvR = random('exp',1,1,k);
35 for i=1:k-1
36     NumPoints(i) = floor(Nc*nvR(i)/sum(nvR));
37 end
38 NumPoints(k) = Nc-sum(NumPoints);
39
40 data = zeros(n,d+1);
41 nTemp = 0;
42 for i=1:k
43     index = (nTemp+1):(nTemp+NumPoints(i));
44     data(index,1)=i;
45     for j=1:d
46         if Dim(i,j)==0
47             data(index,j+1) = rand(NumPoints(i),1)*100;
48         else
49             sij = rand*(s-1)+1;
50             vRec = random('norm',Anchor(i,j),sij*r,NumPoints(i),1);
51             data(index,j+1) = vRec;
52         end
53     end
54     nTemp = nTemp + NumPoints(i);
55 end
56 data((nTemp+1):n,1)=k+1;
57 data((nTemp+1):n,2:(d+1))=rand(n-Nc,d)*100;
58
59 data = data(randperm(n),:);
60
61 % Print the data set information to a text file
62 fp = fopen('10000data100c.txt','w');
63 fprintf(fp,'Number of points %d, outliers %d', n,n-Nc);
64 for i=1:k
65     fprintf(fp, '\n Cluster %d (%d)\n', i, NumPoints(i));
66     fprintf(fp, 'Dimensions : ');
67     for j=1:d

```

```

68      if Dim(i,j)==1
69          fprintf(fp, '%d', j);
70      end
71  end
72 end
73 fclose(fp);
74
75 % Save the data set to a .csv file
76 csvwrite('10000data100c.csv', data);
77
78 %
79 % Generate nNum nonrepeating integers from 1,2,\ldots,nLen
80 %
81 function Out=GenRand(nNum,nLen)
82 select = zeros(1,nLen);
83 select(floor(rand*nLen+1)) = 1;
84 for i=2:nNum
85     nonselect = find(select==0);
86     nTag = nonselect(floor(rand*(nLen-i+1)+1));
87     select(nTag)=1;
88 end
89 Out = select;

```

C.2 The MATLAB Code for the k -modes Algorithm

There are two versions of the k -modes algorithm, one proposed by Huang (1998) and the other independently proposed by Chaturvedi et al. (2001). The following code is the implementation of the k -modes algorithm proposed by Chaturvedi et al. (2001) in MATLAB. The k -modes algorithm proposed by Chaturvedi et al. (2001) is briefly reviewed in Section 9.9.

```

1 function flag=kmodes(data,k)
2 %
3 % Matlab code for k-Modes algorithm
4 % Reference: Chaturvedi et al. K-Modes Clustering ,
5 % Journal of Classification ,
6 % 18:35–55(2001)
7 %
8 % data – input data matrix
9 % k – input number of clusters
10 %
11
12 % get the dimension of the data matrix
13 dim = size(data);
14 n=dim(1);
15 d=dim(2);
16
17 %Declarations
18 %Memberships, vShip[i]=j means x_i is in the jth cluster
19 vShip=zeros(1,n);
20 mModes=zeros(k,d); %Mode of each cluster
21 Lprev=0; L=0;%Loss function values
22
23 %Initialize modes
24 vrnd = zeros(1,k);
25 vrnd(1) = floor(n*rand+1);
26 mModes(1,:) = data(vrnd(1),:);
27
28 for i=2:k
29     bTag = 0;
30     while bTag==0
31         bTag = 1;
32         j=floor(n*rand+1);

```

```

33     for s=1:(i-1)
34         if j==vrand(s)
35             bTag=0;
36         end
37     end
38     vrnd(i) = j;
39     mModes(i,:)= data(vrnd(i),:);
40 end
41 clear vrnd;
42
43 %Estimate vShip given the initial mModes
44 for i=1:n
45     fprev = length(find(abs(data(i,:)-mModes(1,:))>0));
46     vShip(i) = 1;
47     for s=2:k
48         f = length(find(abs(data(i,:)-mModes(s,:))>0));
49         if fprev>f
50             fprev=f;
51             vShip(i)=s;
52         end
53     end
54     L = L+fprev;
55 end
56
57 %Iteration phase, estimate vShip, estimate mModes
58 Lprev=n*d;
59 while abs(Lprev-L)>0
60     Lprev=L;
61     L=0;
62     %Estimate mModes given the revised vShip
63     for s=1:k
64         index=find(vShip==s);
65         for j=1:d
66             A=sort(data(index,j));
67             [b,m,nn]=unique(A);
68             nL = length(m);
69             nMax = m(1);
70             mModes(s,j)=b(1);
71             for i=2:nL
72                 if (m(i)-m(i-1))>nMax
73                     nMax=m(i)-m(i-1);
74                     mModes(s,j)=b(i);
75                 end
76             end
77         end
78     end
79     %Estimate vShip given the estimate of mModes
80     for i=1:n
81         fprev = length(find(abs(data(i,:)-mModes(1,:))>0));
82         vShip(i) = 1;
83         for s=2:k
84             f = length(find(abs(data(i,:)-mModes(s,:))>0));
85             if fprev>f
86                 fprev=f;
87                 vShip(i)=s;
88             end
89         end
90         L = L+fprev;
91     end
92     Lprev
93     L
94 end
95 flag=vShip;
96

```

C.3 The MATLAB Code for the MSSC Algorithm

The mean shift for subspace clustering (MSSC) algorithm is proposed by Gan (2006). This MATLAB program is used to demonstrate the MSSC algorithm graphically. This code is tested in MATLAB 7 (R14).

```

1 function mssc
2 %
3 % mssc.m
4 % The mean shift for subspace clustering (MSSC) algorithm
5 %
6 clear all
7 close all
8 profile on
9
10 % Generate a two-dimensional data set
11 n = 100;
12 data = zeros(3*n,3);
13 for i=1:n
14     data(i,3) = 1 + 4*i/n;
15     data(i,2) = 3 + rand;
16     data(i,1) = 1;
17     data(n+i,1) = 2;
18     data(n+i,2) = 3 + rand;
19     data(n+i,3) = 8 + 4*i/n;
20     data(2*n+i,1) = 3;
21     data(2*n+i,2) = 6 + rand;
22     data(2*n+i,3) = 1 + 11*i/n;
23 end
24 index =randperm(3*n);
25 data = data(index ',:);
26
27 label=data (:,1)';
28 data (:,1)=[] ;
29 [n,d] = size(data);
30
31 alpha = 2.1;
32 beta = 0.4;
33 k = 8;
34 covMat = cov(data);
35 varMat = diag(covMat).^( -1/(alpha - 1));
36 vInitW = varMat/sum(varMat);
37 betac = 1/(2*max(eig(repmat(vInitW.^alpha ,d ,1).*covMat)));
38
39 [W,Z,vShip ,vUni ] = MSSC(data ,alpha ,beta ,k ,1);
40
41 % Misclassification matrix
42 nlabel = length(unique(label));
43 kc = length(vUni);
44 mismatrix = zeros(nlabel ,kc);
45 for i=1:nlabel
46     for j=1:kc
47         mismatrix(i ,j)=length(intersect(find(label==i),find(vShip==j)));
48     end
49 end
50 mismatrix
51
52 close all;
53 figure(1),clf ,hold on
54 cVec = 'bgrcmykbgrcmykbgrcmykbgrcmyk';
55 sVec = 'o*+hdpsx';
56 MEC = 'MarkerEdgeColor';
57 MFC = 'MarkerFaceColor';
58 if d==2
59     for j = 1:min(kc ,length(cVec))
60         myMembers = find(vShip==j);
61         CC = Z(vUni(j ),:);

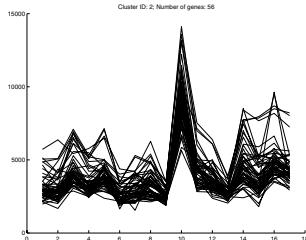
```

```

62 plot(data(myMembers,1)', data(myMembers,2)', [cVec(j) sVec(j)])
63 plot(CC(1),CC(2), 'p', MEC, 'k', MFC, cVec(j), 'MarkerSize', 14)
64 end
65 sTitle = [ 'k=' int2str(k) ',\nk_c=' int2str(kc)];
66 sTitle = [sTitle ',\nalpha=' num2str(alpha) ',\nbeta=' num2str(beta)];
67 title(sTitle)
68 end
69
70 function [W,Z,vShip ,vUni] = MSSC(data ,alpha ,beta ,k ,plotFlag );
71 % ——————INPUT—————
72 % data — the input data matrix (n x d)
73 % alpha — the fuzzifier
74 % beta — the Lagrange multiplier
75 % k — the number of starting points
76 % ——————OUTPUT—————
77 % W — the fuzzy dimension weight matrix (k x d)
78 % Z — the set of centers (k x d)
79 % vShip — for every point which cluster it belongs to ( 1 x n )
80
81 %Initialization
82 cVec = 'bgrcmykbgrcmykbgrcmykbgrcmyk';
83 sVec = 'o*xh+.sdp';
84 MEC = 'MarkerEdgeColor';
85 MFC = 'MarkerFaceColor';
86 [n,d] = size(data);
87 vRand = floor(n*rand(k,1)+1);
88 Z = data(vRand,:);
89 W = ones(k,d)/d;
90 U = zeros(k,n);
91 stopThresh = 1e-5;
92
93 U = UpdateU(W,Z,data ,alpha ,beta ,k ,n);
94 newZ = UpdateZ(data ,U,k ,n,d);
95 W = UpdateW(data ,U,newZ ,alpha ,k ,n,d);
96
97 nIter = 1;
98 while max(max(abs(newZ-Z)))>stopThresh
99     if plotFlag>0
100         if d == 2
101             figure(12345),clf,hold on
102             title([ 'k=' int2str(k)]);
103             plot(data(:,1)', data(:,2)', '.', 'MarkerSize', 8)
104             plot(Z(:,1)',Z(:,2)', 'o',MEC, 'g', 'MarkerSize',12)
105             plot(newZ(:,1)',newZ(:,2)', 'd',MEC, 'r', 'MarkerSize',11)
106             pause
107         end
108     end
109     Z = newZ;
110     U = UpdateU(W,Z,data ,alpha ,beta ,k ,n);
111     newZ = UpdateZ(data ,U,k ,n,d);
112     W = UpdateW(data ,U,newZ ,alpha ,k ,n,d);
113 end
114
115 % Get distinct centers
116 bCent = ones(1,k);
117 for j=2:k
118     for i=1:(j-1)
119         if bCent(i) == 1
120             if max(max(abs(Z(j,:)-Z(i,:)))) < 0.05
121                 bCent(j) = 0;
122                 break;
123             end
124         end
125     end
126 end
127 vUni = find(bCent==1);
128
129 % Get hard cluster membership

```

```
130 if length(vUni)==1
131     vShip=ones(1,n);
132 else
133     [vFM vShip]=max(U(vUni,:));
134 end
135
136 function outU = UpdateU(W,Z,data,alpha,beta,k,n)
137 outU = zeros(k,n);
138 for i=1:n
139     for j=1:k
140         temp = (W(j,:).^alpha)*((data(i,:)-Z(j,:)).^2)';
141         outU(j,i)=exp(-beta*sum(temp));
142     end
143     dTemp = sum(outU(:,i));
144     for j=1:k
145         outU(j,i) = outU(j,i)/dTemp;
146     end
147 end
148
149 function outZ = UpdateZ(data,U,k,n,d)
150 outZ = zeros(k,d);
151 for j=1:k
152     outZ(j,:)=sum(data.*repmat(U(j,:)',1,d))./sum(repmat(U(j,:)',1,d));
153 end
154
155 function outW = UpdateW(data,U,Z,alpha,k,n,d)
156 outW = zeros(k,d);
157 for j=1:k
158     for h=1:d
159         temp = U(j,:)*((data(:,h)-repmat(Z(j,h),n,1)).^2);
160         outW(j,h) = temp.^(-1/(alpha-1));
161     end
162     dTemp = sum(outW(j,:));
163     for h=1:d
164         outW(j,h) = outW(j,h)/dTemp;
165     end
166 end
```

Appendix D C++ Codes

This appendix gives two C++ codes mentioned in this book. The first code is used to convert categorical values to integers (see Section 19.2), and the second code is the program of the FSC algorithm (see Section 15.11).

D.1 The C++ Code for Converting Categorical Values to Integers

This section gives the full code of the small program which converts categorical values in a data set to integers.

```

1  /*
2   * Convert categorical values in a data set to integers
3   */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <iostream>
8 #include <fstream>
9 #include <vector>
10 #include <string>
11 #include <list>
12
13 using namespace std;
14
15 // vectors
16 typedef vector<int> nVector;
17 typedef vector<string> sVector;
18 typedef vector<double> dVector;
19
20 typedef vector<sVector> sMatrix;
21 typedef vector<nVector> nMatrix;
22
23 // iterators
24 typedef nVector::iterator nlIter;
25 typedef sVector::iterator slIter;
26 typedef dVector::iterator dlIter;
27
28 // list
29 typedef list<int> nList;
30 typedef list<double> dList;

```

```

31 | typedef list<string> sList;
32 |
33 | typedef sList::iterator sIIter;
34 | typedef nList::iterator nIIter;
35 | typedef dList::iterator dIIter;
36 |
37 | typedef vector<sList> sIVector;
38 |
39 | #if !defined(LIMITS)
40 | #define LIMITS
41 |     // max chars in a field
42 |     #define MAX_FIELD_BUFFER 1024
43 |     // max chars in one line
44 |     #define MAX_LINE_BUFFER 0x00FFFFFF
45 | #endif
46 |
47 | int ReadData(
48 |     const char *filename,
49 |     sVector &svLabel,
50 |     sVector &svAtt,
51 |     sMatrix &smData
52 | ){
53 |     int i,j,n,d;
54 |     ifstream ifsFile;
55 |     char buff[MAX_FIELD_BUFFER];
56 |
57 |     try{
58 |         ifsFile.open(filename);
59 |         // count the number of attributes;
60 |         i = 1;
61 |         do{
62 |             if (ifsFile.peek() == ',')
63 |                 i++;
64 |         } while (ifsFile.get() != '\n');
65 |         d = i-1; // the first column is record name
66 |
67 |         // now count the number of records
68 |         i = 0;
69 |         do{
70 |             ifsFile.ignore(MAX_LINE_BUFFER, '\n');
71 |             i++;
72 |         } while (!ifsFile.eof());
73 |         n = i-1; // one return at the end of the last line of the file
74 |             // NO return after the last line of the file
75 |         ifsFile.close();
76 |
77 |         // open the file again
78 |         ifsFile.open(filename);
79 |
80 |         svAtt = sVector(d,"v");
81 |         ifsFile.getline(buff, MAX_FIELD_BUFFER, ',');
82 |         for (i=0; i<d; i++){
83 |             if (i == d-1){
84 |                 ifsFile.getline(buff, MAX_FIELD_BUFFER, '\n');
85 |                 // remove '\n'
86 |                 for(j=0;j<sizeof(buff);j++){
87 |                     if(buff[j] == '\n'){
88 |                         buff[j] = ' ';
89 |                     }
90 |                 }
91 |                 svAtt[i] = buff;
92 |             } else{
93 |                 ifsFile.getline(buff, MAX_FIELD_BUFFER, ',');
94 |                 svAtt[i] =buff;
95 |             }
96 |         }
97 |
98 |         smData = sMatrix(n);

```

```

99     i = 0;
100    while (! ifsFile.eof() && i<n){
101        // get the record name
102        ifsFile.getline(buff, MAX_FIELD_BUFFER, ' ', ' ');
103        svLabel.push_back(buff);
104        for(j=0; j<d; j++){
105            if (j == d-1){ // denotes the end of the line
106                ifsFile.getline(buff, MAX_FIELD_BUFFER, '\n');
107                // remove '\n'
108                for(j=0;j<sizeof(buff);j++){
109                    if(buff[j] == '\n'){
110                        buff[j] = ' ';
111                    }
112                }
113                smData[i].push_back(buff);
114            } else{
115                ifsFile.getline(buff, MAX_FIELD_BUFFER, ' ', ' ');
116                smData[i].push_back(buff);
117            }
118        }
119        i++;
120    }
121
122    ifsFile.close();
123    return 0;
124 }
125 catch(\ldots){
126     cout<<"reading\_data\_error "<<endl;
127     return -1;
128 }
129 } //ReadData()

130
131 int Cat2Int(
132     sMatrix &smData, // Input categorical data matrix
133     nMatrix &nmData // Output Integer matrix
134 ){
135     int n,d,i,j,s,nTemp;
136     sMatrix smSymbol;
137     sList sITemp;
138     sIIter sliTemp;
139
140     // Get the dimension of the data matrix
141     n = smData.size();
142     d = smData[0].size();
143
144     // Count distinct attribute values for each attribute
145     smSymbol = sMatrix(d);
146     for(j=0;j<d;j++){
147         for(i=0;i<n; i++){
148             sITemp.push_back(smData[i][j]);
149         }
150         sITemp.sort();
151         sITemp.unique();
152
153         sliTemp = sITemp.begin();
154         while( sliTemp != sITemp.end() ){
155             smSymbol[j].push_back(*sliTemp);
156             sliTemp++;
157         }
158         sITemp.clear();
159     }
160
161     // Convert categorical values to integers
162     nmData = nMatrix(n);
163     for(i=0;i<n; i++)
164         nmData[i] = nVector(d,0);
165     for(j=0;j<d; j++){
166         nTemp = smSymbol[j].size();

```

```

167     for( i=0; i<n ; i ++){
168         for(s=0;s<nTemp; s++){
169             if(smSymbol[ j ][ s ] == smData[ i ][ j ]){
170                 nmData[ i ][ j ] = s+1;
171                 break;
172             }
173         }
174     }
175 }
176
177 return 0;
178 } // Cat2Int
179
180 int main(int argc , char *argv [])
181 {
182     int i,j,n,d;
183     sMatrix smData;
184     sVector svLabel ,svAtt ;
185     nMatrix nmData;
186     ofstream ofsFile ;
187
188     // Data set file name
189     char fname []="catdata.csv";
190
191     // Read data from file
192     ReadData(fname , svLabel , svAtt ,smData );
193
194     // Convert categorical values to integers
195     Cat2Int (smData ,nmData );
196
197     // Output converted data to a .csv file
198     ofsFile .open("intdata.csv");
199     n = nmData .size ();
200     d = nmData [ 0 ].size ();
201     for(i=0;i<n;i++){
202         for(j=0;j<d-1;j++){
203             ofsFile <<nmData [ i ][ j ]<<"," ;
204         }
205         ofsFile <<nmData [ i ][ d-1 ]<<endl ;
206     }
207     ofsFile .close ();
208
209     system("PAUSE");
210
211 }

```

D.2 The C++ Code for the FSC Algorithm

This appendix gives the complete C++ code of the fuzzy subspace clustering (FSC) algorithm for clustering the gene expression data gathered by (Cho et al., 1998). The clustering results are presented in Section 18.6.

```

1  /*
2   * Fuzzy subspace clustering algorithm
3   */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <time.h>
8 #include <iostream>
9 #include <fstream>
10 #include <vector>
11 #include <string>
12 #include <list>

```

```
13 #include <math.h>
14
15 using namespace std;
16
17 // vectors
18 typedef vector<int> nVector;
19 typedef vector<string> sVector;
20 typedef vector<double> dVector;
21
22 typedef vector<dVector> dMatrix;
23 typedef vector<nVector> nMatrix;
24
25 // iterators
26 typedef nVector::iterator nIter;
27 typedef sVector::iterator sIter;
28 typedef dVector::iterator dIter;
29
30 // list
31 typedef list<int> nList;
32 typedef list<double> dList;
33 typedef list<string> sList;
34
35 typedef sList::iterator sIter;
36 typedef nList::iterator nIter;
37 typedef dList::iterator dIter;
38
39 typedef vector<sList> slVector;
40
41 #if !defined(LIMITS)
42 #define LIMITS
43     // max chars in a field
44     #define MAX_FIELD_BUFFER      1024
45     // max chars in 1 line
46     #define MAX_LINE_BUFFER       0x00FFFFFF
47 #endif
48
49 //Declarations of global variables
50 static int n=0; // the number of records
51 static int k=0; // the number of clusters
52 static int d=0; // the number of attributes
53 static double alpha = 2.1; // the fuzzier
54 static double epsilon = 0;
55
56 /*
57 * http://linux.wku.edu/~lamonml/algor/sort/quick.html
58 * sorted sequence a_1<a_2<\ldots<a_m
59 */
60 int QuickSort(
61     dVector &dvData,
62     nVector &nvIndex
63 ){
64     double l_hold , r_hold , dTemp;
65     int left ,right ,nSize ,nTemp ,nIndex ;
66
67     nSize = (int)dvData.size();
68     for(left=0;left < nSize -1;left ++){
69         l_hold = dvData[left];
70         r_hold = nvIndex[left];
71         nIndex = left;
72         for(right=left+1;right < nSize ;right ++){
73             if(dvData[right] < l_hold){
74                 l_hold = dvData[right];
75                 r_hold = nvIndex[right];
76                 nIndex = right;
77             }
78         }
79         if(left != nIndex){
```

```

81     dTemp = dvData[ left ];
82     dvData[ left ] = l_hold;
83     dvData[ nIndex ] = dTemp;
84
85     nIndex = nvIndex[ left ];
86     nvIndex[ left ] = nvIndex[ nIndex ];
87     nvIndex[ nIndex ] = nIndex;
88 }
89
90     return 0;
91 } //QuickSort()
92
93 int Initialization(
94     dMatrix &dmZ,
95     dMatrix &dmW,
96     dMatrix &dmData
97 ){
98     int i,j,nIndex;
99
100    for(i=0;i<k;i++){
101        nIndex = (int)(n*rand()/(RAND_MAX+1.0));
102        for(j=0;j<d;j++){
103            dmZ[i][j] = dmData[nIndex][j];
104            dmW[i][j] = 1.0/d;
105        }
106    }
107
108    return 0;
109 } //Initialization()
110
111 /*
112  * dvW - weight , dvZ - center , dvX - data point
113  */
114 double Distance(
115     dVector &dvW,
116     dVector &dvZ,
117     dVector &dvX
118 ){
119     int h;
120     double dTemp;
121
122     dTemp = 0.0;
123     for(h=0;h<d;h++){
124         dTemp += pow(dvW[h],alpha)*pow(dvX[h]-dvZ[h],2);
125     }
126
127     return dTemp;
128 } //Distance()
129
130 int GetShip(
131     dMatrix &dmData,
132     dMatrix &dmW,
133     dMatrix &dmZ,
134     nVector &nvShip
135 ){
136     int i,j;
137     double dTemp;
138     dVector dvDist;
139     nVector nvIndex;
140
141     dvDist = dVector(k,0.0);
142     nvIndex = nVector(k,0);
143
144     for(i=0;i<n;i++){
145         for(j=0;j<k;j++){
146             dvDist[j] = Distance(dmW[j],dmZ[j],dmData[i]);
147             nvIndex[j] = j;
148         }
149     }
150
151     for(i=0;i<n;i++){
152         for(j=0;j<k;j++){
153             if(dvDist[i] > dvDist[j])
154                 nvIndex[i] = j;
155         }
156     }
157
158     nvShip = nvIndex;
159
160     return 0;
161 } //GetShip()
162
163 void QuickSort(
164     dVector &dvData,
165     int left,
166     int right
167 ){
168     int l_hold,r_hold;
169     double dTemp;
170     nVector nvIndex;
171
172     if(left < right){
173         nIndex = nvIndex[ left ];
174         nvIndex[ left ] = nvIndex[ nIndex ];
175         nvIndex[ nIndex ] = nIndex;
176
177         l_hold = nvIndex[ left ];
178         r_hold = nvIndex[ right ];
179
180         for(i=left+1;i<right;i++){
181             if(nvIndex[i] < nIndex)
182                 nvIndex[i] = l_hold;
183             else if(nvIndex[i] > nIndex)
184                 nvIndex[i] = r_hold;
185             else
186                 nvIndex[i] = nvIndex[i];
187         }
188
189         dTemp = dvData[ left ];
190         dvData[ left ] = dvData[ nIndex ];
191         dvData[ nIndex ] = dTemp;
192
193         nIndex = nvIndex[ left ];
194         nvIndex[ left ] = nvIndex[ nIndex ];
195         nvIndex[ nIndex ] = nIndex;
196
197         l_hold = nIndex;
198         r_hold = nvIndex[ right ];
199
200         for(i=left+1;i<right;i++){
201             if(nvIndex[i] < l_hold)
202                 nvIndex[i] = l_hold;
203             else if(nvIndex[i] > r_hold)
204                 nvIndex[i] = r_hold;
205             else
206                 nvIndex[i] = nvIndex[i];
207         }
208
209         nIndex = nvIndex[ left ];
210         nvIndex[ left ] = nvIndex[ nIndex ];
211         nvIndex[ nIndex ] = nIndex;
212
213         dTemp = dvData[ left ];
214         dvData[ left ] = dvData[ nIndex ];
215         dvData[ nIndex ] = dTemp;
216
217         nIndex = nvIndex[ left ];
218         nvIndex[ left ] = nvIndex[ nIndex ];
219         nvIndex[ nIndex ] = nIndex;
220
221         l_hold = nIndex;
222         r_hold = nvIndex[ right ];
223
224         for(i=left+1;i<right;i++){
225             if(nvIndex[i] < l_hold)
226                 nvIndex[i] = l_hold;
227             else if(nvIndex[i] > r_hold)
228                 nvIndex[i] = r_hold;
229             else
230                 nvIndex[i] = nvIndex[i];
231         }
232
233         nIndex = nvIndex[ left ];
234         nvIndex[ left ] = nvIndex[ nIndex ];
235         nvIndex[ nIndex ] = nIndex;
236
237         dTemp = dvData[ left ];
238         dvData[ left ] = dvData[ nIndex ];
239         dvData[ nIndex ] = dTemp;
240
241         nIndex = nvIndex[ left ];
242         nvIndex[ left ] = nvIndex[ nIndex ];
243         nvIndex[ nIndex ] = nIndex;
244
245         l_hold = nIndex;
246         r_hold = nvIndex[ right ];
247
248         for(i=left+1;i<right;i++){
249             if(nvIndex[i] < l_hold)
250                 nvIndex[i] = l_hold;
251             else if(nvIndex[i] > r_hold)
252                 nvIndex[i] = r_hold;
253             else
254                 nvIndex[i] = nvIndex[i];
255         }
256
257         nIndex = nvIndex[ left ];
258         nvIndex[ left ] = nvIndex[ nIndex ];
259         nvIndex[ nIndex ] = nIndex;
260
261         dTemp = dvData[ left ];
262         dvData[ left ] = dvData[ nIndex ];
263         dvData[ nIndex ] = dTemp;
264
265         nIndex = nvIndex[ left ];
266         nvIndex[ left ] = nvIndex[ nIndex ];
267         nvIndex[ nIndex ] = nIndex;
268
269         l_hold = nIndex;
270         r_hold = nvIndex[ right ];
271
272         for(i=left+1;i<right;i++){
273             if(nvIndex[i] < l_hold)
274                 nvIndex[i] = l_hold;
275             else if(nvIndex[i] > r_hold)
276                 nvIndex[i] = r_hold;
277             else
278                 nvIndex[i] = nvIndex[i];
279         }
280
281         nIndex = nvIndex[ left ];
282         nvIndex[ left ] = nvIndex[ nIndex ];
283         nvIndex[ nIndex ] = nIndex;
284
285         dTemp = dvData[ left ];
286         dvData[ left ] = dvData[ nIndex ];
287         dvData[ nIndex ] = dTemp;
288
289         nIndex = nvIndex[ left ];
290         nvIndex[ left ] = nvIndex[ nIndex ];
291         nvIndex[ nIndex ] = nIndex;
292
293         l_hold = nIndex;
294         r_hold = nvIndex[ right ];
295
296         for(i=left+1;i<right;i++){
297             if(nvIndex[i] < l_hold)
298                 nvIndex[i] = l_hold;
299             else if(nvIndex[i] > r_hold)
300                 nvIndex[i] = r_hold;
301             else
302                 nvIndex[i] = nvIndex[i];
303         }
304
305         nIndex = nvIndex[ left ];
306         nvIndex[ left ] = nvIndex[ nIndex ];
307         nvIndex[ nIndex ] = nIndex;
308
309         dTemp = dvData[ left ];
310         dvData[ left ] = dvData[ nIndex ];
311         dvData[ nIndex ] = dTemp;
312
313         nIndex = nvIndex[ left ];
314         nvIndex[ left ] = nvIndex[ nIndex ];
315         nvIndex[ nIndex ] = nIndex;
316
317         l_hold = nIndex;
318         r_hold = nvIndex[ right ];
319
320         for(i=left+1;i<right;i++){
321             if(nvIndex[i] < l_hold)
322                 nvIndex[i] = l_hold;
323             else if(nvIndex[i] > r_hold)
324                 nvIndex[i] = r_hold;
325             else
326                 nvIndex[i] = nvIndex[i];
327         }
328
329         nIndex = nvIndex[ left ];
330         nvIndex[ left ] = nvIndex[ nIndex ];
331         nvIndex[ nIndex ] = nIndex;
332
333         dTemp = dvData[ left ];
334         dvData[ left ] = dvData[ nIndex ];
335         dvData[ nIndex ] = dTemp;
336
337         nIndex = nvIndex[ left ];
338         nvIndex[ left ] = nvIndex[ nIndex ];
339         nvIndex[ nIndex ] = nIndex;
340
341         l_hold = nIndex;
342         r_hold = nvIndex[ right ];
343
344         for(i=left+1;i<right;i++){
345             if(nvIndex[i] < l_hold)
346                 nvIndex[i] = l_hold;
347             else if(nvIndex[i] > r_hold)
348                 nvIndex[i] = r_hold;
349             else
350                 nvIndex[i] = nvIndex[i];
351         }
352
353         nIndex = nvIndex[ left ];
354         nvIndex[ left ] = nvIndex[ nIndex ];
355         nvIndex[ nIndex ] = nIndex;
356
357         dTemp = dvData[ left ];
358         dvData[ left ] = dvData[ nIndex ];
359         dvData[ nIndex ] = dTemp;
360
361         nIndex = nvIndex[ left ];
362         nvIndex[ left ] = nvIndex[ nIndex ];
363         nvIndex[ nIndex ] = nIndex;
364
365         l_hold = nIndex;
366         r_hold = nvIndex[ right ];
367
368         for(i=left+1;i<right;i++){
369             if(nvIndex[i] < l_hold)
370                 nvIndex[i] = l_hold;
371             else if(nvIndex[i] > r_hold)
372                 nvIndex[i] = r_hold;
373             else
374                 nvIndex[i] = nvIndex[i];
375         }
376
377         nIndex = nvIndex[ left ];
378         nvIndex[ left ] = nvIndex[ nIndex ];
379         nvIndex[ nIndex ] = nIndex;
380
381         dTemp = dvData[ left ];
382         dvData[ left ] = dvData[ nIndex ];
383         dvData[ nIndex ] = dTemp;
384
385         nIndex = nvIndex[ left ];
386         nvIndex[ left ] = nvIndex[ nIndex ];
387         nvIndex[ nIndex ] = nIndex;
388
389         l_hold = nIndex;
390         r_hold = nvIndex[ right ];
391
392         for(i=left+1;i<right;i++){
393             if(nvIndex[i] < l_hold)
394                 nvIndex[i] = l_hold;
395             else if(nvIndex[i] > r_hold)
396                 nvIndex[i] = r_hold;
397             else
398                 nvIndex[i] = nvIndex[i];
399         }
400
401         nIndex = nvIndex[ left ];
402         nvIndex[ left ] = nvIndex[ nIndex ];
403         nvIndex[ nIndex ] = nIndex;
404
405         dTemp = dvData[ left ];
406         dvData[ left ] = dvData[ nIndex ];
407         dvData[ nIndex ] = dTemp;
408
409         nIndex = nvIndex[ left ];
410         nvIndex[ left ] = nvIndex[ nIndex ];
411         nvIndex[ nIndex ] = nIndex;
412
413         l_hold = nIndex;
414         r_hold = nvIndex[ right ];
415
416         for(i=left+1;i<right;i++){
417             if(nvIndex[i] < l_hold)
418                 nvIndex[i] = l_hold;
419             else if(nvIndex[i] > r_hold)
420                 nvIndex[i] = r_hold;
421             else
422                 nvIndex[i] = nvIndex[i];
423         }
424
425         nIndex = nvIndex[ left ];
426         nvIndex[ left ] = nvIndex[ nIndex ];
427         nvIndex[ nIndex ] = nIndex;
428
429         dTemp = dvData[ left ];
430         dvData[ left ] = dvData[ nIndex ];
431         dvData[ nIndex ] = dTemp;
432
433         nIndex = nvIndex[ left ];
434         nvIndex[ left ] = nvIndex[ nIndex ];
435         nvIndex[ nIndex ] = nIndex;
436
437         l_hold = nIndex;
438         r_hold = nvIndex[ right ];
439
440         for(i=left+1;i<right;i++){
441             if(nvIndex[i] < l_hold)
442                 nvIndex[i] = l_hold;
443             else if(nvIndex[i] > r_hold)
444                 nvIndex[i] = r_hold;
445             else
446                 nvIndex[i] = nvIndex[i];
447         }
448
449         nIndex = nvIndex[ left ];
450         nvIndex[ left ] = nvIndex[ nIndex ];
451         nvIndex[ nIndex ] = nIndex;
452
453         dTemp = dvData[ left ];
454         dvData[ left ] = dvData[ nIndex ];
455         dvData[ nIndex ] = dTemp;
456
457         nIndex = nvIndex[ left ];
458         nvIndex[ left ] = nvIndex[ nIndex ];
459         nvIndex[ nIndex ] = nIndex;
460
461         l_hold = nIndex;
462         r_hold = nvIndex[ right ];
463
464         for(i=left+1;i<right;i++){
465             if(nvIndex[i] < l_hold)
466                 nvIndex[i] = l_hold;
467             else if(nvIndex[i] > r_hold)
468                 nvIndex[i] = r_hold;
469             else
470                 nvIndex[i] = nvIndex[i];
471         }
472
473         nIndex = nvIndex[ left ];
474         nvIndex[ left ] = nvIndex[ nIndex ];
475         nvIndex[ nIndex ] = nIndex;
476
477         dTemp = dvData[ left ];
478         dvData[ left ] = dvData[ nIndex ];
479         dvData[ nIndex ] = dTemp;
480
481         nIndex = nvIndex[ left ];
482         nvIndex[ left ] = nvIndex[ nIndex ];
483         nvIndex[ nIndex ] = nIndex;
484
485         l_hold = nIndex;
486         r_hold = nvIndex[ right ];
487
488         for(i=left+1;i<right;i++){
489             if(nvIndex[i] < l_hold)
490                 nvIndex[i] = l_hold;
491             else if(nvIndex[i] > r_hold)
492                 nvIndex[i] = r_hold;
493             else
494                 nvIndex[i] = nvIndex[i];
495         }
496
497         nIndex = nvIndex[ left ];
498         nvIndex[ left ] = nvIndex[ nIndex ];
499         nvIndex[ nIndex ] = nIndex;
500
501         dTemp = dvData[ left ];
502         dvData[ left ] = dvData[ nIndex ];
503         dvData[ nIndex ] = dTemp;
504
505         nIndex = nvIndex[ left ];
506         nvIndex[ left ] = nvIndex[ nIndex ];
507         nvIndex[ nIndex ] = nIndex;
508
509         l_hold = nIndex;
510         r_hold = nvIndex[ right ];
511
512         for(i=left+1;i<right;i++){
513             if(nvIndex[i] < l_hold)
514                 nvIndex[i] = l_hold;
515             else if(nvIndex[i] > r_hold)
516                 nvIndex[i] = r_hold;
517             else
518                 nvIndex[i] = nvIndex[i];
519         }
520
521         nIndex = nvIndex[ left ];
522         nvIndex[ left ] = nvIndex[ nIndex ];
523         nvIndex[ nIndex ] = nIndex;
524
525         dTemp = dvData[ left ];
526         dvData[ left ] = dvData[ nIndex ];
527         dvData[ nIndex ] = dTemp;
528
529         nIndex = nvIndex[ left ];
530         nvIndex[ left ] = nvIndex[ nIndex ];
531         nvIndex[ nIndex ] = nIndex;
532
533         l_hold = nIndex;
534         r_hold = nvIndex[ right ];
535
536         for(i=left+1;i<right;i++){
537             if(nvIndex[i] < l_hold)
538                 nvIndex[i] = l_hold;
539             else if(nvIndex[i] > r_hold)
540                 nvIndex[i] = r_hold;
541             else
542                 nvIndex[i] = nvIndex[i];
543         }
544
545         nIndex = nvIndex[ left ];
546         nvIndex[ left ] = nvIndex[ nIndex ];
547         nvIndex[ nIndex ] = nIndex;
548
549         dTemp = dvData[ left ];
550         dvData[ left ] = dvData[ nIndex ];
551         dvData[ nIndex ] = dTemp;
552
553         nIndex = nvIndex[ left ];
554         nvIndex[ left ] = nvIndex[ nIndex ];
555         nvIndex[ nIndex ] = nIndex;
556
557         l_hold = nIndex;
558         r_hold = nvIndex[ right ];
559
560         for(i=left+1;i<right;i++){
561             if(nvIndex[i] < l_hold)
562                 nvIndex[i] = l_hold;
563             else if(nvIndex[i] > r_hold)
564                 nvIndex[i] = r_hold;
565             else
566                 nvIndex[i] = nvIndex[i];
567         }
568
569         nIndex = nvIndex[ left ];
570         nvIndex[ left ] = nvIndex[ nIndex ];
571         nvIndex[ nIndex ] = nIndex;
572
573         dTemp = dvData[ left ];
574         dvData[ left ] = dvData[ nIndex ];
575         dvData[ nIndex ] = dTemp;
576
577         nIndex = nvIndex[ left ];
578         nvIndex[ left ] = nvIndex[ nIndex ];
579         nvIndex[ nIndex ] = nIndex;
580
581         l_hold = nIndex;
582         r_hold = nvIndex[ right ];
583
584         for(i=left+1;i<right;i++){
585             if(nvIndex[i] < l_hold)
586                 nvIndex[i] = l_hold;
587             else if(nvIndex[i] > r_hold)
588                 nvIndex[i] = r_hold;
589             else
590                 nvIndex[i] = nvIndex[i];
591         }
592
593         nIndex = nvIndex[ left ];
594         nvIndex[ left ] = nvIndex[ nIndex ];
595         nvIndex[ nIndex ] = nIndex;
596
597         dTemp = dvData[ left ];
598         dvData[ left ] = dvData[ nIndex ];
599         dvData[ nIndex ] = dTemp;
600
601         nIndex = nvIndex[ left ];
602         nvIndex[ left ] = nvIndex[ nIndex ];
603         nvIndex[ nIndex ] = nIndex;
604
605         l_hold = nIndex;
606         r_hold = nvIndex[ right ];
607
608         for(i=left+1;i<right;i++){
609             if(nvIndex[i] < l_hold)
610                 nvIndex[i] = l_hold;
611             else if(nvIndex[i] > r_hold)
612                 nvIndex[i] = r_hold;
613             else
614                 nvIndex[i] = nvIndex[i];
615         }
616
617         nIndex = nvIndex[ left ];
618         nvIndex[ left ] = nvIndex[ nIndex ];
619         nvIndex[ nIndex ] = nIndex;
620
621         dTemp = dvData[ left ];
622         dvData[ left ] = dvData[ nIndex ];
623         dvData[ nIndex ] = dTemp;
624
625         nIndex = nvIndex[ left ];
626         nvIndex[ left ] = nvIndex[ nIndex ];
627         nvIndex[ nIndex ] = nIndex;
628
629         l_hold = nIndex;
630         r_hold = nvIndex[ right ];
631
632         for(i=left+1;i<right;i++){
633             if(nvIndex[i] < l_hold)
634                 nvIndex[i] = l_hold;
635             else if(nvIndex[i] > r_hold)
636                 nvIndex[i] = r_hold;
637             else
638                 nvIndex[i] = nvIndex[i];
639         }
640
641         nIndex = nvIndex[ left ];
642         nvIndex[ left ] = nvIndex[ nIndex ];
643         nvIndex[ nIndex ] = nIndex;
644
645         dTemp = dvData[ left ];
646         dvData[ left ] = dvData[ nIndex ];
647         dvData[ nIndex ] = dTemp;
648
649         nIndex = nvIndex[ left ];
650         nvIndex[ left ] = nvIndex[ nIndex ];
651         nvIndex[ nIndex ] = nIndex;
652
653         l_hold = nIndex;
654         r_hold = nvIndex[ right ];
655
656         for(i=left+1;i<right;i++){
657             if(nvIndex[i] < l_hold)
658                 nvIndex[i] = l_hold;
659             else if(nvIndex[i] > r_hold)
660                 nvIndex[i] = r_hold;
661             else
662                 nvIndex[i] = nvIndex[i];
663         }
664
665         nIndex = nvIndex[ left ];
666         nvIndex[ left ] = nvIndex[ nIndex ];
667         nvIndex[ nIndex ] = nIndex;
668
669         dTemp = dvData[ left ];
670         dvData[ left ] = dvData[ nIndex ];
671         dvData[ nIndex ] = dTemp;
672
673         nIndex = nvIndex[ left ];
674         nvIndex[ left ] = nvIndex[ nIndex ];
675         nvIndex[ nIndex ] = nIndex;
676
677         l_hold = nIndex;
678         r_hold = nvIndex[ right ];
679
680         for(i=left+1;i<right;i++){
681             if(nvIndex[i] < l_hold)
682                 nvIndex[i] = l_hold;
683             else if(nvIndex[i] > r_hold)
684                 nvIndex[i] = r_hold;
685             else
686                 nvIndex[i] = nvIndex[i];
687         }
688
689         nIndex = nvIndex[ left ];
690         nvIndex[ left ] = nvIndex[ nIndex ];
691         nvIndex[ nIndex ] = nIndex;
692
693         dTemp = dvData[ left ];
694         dvData[ left ] = dvData[ nIndex ];
695         dvData[ nIndex ] = dTemp;
696
697         nIndex = nvIndex[ left ];
698         nvIndex[ left ] = nvIndex[ nIndex ];
699         nvIndex[ nIndex ] = nIndex;
700
701         l_hold = nIndex;
702         r_hold = nvIndex[ right ];
703
704         for(i=left+1;i<right;i++){
705             if(nvIndex[i] < l_hold)
706                 nvIndex[i] = l_hold;
707             else if(nvIndex[i] > r_hold)
708                 nvIndex[i] = r_hold;
709             else
710                 nvIndex[i] = nvIndex[i];
711         }
712
713         nIndex = nvIndex[ left ];
714         nvIndex[ left ] = nvIndex[ nIndex ];
715         nvIndex[ nIndex ] = nIndex;
716
717         dTemp = dvData[ left ];
718         dvData[ left ] = dvData[ nIndex ];
719         dvData[ nIndex ] = dTemp;
720
721         nIndex = nvIndex[ left ];
722         nvIndex[ left ] = nvIndex[ nIndex ];
723         nvIndex[ nIndex ] = nIndex;
724
725         l_hold = nIndex;
726         r_hold = nvIndex[ right ];
727
728         for(i=left+1;i<right;i++){
729             if(nvIndex[i] < l_hold)
730                 nvIndex[i] = l_hold;
731             else if(nvIndex[i] > r_hold)
732                 nvIndex[i] = r_hold;
733             else
734                 nvIndex[i] = nvIndex[i];
735         }
736
737         nIndex = nvIndex[ left ];
738         nvIndex[ left ] = nvIndex[ nIndex ];
739         nvIndex[ nIndex ] = nIndex;
740
741         dTemp = dvData[ left ];
742         dvData[ left ] = dvData[ nIndex ];
743         dvData[ nIndex ] = dTemp;
744
745         nIndex = nvIndex[ left ];
746         nvIndex[ left ] = nvIndex[ nIndex ];
747         nvIndex[ nIndex ] = nIndex;
748
749         l_hold = nIndex;
750         r_hold = nvIndex[ right ];
751
752         for(i=left+1;i<right;i++){
753             if(nvIndex[i] < l_hold)
754                 nvIndex[i] = l_hold;
755             else if(nvIndex[i] > r_hold)
756                 nvIndex[i] = r_hold;
757             else
758                 nvIndex[i] = nvIndex[i];
759         }
760
761         nIndex = nvIndex[ left ];
762         nvIndex[ left ] = nvIndex[ nIndex ];
763         nvIndex[ nIndex ] = nIndex;
764
765         dTemp = dvData[ left ];
766         dvData[ left ] = dvData[ nIndex ];
767         dvData[ nIndex ] = dTemp;
768
769         nIndex = nvIndex[ left ];
770         nvIndex[ left ] = nvIndex[ nIndex ];
771         nvIndex[ nIndex ] = nIndex;
772
773         l_hold = nIndex;
774         r_hold = nvIndex[ right ];
775
776         for(i=left+1;i<right;i++){
777             if(nvIndex[i] < l_hold)
778                 nvIndex[i] = l_hold;
779             else if(nvIndex[i] > r_hold)
780                 nvIndex[i] = r_hold;
781             else
782                 nvIndex[i] = nvIndex[i];
783         }
784
785         nIndex = nvIndex[ left ];
786         nvIndex[ left ] = nvIndex[ nIndex ];
787         nvIndex[ nIndex ] = nIndex;
788
789         dTemp = dvData[ left ];
790         dvData[ left ] = dvData[ nIndex ];
791         dvData[ nIndex ] = dTemp;
792
793         nIndex = nvIndex[ left ];
794         nvIndex[ left ] = nvIndex[ nIndex ];
795         nvIndex[ nIndex ] = nIndex;
796
797         l_hold = nIndex;
798         r_hold = nvIndex[ right ];
799
800         for(i=left+1;i<right;i++){
801             if(nvIndex[i] < l_hold)
802                 nvIndex[i] = l_hold;
803             else if(nvIndex[i] > r_hold)
804                 nvIndex[i] = r_hold;
805             else
806                 nvIndex[i] = nvIndex[i];
807         }
808
809         nIndex = nvIndex[ left ];
810         nvIndex[ left ] = nvIndex[ nIndex ];
811         nvIndex[ nIndex ] = nIndex;
812
813         dTemp = dvData[ left ];
814         dvData[ left ] = dvData[ nIndex ];
815         dvData[ nIndex ] = dTemp;
816
817         nIndex = nvIndex[ left ];
818         nvIndex[ left ] = nvIndex[ nIndex ];
819         nvIndex[ nIndex ] = nIndex;
820
821         l_hold = nIndex;
822         r_hold = nvIndex[ right ];
823
824         for(i=left+1;i<right;i++){
825             if(nvIndex[i] < l_hold)
826                 nvIndex[i] = l_hold;
827             else if(nvIndex[i] > r_hold)
828                 nvIndex[i] = r_hold;
829             else
830                 nvIndex[i] = nvIndex[i];
831         }
832
833         nIndex = nvIndex[ left ];
834         nvIndex[ left ] = nvIndex[ nIndex ];
835         nvIndex[ nIndex ] = nIndex;
836
837         dTemp = dvData[ left ];
838         dvData[ left ] = dvData[ nIndex ];
839         dvData[ nIndex ] = dTemp;
840
841         nIndex = nvIndex[ left ];
842         nvIndex[ left ] = nvIndex[ nIndex ];
843         nvIndex[ nIndex ] = nIndex;
844
845         l_hold = nIndex;
846         r_hold = nvIndex[ right ];
847
848         for(i=left+1;i<right;i++){
849             if(nvIndex[i] < l_hold)
850                 nvIndex[i] = l_hold;
851             else if(nvIndex[i] > r_hold)
852                 nvIndex[i] = r_hold;
853             else
854                 nvIndex[i] = nvIndex[i];
855         }
856
857         nIndex = nvIndex[ left ];
858         nvIndex[ left ] = nvIndex[ nIndex ];
859         nvIndex[ nIndex ] = nIndex;
860
861         dTemp = dvData[ left ];
862         dvData[ left ] = dvData[ nIndex ];
863         dvData[ nIndex ] = dTemp;
864
865         nIndex = nvIndex[ left ];
866         nvIndex[ left ] = nvIndex[ nIndex ];
867         nvIndex[ nIndex ] = nIndex;
868
869         l_hold = nIndex;
870         r_hold = nvIndex[ right ];
871
872         for(i=left+1;i<right;i++){
873             if(nvIndex[i] < l_hold)
874                 nvIndex[i] = l_hold;
875             else if(nvIndex[i] > r_hold)
876                 nvIndex[i] = r_hold;
877             else
878                 nvIndex[i] = nvIndex[i];
879         }
880
881         nIndex = nvIndex[ left ];
882         nvIndex[ left ] = nvIndex[ nIndex ];
883         nvIndex[ nIndex ] = nIndex;
884
885         dTemp = dvData[ left ];
886         dvData[ left ] = dvData[ nIndex ];
887         dvData[ nIndex ] = dTemp;
888
889         nIndex = nvIndex[ left ];
890         nvIndex[ left ] = nvIndex[ nIndex ];
891         nvIndex[ nIndex ] = nIndex;
892
893         l_hold = nIndex;
894         r_hold = nvIndex[ right ];
895
896         for(i=left+1;i<right;i++){
897             if(nvIndex[i] < l_hold)
898                 nvIndex[i] = l_hold;
899             else if(nvIndex[i] > r_hold)
900                 nvIndex[i] = r_hold;
901             else
902                 nvIndex[i] = nvIndex[i];
903         }
904
905         nIndex = nvIndex[ left ];
906         nvIndex[ left ] = nvIndex[ nIndex ];
907         nvIndex[ nIndex ] = nIndex;
908
909         dTemp = dvData[ left ];
910         dvData[ left ] = dvData[ nIndex ];
911         dvData[ nIndex ] = dTemp;
912
913         nIndex = nvIndex[ left ];
914         nvIndex[ left ] = nvIndex[ nIndex ];
915         nvIndex[ nIndex ] = nIndex;
916
917         l_hold = nIndex;
918         r_hold = nvIndex[ right ];
919
920         for(i=left+1;i<right;i++){
921             if(nvIndex[i] < l_hold)
922                 nvIndex[i] = l_hold;
923             else if(nvIndex[i] > r_hold)
924                 nvIndex[i] = r_hold;
925             else
926                 nvIndex[i] = nvIndex[i];
927         }
928
929         nIndex = nvIndex[ left ];
930         nvIndex[ left ] = nvIndex[ nIndex ];
931         nvIndex[ nIndex ] = nIndex;
932
933         dTemp = dvData[ left ];
934         dvData[ left ] = dvData[ nIndex ];
935         dvData[ nIndex ] = dTemp;
936
937         nIndex = nvIndex[ left ];
938         nvIndex[ left ] = nvIndex[ nIndex ];
939         nvIndex[ nIndex ] = nIndex;
940
941         l_hold = nIndex;
942         r_hold = nvIndex[ right ];
943
944         for(i=left+1;i<right;i++){
945             if(nvIndex[i] < l_hold)
946                 nvIndex[i] = l_hold;
947             else if(nvIndex[i] > r_hold)
948                 nvIndex[i] = r_hold;
949             else
950                 nvIndex[i] = nvIndex[i];
951         }
952
953         nIndex = nvIndex[ left ];
954         nvIndex[ left ] = nvIndex[ nIndex ];
955         nvIndex[ nIndex ] = nIndex;
956
957         dTemp = dvData[ left ];
958         dvData[ left ] = dvData[ nIndex ];
959         dvData[ nIndex ] = dTemp;
960
961         nIndex = nvIndex[ left ];
962         nvIndex[ left ] = nvIndex[ nIndex ];
963         nvIndex[ nIndex ] = nIndex;
964
965         l_hold = nIndex;
966         r_hold = nvIndex[ right ];
967
968         for(i=left+1;i<right;i++){
969             if(nvIndex[i] < l_hold)
970                 nvIndex[i] = l_hold;
971             else if(nvIndex[i] > r_hold)
972                 nvIndex[i] = r_hold;
973             else
974                 nvIndex[i] = nvIndex[i];
975         }
976
977         nIndex = nvIndex[ left ];
978         nvIndex[ left ] = nvIndex[ nIndex ];
979         nvIndex[ nIndex ] = nIndex;
980
981         dTemp = dvData[ left ];
982         dvData[ left ] = dvData[ nIndex ];
983         dvData[ nIndex ] = dTemp;
984
985         nIndex = nvIndex[ left ];
986         nvIndex[ left ] = nvIndex[ nIndex ];
987         nvIndex[ nIndex ] = nIndex;
988
989         l_hold = nIndex;
990         r_hold = nvIndex[ right ];
991
992         for(i=left+1;i<right;i++){
993             if(nvIndex[i] < l_hold)
994                 nvIndex[i] = l_hold;
995             else if(nvIndex[i] > r_hold)
996                 nvIndex[i] = r_hold;
997             else
998                 nvIndex[i] = nvIndex[i];
999         }
1000
1001         nIndex = nvIndex[ left ];
1002         nvIndex[ left ] = nvIndex[ nIndex ];
1003         nvIndex[ nIndex ] = nIndex;
1004
1005         dTemp = dvData[ left ];
1006         dvData[ left ] = dvData[ nIndex ];
1007         dvData[ nIndex ] = dTemp;
1008
1009         nIndex = nvIndex[ left ];
1010         nvIndex[ left ] = nvIndex[ nIndex ];
1011         nvIndex[ nIndex ] = nIndex;
1012
1013         l_hold = nIndex;
1014         r_hold = nvIndex[ right ];
1015
1016         for(i=left+1;i<right;i++){
1017             if(nvIndex[i] < l_hold)
1018                 nvIndex[i] = l_hold;
1019             else if(nvIndex[i] > r_hold)
1020                 nvIndex[i] = r_hold;
1021             else
1022                 nvIndex[i] = nvIndex[i];
1023         }
1024
1025         nIndex = nvIndex[ left ];
1026         nvIndex[ left ] = nvIndex[ nIndex ];
1027         nvIndex[ nIndex ] = nIndex;
1028
1029         dTemp = dvData[ left ];
1030         dvData[ left ] = dvData[ nIndex ];
1031         dvData[ nIndex ] = dTemp;
1032
1033         nIndex = nvIndex[ left ];
1034         nvIndex[ left ] = nvIndex[ nIndex ];
1035         nvIndex[ nIndex ] = nIndex;
1036
1037         l_hold = nIndex;
1038         r_hold = nvIndex[ right ];
1039
1040         for(i=left+1;i<right;i++){
1041             if(nvIndex[i] < l_hold)
1042                 nvIndex[i] = l_hold;
1043             else if(nvIndex[i] > r_hold)
1044                 nvIndex[i] = r_hold;
1045             else
1046                 nvIndex[i] = nvIndex[i];
1047         }
1048
1049         nIndex = nvIndex[ left ];
1050         nvIndex[ left ] = nvIndex[ nIndex ];
1051         nvIndex[ nIndex ] = nIndex;
1052
1053         dTemp = dvData[ left ];
1054         dvData[ left ] = dvData[ nIndex ];
1055         dvData[ nIndex ] = dTemp;
1056
1057         nIndex = nvIndex[ left ];
1058         nvIndex[ left ] = nvIndex[ nIndex ];
1059         nvIndex[ nIndex ] = nIndex;
1060
1061         l_hold = nIndex;
1062         r_hold = nvIndex[ right ];
1063
1064         for(i=left+1;i<right;i++){
1065             if(nvIndex[i] < l_hold)
1066                 nvIndex[i] = l_hold;
1067             else if(nvIndex[i] > r_hold)
1068                 nvIndex[i] = r_hold;
1069             else
1070                 nvIndex[i] = nvIndex[i];
1071         }
1072
1073         nIndex = nvIndex[ left ];
1074         nvIndex[ left ] = nvIndex[ nIndex ];
1075         nvIndex[ nIndex ] = nIndex;
1076
1077         dTemp = dvData[ left ];
1078         dvData[ left ] = dvData[ nIndex ];
1079         dvData[ nIndex ] = dTemp;
1080
1081         nIndex = nvIndex[ left ];
1082         nvIndex[ left ] = nvIndex[ nIndex ];
1083         nvIndex[ nIndex ] = nIndex;
1084
1085         l_hold = nIndex;
1086         r_hold = nvIndex[ right ];
1087
1088         for(i=left+1;i<right;i++){
1089             if(nvIndex[i] < l_hold)
1090                 nvIndex[i] = l_hold;
1091             else if(nvIndex[i] > r_hold)
1092                 nvIndex[i] = r_hold;
1093             else
1094                 nvIndex[i] = nvIndex[i];
1095         }
1096
1097         nIndex = nvIndex[ left ];
1098         nvIndex[ left ] = nvIndex[ nIndex ];
1099         nvIndex[ nIndex ] = nIndex;
1100
1101         dTemp = dvData[ left ];
1102         dvData[ left ] = dvData[ nIndex ];
1103         dvData[ nIndex ] = dTemp;
1104
1105         nIndex = nvIndex[ left ];
1106         nvIndex[ left ] = nvIndex[ nIndex ];
1107         nvIndex[ nIndex ] = nIndex;
1108
1109         l_hold = nIndex;
1110         r_hold = nvIndex[ right ];
1111
1112         for(i=left+1;i<right;i++){
1113             if(nvIndex[i] < l_hold)
1114                 nvIndex[i] = l_hold;
1115             else if(nvIndex[i] > r_hold)
1116                 nvIndex[i] = r_hold;
1117             else
1118                 nvIndex[i] = nvIndex[i];
1119         }
1120
1121         nIndex = nvIndex[ left ];
1122         nvIndex[ left ] = nvIndex[ nIndex ];
1123         nvIndex[ nIndex ] = nIndex;
1124
1125         dTemp = dvData[ left ];
1126         dvData[ left ] = dvData[ nIndex ];
1127         dvData[ nIndex ] = dTemp;
1128
1129         nIndex = nvIndex[ left ];
1130         nvIndex[ left ] = nvIndex[ nIndex ];
1131         nvIndex[ nIndex ] = nIndex;
1132
1133         l_hold = nIndex;
1134         r_hold = nvIndex[ right ];
1135
1136         for(i=left+1;i<right;i++){
1137             if(nvIndex[i] < l_hold)
1138                 nvIndex[i] = l_hold;
1139             else if(nvIndex[i] > r_hold)
1140                 nvIndex[i] = r_hold;
1141             else
1142                 nvIndex[i] = nvIndex[i];
1143         }
1144
1145         nIndex = nvIndex[ left ];
1146         nvIndex[ left ] = nvIndex[ nIndex ];
1147         nvIndex[ nIndex ] = nIndex;
1148
1149         dTemp = dvData[ left ];
1150         dvData[ left ] = dvData[ nIndex ];
1151         dvData[ nIndex ] = dTemp;
1152
1153         nIndex = nvIndex[ left ];
1154         nvIndex[ left ] = nvIndex[ nIndex ];
1155         nvIndex[ nIndex ] = nIndex;
1156
1157         l_hold = nIndex;
1158         r_hold = nvIndex[ right ];
1159
1160         for(i=left+1;i<right;i++){
1161             if(nvIndex[i] < l_hold)
1162                 nvIndex[i] = l_hold;
1163             else if(nvIndex[i] > r_hold)
1164                 nvIndex[i] = r_hold;
1165             else
1166                 nvIndex[i] = nvIndex[i];
1167         }
1168
1169         nIndex = nvIndex[ left ];
1170         nvIndex[ left ] = nvIndex[ nIndex ];
1171         nvIndex[ nIndex ] = nIndex;
1172
1173         dTemp = dvData[ left ];
1174         dvData[ left ] = dvData[ nIndex ];
1175         dvData[ nIndex ] = dTemp;
1176
1177         nIndex = nvIndex[ left ];
1178         nvIndex[ left ] = nvIndex[ nIndex ];
1179         nvIndex[ nIndex ] = nIndex;
1180
1181         l_hold = nIndex;
1182         r_hold = nvIndex[ right ];
1183
1184         for(i=left+1;i<right;i++){
1185             if(nvIndex[i] < l_hold)
1186                 nvIndex[i] = l_hold;
1187             else if(nvIndex[i] > r_hold)
1188                 nvIndex[i] = r_hold;
1189             else
1190                 nvIndex[i] = nvIndex[i];
1191         }
1192
1193         nIndex = nvIndex[ left ];
1194         nvIndex[ left ] = nvIndex[ nIndex ];
1195         nvIndex[ nIndex ] = nIndex;
1196
1197         dTemp = dvData[ left ];
1198         dvData[ left ] = dvData[ nIndex ];
1199         dvData[ nIndex ] = dTemp;
1200
1201         nIndex = nvIndex[ left ];
1202         nvIndex[ left ] = nvIndex[ nIndex ];
1203         nvIndex[ nIndex ] = nIndex;
1204
1205         l_hold = nIndex;
1206         r_hold = nvIndex[ right ];
1207
1208         for(i=left+1;i<right;i++){
1209             if(nvIndex[i] < l_hold)
1210                 nvIndex[i] = l_hold;
1211             else if(nvIndex[i] > r_hold)
1212                 nvIndex[i] = r_hold;
1213             else
1214                 nvIndex[i] = nvIndex[i];
1215         }
1216
1217         nIndex = nvIndex[ left ];
1218         nvIndex[ left ] = nvIndex[ nIndex ];
1219         nvIndex[ nIndex ] = nIndex;
1220
1221         dTemp = dvData[ left ];
1222         dvData[ left ] = dvData[ nIndex ];
1223         dvData[ nIndex ] = dTemp;
1224
1225         nIndex = nvIndex[ left ];
1226         nvIndex[ left ] = nvIndex[ nIndex ];
1227         nvIndex[ nIndex ] = nIndex;
1228
1229         l_hold = nIndex;
1230         r_hold = nvIndex[ right ];
1231
1232         for(i=left+1;i<right;i++){
1233             if(nvIndex[i] &lt
```

```

149     }
150     dTemp = dvDist[nvShip[i]];
151     QuickSort(dvDist, nvIndex);
152     if(dTemp > dvDist[0])
153         nvShip[i] = nvIndex[0];
154     }
155     return 0;
156 } // GetShip()

157
158 /* Get subspace dimension from fuzzy subspace memberships
159 */
160 int GetDim(
161     dMatrix &dmW,
162     nMatrix &nMdim
163 ){
164     dVector dvW;
165     nVector nvIndex;
166     int i,j,nCut;
167     double dSum1,dSum,dTemp,dTemp1;
168
169     dvW = dVector(d,0.0);
170     nvIndex = nVector(d,0);
171     for(i=0;i<k;i++){
172         dSum = 0.0;
173         for(j=0;j<d;j++){
174             dvW[j] = dmW[i][j];
175             nvIndex[j] = j;
176             dSum += dvW[j];
177         }
178         QuickSort(dvW, nvIndex);
179         dSum1 = dvW[0];
180         nCut = 0;
181         dTemp = pow(dSum1,2)+ pow(dSum-dSum1,2)/(d-1);
182         for(j=1;j<d-1;j++){
183             dSum1 += dvW[j];
184             dTemp1 = pow(dSum1,2)/(j+1) + pow(dSum-dSum1,2)/(d-j-1);
185             if(dTemp1 > dTemp){
186                 dTemp = dTemp1;
187                 nCut = j;
188             }
189         }
190         for(j=nCut+1;j<d;j++)
191             nmDim[i].push_back(nvIndex[j]+1);
192     }
193
194     return 0;
195 } // GetDim()

196
197 double CalObj(
198     dMatrix &dmData,
199     dMatrix &dmW,
200     dMatrix &dmZ,
201     nVector &nvShip
202 ){
203     int i,j;
204     double dTemp;
205
206     dTemp = 0.0;
207     for(i=0;i<n;i++){
208         dTemp += Distance(dmW[nvShip[i]],dmZ[nvShip[i]],dmData[i]);
209     }
210
211     return dTemp;
212 } // CalObj()

213
214 double Variance(
215     int j,
216

```

```

217     int h,
218     dMatrix &dmData,
219     dMatrix &dmZ,
220     nVector &nvShip
221   ){
222     int i;
223     double dTemp;
224
225     dTemp = 0.0;
226     for(i=0;i<n;i++){
227       if(nvShip[i]==j)
228         dTemp += pow(dmData[i][h] - dmZ[j][h],2);
229     }
230
231     return dTemp;
232   } // Variance()
233
234   int EstW(
235     dMatrix &dmData,
236     dMatrix &dmZ,
237     dMatrix &dmW,
238     nVector &nvShip
239   ){
240     int j,h;
241     double dTemp;
242     dVector dvVar;
243
244     dvVar = dVector(d,0.0);
245     for(j=0;j<k;j++){
246       dTemp = 0.0;
247       for(h=0;h<d;h++){
248         dvVar[h] = Variance(j,h,dmData,dmZ,nvShip)+epsilon;
249         dTemp += pow(1/dvVar[h],1/(alpha-1));
250       }
251       for(h=0;h<d;h++)
252         dmW[j][h] = pow(1/dvVar[h],1/(alpha-1))/dTemp;
253     }
254     return 0;
255   } // EstW()
256
257   int EstZ(
258     dMatrix &dmData,
259     dMatrix &dmZ,
260     nVector &nvShip
261   ){
262     int i,j,h;
263     double dTemp;
264     nVector nvSize;
265
266     nvSize = nVector(k,0);
267     for(j=0;j<k;j++){
268       for(h=0;h<d;h++){
269         dmZ[j][h] = 0.0;
270       }
271     }
272     for(i=0;i<n;i++){
273       nvSize[nvShip[i]]++;
274       for(h=0;h<d;h++)
275         dmZ[nvShip[i]][h] += dmData[i][h];
276     }
277     for(j=0;j<k;j++){
278       for(h=0;h<d;h++){
279         dmZ[j][h] /= nvSize[j];
280       }
281     }
282
283     return 0;
284   } // EstZ()

```

```

285
286 double FuzzySub(
287     dMatrix &dmData ,
288     dMatrix &dmW,
289     dMatrix &dmZ,
290     nVector &nvShip
291 ){
292     int i,j;
293     double dObj ,dNewObj , dError ;
294
295     Initialization (dmZ,dmW,dmData );
296     GetShip (dmData ,dmW,dmZ, nvShip );
297     dObj = CalObj (dmData ,dmW,dmZ, nvShip );
298
299     dError = 1.0;
300     while(dError >0){
301         EstZ (dmData ,dmZ, nvShip );
302         EstW (dmData ,dmZ,dmW, nvShip );
303         GetShip (dmData ,dmW,dmZ, nvShip );
304         dNewObj = CalObj (dmData ,dmW,dmZ, nvShip );
305         dError = fabs (dNewObj-dObj );
306         dObj = dNewObj ;
307     }
308
309     return dObj;
310 } //FuzzySub()
311
312 int ReadData(
313     const char *filename ,
314     sVector &svLabel ,
315     sVector &svAtt ,
316     dMatrix &dmData
317 ){
318     int i,j;
319     ifstream ifsFile ;
320     char buff[MAX_FIELD_BUFFER];
321
322     try{
323         ifsFile .open(filename );
324         // count the number of attributes;
325         i = 1;
326         do{
327             if (ifsFile .peek() == ',')
328                 i++;
329         } while (ifsFile .get() != '\n');
330         d = i-1; // the first column is record name
331
332         // now count the number of records
333         i = 0;
334         do{
335             ifsFile .ignore(MAX_LINE_BUFFER, '\n');
336             i++;
337         } while (!ifsFile .eof());
338         n = i-1;// one return at the end of the last line of the file
339         // NO return after the last line of the file
340         ifsFile .close();
341
342         // open the file again
343         ifsFile .open(filename );
344
345         svAtt = sVector(d,"v");
346         ifsFile .getline(buff , MAX_FIELD_BUFFER, ',' );
347         for (i=0; i<d; i++){
348             if (i == d-1){
349                 ifsFile .getline(buff , MAX_FIELD_BUFFER, '\n');
350                 // remove '\n'
351                 for(j=0;j<sizeof(buff);j++){
352                     if(buff[j] == '\n'){

```

```

353             buff[j] = 'u';
354         }
355     }
356     svAtt[i] = buff;
357 } else {
358     ifsFile.getline(buff, MAX_FIELD_BUFFER, ',');
359     svAtt[i] =buff;
360 }
361 }
362 dmData = dMatrix(n);
363 i = 0;
364 while (!ifsFile.eof() && i<n){
365     // get the record name
366     ifsFile.getline(buff, MAX_FIELD_BUFFER, ',');
367     svLabel.push_back(buff);
368     for(j=0; j<d; j++){
369         if (j == d-1){ // denotes the end of the line
370             ifsFile.getline(buff, MAX_FIELD_BUFFER, '\n');
371             dmData[i].push_back(atof(buff));
372         } else{
373             ifsFile.getline(buff, MAX_FIELD_BUFFER, ',');
374             dmData[i].push_back(atof(buff));
375         }
376     }
377     i++;
378 }
380
381     ifsFile.close();
382     return 0;
383 }
384 catch(\dots){
385     cout<<"reading data error"<<endl;
386     return -1;
387 }
388 } //ReadData()

389 int main(int argc, char *argv[])
390 {
391     dMatrix dmData,dmZ,dmW,dmBestW,dmBestZ;
392     nMatrix nmDim;
393     nVector nvShip;
394     sVector svLabel,svAtt;
395     double dObj,dBestObj,dAvg;
396     int i,j,jj,tag,nRun;
397     ofstream ofsFile;
398
399     // Initialize random generator
400     srand( time(NULL) );
401
402     // Read data from file
403     ReadData("Data.csv",svLabel,svAtt,dmData);
404
405     // Get the number of clusters
406     cout<<"please input the number of clusters :\n";
407     cin>>k;
408
409     cout<<n<<endl<<d<<endl<<k<<endl;
410
411     // Initialize necessary parameters
412     nmDim = nMatrix(k);
413     nvShip = nVector(n,0);
414     dmZ = dMatrix(k);
415     dmBestZ = dMatrix(k);
416     dmW = dMatrix(k);
417     dmBestW = dMatrix(k);
418     for(i=0;i<k;i++){
419         dmZ[i] = dVector(d,0.0);
420

```

```

421     dmBestZ[ i ] = dVector( d , 0.0 );
422     dmW[ i ] = dVector( d , 1.0 );
423     dmBestW[ i ] = dVector( d , 1.0 );
424 }
425
426 // Specify the number of runs
427 nRun = 5;
428
429 cout << "Run_1" << endl ;
430 dBestObj = FuzzySub( dmData , dmBestW , dmBestZ , nvShip );
431 dAvg = dBestObj ;
432 for( i=1; i<nRun; i++ ){
433     cout << "Run_" << i+1 << endl ;
434     dObj = FuzzySub( dmData , dmW , dmZ , nvShip );
435     dAvg += dObj ;
436     if( dObj < dBestObj ){
437         dBestObj = dObj ;
438         for( j=0; j < k; j++ ){
439             for( jj=0; jj < d; jj++ ){
440                 dmBestW[ j ][ jj ] = dmW[ j ][ jj ];
441                 dmBestZ[ j ][ jj ] = dmZ[ j ][ jj ];
442             }
443         }
444     }
445 }
446
447 // Calculate the subspace dimensions for each cluster
448 GetDim( dmBestW , nmDim );
449
450 // Print record name and record membership to a file
451 ofsFile .open( "DataMembership.csv" );
452 for( i=0; i < n; i++ ){
453     ofsFile << svLabel[ i ] << "," << nvShip[ i ] + 1 << endl ;
454 }
455 ofsFile .close ();
456
457 // Print record membership and records to a file so that
458 // the clusters can be plotted in MATLAB
459 ofsFile .open( "DataResult.csv" );
460 for( i=0; i < n; i++ ){
461     ofsFile << nvShip[ i ] + 1 << "," ;
462     for( j=0; j < d - 1; j++ ){
463         ofsFile << dmData[ i ][ j ] << "," ;
464         ofsFile << dmData[ i ][ d - 1 ] << endl ;
465     }
466     ofsFile .close ();
467
468 system( "PAUSE" );
469 return 0;
470 }
```


Bibliography

- Achlioptas, D. and McSherry, F. (2001). Fast computation of low rank matrix approximations. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on theory of computing*, pages 611–618. New York: ACM Press.
- Agarwal, P. and Mustafa, N. (2004). k -means projective clustering. In *Proceedings of the twenty-third ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems (PODS)*, pages 155–165. Paris: ACM Press.
- Aggarwal, C., Han, J., Wang, J., and Yu, P. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on very large data bases*, pages 81–92. Berlin: Morgan Kaufmann.
- Aggarwal, C., Han, J., Wang, J., and Yu, P. (2004). A framework for projected clustering of high dimensional data streams. In *Proceedings of the thirtieth international conference on very large data bases*, pages 852–863. Toronto: Morgan Kaufmann.
- Aggarwal, C., Wolf, J., Yu, P., Procopiuc, C., and Park, J. (1999). Fast algorithms for projected clustering. In *Proceedings of the 1999 ACM SIGMOD international conference on management of data*, pages 61–72. Philadelphia: ACM Press.
- Aggarwal, C. and Yu, P. (2000). Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the 2000 ACM SIGMOD international conference on management of data*, pages 70–81. Dallas, TX: ACM Press.
- Aggarwal, C. and Yu, P. (2002). Redefining clustering for high-dimensional applications. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):210–225.
- Agrawal, R., Faloutsos, C., and Swami, A. (1993). Efficient similarity search in sequence databases. In *FODO '93: Proceedings of the 4th international conference on foundations of data organization and algorithms*, volume 730 of *Lecture Notes in Computer Science*, pages 69–84. New York: Springer-Verlag.
- Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Record ACM Special Interest Group on Management of Data*, pages 94–105. New York: ACM Press.

- Agrawal, R., Lin, K., Sawhney, H., and Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB '95: Proceedings of the 21st international conference on very large data bases*, pages 490–501. Zurich: Morgan Kaufmann.
- Aho, A., Hopcroft, J., and Ullman, J. (1974). *The design and analysis of computer algorithms*. Addison-Wesley series in computer science and information processing. Reading, MA; Don Mills, ON: Addison-Wesley.
- Al-Sultan, K. (1995). A tabu search approach to the clustering problem. *Pattern Recognition*, 28(9):1443–1451.
- Al-Sultan, K. and Fedjki, C. (1997). A tabu search-based algorithm for the fuzzy clustering problem. *Pattern Recognition*, 30(12):2023–2030.
- Alon, N., Matias, Y., and Szegedy, M. (1996). The space complexity of approximating the frequency moments. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on theory of computing*, pages 20–29. New York: ACM Press.
- Alpert, C. and Yao, S. (1995). Spectral partitioning: the more eigenvectors, the better. In *DAC '95: Proceedings of the 32nd ACM/IEEE conference on design automation*, pages 195–200. San Francisco, CA: ACM Press.
- Alsabti, K., Ranka, S., and Singh, V. (1998). An efficient k-means clustering algorithm. In *Proceedings of IPPS/SPDP Workshop on high performance data mining*, pages 881–892. Orlando, Florida: IEEE Computer Society.
- Amir, A., Kashi, R., Netanyahu, N., Keim, D., and Wawryniuk, M. (2003). Analyzing high-dimensional data by subspace validity. In *Third IEEE international conference on data mining, 2003. ICDM 2003*, pages 473–476. Melbourne, FL: IEEE.
- Anderberg, M. (1973). *Cluster analysis for applications*. New York: Academic Press.
- Andrews, D. (1972). Plots of high-dimensional data. *Biometrics*, 28(1):125–136.
- Andrews, H. and Patterson, C. (1976a). Singular value decomposition image coding. *IEEE Transactions on Communications*, 4:425–432.
- Andrews, H. and Patterson, C. (1976b). Singular value decompositions and digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24:26–53.
- Andrienko, G. and Andrienko, N. (2004). Parallel coordinates for exploring properties of subsets. In *CMV '04: Proceedings of the second international conference on coordinated & multiple views in exploratory visualization (CMV'04)*, pages 93–104. Washington, DC: IEEE Computer Society.
- Augustson, J. and Minker, J. (1970). An analysis of some graph theoretical cluster techniques. *Journal of the Association for Computing Machinery*, 17(4):571–588.
- Azoff, E. (1994). *Neural Network Time Series Forecasting of Financial Markets*. New York: John Wiley & Sons.

- Babcock, B., Datar, M., and Motwani, R. (2002). Sampling from a moving window over streaming data. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 633–634. Philadelphia, PA: SIAM.
- Babcock, B., Datar, M., Motwani, R., and O’Callaghan, L. (2003). Maintaining variance and k -medians over data stream windows. In *PODS ’03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pages 234–243. New York: ACM Press.
- Babu, G. and Murty, M. (1993). A near-optimal initial seed value selection in k -means algorithm using a genetic algorithm. *Pattern Recognition Letters*, 14(10):763–769.
- Bagnall, A. and Janacek, G. (2004). Clustering time series from ARMA models with clipped data. In *KDD ’04: Proceedings of the 2004 ACM SIGKDD international conference on knowledge discovery and data mining*, pages 49–58. New York: ACM Press.
- Bagnall, A., Janacek, G., and Zhang, M. (2003). Clustering time series from mixture polynomial models with discretised data. Technical Report CMP-C03-17, School of Computing Sciences, University of East Anglia.
- Baker, F. and Hubert, L. (1976). A graph-theoretic approach to goodness-of-fit in complete-link hierarchical clustering. *Journal of the American Statistical Association*, 71(356):870–878.
- Bandyopadhyay, S. and Maulik, U. (2002). An evolutionary technique based on k-means algorithm for optimal clustering in r^n . *Information Sciences*, 146(14):221–237.
- Bandyopadhyay, S., Murthy, C., and Pal, S. (1995). Pattern classification with genetic algorithms. *Pattern Recognition Letters*, 16(8):801–808.
- Banfield, C. (1976). Statistical algorithms: Algorithm AS 102: Ultrametric distances for a single linkage dendrogram. *Applied Statistics*, 25(3):313–315.
- Banfield, J. and Raftery, A. (1993). Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49(3):803–821.
- Bar-Joseph, Z., Gerber, G., Gifford, D., Jaakkola, T., and Simon, I. (2002). A new approach to analyzing gene expression time series data. In *RECOMB ’02: Proceedings of the sixth annual international conference on computational biology*, pages 39–48. New York: ACM Press.
- Baraldi, A. and Blonda, P. (1999a). A survey of fuzzy clustering algorithms for pattern recognition. I. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 29(6):778–785.
- Baraldi, A. and Blonda, P. (1999b). A survey of fuzzy clustering algorithms for pattern recognition. II. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 29(6):786–801.
- Barbará, D. (2002). Requirements for clustering data streams. *ACM SIGKDD Explorations Newsletter*, 3(2):23–27.

- Barbará, D., Li, Y., and Couto, J. (2002). COOLCAT: An entropy-based algorithm for categorical clustering. In *Proceedings of the eleventh international conference on information and knowledge management*, pages 582–589. McLean, VA: ACM Press.
- Barber, C., Dobkin, D., and Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483.
- Batagelj, V. (1981). Note on ultrametric hierarchical clustering algorithms. *Psychometrika*, 46(3):351–352.
- Baulieu, F. (1989). A classification of presence/absence based dissimilarity coefficients. *Journal of Classification*, 6:233–246.
- Baulieu, F. (1997). Two variant axiom systems for presence/absence based dissimilarity coefficients. *Journal of Classification*, 14:159–170.
- Bay, S. and Pazzani, M. (1999). Detecting change in categorical data: Mining contrast sets. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 302–306. New York: ACM Press.
- Belacel, N., Hansen, P., and Mladenović, N. (2002). Fuzzy J -means: A new heuristic for fuzzy clustering. *Pattern Recognition*, 35(10):2193–2200.
- Bell, D., McErlean, F., and Stewart, P. (1990). Application of simulated annealing to clustering tuples in databases. *Journal of the American Society for Information Science and Technology*, 42(2):98–110.
- Bellman, R., Kalaba, R., and Zadeh, L. (1966). Abstraction and pattern classification. *Journal of Mathematical Analysis and Applications*, 2:581–586.
- Bensmail, H., Celeux, G., Raftery, A., and Robert, C. (1997). Inference in model-based cluster analysis. *Statistics and Computing*, 7(1):1–10.
- Bentley, J. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.
- Bentley, J. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229.
- Bentley, J. and Friedman, J. (1979). Data structures for range searching. *ACM Computing Surveys (CSUR)*, 11(4):397–409.
- Berndt, D. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *AAAI-94 Workshop on knowledge discovery in databases*, pages 229–248. Seattle, Washington: AAAI Press.
- Berry, M. and Linoff, G. (2000). *Mastering Data Mining*. New York: John Wiley & Sons.
- Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is “nearest neighbor” meaningful? In *Database Theory—ICDT ’99, 7th international conference, Jerusalem, Israel, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. London, UK: Springer.

- Beygelzimer, A., Perng, C., and Ma, S. (2001). Fast ordering of large categorical datasets for better visualization. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*, pages 239–244. New York: ACM Press.
- Bezdek, J. (1974a). Cluster validity with fuzzy sets. *Journal of Cybernetics*, 3(3):58–72.
- Bezdek, J. (1974b). *Fuzzy Mathematics in Pattern Classification*. PhD thesis, Cornell University, Ithaca, NY.
- Bezdek, J. (1980). A convergence theorem for the fuzzy ISODATA clustering algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):1–8.
- Bezdek, J. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press.
- Bezdek, J., Hathaway, R., Sabin, M., and Tucker, W. (1992). Convergence theory for fuzzy c -means: Counterexamples and repairs. In Bezdek, J., and Pal, S., editors, *Fuzzy Models for Pattern Recognition: Methods that Search for Approximate Structures in Data*, pages 138–142. New York: IEEE Press.
- Bijnen, E. (1973). *Cluster Analysis: Survey and Evaluation of Techniques*. Tilburg, The Netherlands: Tilburg University Press.
- Binder, D. (1978). Bayesian cluster analysis. *Biometrika*, 65(1):31–38.
- Blake, C. and Merz, C. (1998). UCI repository of machine learning databases. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- Blum, A. (1992). *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems*. New York: John Wiley & Sons.
- Bobisud, H. and Bobisud, L. (1972). A metric for classification. *Taxon*, 21:607–613.
- Bobrowski, L. and Bezdek, J. (1991). c -means clustering with the l_1 and l_∞ norms. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):545–554.
- Bock, H. (1985). On some significance tests in cluster analysis. *Journal of Classification*, 2:77–108.
- Bock, H. (1989). Probabilistic aspects in cluster analysis. In Opitz, O., editor, *Conceptual and Numerical Analysis of Data*, pages 12–44. Augsburg, FRG: Springer-Verlag.
- Bock, H. (1996). Probabilistic models in cluster analysis. *Computational Statistics and Data Analysis*, 23(1):5–28.
- Bollobás, B., Das, G., Gunopulos, D., and Mannila, H. (1997). Time-series similarity problems and well-separated geometric sets. In *SCG '97: Proceedings of the thirteenth annual symposium on computational geometry*, pages 454–456. New York: ACM Press.
- Borg, I. and Groenen, P. (1997). *Modern Multidimensional Scaling*. Springer Series in Statistics. Berlin: Springer.

- Borodin, A., Ostrovsky, R., and Rabani, Y. (1999). Subquadratic approximation algorithms for clustering problems in high dimensional spaces. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on theory of computing*, pages 435–444. New York: ACM Press.
- Bottou, L. and Bengio, Y. (1995). Convergence properties of the k -means algorithms. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 585–592. Cambridge, MA: The MIT Press.
- Boyles, R. (1983). On the convergence of the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 45(1):47–50.
- Bozdogan, H. (1987). Model selection and Akaike's information criterion (AIC): The general theory and its analytical extensions. *Psychometrika*, 52:345–370.
- Bozkaya, T., Yazdani, N., and Özsoyoglu, M. (1997). Matching and indexing sequences of different lengths. In *CIKM '97: Proceedings of the sixth international conference on information and knowledge management*, pages 128–135. New York: ACM Press.
- Bradley, P. and Fayyad, U. (1998). Refining initial points for k -means clustering. In *Proceedings of the fifteenth international conference on machine learning*, pages 91–99. San Francisco: Morgan Kaufmann.
- Bradley, P., Fayyad, U., and Reina, C. (1998). Scaling clustering algorithms to large databases. In *Proceedings of 4th international conference on knowledge discovery and data mining*, pages 9–15. New York: AAAI Press.
- Bradley, P. and Mangasarian, O. (2000). k -plane clustering. *Journal of Global Optimization*, 16(1):23–32.
- Brazma, A. and Vilo, J. (2000). Gene expression data analysis. *Biochemical Societies*, 480:17–24.
- Broder, A., Glassman, S., Manasse, M., and Zweig, G. (1997). Syntactic clustering of the web. In *Selected papers from the sixth international conference on the World Wide Web*, pages 1157–1166. Elsevier Science Publishers Ltd.
- Buhmann, J. (2003). Data clustering and learning. In Arbib, M., editor, *The Handbook of Brain Theory and Neural Networks*, pages 308–312. Cambridge, MA: The MIT Press.
- Calinski, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3:1–27.
- Cao, Y., and Wu, J. (2002). Projective ART for clustering data sets in high dimensional spaces. *Neural Networks*, 15(1):105–120.
- Caraça-Valente, J. and López-Chavarrías, I. (2000). Discovering similar patterns in time series. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 497–505. New York: ACM Press.

- Carmichael, J., George, J. A., and Julius, R. (1968). Finding natural clusters. *Systematic Zoology*, 17(2):144–150.
- Carpenter, G. and Grossberg, S. (1987a). ART2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930.
- Carpenter, G. and Grossberg, S. (1987b). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing*, 37:54–115.
- Carpenter, G. and Grossberg, S. (1990). ART3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, 3:129–152.
- Carroll, J. and Arabie, P. (1980). Multidimensional scaling. *Annual Review of Psychology*, 31:607–649.
- Cattell, R. (1949). r_p and other coefficients of pattern similarity. *Psychometrika*, 14(4):279–298.
- Celeux, G. and Govaert, G. (1995). Gaussian parsimonious clustering models. *Pattern Recognition*, 28(5):781–793.
- Cezkanowski, J. (1909). Zur differentialdiagnose der neandertalgruppe. *Korrespondenz-Blatt deutsch. Ges. Anthropol. Ethnol. Urgesch*, 40:44–47.
- Chakrabarti, K., Keogh, E., Mehrotra, S., and Pazzani, M. (2002). Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems*, 27(2):188–228.
- Chan, K. and Fu, W. (1999). Efficient time series matching by wavelets. In *ICDE '99: Proceedings of the 15th international conference on data engineering*, pages 126–133. Washington, DC: IEEE Computer Society.
- Chan, P., Schlag, M., and Zien, J. (1994). Spectral k-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1088–1096.
- Chang, C. and Ding, Z. (2004). Categorical data visualization and clustering using subjective factors. In *6th international conference on data warehousing and knowledge discovery*, volume 3181, pages 229–238. Zaragoza, Spain: Springer-Verlag.
- Chang, C. and Ding, Z. (2005). Categorical data visualization and clustering using subjective factors. *Data & Knowledge Engineering*, 53(3):243–262.
- Chang, J. and Jin, D. (2002). A new cell-based clustering method for large, high-dimensional data in data mining applications. In *SAC '02: Proceedings of the 2002 ACM symposium on applied computing*, pages 503–507. New York: ACM Press.
- Charikar, M., Chaudhuri, S., Motwani, R., and Narasayya, V. (2000). Towards estimation error guarantees for distinct values. In *PODS '00: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pages 268–279. New York: ACM Press.

- Charikar, M. and Guha, S. (1999). Improved combinatorial algorithms for the facility location and k -median problems. In *FOCS '99: Proceedings of the 40th annual symposium on foundations of computer science*, pages 378–388. Washington: IEEE Computer Society.
- Charikar, M., O'Callaghan, L., and Panigrahy, R. (2003). Better streaming algorithms for clustering problems. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on theory of computing*, pages 30–39. New York: ACM Press.
- Chaturvedi, A., Green, P., and Carroll, J. (2001). k -modes clustering. *Journal of Classification*, 18(1):35–55.
- Chen, H. and Meer, P. (2005). Robust fusion of uncertain information. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(3):578–586.
- Chen, J., Ching, R., and Lin, Y. (2004). An extended study of the k -means algorithm for data clustering and its applications. *Journal of the Operational Research Society*, 55(9):976–987.
- Chen, N., Chen, A., and Zhou, L. (2002). An incremental grid density-based clustering algorithm. *Journal of Software*, 13(1):1–7.
- Cheng, C., Fu, A., and Zhang, Y. (1999). Entropy-based subspace clustering for mining numerical data. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 84–93. New York: ACM Press.
- Cheng, C., Lee, W., and Wong, K. (2002). A genetic algorithm-based clustering approach for database partitioning. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 32(3):215–230.
- Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799.
- Chernoff, H. (1973). The use of faces to represent points in k -dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368.
- Chiou, Y. and Lan, L. (2001). Genetic clustering algorithms. *European Journal of Operational Research*, 135(2):413–427.
- Chiu, B., Keogh, E., and Lonardi, S. (2003). Probabilistic discovery of time series motifs. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 493–498. New York: ACM Press.
- Cho, R., Campbell, M., Winzeler, E., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T., Gabrielian, A., Landsman, D., Lockhart, D., and Davis, R. (1998). A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular Cell*, 2(1):65–73.
- Chrétien, S. and Hero III, A. (1998). Acceleration of the EM algorithm via proximal point iterations. In *Proceedings of the IEEE International Symposium on Information Theory*, page 444. New York: IEEE Information Theory Society.

- Christopher, M. (1969). Cluster analysis and market segmentation. *British Journal of Marketing*, 3(2):99–102.
- Chu, K. and Wong, M. (1999). Fast time-series searching with scaling and shifting. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pages 237–248. New York: ACM Press.
- Clatworthy, J., Buick, D., Hankins, M., Weinman, J., and Horne, R. (2005). The use and reporting of cluster analysis in health psychology: A review. *British Journal of Health Psychology*, 10(3):329–358.
- Cochran, W. and Hopkins, C. (1961). Some classification problems with multivariate qualitative data. *Biometrics*, 17(1):10–23.
- Comaniciu, D. and Meer, P. (1999). Mean shift analysis and applications. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1197–1203. Kerkyra, Greece: IEEE Computer Society.
- Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.
- Conover, W. and Iman, R. (1981). Rank transformations as a bridge between parametric and nonparametric statistics. *The American Statistician*, 35(3):124–129.
- Constantine, A. and Gower, J. (1978). Graphical representation of asymmetric matrices. *Applied Statistics*, 27:297–304.
- Constantinescu, P. (1966). The classification of a set of elements with respect to a set of properties. *The Computer Journal*, 8(4):352–357.
- Cormack, R. (1971). A review of classification. *Journal of the Royal Statistical Society. Series A (General)*, 134(3):321–367.
- Cormen, T., Stein, C., Rivest, R., and Leiserson, C. (2001). *Introduction to Algorithms*. New York: McGraw-Hill Higher Education.
- Costa, I., de Carvalho, F., and de Souto, M. (2004). Comparative analysis of clustering methods for gene expression time course data. *Genetics and Molecular Biology*, 27:623–631.
- Cotofrei, P. and Stoffel, K. (2002). Classification rules + time = temporal rules. In *ICCS '02: Proceedings of the international conference on computational science—Part I*, volume 2329 of *Lecture Notes in Computer Science*, pages 572–581. London, UK: Springer-Verlag.
- Cowgill, M., Harvey, R., and Watson, L. (1999). A genetic algorithm approach to cluster analysis. *Computers & Mathematics with Applications*, 37(7):99–108.
- Cox, T. and Cox, M. (1994). *Multidimensional Scaling*, volume 59 of *Monographs on Statistics and Applied Probability*. London: Chapman & Hall.

- Cuesta-Albertos, J., Gordaliza, A., and Matrán, C. (1997). Trimmed k -means: An attempt to robustify quantizers. *The Annals of Statistics*, 25(2):553–576.
- Cunningham, K. and Ogilvie, J. (1972). Evaluation of hierarchical grouping techniques: A preliminary study. *The Computer Journal*, 15(3):209–213.
- Das, G., Gunopulos, D., and Mannila, H. (1997). Finding similar time series. In *Proceedings of the first European symposium on principles of data mining and knowledge discovery*, pages 88–100. New York: Springer-Verlag.
- Das, G., Lin, K., Mannila, H., Renganathan, G., and Smyth, P. (1998). Rule discovery from time series. In *KDD '98: Proceedings of the 4th international conference on knowledge discovery and data mining*, pages 16–22. New York: ACM Press.
- Dasgupta, A. and Raftery, A. (1998). Detecting features in spatial point processes with clutter via model-based clustering. *Journal of the American Statistical Association*, 93(441):294–302.
- Dash, M., Liu, H., and Xu, X. (2001). ‘1+1>2’: merging distance and density based clustering. In *Proceedings of the seventh international conference on database systems for advanced applications*, pages 32–39. Hong Kong: IEEE Computer Society.
- Datar, M., Gionis, A., Indyk, P., and Motwani, R. (2002). Maintaining stream statistics over sliding windows (extended abstract). In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms*, pages 635–644. Philadelphia: SIAM.
- Dave, R. (1996). Validating fuzzy partitions obtained through c -shells clustering. *Pattern Recognition Letters*, 17(6):613–623.
- Davies, D. and Bouldin, D. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227.
- Day, N. (1969). Estimating the components of a mixture of normal distributions. *Biometrika*, 56(3):463–474.
- Day, W. and Edelsbrunner, H. (1984). Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(7):7–24.
- De Backer, S., Naud, A., and Scheunders, P. (1998). Non-linear dimensionality reduction techniques for unsupervised feature extraction. *Pattern Recognition Letters*, 19(8):711–720.
- De Smet, F., Mathys, J., Marchal, K., Thijs, G., De Moor, B., and Moreau, Y. (2002). Adaptive quality-based clustering of gene expression profiles. *Bioinformatics*, 18(5):735–746.
- Debregeas, A. and Hebrail, G. (1998). Interactive interpretation of Kohonen maps applied to curves. In *KDD '98: Proceedings of the 4th international conference on knowledge discovery and data mining*, pages 179–183. New York: ACM Press.

- Defays, D. (1977). An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366.
- Delattre, M. and Hansen, P. (1980). Bicriterion cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4):277–291.
- Delgado, M., Skárm̄eta, A., and Barberá, H. (1997). A tabu search approach to the fuzzy clustering problem. In *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems*, volume 1, pages 125–130. Barcelona, Spain: IEEE.
- Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.
- Deng, K. and Moore, A. (1995). Multiresolution instance-based learning. In *Proceedings of the twelfth international joint conference on artificial intelligence*, pages 1233–1239. San Francisco: Morgan Kaufmann.
- D’haeseleer, P. (2005). How does gene expression clustering work? *Nature Biotechnology*, 23:1499–1501.
- D’haeseleer, P., Wen, X., Fuhrman, S., and Somogyi, R. (1998). Mining the gene expression matrix: Inferring gene relationships from large scale gene expression data. In *Proceedings of the second international workshop on information processing in cell and tissues*, pages 203–212. New York: Plenum Press.
- Dhillon, I., Modha, D., and Spangler, W. (1998). Visualizing class structure of multidimensional data. In *Proceedings of the 30th Symposium on the interface: Computing science and statistics*, volume 30, pages 488–493. Minneapolis, MN: Interface Foundation of North America.
- Dinesh, M., Gowda, K., and Ravi, T. (1997). Classification of symbolic data using fuzzy set theory. In *Proceedings of the first international conference on knowledge-based intelligent electronic systems*, volume 2, pages 383–386. Adelaide, Australia: IEEE Press.
- Ding, C. and He, X. (2004). k -means clustering via principal component analysis. In *Twenty-first international conference on machine learning*. New York: ACM Press.
- Domeniconi, C., Papadopoulos, D., Gunopulos, D., and Ma, S. (2004). Subspace clustering of high dimensional data. In *Proceedings of the SIAM International Conference on Data Mining*, Lake Buena Vista, FL: SIAM.
- Drineas, P., Frieze, A., Kannan, R., Vempala, S., and Vinay, V. (1999). Clustering in large graphs and matrices. In *Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms*, pages 291–299. Philadelphia: SIAM.
- Drineas, P., Frieze, A., Kannan, R., Vempala, S., and Vinay, V. (2004). Clustering large graphs via the singular value decomposition. *Machine Learning*, 56(1–3):9–33.
- Dubes, R. (1987). How many clusters are best? An experiment. *Pattern Recognition*, 20(6):645–663.

- DuBien, J. and Warde, W. (1979). A mathematical comparison of the members of an infinite family of agglomerative clustering algorithms. *The Canadian Journal of Statistics*, 7:29–38.
- Dunn, J. (1974a). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57.
- Dunn, J. (1974b). Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4:95–104.
- Dunn, J. (1977). Indices of partition fuzziness and the detection of clusters in large data sets. In Gupta, M., Saridis, G., and Gaines, B., editors, *Fuzzy Automata and Decision Processes*, pages 271–284. New York: North-Holland.
- Duran, B. and Odell, P. (1974). *Cluster Analysis: A Survey*, volume 100 of *Lecture Notes in Economics and Mathematical Systems*. Berlin, Heidelberg, New York: Springer-Verlag.
- Edwards, A. and Cavalli-Sforza, L. (1965). A method for cluster analysis. *Biometrics*, 21(2):362–375.
- Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26.
- Efron, B. (1982). *The Jackknife, the Bootstrap, and Other Resampling Plans*, volume 38 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Philadelphia: SIAM.
- Efron, B. and Gong, G. (1983). A leisurely look at the bootstrap, the jackknife, and cross-validation. *The American Statistician*, 37(1):36–48.
- Egan, M., Krishnamoorthy, M., and Rajan, K. (1998). FCLUS: A visualization tool for fuzzy clustering. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on computer science education*, pages 227–231. New York: ACM Press.
- Eisen, M. and Brown, P. (1999). DNA arrays for analysis of gene expression. *Methods in Enzymology*, 303:179–205.
- Eisen, M., Spellman, P., Brown, P., and Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the United States of America*, 95(25):14863–14868.
- Eklund, T., Back, B., Vanharanta, H., and Visa, A. (2003). Using the self-organizing map as a visualization tool in financial benchmarking. *Information Visualization*, 2(3):171–181.
- El-Sonbaty, Y., Ismail, M., and Farouk, M. (2004). An efficient density based clustering algorithm for large databases. In *16th IEEE international conference on tools with artificial intelligence*, pages 673–677. Boca Raton FL: IEEE Computer Society.
- Engelman, L. and Hartigan, J. (1969). Percentage points of a test for clusters. *Journal of the American Statistical Association*, 64(328):1647–1648.

- Estabrook, G. (1966). A mathematical model in graph theory for biological classification. *Journal of Theoretical Biology*, 12:297–310.
- Estabrook, G. and Rogers, D. (1966). A general method of taxonomic description for a computed similarity measure. *BioScience*, 16:789–793.
- Ester, M., Kriegel, H., Sander, J., Wimmer, M., and Xu, X. (1998). Incremental clustering for mining in a data warehousing environment. In *VLDB '98: Proceedings of the 24th international conference on very large data bases*, pages 323–333. New York: Morgan Kaufmann.
- Ester, M., Kriegel, H., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second international conference on knowledge discovery and data mining*, pages 226–231. Portland, OR: AAAI Press.
- Ester, M., Kriegel, H., Sander, J., and Xu, X. (1997). Density-connected sets and their application for trend detection in spatial databases. In *Third International Conference on Knowledge Discovery and Data Mining*, pages 10–15. Menlo Park, CA: AAAI Press.
- Estivill-Castro, V. (2002). Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorations Newsletter*, 4(1):65–75.
- Everitt, B. (1993). *Cluster analysis*, 3rd edition. New York, Toronto: Halsted Press.
- Everitt, B. and Hand, D. (1981). *Finite Mixture Distributions*. London: Chapman & Hall.
- Everitt, B., Landau, S., and Leese, M. (2001). *Cluster Analysis*, 4th edition. New York: Oxford University Press.
- Everitt, B. and Nicholls, P. (1975). Visual techniques for representing multivariate data. *The Statistician*, 24(1):37–49.
- Faber, V. (1994). Clustering and the continuous k -means algorithm. *Los Alamos Science*, 22:138–144.
- Faloutsos, C., Jagadish, H., Mendelzon, A., and Milo, T. (1997). A signature technique for similarity-based queries. In *Proceedings on compression and complexity of sequences 1997*, pages 2–20. Salerno: IEEE Computer Society.
- Faloutsos, C. and Lin, K. (1995). FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on management of data*, pages 163–174. New York: ACM Press.
- Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on management of data*, pages 419–429. New York: ACM Press.
- Farris, J. (1969). On the cophenetic correlation coefficient. *Systematic Zoology*, 18(3):279–285.

- Fashing, M. and Tomasi, C. (2005). Mean shift is a bound optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):471–474.
- Feder, T. and Greene, D. (1988). Optimal algorithms for approximate clustering. In *STOC '88: Proceedings of the twentieth annual ACM symposium on theory of computing*, pages 434–444. New York: ACM Press.
- Feigenbaum, J., Kannan, S., Strauss, M., and Viswanathan, M. (1999). An approximate 11-difference algorithm for massive data streams. In *FOCS '99: Proceedings of the 40th annual symposium on foundations of computer science*, page 501. Washington, DC: IEEE Computer Society.
- Felsenstein, J. (1985). Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39(4):783–791.
- Filho, J., Treleaven, P., and Alippi, C. (1994). Genetic-algorithm programming environments. *IEEE Computer*, 27(6):28–43.
- Fisher, W. (1958). On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53(284):789–798.
- Fitzgibbon, L., Allison, L., and Dowe, D. (2000). Minimum message length grouping of ordered data. *Lecture Notes in Artificial Intelligence*, 1968:56–70.
- Florek, K., Lukaszewicz, J., Steinhaus, H., and Zubrzycki, S. (1951). Sur la liaison et la division des points d'un ensemble fini. *Colloquium Mathematicum*, 2:282–285.
- Fowlkes, C., Belongie, S., Chung, F., and Malik, J. (2004). Spectral grouping using the Nystrom method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225.
- Fraley, C. (1998). Algorithms for model-based Gaussian hierarchical clustering. *SIAM Journal on Scientific Computing*, 20(1):270–281.
- Fraley, C. and Raftery, A. (1998). How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, 41(8):578–588.
- Fraley, C. and Raftery, A. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, 97(458):611–631.
- Frank, R. and Green, P. (1968). Numerical taxonomy in marketing analysis: A review article. *Journal of Marketing Research (pre-1986)*, 5(000001):83–94.
- Friedman, H. and Rubin, J. (1967). On some invariant criteria for grouping data. *Journal of the American Statistical Association*, 62(320):1159–1178.
- Friedman, J. and Rafsky, L. (1981). Graphics for the multivariate two-sample problem. *Journal of the American Statistical Association*, 76(374):277–287.

- Frieze, A., Kannan, R., and Vempala, S. (1998). Fast Monte-Carlo algorithms for finding low-rank approximations. In *FOCS '98: Proceedings of the 39th annual symposium on foundations of computer science*, pages 370–378. Washington, DC: IEEE Computer Society.
- Frieze, A., Kannan, R., and Vempala, S. (2004). Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041.
- Fua, Y., Ward, M., and Rundensteiner, E. (1999). Hierarchical parallel coordinates for exploration of large datasets. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 43–50. Los Alamitos, CA: IEEE Computer Society Press.
- Fujikawa, Y. and Ho, T. (2002). Cluster-based algorithms for dealing with missing values. In Cheng, M.-S., Yu, P. S., and Liu, B., editors, *Advances in Knowledge Discovery and Data Mining, Proceedings of the 6th Pacific-Asia Conference, PAKDD 2002*, Taipei, Taiwan, volume 2336 of *Lecture Notes in Computer Science*, pages 549–554. New York: Springer.
- Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition*, 2nd edition. Computer Science and Scientific Computing. San Diego, CA: Academic Press.
- Fukunaga, K. and Hostetler, L. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40.
- Fukuyama, Y. and Sugeno, M. (1989). A new method of choosing the number of clusters for the fuzzy c -means method. In *Proceedings of the 5th Fuzzy Systems Symposium*, pages 247–250. Kobe, Japan (In Japanese.).
- Gaber, M., Zaslavsky, A., and Krishnaswamy, S. (2005). Mining data streams: A review. *ACM SIGMOD Record*, 34(2):18–26.
- Gaede, V. and Günther, O. (1998). Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231.
- Gan, G. (2003). *Subspace Clustering for High Dimensional Categorical Data*. Master's thesis, Department of Mathematics and Statistics, York University, Toronto, Canada.
- Gan, G. (2006). *Subspace Clustering Based on Fuzzy Models and Mean Shifts*. PhD thesis, Department of Mathematics and Statistics, York University, ON, Canada.
- Gan, G. and Wu, J. (2004). Subspace clustering for high dimensional categorical data. *ACM SIGKDD Explorations Newsletter*, 6(2):87–94.
- Gan, G., Wu, J., and Yang, Z. (2006). A fuzzy subspace algorithm for clustering high dimensional data. In Li, X., Zaiane, O., and Li, Z., editors, *Lecture Notes in Artificial Intelligence*, volume 4093, pages 271–278. New York: Springer.

- Gan, G., Yang, Z., and Wu, J. (2005). A genetic k -modes algorithm for clustering categorical data. In Li, X., Wang, S., and Dong, Z., editors, *Proceedings on Advanced Data Mining and Applications: First International Conference, ADMA, Wuhan, China*, volume 3584 of *Lecture Notes in Artificial Intelligence*, pages 195–202. New York: Springer-Verlag.
- Ganti, V., Gehrke, J., and Ramakrishnan, R. (1999). CACTUS: Clustering categorical data using summaries. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 73–83. New York: ACM Press.
- Ganti, V., Gehrke, J., and Ramakrishnan, R. (2000). DEMON: Mining and monitoring evolving data. In *ICDE '00: Proceedings of the 16th international conference on data engineering*, pages 439–448. Washington, DC: IEEE Computer Society.
- Ganti, V., Gehrke, J., and Ramakrishnan, R. (2001). DEMON: Mining and monitoring evolving data. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):50–63.
- Garai, G. and Chaudhuri, B. (2004). A novel genetic algorithm for automatic clustering. *Pattern Recognition Letters*, 25(2):173–187.
- García-Escudero, L. and Gordaliza, A. (1999). Robustness properties of k -means and trimmed k -means. *Journal of the American Statistical Association*, 94(447):956–969.
- García-Escudero, L., Gordaliza, A., and Matrán, C. (1999a). Asymptotics for trimmed k -means and associated tolerance zones. *Journal of Statistical Planning and Inference*, 77(2):247–262.
- García-Escudero, L., Gordaliza, A., and Matrán, C. (1999b). A central limit theorem for multivariate generalized trimmed k -means. *The Annals of Statistics*, 27(3):1061–1079.
- Gath, I. and Geva, A. (1989). Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):773–780.
- Gavrilov, M., Anguelov, D., Indyk, P., and Motwani, R. (2000). Mining the stock market: which measure is best? In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 487–496. New York: ACM Press.
- Ge, X. and Smyth, P. (2000). Deformable Markov model templates for time-series pattern matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 81–90. New York: ACM Press.
- Gehrke, J., Ganti, V., Ramakrishnan, R., and Loh, W. (1999). BOAT—optimistic decision tree construction. In *Proceedings of the 1999 ACM SIGMOD international conference on management of data*, pages 169–180. New York: ACM Press.
- Gehrke, J., Ramakrishnan, R., and Ganti, V. (1998). RainForest—A framework for fast decision tree construction of large datasets. In *Proceedings of the 24th international conference on very large data bases, VLDB*, pages 416–427. New York: Morgan Kaufmann.

- Georgescu, B., Shimshoni, I., and Meer, P. (2003). Mean shift based clustering in high dimensions: a texture classification example. In *Proceedings of the ninth IEEE international conference on computer vision*, pages 456–463. Nice, France: IEEE Computer Society.
- Geurts, P. (2001). Pattern extraction for time series classification. In *PKDD '01: Proceedings of the 5th European conference on principles of data mining and knowledge discovery*, volume 2168 of *Lecture Notes in Computer Science*, pages 115–127. London, UK: Springer-Verlag.
- Gibbons, F. and Roth, F. (2002). Judging the quality of gene expression-based clustering methods using gene annotation. *Genome Research*, 12:1574–1581.
- Gibbons, P. and Matias, Y. (1999). Synopsis data structures for massive data sets. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms*, pages 909–910. Philadelphia: SIAM.
- Gibson, D., Kleinberg, J., and Raghavan, P. (2000). Clustering categorical data: An approach based on dynamical systems. *The VLDB Journal*, 8(3–4):222–236.
- Gilbert, A., Guha, S., Indyk, P., Muthukrishnan, S., and Strauss, M. (2002a). Near-optimal sparse Fourier representations via sampling. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on theory of computing*, pages 152–161. New York: ACM Press.
- Gilbert, A., Kotidis, Y., Muthukrishnan, S., and Strauss, M. (2002b). How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB 2002, Proceedings of the 28th international conference on very large data bases*, pages 454–465. Hong Kong: Morgan Kaufmann.
- Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing*, 1(3):190–206.
- Glover, F. (1990). Tabu search—Part II. *ORSA Journal on Computing*, 2(1):4–32.
- Glover, F., Taillard, E., and de Werra, D. (1993). A user’s guide to tabu search. *Annals of Operations Research*, 41:3–28.
- Gluck, A. and Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–287. Irvine, CA: Lawrence Erlbaum Associates.
- Goil, S., Nagesh, H., and Choudhary, A. (1999). *MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets*. Technical Report CPDC-TR-9906-010, Center for Parallel and Distributed Computing, Department of Electrical & Computer Engineering, Northwestern University.
- Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

- Goldin, D. and Kanellakis, P. (1995). On similarity queries for time-series data: Constraint specification and implementation. In Montanari, U., and Rossi, F., editors, *The First International Conference on Principles and Practice of Constraint Programming—CP'95*, volume 976 of *Lecture Notes in Computer Science*, pages 137–153. New York: Springer.
- Golub, T., Slonim, D., Tamayo, P., Huard, C., Gassenbeek, M., Mesirov, J., Coller, H., and Loh, M. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 268(15):531–537.
- Gonzalez, T. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(2–3):293–306.
- Goodall, D. (1966). A new similarity index based on probability. *Biometrics*, 22(4):882–907.
- Goodman, L. (1974). Exploratory latent structure analysis using both identifiable and unidentifiable models. *Biometrika*, 61(2):215–231.
- Goodman, L. and Kruskal, W. (1954). Measures of association for cross classifications. *Journal of the American Statistical Association*, 49(268):732–764.
- Gordon, A. (1987). A review of hierarchical classification. *Journal of the Royal Statistical Society. Series A (General)*, 150(2):119–137.
- Gordon, A. (1996). Hierarchical classification. In Arabie, P., Hubert, L., and Soete, G., editors, *Clustering and Classification*, pages 65–121. River Edge, NJ: World Scientific.
- Gordon, A. (1998). Cluster validation. In Hayashi, C., Ohsumi, N., Yajima, K., Tanaka, Y., Bock, H., and Baba, Y., editors, *Data Science, Classification, and Related Methods*, pages 22–39. Tokyo: Springer.
- Gordon, A. (1999). *Classification*, 2nd edition. Boca Raton, FL: Chapman & Hall/CRC.
- Gotlieb, C. and Kumar, S. (1968). Semantic clustering of index terms. *Journal of the Association for Computing Machinery*, 15(4):493–513.
- Gowda, K. and Diday, E. (1992). Symbolic clustering using a new similarity measure. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):368–378.
- Gower, J. (1967). A comparison of some methods of cluster analysis. *Biometrics*, 23(4):623–637.
- Gower, J. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–874.
- Gower, J. and Legendre, P. (1986). Metric and Euclidean properties of dissimilarity coefficients. *Journal of Classification*, 3:5–48.
- Gower, J. and Ross, G. (1969). Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18(1):54–64.

- Grabusts, P. and Borisov, A. (2002). Using grid-clustering methods in data classification. In *Proceedings of the International Conference on Parallel Computing in Electrical Engineering, 2002. PARELEC '02*. pages 425–426. Warsaw, Poland: IEEE Computer Society.
- Green, P. and Rao, V. (1969). A note on proximity measures and cluster analysis. *Journal of Marketing Research*, 6(000003):359–364.
- Greene, W. (2003). Unsupervised hierarchical clustering via a genetic algorithm. In *The 2003 congress on evolutionary computation*, volume 2, pages 998–1005. Canberra, Australia: IEEE Press.
- Greenwald, M. and Khanna, S. (2001). Space-efficient online computation of quantile summaries. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on management of data*, pages 58–66. New York: ACM Press.
- Guha, S. and Koudas, N. (2002). Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *ICDE '02: Proceedings of the 18th international conference on data engineering*, pages 567–576. Washington, DC: IEEE Computer Society.
- Guha, S., Koudas, N., and Shim, K. (2001). Data-streams and histograms. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on theory of computing*, pages 471–475. New York: ACM Press.
- Guha, S., Meyerson, A., Mishra, N., Motwani, R., and Callaghan, L. (2003). Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528.
- Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L. (2000a). Clustering data streams. In *FOCS '00: Proceedings of the 41st annual symposium on foundations of computer science*, pages 359–366. Washington, DC: IEEE Computer Society.
- Guha, S., Rastogi, R., and Shim, K. (1998). CURE: An efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM SIGMOD international conference on management of data*, pages 73–84. New York: ACM Press.
- Guha, S., Rastogi, R., and Shim, K. (2000b). ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366.
- Gunopulos, D., and Das, G. (2000). Time series similarity measures (tutorial PM-2). In *Tutorial notes of the sixth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 243–307. Boston, MA: ACM Press.
- Gunopulos, D. and Das, G. (2001). Time series similarity measures and time series indexing. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on management of data*, page 624. New York: ACM Press.
- Gupta, S., Rao, K., and Bhatnagar, V. (1999). k -means clustering algorithm for categorical attributes. In *Proceedings of the first international conference on data warehousing and knowledge discovery*, pages 203–208. New York: Springer-Verlag.

- Haas, P., Naughton, J., Seshadri, S., and Stokes, L. (1995). Sampling-based estimation of the number of distinct values of an attribute. In *VLDB '95: Proceedings of the 21st international conference on very large data bases*, pages 311–322. San Francisco: Morgan Kaufmann.
- Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2001a). Clustering algorithms and validity measures. *Thirteenth international conference on scientific and statistical database management*, pages 3–22. Fairfax, VA: IEEE Computer Society.
- Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2001b). On clustering validation techniques. *Journal of Intelligent Information Systems*, 17:107–145.
- Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2002a). Cluster validity methods: Part I. *ACM SIGMOD Record*, 31(2).
- Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2002b). Clustering validity checking methods: Part II. *ACM SIGMOD Record*, 31(3).
- Halkidi, M. and Vazirgiannis, M. (2001). Clustering validity assessment: Finding the optimal partitioning of a data set. *ICDM 2001, Proceedings of the IEEE international conference on data mining*, pages 187–194. San Jose, CA: IEEE Computer Society.
- Halkidi, M., Vazirgiannis, M., and Batistakis, I. (2000). Quality scheme assessment in the clustering process. *Proceedings of PKDD*, pages 265–276. Freiburg, Germany: Springer.
- Hall, L., Özyurt, I., and Bezdek, J. (1999). Clustering with a genetically optimized approach. *IEEE Transactions on Evolutionary Computation*, 3(2):103–112.
- Hamerly, G. and Elkan, C. (2002). Alternatives to the k -means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on information and knowledge management*, pages 600–607. McLean, VA: ACM Press.
- Hampel, F. (1971). A general qualitative definition of robustness. *The Annals of Mathematical Statistics*, 42(6):1887–1896.
- Handl, J., Knowles, J., and Kell, D. (2005). Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21:3201–3212.
- Hansen, P. and Mladenović, N. (2001a). J -means: A new local search heuristic for minimum sum of squares clustering. *Pattern Recognition*, 34(2):405–413.
- Hansen, P. and Mladenović, N. (2001b). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.
- Har-Peled, S. and Mazumdar, S. (2004). On coresets for k -means and k -median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on theory of computing*, pages 291–300. New York: ACM Press.
- Har-Peled, S. and Varadarajan, K. (2002). Projective clustering in high dimensions using core-sets. In *Proceedings of the eighteenth annual symposium on computational geometry*, pages 312–318. New York: ACM Press.

- Harding, E. (1967). The number of partitions of a set of n points in k dimensions induced by hyperplanes. In *Proceedings of the Edinburgh Mathematical Society (Series II)*, volume 15, pages 285–289. Edinburgh: Scottish Academic Press.
- Hartigan, J. (1967). Representation of similarity matrices by trees. *Journal of the American Statistical Association*, 62(320):1140–1158.
- Hartigan, J. (1975). *Clustering Algorithms*. Toronto: John Wiley & Sons.
- Hartigan, J. and Mohanty, S. (1992). The RUNT test for multimodality. *Journal of Classification*, 9(1):63–70.
- Hartigan, J. and Wong, M. (1979). Algorithm AS136: A k -means clustering algorithm. *Applied Statistics*, 28(1):100–108.
- Hathaway, R. and Bezdek, J. (1984). Local convergence of the fuzzy c -means algorithm. *Pattern Recognition*, 19(6):477–480.
- Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*, 2nd edition. Upper Saddle River, NJ: Prentice-Hall.
- Henzinger, M., Raghavan, P., and Rajagopalan, S. (1998). *Computing on Data Streams*. Technical Report TR-1998-011, Digital Equipment Corp.
- Herniter, M. (2001). *Programming in MATLAB*. Pacific Grove, CA: Brooks/Cole.
- Herrero, J., Diaz-Uriarte, R., and Dopazo, J. (2003). Gene expression data preprocessing. *Bioinformatics*, 19(5):655–656.
- Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation*. Boston: Addison-Wesley Longman.
- Hetland, M. (2004). A survey of recent methods for efficient retrieval of similar time sequences. In Last, M., Kandel, A., and Bunke, H., editors, *Data Mining in Time Series Databases*, volume 57 of *Machine Perception and Artificial Intelligence*. River Edge, NJ: World Scientific.
- Hettich, S. and Bay, S. (1999). The UCI KDD archive. <http://kdd.ics.uci.edu>.
- Heyer, L., Kruglyak, S., and Yoosiph, S. (1999). Exploring expression data: Identification and analysis of coexpressed genes. *Genome Research*, 9(11):1106–1115.
- Hill, A., Brown, E., Whitley, M., Tucker-Kellogg, G., Hunter, C., and Slonim, D. (2001). Evaluation of normalization procedures for oligonucleotide array data based on spiked cRNA controls. *Genome Biology*, 2(12):research0055.1–research0055.13.
- Hinneburg, A. and Keim, D. (1998). An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 58–65. New York: AAAI Press.
- Hipp, J., Güntzer, U., and Nakhaeizadeh, G. (2000). Algorithms for association rule mining: A general survey and comparison. *ACM SIGKDD Explorations Newsletter*, 2(1):58–64.

- Hoare, C. (1961). Algorithm 64: Quicksort. *Communications of the ACM*, 4(7):321.
- Hodges, K. and Wotring, J. (2000). Client typology based on functioning across domains using the CAFAS: Implications for service planning. *Journal of Behavioral Health Services and Research*, 27(3):257–270.
- Hodson, F. (1970). Cluster analysis and archaeology: Some new developments and applications. *World Archaeology*, 1(3):299–320.
- Hofmann, T. and Buhmann, J. (1995). Multidimensional scaling and data clustering. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 459–466. Cambridge, MA: The MIT Press.
- Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.
- Holman, E. (1992). Statistical properties of large published classifications. *Journal of Classification*, 9(2):187–210.
- Hopcroft, J. and Tarjan, R. (1973). Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378.
- Höppner, F., Klawonn, F., Kruse, R., and Runkler, T. (1999). *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*. New York: Wiley.
- Howard, R. (1966). Classifying a population into homogeneous groups. In Lawrence, J., editor, *Operational Research and the Social Sciences*. London: Tavistock.
- Hsu, C. (2006). Generalizing self-organizing map for categorical data. *IEEE Transactions on Neural Networks*, 17(2):294–304.
- Hua, K., Lang, S., and Lee, W. (1994). A decomposition-based simulated annealing technique for data clustering. In *PODS '94. Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems*, Minneapolis, MN, volume 13, pages 117–128. New York: ACM Press.
- Huang, Z. (1997a). Clustering large data sets with mixed numeric and categorical values. In *Knowledge Discovery and Data Mining: Techniques and Applications*. River Edge, NJ: World Scientific.
- Huang, Z. (1997b). A fast clustering algorithm to cluster very large categorical data sets in data mining. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 146–151. Tucson, AZ: ACM Press.
- Huang, Z. (1998). Extensions to the k -means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304.
- Huang, Z. and Ng, M. (1999). A fuzzy k -modes algorithm for clustering categorical data. *IEEE Transactions on Fuzzy Systems*, 7(4):446–452.

- Hubálek, Z. (1982). Coefficients of association and similarity based on binary (presence-absence) data. An evaluation. *Biological Reviews of the Cambridge Philosophical Society*, 57:669–689.
- Hubert, L. (1974). Some applications of graph theory to clustering. *Psychometrika*, 39(3):283–309.
- Hussein, N. (2003). A Fast Greedy k -means Algorithm. Master’s thesis, Computer Science, University of Amsterdam, Amsterdam, Netherlands.
- Ibaraki, T. and Katoh, N. (1988). *Resource Allocation Problems: Algorithmic Approaches*. Cambridge MA: MIT Press.
- Ichino, M. (1988). General metrics for mixed features—The Cartesian space theory for pattern recognition. In *Proceedings of the 1988 IEEE international conference on systems, man, and cybernetics*, volume 1, pages 494–497. Beijing, China: IEEE.
- Ichino, M. and Yaguchi, H. (1994). Generalized Minkowski metrics for mixed feature-type data analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):698–708.
- Indyk, P. (1999). Sublinear time algorithms for metric space problems. In *STOC ’99: Proceedings of the thirty-first annual ACM symposium on theory of computing*, pages 428–434. New York: ACM Press.
- Indyk, P. (2000). Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS ’00: Proceedings of the 41st annual symposium on foundations of computer science*, page 189. Washington, DC: IEEE Computer Society.
- Indyk, P., Koudas, N., and Muthukrishnan, S. (2000). Identifying representative trends in massive time series data sets using sketches. In *VLDB ’00: Proceedings of the 26th international conference on very large data bases*, pages 363–372. San Francisco, CA: Morgan Kaufmann.
- Inselberg, A. and Dimsdale, B. (1990). Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *VIS ’90: Proceedings of the 1st conference on visualization*, pages 361–378. Los Alamitos, CA: IEEE Computer Society.
- Itoh, T., Yamaguchi, Y., Ikehata, Y., and Kajinaga, Y. (2004). Hierarchical data visualization using a fast rectangle-packing algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):302–313.
- Jain, A. and Dubes, R. (1988). *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall.
- Jain, A., Duin, R., and Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37.
- Jain, A., Murty, M., and Flynn, P. (1999). Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323.
- Jambu, M. (1978). *Classification automatique pour l’analyse de données*. Paris: Dunod.

- Jamshidian, M. and Jennrich, R. (1993). Conjugate gradient acceleration of the EM algorithm. *Journal of the American Statistical Association*, 88(421):221–228.
- Jamshidian, M. and Jennrich, R. (1997). Acceleration of the EM algorithm by using quasi-Newton methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, 59(3):569–587.
- Jardine, C., Jardine, N., and Sibson, R. (1967). The structure and construction of taxonomic hierarchies. *Mathematical Biosciences*, 1(2):173–179.
- Jardine, N. (1971). A new approach to pattern recognition. *Nature*, 234:526–528.
- Jardine, N. and Sibson, R. (1968). The construction of hierachic and non-hierachic classifications. *The Computer Journal*, 11(2):177–184.
- Jeffreys, H. (1935). Some tests of significance, treated by the theory of probability. *Proceedings of the Cambridge Philosophy Society*, 31:203–222.
- Jeffreys, H. (1961). *Theory of Probability*, 3rd edition. Oxford UK: Oxford University Press.
- Jiang, D., Pei, J., and Zhang, A. (2003). DHC: A density-based hierarchical clustering method for time series gene expression data. In *Proceedings of the third IEEE symposium on bioinformatics and bioengineering*, pages 393–400. Washington DC: IEEE Computer Society.
- Jiang, D., Tang, C., and Zhang, A. (2004). Cluster analysis for gene expression data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1370–1386.
- Jiang, T. and Ma, S. (1996). Cluster analysis using genetic algorithms. In *Proceedings of the third international conference on signal processing*, volume 2, pages 1277–1279, Beijing: IEEE Computer Society.
- Jin, X., Lu, Y., and Shi, C. (2002). Similarity measure based on partial information of time series. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 544–549. New York: ACM Press.
- Johansson, J., Ljung, P., Jern, M., and Cooper, M. (2006). Revealing structure in visualizations of dense 2D and 3D parallel coordinates. *Information Visualization*, 5(2):125–136.
- Johnson, R. and Wichern, D. (1998). *Applied Multivariate Statistical Analysis*. Englewood Cliffs, NJ: Prentice-Hall.
- Johnson, S. (1967). Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254.
- Jolliffe, I. (2002). *Principal Component Analysis*, 2nd edition. Springer Series in Statistics. New York: Springer-Verlag.
- Jones, D. and Beltramo, M. (1991). Solving partitioning problems with genetic algorithms. In *Proceedings of the 4th international conference on genetic algorithms*, pages 442–449. San Mateo, CA: Morgan Kaufman.

- Kahveci, T., Singh, A., and Gurel, A. (2002). An efficient index structure for shift and scale invariant search of multi-attribute time sequences. In *ICDE '02: Proceedings. 18th International Conference on Data Engineering*, page 266. San Jose, CA: IEEE Computer Society.
- Kalpakis, K., Gada, D., and Puttagunta, V. (2001). Distance measures for effective clustering of ARIMA time-series. In *ICDM 2001: Proceedings of the IEEE international conference on data mining*, pages 273–280. San Jose, CA: IEEE Press.
- Kandogan, E. (2001). Visualizing multi-dimensional clusters, trends, and outliers using star coordinates. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*, pages 107–116. New York: ACM Press.
- Kannan, R., Vempala, S., and Vetta, A. (2004). On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515.
- Kantabutra, S. and Couch, A. (2000). Parallel k -means clustering algorithm on NOWs. *NECTEC Technical Journal*, 1(6):243–248. <http://www.nectec.or.th>.
- Kanth, K., Agrawal, D., and Singh, A. (1998). Dimensionality reduction for similarity searching in dynamic databases. In *Proceedings of the 1998 ACM SIGMOD international conference on management of data*, pages 166–176. New York: ACM Press.
- Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., and Wu, A. (2002). An efficient k -means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892.
- Karypis, G., Han, E., and Kumar, V. (1999). Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75.
- Kass, R. and Raftery, A. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795.
- Kaufman, L. and Rousseeuw, P. (1990). *Finding Groups in Data—An Introduction to Cluster Analysis*. Wiley Series in Probability and Mathematical Statistics. New York: John Wiley & Sons, Inc.
- Ke, Q. and Kanade, T. (2004). Robust subspace clustering by combined use of KNND metric and SVD algorithm. In *Proceedings of the 2004 IEEE computer society conference on computer vision and pattern recognition*, volume 2, pages 592–599. Washington, DC: IEEE Computer Society.
- Keim, D. and Hinneburg, A. (1999). Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proceedings of the 25th international conference on very large data bases (VLDB '99)*, pages 506–517. San Francisco: Morgan Kaufmann.
- Keim, D. and Kriegel, H. (1996). Visualization techniques for mining large databases: A comparison. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):923–938.

- Kendall, S. and Ord, J. (1990). *Time Series*, 3rd edition. Seven Oaks, U.K.: Edward Arnold.
- Keogh, E. (2001). A tutorial on indexing and mining time series data. In *Proceedings of the 2001 IEEE international conference on data mining*, San Jose, CA. http://www.cse.cuhk.edu.hk/~mzhou/ClassifiedReferences/Survey/tutorial_on_time_series.ppt.
- Keogh, E., Chu, S., and Pazzani, M. (2001). Ensemble-index: A new approach to indexing large databases. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*, pages 117–125. New York: ACM Press.
- Keogh, E. and Kasetty, S. (2003). On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371.
- Keogh, E., Lin, J., and Truppel, W. (2003). Clustering of time series subsequences is meaningless: Implications for previous and future research. In *ICDM '03: Proceedings of the third IEEE international conference on data mining*, pages 115–122. Washington, DC: IEEE Computer Society.
- Keogh, E. and Pazzani, M. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Fourth international conference on knowledge discovery and data mining (KDD'98)*, pages 239–241. New York: ACM Press.
- Keogh, E. and Pazzani, M. (2000). Scaling up dynamic time warping for datamining applications. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 285–289. New York: ACM Press.
- Keogh, E. and Ratanamahatana, C. (2005). Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386.
- Keogh, E. and Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. In *KDD '97: Proceedings of the 3rd international conference of knowledge discovery and data mining*, pages 24–30. Newport Beach, CA: AAAI Press.
- Khan, S. and Ahmad, A. (2004). Cluster center initialization algorithm for k -means clustering. *Pattern Recognition Letters*, 25(11):1293–1302.
- Kim, E., Lam, J., and Han, J. (2000a). AIM: Approximate intelligent matching for time series data. In *DaWaK 2000: Proceedings of the second international conference on data warehousing and knowledge discovery*, pages 347–357. London, UK: Springer-Verlag.
- Kim, H., Golub, G., and Park, H. (2005). Missing value estimation for DNA microarray gene expression data: Local least squares imputation. *Bioinformatics*, 21(2):187–198.
- Kim, S., Kwon, S., and Cook, D. (2000b). Interactive visualization of hierarchical clusters using MDS and MST. *Metrika*, 51(1):39–51.

- Klein, R. and Dubes, R. (1989). Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, 22(2):213–220.
- Klock, H. and Buhmann, J. (2000). Data visualization by multidimensional scaling: A deterministic annealing approach. *Pattern Recognition*, 33(4):651–669.
- Kohonen, T. (1989). *Self-Organization and Associative Memory*, 3rd edition. New York: Springer-Verlag.
- Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.
- Konig, A. (2000). Interactive visualization and analysis of hierarchical neural projections for data mining. *IEEE Transactions on Neural Networks*, 11(3):615–624.
- Koren, Y. and Harel, D. (2003). A two-way visualization method for clustered data. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 589–594. New York: ACM Press.
- Krishna, K. and Narasimha, M. (1999). Genetic k -means algorithm. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 29(3):433–439.
- Krishnapuram, R. and Freg, C. (1991). Fuzzy algorithms to find linear and planar clusters and their applications. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition*, pages 426–431. Maui, HI: IEEE.
- Kroeger, P., Kriegel, H., and Kailing, K. (2004). Density-connected subspace clustering for high-dimensional data. In *Proceedings of the SIAM International Conference on Data Mining*, pages 246–257. Lake Buena Vista, FL: SIAM.
- Kruskal, J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50.
- Kruskal, J. (1964). Nonmetric multidimensional scaling: Numerical method. *Psychometrika*, 29(2):115–129.
- Kruskal, J. (1971). Comments on “A nonlinear mapping for data structure analysis.” *IEEE Transactions on Computers*, C-20(12):1614.
- Kruskal, J. and Landwehr, J. (1983). Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168.
- Krzanowski, W. and Lai, Y. (1988). A criterion for determining the number of groups in a data set using sum-of-squares clustering. *Biometrics*, 44(1):23–34.
- Kuiper, F. and Fisher, L. (1975). Shorter communications 391: A Monte Carlo comparison of six clustering procedures. *Biometrics*, 31(3):777–783.
- Kuncicky, D. (2004). *MATLAB Programming*. Upper Saddle River, NJ: Pearson Education.
- Lance, G. and Williams, W. (1966). A generalised sorting strategy for computer classifications. *Nature*, 212:218.

- Lance, G. and Williams, W. (1967a). A general theory of classificatory sorting strategies I. Hierarchical systems. *The Computer Journal*, 9(4):373–380.
- Lance, G. and Williams, W. (1967b). A general theory of classificatory sorting strategies II. Clustering systems. *The Computer Journal*, 10(3):271–277.
- Lance, G. and Williams, W. (1967c). Mixed-data classificatory programs I. Agglomerative systems. *Australian Computer Journal*, 1(1):15–20.
- Lanyon, S. (1985). Detecting internal inconsistencies in distance data. *Systematic Zoology*, 34(4):397–403.
- Lee, R., Slagle, J., and Mong, C. (1976). Application of clustering to estimate missing data and improve data integrity. In *Proceedings of the 2nd international conference on software engineering*, pages 539–544. San Francisco: IEEE Computer Society Press.
- Lee, S., Chun, S., Kim, D., Lee, J., and Chung, C. (2000). Similarity search for multidimensional data sequences. In *Proceedings of the 16th international conference on data engineering*, pages 599–608. San Diego, CA: IEEE Press.
- Legendre, L. and Legendre, P. (1983). *Numerical Ecology*. New York: Elsevier Scientific.
- Legendre, P. and Rogers, D. (1972). Characters and clustering in taxonomy: A synthesis of two taximetric procedures. *Taxon*, 21:567–606.
- Li, C., Yu, P., and Castelli, V. (1998). MALM: A framework for mining sequence database at multiple abstraction levels. In *CIKM '98: Proceedings of the seventh international conference on information and knowledge management*, pages 267–272. New York: ACM Press.
- Likas, A. and Verbeek, N. V. J. (2003). The global k -means clustering algorithm. *Pattern Recognition*, 36(2):45–461.
- Lin, J., Keogh, E., and Truppel, W. (2003). Clustering of streaming time series is meaningless. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, pages 56–65. New York: ACM Press.
- Liu, B., Xia, Y., and Yu, P. (2000). Clustering through decision tree construction. In *Proceedings of the ninth international conference on information and knowledge management*, pages 20–29. McLean, VA: ACM Press.
- Liu, Y., Chen, K., Liao, X., and Zhang, W. (2004). A genetic clustering method for intrusion detection. *Pattern Recognition*, 37(5):927–942.
- Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.
- Lockhart, D., Dong, H., Byrne, M., Follettie, M., Gallo, M., Chee, M., Mittmann, M., Wang, C., Kobayashi, M., Norton, H., and Brown, E. (1996). Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, 14:1675–1680.

- Lorr, M. (1983). *Cluster Analysis for Social Scientists*. The Jossey-Bass Social and Behavioral Science Series. San Francisco, Washington, London: Jossey-Bass.
- Lu, Y. and Huang, Y. (2005). Mining data streams using clustering. In *Proceedings of the 2005 international conference on machine learning and cybernetics*, volume 4, pages 2079–2083. Guangzhou, China: IEEE Computer Society.
- Lu, Y., Lu, S., Fotouhi, F., Deng, Y., and Brown, S. (2004a). Incremental genetic k -means algorithm and its application in gene expression data analysis. *BMC Bioinformatics*, 5(172):1–27.
- Lu, Y., Lu, S., Fotouhi, F., Deng, Y., and Brown, S. (2004b). FGKA: A fast genetic k -means clustering algorithm. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 622–623. New York: ACM Press.
- Ma, E. and Chow, T. (2004). A new shifting grid clustering algorithm. *Pattern Recognition*, 37(3):503–514.
- Macnaughton-Smith, P., Williams, W., Dale, M., and Mockett, L. (1964). Dissimilarity analysis: A new technique of hierarchical sub-division. *Nature*, 202:1034–1035.
- Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Berkeley, CA: University of California Press.
- Maharaj, E. (2000). Clusters of time series. *Journal of Classification*, 17(2):297–314.
- Malerba, D., Esposito, F., Gioviale, V., and Tamma, V. (2001). Comparing dissimilarity measures for symbolic data analysis. In *Proceedings of the joint conferences on “New techniques and technologies for statistics” and “exchange of technology and know-how”(ETK-NTTS’01)*, pages 473–481. Crete, Greece: Eurostat.
- Manku, G., Rajagopalan, S., and Lindsay, B. (1998). Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD ’98: Proceedings of the 1998 ACM SIGMOD international conference on management of data*, pages 426–435. New York: ACM Press.
- Manku, G., Rajagopalan, S., and Lindsay, B. (1999). Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD ’99: Proceedings of the 1999 ACM SIGMOD international conference on management of data*, pages 251–262. New York: ACM Press.
- Mántaras, R. (1991). A distance-based attribute selection measure for decision tree induction. In *Machine Learning*, volume 6, pages 81–92. Boston: Kluwer Academic.
- Mao, J. and Jain, A. (1996). A self-organizing network for hyperellipsoidal clustering (HEC). *IEEE Transactions on Neural Networks*, 7(1):16–29.
- Marriott, F. (1971). Practical problems in a method of cluster analysis. *Biometrics*, 27(3):501–514.

- Marshall, A. and Hodgson, J. (1998). DNA chips: An array of possibilities. *Nature Biotechnology*, 16:27–31.
- Martinez, W. and Martinez, A. (2005). *Exploratory data analysis with MATLAB*. Computer Science and Data Analysis. Boca Raton, FL: Chapman & Hall/CRC.
- Matoušek, J. (2000). On approximate geometric k -clustering. *Discrete and Computational Geometry*, 24(1):61–84.
- Maulik, U. and Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique. *Pattern Recognition*, 33(9):1455–1465.
- Maulik, U. and Bandyopadhyay, S. (2002). Performance evaluation of some clustering algorithms and validity indices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12):1650–1654.
- McErlean, F., Bell, D., and McClean, S. (1990). The use of simulated annealing for clustering data in databases. *Information Systems*, 15(2):233–245.
- McLachlan, G. and Basford, K. (1988). *Mixture Models: Inference and Applications to Clustering*, volume 84 of *STATISTICS: Textbooks and Monographs*. New York: Marcel Dekker, Inc.
- McLachlan, G., Bean, R., and Peel, D. (2002). A mixture model-based approach to the clustering of microarray expression data. *Bioinformatics*, 18(3):413–422.
- McLachlan, G. and Krishnan, T. (1997). *The EM Algorithm and Extensions*. New York: Wiley.
- McLachlan, G. and Peel, D. (2000). *Finite Mixture Models*. New York: Wiley.
- McMorris, F., Meronk, D., and Neumann, D. (1983). A view of some consensus methods for trees. In Felsenstein, J., editor, *Numerical Taxonomy*, pages 122–126. Berlin: Springer-Verlag.
- McQuitty, L. (1957). Elementary linkage analysis for isolating orthogonal and oblique types and typal relevancies. *Educational and Psychological Measurement*, 17:207–222.
- McQuitty, L. (1960). Hierarchical linkage analysis for the isolation of types. *Educational and Psychological Measurement*, 20:55–67.
- McQuitty, L. (1966). Similarity analysis by reciprocal pairs for discrete and continuous data. *Educational and Psychological Measurement*, 26:825–831.
- McQuitty, L. (1967). Expansion of similarity analysis by reciprocal pairs for discrete and continuous data. *Educational and Psychological Measurement*, 27:253–255.
- Meng, X. and van Dyk, D. (1997). The EM algorithm—An old folk-song sung to a fast new tune. *Journal of the Royal Statistical Society. Series B (Methodological)*, 59(3):511–567.
- Mettu, R. and Plaxton, C. (2004). Optimal time bounds for approximate clustering. *Machine Learning*, 56(1–3):35–60.

- Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, New York: Springer-Verlag.
- Michaud, P. (1997). Clustering techniques. *Future Generation Computer Systems*, 13(2–3):135–147.
- Miller, J. and Wegman, E. (1991). Construction of line densities for parallel coordinate plots. In Bujag A., and Tukey, P., editors, *Computing and Graphics in Statistics*, pages 107–123. New York: Springer-Verlag.
- Milligan, G. (1979). Ultrametric hierarchical clustering algorithms. *Psychometrika*, 44:343–346.
- Milligan, G. (1989). A study of the beta-flexible clustering method. *Multivariate Behavioral Research*, 24:163–176.
- Milligan, G. and Cooper, M. (1988). A study of standardization of variables in cluster analysis. *Journal of Classification*, 5:181–204.
- Mirkin, B. (2005). *Clustering for Data Mining: A Data Recovery Approach*. Computer Science and Data Analysis Series. Boca Raton, FL: Chapman & Hall/CRC.
- Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100.
- Moore, A. (1990). *Efficient Memory-based Learning for Robot Control*. PhD thesis, Computer Laboratory, University of Cambridge, Cambridge, UK.
- Moore, A. (1999). Very fast EM-based mixture model clustering using multiresolution *kd*-trees. In *Proceedings of the 1998 conference on advances in neural information processing systems II*, pages 543–549. Cambridge, MA: MIT Press.
- Morgan, B. (1981). Three applications of methods of cluster-analysis. *The Statistician*, 30(3):205–223.
- Morrison, D. (1967). Measurement problems in cluster analysis. *Management Science (Series B, Managerial)*, 13(12):B775–B780.
- Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*. New York: Cambridge University Press.
- Munro, J. and Paterson, M. (1980). Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323.
- Murtagh, F. (1983). A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359.
- Murtagh, F. (1984a). Complexities of hierachic clustering algorithms: State of the art. *Computational Statistics Quarterly*, 1(2):101–113.
- Murtagh, F. (1984b). Counting dendograms: A survey. *Discrete Applied Mathematics*, 7(2):191–199.

- Murtagh, F. and Raftery, A. (1984). Fitting straight lines to points patterns. *Pattern Recognition*, 17:479–483.
- Murthy, C. and Chowdhury, N. (1996). In search of optimal clusters using genetic algorithms. *Pattern Recognition Letters*, 17(8):825–832.
- Nabney, I. (2002). *NETLAB: Algorithms for Pattern Recognition*. Advances in Pattern Recognition. London, New York: Springer.
- Nagesh, H., Goil, S., and Choudhary, A. (2000). A scalable parallel subspace clustering algorithm for massive data sets. In *2000 International Conference on Parallel Processing (ICPP '00)*, pages 477–486. Washington, Brussels, Tokyo: IEEE.
- Nagesh, H., Goil, S., and Choudhary, A. (2001). Adaptive grids for clustering massive data sets. In *first SIAM international conference on data mining*, Chicago: SIAM.
- Narahashi, M. and Suzuki, E. (2002). Subspace clustering based on compressibility. *Lecture Notes in Computer Science*, 2534:435–440.
- Ng, A., Jordan, M., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14, Cambridge, MA: MIT Press.
- Ng, M. and Wong, J. (2002). Clustering categorical data sets using tabu search techniques. *Pattern Recognition*, 35(12):2783–2790.
- Ng, R. and Han, J. (1994). Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th international conference on very large data bases*, Santiago, Chile, pages 144–155. Los Altos, CA: Morgan Kaufmann.
- Nievergelt, J., Hinterberger, H., and Sevcik, K. (1984). The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71.
- Oates, T., Firoiu, L., and Cohen, P. (2001). Using dynamic time warping to bootstrap HMM-based clustering of time series. In *Sequence Learning—Paradigms, Algorithms, and Applications*, volume 1828 of *Lecture Notes in Computer Science*, pages 35–52. London: Springer-Verlag.
- O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S., and Motwani, R. (2002). Streaming-data algorithms for high-quality clustering. In *ICDE '02: Proceedings of the 18th international conference on data engineering*, pages 685–694. Washington, DC: IEEE Computer Society.
- Ordonez, C. (2003). Clustering binary data streams with k -means. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, pages 12–19. New York: ACM Press.
- Orlóci, L. (1967). An agglomerative method for classification of plant communities. *Journal of Ecology*, 55:193–205.

- Overall, J. and Klett, C. (1972). *Applied Multivariate Analysis*. McGraw-Hill Series in Psychology. New York: McGraw-Hill.
- Pal, N. and Biswas, J. (1997). Cluster validation using graph theoretic concepts. *Pattern Recognition*, 30(6):847–857.
- Pandya, A. and Macy, R. (1996). *Pattern Recognition with Neural Networks in C++*. Boca Raton, FL: CRC Press.
- Park, N. and Lee, W. (2004). Statistical grid-based clustering over data streams. *SIGMOD Record*, 33(1):32–37.
- Park, S. and Chu, D. L. W. (1999). Fast retrieval of similar subsequences in long sequence databases. In *Proceedings of the 1999 workshop on knowledge and data engineering exchange*, pages 60–67. Chicago: IEEE Computer Society.
- Park, S., Chu, W., Yoon, J., and Hsu, C. (2000). Efficient searches for similar subsequences of different lengths in sequence databases. In *ICDE '00: Proceedings of the 16th international conference on data engineering*, pages 23–32. San Diego, CA: IEEE Computer Society.
- Park, S., Kim, S., and Chu, W. (2001). Segment-based approach for subsequence searches in sequence databases. In *SAC '01: Proceedings of the 2001 ACM symposium on applied computing*, pages 248–252. New York: ACM Press.
- Parsons, L., Haque, E., and Liu, H. (2004a). Evaluating subspace clustering algorithms. In *Workshop on clustering high dimensional data and its applications, SIAM international conference on data mining (SDM 2004)*, pages 48–56. Philadelphia: SIAM.
- Parsons, L., Haque, E., and Liu, H. (2004b). Subspace clustering for high dimensional data: A review. *SIGKDD, Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, 6(1):90–105.
- Pei, J. and Yang, X. (2000). Study of clustering validity based on fuzzy similarity. In *Proceedings of the 3rd world congress on intelligent control and automation*, volume 4, pages 2444–2447. Hefei, China: IEEE.
- Pelleg, D. and Moore, A. (1999). Accelerating exact k -means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 277–281. San Diego, CA: ACM Press.
- Pelleg, D. and Moore, A. (2000). x -means: Extending k -means with efficient estimation of the number of clusters. In *Proceedings of the seventeenth international conference on machine learning*, pages 727–734. San Francisco: Morgan Kaufmann.
- Peña, J., Lozano, J., and Larrañaga, P. (1999). An empirical comparison of four initialization methods for the k -means algorithm. *Pattern Recognition Letters*, 20(10):1027–1040.

- Perng, C., Wang, H., Zhang, S., and Parker, D. (2000). Landmarks: A new model for similarity-based pattern querying in time series databases. In *ICDE '00: Proceedings of the 16th international conference on data Engineering*, pages 33–42. Washington, DC: Computer Society.
- Phillips, S. (2002). Acceleration of k -means and related clustering algorithms. In Mount, D. and Stein, C., editors, *ALENEX: International workshop on algorithm engineering and experimentation, LNCS*, volume 2409, pages 166–177. San Francisco: Springer-Verlag.
- Podani, J. (1989). New combinatorial clustering methods. *Vegetatio*, 81:61–77.
- Pollard, D. (1981). Strong consistency of k -means clustering. *The Annals of Statistics*, 9(1):135–140.
- Pollard, D. (1982). A central limit theorem for k -means clustering. *The Annals of Probability*, 10(4):919–926.
- Pölzlauer, G., Dittenbach, M., and Rauber, A. (2006). Advanced visualization of self-organizing maps with vector fields. *Neural Networks*, 19(6–7):911–922.
- Popivanov, I. and Miller, R. (2002). Similarity search over time-series data using wavelets. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 212. Washington, DC: IEEE Computer Society.
- Posse, C. (2001). Hierarchical model-based clustering for large datasets. *Journal of Computational & Graphical Statistics*, 10(3):464–486.
- Pratt, K. and Fink, E. (2002). Search for patterns in compressed time series. *International Journal of Image and Graphics*, 2(1):89–106.
- Preparata, F. and Shamos, M. (1985). *Computational Geometry: An Introduction*. New York: Springer-Verlag.
- Prim, R. (1957). Shortest connection matrix network and some generalizations. *Bell Systems Technical Journal*, 36:1389–1401.
- Procopiuc, C., Jones, M., Agarwal, P., and Murali, T. (2002). A Monte Carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD international conference on management of data*, pages 418–427. New York: ACM Press.
- Qu, Y., Wang, C., and Wang, X. (1998). Supporting fast search in time series for movement patterns in multiple scales. In *CIKM '98: Proceedings of the seventh international conference on information and knowledge management*, pages 251–258. New York: ACM Press.
- Quinlan, J. (1993). *C4.5: Program for Machine Learning*. San Mateo, CA: Morgan Kaufmann.
- Rafiei, D. and Mendelzon, A. (1997). Similarity-based queries for time series data. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on management of data*, pages 13–25. New York: ACM Press.

- Rafiei, D. and Mendelzon, A. (1998). Efficient retrieval of similar time sequences using DFT. In *FODO '98: Proceedings of the 5th international conference on foundations of data organizations and algorithms*, pages 249–257. Kobe, Japan.
- Ramoni, M., Sebastiani, P., and Cohen, P. (2002). Bayesian clustering by dynamics. *Machine Learning*, 47(1):91–121.
- Ramsay, G. (1998). DNA chips: State-of-the-art. *Nature Biotechnology*, 16:40–44.
- Rand, W. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850.
- Rao, V. and Rao, H. (1995). *C++ Neural Networks and Fuzzy Logic*, 2nd edition. New York: MIS Press.
- Ray, S. and Turi, R. (1999). Determination of number of clusters in k -means clustering and application in colour image segmentation. In *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques (ICAPRDT '99)*, pages 137–143. Calcutta, India: Narosa Publishing House.
- Reymond, P., Weber, H., Damond, M., and Farmer, E. (2000). Differential gene expression in response to mechanical wounding and insect feeding in arabidopsis. *The Plant Cell*, 12(5):707–720.
- Rhee, H. and Oh, K. (1996). A performance measure for the fuzzy cluster validity. *IEEE soft computing in intelligent systems and information processing, Proceedings of the 1996 Asian fuzzy systems symposium*, pages 364–369. Kenting, Taiwan: IEEE Computer Society.
- Rogers, D. and Tanimoto, T. (1960). A computer program for classifying plants. *Science*, 132:1115–1118.
- Rohlf, F. (1970). Adaptive hierarchical clustering schemes. *Systematic Zoology*, 19(1):58–82.
- Rohlf, F. (1973). Algorithm 76: Hierarchical clustering using the minimum spanning tree. *The Computer Journal*, 16(1):93–95.
- Rohlf, F. (1974). Algorithm 81: Dendrogram plot. *The Computer Journal*, 17(1):89–91.
- Rohlf, F. (1982). Single link clustering algorithms. In Krishnaiah, P. and Kanal, L., editors, *Handbook of Statistics*, volume 2, pages 267–284. Amsterdam: North-Holland.
- Rose, K., Gurewitz, E., and Fox, G. (1990). Statistical mechanics and phase transitions in clustering. *Physical Review Letters*, 65(8):945–948.
- Ross, G. (1969). Algorithm AS 15: Single linkage cluster analysis. *Applied Statistics*, 18(1):106–110.
- Rousseeuw, P. (1986). A visual display for hierarchical classification. In Diday, E., Escoufier, Y., Lebart, L., Pagès, J., Schektmann, Y., and Tomassone, R., editors, *Data Analysis and Informatics 4*, pages 743–748. Amsterdam: North-Holland.

- Rousseeuw, P. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.
- Rummel, R. (1970). *Applied Factor Analysis*. Evanston, IL: Northwestern University Press.
- Ruspini, E. (1969). A new approach to clustering. *Information and Control*, 15:22–32.
- Salton, G. and McGill, M. (1983). *Introduction to Modern Information Retrieval*. New York, Tokyo: McGraw-Hill.
- Sammon Jr., J. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409.
- Sander, J., Ester, M., Kriegel, H., and Xu, X. (1998). Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194.
- Sarafis, I., Trinder, P., and Zalzala, A. (2003). Towards effective subspace clustering with an evolutionary algorithm. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03*, volume 2, pages 797–806. Canberra, Australia: IEEE.
- SAS Institute Inc. (1983). SAS Technical Report A-108, Cubic Clustering Criterion. SAS Institute Inc.
- Saunders, J. (1980). Cluster analysis for market segmentation. *European Journal of Marketing*, 14(7):422–435.
- Scheibler, D. and Schneider, W. (1985). Monte Carlo test of the accuracy of cluster analysis algorithms—A comparison of hierarchical and nonhierarchical methods. *Multivariate Behavioral Research*, 20:283–304.
- Schena, M., Shalon, D., Davis, R., and Brown, P. (1995). Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270(5235):467–470.
- Schikuta, E. (1996). Grid-clustering: An efficient hierarchical clustering method for very large data sets. In *Proceedings of the 13th International Conference on Pattern Recognition*, volume 2, pages 101–105. Vienna, Austria: IEEE.
- Schikuta, E. and Erhart, M. (1997). The BANG-clustering system: Grid-based data analysis. In Liu, X., Cohen, P., and Berthold, M., editors, *Lecture Notes in Computer Science*, volume 1280, pages 513–524. Berlin, Heidelberg: Springer-Verlag.
- Schuchhardt, J., Beule, D., Malik, A., Wolski, E., Eickhoff, H., Lehrach, H., and Herzel, H. (2000). Normalization strategies for cDNA microarrays. *Nucleic Acids Research*, 28(10):e47–e47.
- Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

- Scott, A. and Symons, M. (1971a). 297. Note: On the Edwards and Cavalli-Sforza method of cluster analysis. *Biometrics*, 27(1):217–219.
- Scott, A. and Symons, M. (1971b). Clustering methods based on likelihood ratio criteria. *Biometrics*, 27(2):387–397.
- Sedgewick, R. (1978). Implementing quicksort programs. *Communications of the ACM*, 21(10):847–857.
- Selim, S. and Al-Sultan, K. (1991). A simulated annealing algorithm for the clustering problem. *Pattern Recognition*, 24:1003–1008.
- Selim, S. and Ismail, M. (1984). k -means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):81–87.
- Selim, S. and Ismail, M. (1986). Fuzzy c -means: optimality of solutions and effective termination of the algorithm. *Pattern Recognition*, 19(6):651–663.
- Serinko, R. and Babu, G. (1992). Weak limit theorems for univariate k -means clustering under a nonregular condition. *Journal of Multivariate Analysis*, 41:273–296.
- Shadbolt, J. and Taylor, J., editors (2002). *Neural Networks and the Financial Markets*. London: Springer.
- Shafer, J., Agrawal, R., and Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining. In *VLDB '96, Proceedings of 22nd international conference on very large data bases*, pages 544–555. Mumbai (Bombay), India: Morgan Kaufmann.
- Sharan, R. and Shamir, R. (2000). CLICK: A clustering algorithm with applications to gene expression analysis. In *Proceedings of the eighth international conference on intelligent systems for molecular biology (ISMB)*, pages 307–316. Menlo Park, CA: AAAI Press.
- Sharma, S. (1996). *Applied Multivariate Techniques*. New York: John Wiley & Sons.
- Sheikholeslami, G., Chatterjee, S., and Zhang, A. (2000). WaveCluster: a wavelet-based clustering approach for spatial data in very large databases. *The VLDB Journal*, 8(3–4):289–304.
- Shepard, R. (1962). The analysis of proximities: Multidimensional scaling with an unknown distance function I. *Psychometrica*, 27:125–140.
- Shneiderman, B. (1992). Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99.
- Sibson, R. (1973). SLINK: An optimally efficient algorithm for the single link cluster method. *The Computer Journal*, 16(1):30–34.
- Sneath, P. (1957). The applications of computers to taxonomy. *Journal of General Microbiology*, 17:201–226.

- Sneath, P. (1967). Some statistical problems in numerical taxonomy. *The Statistician*, 17(1):1–12.
- Sneath, P. (1969). Evaluation of clustering methods (with discussion). In Cole, A., editor, *Numerical Taxonomy*, pages 257–271. London: Academic Press.
- Sokal, R. and Rohlf, F. (1962). The comparison of dendograms by objective methods. *Taxon*, 11:33–40.
- Sokal, R. and Sneath, P. (1963). *Principles of Numerical Taxonomy*. San Francisco: W.H. Freeman.
- Sokal, R. and Sneath, P. (1973). *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. San Francisco: W.H. Freeman.
- Somorjai, R., Dolenko, B., Demko, A., Mandelzweig, M., Nikulin, A., Baumgartner, R., and Pizzi, N. (2004). Mapping high-dimensional data onto a relative distance plane—An exact method for visualizing and characterizing high-dimensional patterns. *Journal of Biomedical Informatics*, 37(5):366–379.
- Späth, H. (1980). *Cluster Analysis Algorithms*. West Sussex, UK: Ellis Horwood Limited.
- Spielmat, D. and Teng, S. (1996). Spectral partitioning works: Planar graphs and finite element meshes. In *Proceedings of the 37th annual symposium on foundations of computer science*, pages 96–105. Burlington, VT: IEEE Computer Society.
- Sprenger, T., Brunella, R., and Gross, M. (2000). H-BLOB: A hierarchical visual clustering method using implicit surfaces. In *VIS '00: Proceedings of the conference on visualization '00*, pages 61–68. Los Alamitos, CA: IEEE Computer Society Press.
- StatSoft, Inc. (2005). *Time Series Analysis*. <http://www.statsoftinc.com/textbook/sttimser.html>.
- Struyf, A., Hubert, M., and Rousseeuw, P. (1996). Clustering in an object-oriented environment. *Journal of Statistical Software*, 1(4):1–30.
- Struyf, A., Hubert, M., and Rousseeuw, P. (1997). Integrating robust clustering techniques in S-PLUS. *Computational Statistics and Data Analysis*, 26(1):17–37.
- Stute, W. and Zhu, L. (1995). Asymptotics of k -means clustering based on projection pursuit. *The Indian Journal of Statistics*, 57(3):462–471.
- Su, M. and Chou, C. (2001). A modified version of the k -means algorithm with a distance based on cluster symmetry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):674–680.
- Su, Z., Yang, Q., Zhang, H., Xu, X., Hu, Y., and Ma, S. (2002). Correlation-based web document clustering for adaptive web interface design. *Knowledge and Information Systems*, 4(2):151–167.

- Sung, C. and Jin, H. (2000). A tabu-search-based heuristic for clustering. *Pattern Recognition*, 33(5):849–858.
- Tamura, S., Higuchi, S., and Tanaka, K. (1971). Pattern classification based on fuzzy relations. *IEEE Trans. Systems Man Cybernet.*, 1(1):61–66.
- Tang, C. and Zhang, A. (2002). An iterative strategy for pattern discovery in high-dimensional data sets. In *Proceedings of the eleventh international conference on information and knowledge management*, pages 10–17. New York: ACM Press.
- Tang, C., Zhang, L., Ramanathan, M., and Zhang, A. (2001). Interrelated two-way clustering: An unsupervised approach for gene expression data analysis. In *Proceedings of the 2nd IEEE International Symposium on Bioinformatics and Bioengineering*, pages 41–48. Washington, DC: IEEE Computer Society.
- Tarsitano, A. (2003). A computational study of several relocation methods for k -means algorithms. *Pattern Recognition*, 36(12):2955–2966.
- Tavazoie, S., Hughes, J., Campbell, M., Cho, R., and Church, G. (1999). Systematic determination of genetic network architecture. *Nature Genetics*, 22:281–285.
- Thaper, N., Guha, S., Indyk, P., and Koudas, N. (2002). Dynamic multidimensional histograms. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on management of data*, pages 428–439. New York: ACM Press.
- Theodoridis, S. and Koutroubas, K. (1999). *Pattern Recognition*. London: Academic Press.
- Thomasian, A., Castelli, V., and Li, C. (1998). Clustering and singular value decomposition for approximate indexing in high dimensional spaces. In *Proceedings of the seventh international conference on information and knowledge management*, pages 201–207. New York: ACM Press.
- Thorup, M. (2001). Quick k -median, k -center, and facility location for sparse graphs. In *ICALP '01: Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes In Computer Science*, pages 249–260. London, UK: Springer-Verlag.
- Tibshirani, R., Walther, G., and Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *J. Royal Statistical Soc. B*, 63(2):411–411.
- Trauwaert, E. (1988). On the meaning of Dunn's partition coefficient for fuzzy clusters. *Fuzzy Sets and Systems*, 25(2):217–242.
- Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525.
- Tsay, R. (2002). *Analysis of Financial Time Series*. Wiley Series in Probability and Statistics. New York: John Wiley & Sons.

- Tseng, L. and Yang, S. (2000). A genetic clustering algorithm for data with non-spherical-shape clusters. *Pattern Recognition*, 33(7):1251–1259.
- Tseng, L. and Yang, S. (2001). A genetic approach to the automatic clustering problem. *Pattern Recognition*, 34(2):415–424.
- Tsumoto, S. (1999). Rule discovery in large time-series medical databases. In *PKDD '99: Proceedings of the third European conference on principles of data mining and knowledge discovery*, pages 23–31. London: Springer-Verlag.
- Tubbs, J. (1989). A note on binary template matching. *Pattern Recognition*, 22(4):359–365.
- van Groenewoud, H. and Ihm, P. (1974). A cluster analysis based on graph theory. *Vegetatio*, 29:115–120.
- van Rijsbergen, C. (1970). Algorithm 52: A fast hierachic clustering algorithm. *The Computer Journal*, 13(3):324–326.
- Vesanto, J. (1999). SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126.
- Vesanto, J. (2000). Neural network tool for data mining: SOM toolbox. In *Proceedings of the symposium on tool environments and development methods for intelligent systems (TOOLMET2000)*, pages 184–196. Oulu, Finland: Oulun yliopistopaino.
- Vesanto, J., Alhoniemi, E., Himberg, J., Kiviluoto, K., and Parviainen, J. (1999a). Self-organizing map for data mining in MATLAB: The SOM toolbox. *Simulation News Europe*, page 54.
- Vesanto, J., Himberg, J., Alhoniemi, E., and Parhankangas, J. (1999b). Self-organizing map in MATLAB: The SOM toolbox. In *Proceedings of the MATLAB DSP Conference 1999*, pages 35–40. Espoo, Finland.
- Vesanto, J., Himberg, J., Alhoniemi, E., and Parhankangas, J. (2000). SOM Toolbox for MATLAB 5. Report A57, Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland.
- Vlachos, M., Hadjieleftheriou, M., Gunopoulos, D., and Keogh, E. (2003). Indexing multi-dimensional time-series with support for multiple distance measures. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 216–225. New York: ACM Press.
- Wang, C. and Wang, X. (2000). Supporting content-based searches on time series via approximation. In *SSDBM '00: Proceedings of the 12th international conference on scientific and statistical database management*, pages 69–81. Washington, DC: IEEE Computer Society.
- Wang, H., Chu, F., Fan, W., Yu, P., and Pei, J. (2004). A fast algorithm for subspace clustering by pattern similarity. In *Proceedings of the 16th international conference on scientific and statistical database management*, pages 51–60. Santorini, Greece: IEEE.

- Wang, H., Wang, W., Yang, J., and Yu, P. (2002). Clustering by pattern similarity in large data sets. In *Proceedings of the 2002 ACM SIGMOD international conference on management of data*, pages 394–405. New York: ACM Press.
- Wang, K., Xu, C., and Liu, B. (1999a). Clustering transactions using large items. In *Proceedings of the eighth international conference on information and knowledge management*, pages 483–490. Kansas City, Missouri: ACM Press.
- Wang, L. and Wang, Z. (2003). CUBN: A clustering algorithm based on density and distance. In *Proceedings of the 2003 international conference on machine learning and cybernetics*, pages 108–112. Xilan, China: IEEE Press.
- Wang, W., Yang, J., and Muntz, R. (1997). STING: A statistical information grid approach to spatial data mining. In *Twenty-third international conference on very large data bases*, pages 186–195. Athens, Greece: Morgan Kaufmann.
- Wang, W., Yang, J., and Muntz, R. (1999b). STING+: An approach to active spatial data mining. In *Fifteenth international conference on data engineering*, pages 116–125. Sydney, Australia: IEEE Computer Society.
- Ward Jr., J. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244.
- Ward Jr., J. and Hook, M. (1963). Application of a hierarchical grouping procedure to a problem of grouping profiles. *Educational and Psychological Measurement*, 23(1):69–81.
- Wegman, E. (1990). Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675.
- Whittaker, R. (1952). A study of summer foliage insect communities in the Great Smoky Mountains. *Ecol. Monogr.*, 22:1–44.
- Wilks, S. (1962). *Mathematical Statistics*. New York: John Wiley and Sons.
- Willett, P. (1988). Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24(5):577–597.
- Williams, W. and Lambert, J. (1966). Multivariate methods in plant ecology: V. similarity analyses and information-analysis. *Journal of Ecology*, 54(2):427–445.
- Wills, G. (1998). An interactive view for hierarchical clustering. In *INFOVIS '98: Proceedings of the 1998 IEEE symposium on information visualization*, pages 26–31. Washington, DC: IEEE Computer Society.
- Wirth, M., Estabrook, G., and Rogers, D. (1966). A graph theory model for systematic biology, with an example for the oncidiae (orchidaceae). *Systematic Zoology*, 15(1):59–69.
- Wishart, D. (1969). 256. Note: An algorithm for hierarchical classifications. *Biometrics*, 25(1):165–170.

- Wishart, D. (1978). Treatment of missing values in cluster analysis. In Corsten, L., and Hermans, J., editors, *COMPSTAT: Proceedings in Computational Statistics*, pages 281–297. Vienna: Physica Verlag.
- Wishart, D. (2002). *k*-means clustering with outlier detection, mixed variables and missing values. In Schwaiger, M., and Opitz, O., editors, *Exploratory Data Analysis in Empirical Research*, pages 216–226. New York: Springer.
- Wolfe, J. (1970). Pattern clustering by multivariate mixture analysis. *Multivariate Behavioral Research*, 5:329–350.
- Woo, K. and Lee, J. (2002). *FINDIT: A Fast and Intelligent Subspace Clustering Algorithm Using Dimension Voting*. PhD thesis, Korea Advanced Institute of Science and Technology, Department of Electrical Engineering and Computer Science.
- Wu, C. (1983). On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103.
- Wu, C. (1986). Jackknife, bootstrap and other resampling methods in regression analysis. *The Annals of Statistics*, 14(4):1261–1295.
- Wu, L., Faloutsos, C., Sycara, K., and Payne, T. (2000a). FALCON: Feedback adaptive loop for content-based retrieval. In *VLDB '00: Proceedings of the 26th international conference on very large data bases*, pages 297–306. Cairo, Egypt: Morgan Kaufmann.
- Wu, X. and Barbará, D. (2002). Learning missing values from summary constraints. *ACM SIGKDD Explorations Newsletter*, 4(1):21–30.
- Wu, Y., Agrawal, D., and Abbadi, A. (2000b). A comparison of DFT and DWT based similarity search in time-series databases. In *CIKM '00: Proceedings of the ninth international conference on information and knowledge management*, pages 488–495. New York: ACM Press.
- Wu, Z. and Leahy, R. (1993). An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113.
- Xiao, Y. and Dunham, M. (2001). Interactive clustering for transaction data. *Lecture Notes in Computer Science*, 2114:121–130.
- Xie, X. and Beni, G. (1991). A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847.
- Xiong, Y. and Yeung, D. (2002). Mixtures of ARMA models for model-based time series clustering. In *ICDM '02: Proceedings of the 2002 IEEE international conference on data mining*, pages 717–720. Washington, DC: IEEE Computer Society.
- Xu, H., Wang, H., and Li, C. (2002). Fuzzy tabu search method for the clustering problem. In *Proceedings of the 2002 international conference on machine learning and cybernetics*, volume 2, pages 876–880. Beijing: IEEE.

- Xu, J., Xiong, H., Sung, S., and Kumar, V. (2003). A new clustering algorithm for transaction data via caucus. In *Pacific-Asia conference on advances in knowledge discovery and data mining, 7th PAKDD 2003*, volume 2637 of *Lecture Notes in Computer Science*, pages 551–562. New York: Springer.
- Xu, R. and Wunsch II, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678.
- Xu, X., Ester, M., Kriegel, H., and Sander, J. (1998). A distribution-based clustering algorithm for mining in large spatial databases. In *ICDE '98: Proceedings of the fourteenth international conference on data engineering*, pages 324–331. Orlando, FL: IEEE Computer Society.
- Xu, X., Jäger, J., and Kriegel, H. (1999). A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, 3(3):263–290.
- Yang, C., Duraiswami, R., DeMenthon, D., and Davis, L. (2003a). Mean-shift analysis using quasineutron methods. In *Proceedings of the 2003 international conference on image processing (ICIP)*, volume 3, pages II–447–50. Barcelona, Spain: IEEE Computer Society.
- Yang, J., Wang, W., Wang, H., and Yu, P. (2002a). δ -clusters: Capturing subspace correlation in a large data set. In *Proceedings of the 18th international conference on data engineering, 2002*, pages 517–528. San Jose, CA: IEEE Computer Society.
- Yang, J., Ward, M., and Rundensteiner, E. (2003b). Interactive hierarchical displays: A general framework for visualization and exploration of large multivariate data sets. *Computers and Graphics*, 26(2):265–283.
- Yang, K. and Shahabi, C. (2004). A PCA-based similarity measure for multivariate time series. In *MMDB '04: Proceedings of the 2nd ACM international workshop on multimedia databases*, pages 65–74. New York: ACM Press.
- Yang, M. (1993). A survey of fuzzy clustering. *Mathematical and Computer Modelling*, 18(11):1–16.
- Yang, M. and Wu, K. (2001). A new validity index for fuzzy clustering. *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, 1:89–92.
- Yang, Y., Guan, X., and You, J. (2002b). CLOPE: A fast and effective clustering algorithm for transactional data. In *Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 682–687. New York: ACM Press.
- Yazdani, N. and Ozsoyoglu, Z. (1996). Sequence matching of images. In *SSDBM '96: Proceedings of the eighth international conference on scientific and statistical database management*, pages 53–62. Washington, DC: IEEE Computer Society.
- Yeung, K., Fraley, C., Murua, A., Raftery, A., and Ruzzo, W. (2001). Model-based clustering and data transformations for gene expression data. *Bioinformatics*, 17(10):977–987.

- Yeung, K., Medvedovic, M., and Bumgarner, R. (2003). Clustering gene-expression data with repeated measurements. *Genome Biology*, 4(5):G34.1–17.
- Yeung, K. and Ruzzo, W. (2001). Principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9):763–774.
- Yi, B. and Faloutsos, C. (2000). Fast time sequence indexing for arbitrary l_p norms. In *VLDB '00: Proceedings of the 26th international conference on very large data bases*, pages 385–394. New York: Morgan Kaufmann.
- Yi, B., Jagadish, H., and Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping. In *ICDE '98: Proceedings of the fourteenth international conference on data engineering (ICDE)*, pages 201–208. Orlando, FL: IEEE Computer Society.
- Yun, C., Chuang, K., and Chen, M. (2002). Using category-based adherence to cluster market-basket data. In *Proceedings of the 2002 IEEE international conference on data mining (ICDM)*, pages 546–553. Maebashi City, Japan: IEEE Computer Society.
- Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 8:338–353.
- Zadeh, L. (1992). Fuzzy sets. In Bezdek, J., and Pal, S., editors, *Fuzzy Models for Pattern Recognition*, chapter 2, pages 35–45. New York: IEEE Press.
- Zahn, C. (1971). Graph-theoretical methods for detecting and describing Gestalt cluster. *IEEE Transactions on Computers*, C-20:68–86.
- Zaiane, O. and Lee, C. (2002). Clustering spatial data in the presence of obstacles: a density-based approach. In *Proceedings of the international database engineering and applications symposium, 2002*, pages 214–223. Edmonton, Canada: IEEE Computer Society.
- Zait, M. and Messatfa, H. (1997). A comparative study of clustering methods. *Future Generation Computer Systems*, 13(2–3):149–159.
- Zeng, G. and Dubes, R. (1985). A test for spatial randomness based on k -NN distances. *Pattern Recognition Letters*, 3(2):85–91.
- Zhang, B. and Hsu, M. (2000). *Scale up Center-Based Data Clustering Algorithms by Parallelism*. Technical Report HPL-2000-6, Hewlett-Packard Laboratories. <http://www.hpl.hp.com/techreports/2000/HPL-2000-6.html>.
- Zhang, B., Hsu, M., and Dayal, U. (1999). *k -harmonic Means—A Data Clustering Algorithm*. Technical Report HPL-1999-124, Hewlett Packard Laboratories. <http://www.hpl.hp.com/techreports/1999/HPL-1999-124.html>.
- Zhang, B., Hsu, M., and Dayal, U. (2000a). k -harmonic means: A spatial clustering algorithm with boosting. In *International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining, TSDM 2000*, volume 2007 of *Lecture Notes in Artificial Intelligence*, Lyon, France: Springer.

- Zhang, B. and Srihari, S. (2003). *Properties of Binary Vector Dissimilarity Measures*. Technical report, CEDAR, Department of Computer Science & Engineering, University of Buffalo, the State University of New York. <http://www.cedar.buffalo.edu/papers/publications.html>.
- Zhang, T., Ramakrishnan, R., and Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on management of data*, pages 103–114. New York: ACM Press.
- Zhang, Y., Fu, A., Cai, C., and Heng, P. (2000b). Clustering categorical data. In *Proceedings of the 16th international conference on ICDE Data Engineering*, pages 305–305. San Diego, CA: IEEE Computer Society.
- Zhao, L., Tsujimura Y., and Gen, M. (1996). Genetic algorithm for fuzzy clustering. In *Proceedings of the IEEE international conference on evolutionary computation, 1996*, pages 716–719, Nagoya, Japan: IEEE.
- Zhao, Y. and Song, J. (2001). GDILC: A grid-based density-isoline clustering algorithm. In *Proceedings of the international conferences on info-tech and info-net, 2001*, volume 3, pages 140–145. Beijing: IEEE.
- Zhong, S. and Ghosh, J. (2003a). Scalable, balanced model-based clustering. In *Proceedings of the third SIAM international conference on data mining*, pages 1–12. San Francisco: SIAM.
- Zhong, S. and Ghosh, J. (2003b). A unified framework for model-based clustering. *The Journal of Machine Learning Research*, 4:1001–1037.
- Zien, J., Chan, P., and Schlag, M. (1997). Hybrid spectral/iterative partitioning. In *ICCAD '97: Proceedings of the 1997 IEEE/ACM international conference on computer-aided design*, pages 436–440. San Jose, CA: IEEE Computer Society.
- Zien, J., Schlag, M., and Chan, P. (1999). Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1389–1399.

Subject Index

Entries in Courier font refer to MATLAB functions.

- Absolute-valued data, 326
- Adaptive grids, 258
- Adaptive Resonance Theory (ART), 262
- Agglomerative hierarchical, 9, 207
- Akaike information criterion, *see*
Criterion
- Algorithm
 - c -means, 158
 - k -harmonic Means, 171
 - k -means, 11, 38, 161
 - k -medoid, 246
 - k -modes, 176
 - k -probabilities, 179, 372
 - k -prototypes, 181, 372
 - agglomerative hierarchical, 94, 109, 203
 - center-based, 161
 - compare-means, 165
 - continuous k -means, 165
 - density-based, 223
 - depth-first search, 246
 - divisive hierarchical, 94, 109
 - EM, 235
 - fuzzy k -means, 154
 - generalized k -harmonic means, 172
 - graph-based, 203
 - hard c -means, 159
 - hard clustering, 109
 - hierarchical, 109
 - link-based, 203
 - model-based, 227
 - nearest neighbor searching, 116
 - partitional, 109
- search-based, 183
- single-link, 118
- sort-means, 166
- Amplitude transformation, *see*
Transformation
- Anchor point, 377
- AR, *see* Model
- Arithmetic average, 69
- ARMA, *see* Autoregressive moving
average
- ART, *see* Adaptive Resonance Theory
- Association
 - probability, 175
 - rule, 293, 298
- Attribute, 5, 19, 94
 - binary, 21
 - categorical, 181
 - class, 11
 - continuous, 80
 - name, 328
 - ordinal, 80
 - quantitative, 80
- Autoregressive moving average
(ARMA), 288
- average, 358
- Average
 - compactness, 312
 - distance, *see* Distance
 - information, 313
 - linkage, 326
 - scattering, 307
 - standard deviation, 308

- B-tree, 294
- Backpropagation, 59
- Balance weight, 181
- BANG, 214
- Banner, 112, 146
- Bayes
 - classification, 27
 - factor, 237
- Bayes's theorem, 238
- Bayesian approach, 237
- Bayesian information criterion (BIC), 228, 238
- BD*-tree, 376
- BIC, *see* Bayesian information criterion
- Binarization, 25
- Binary, 19, 80
 - data stream, 293
 - feature vector, 77
 - vector unit, 22
- Biological process, 324
- BIRCH, 31, 144, 252, 371
- Bit vector, 93
- Block, 212
- Blurring process, 282
- Bootstrap procedure, 315
- Bootstrapping, 301
- BRIDGE, 221
- BSP*-tree, 376
- CACTUS, 204
- Canberra metric, 75
- Candidate hypothesis, 90
- Cartesian
 - join, 82
 - meet, 82
 - product, 81
 - space model, 82
- Cases by variables, 5
- Categorical attribute, 19, 80
- Categorical data visualization, *see* Visualization
- Categorization, 30
 - automatic, 37
 - cluster-based, 31
 - direct, 30
- cDNA
 - clone, 323
 - microarray, 323
 - sequence, 323
- Cell function, 246
- Cellular process, 324
- Central-extreme principle, 27
- Centroid, 97, 162
 - method, 116
 - weighted, 97
- centroid, 358
- CF, *see* Clustering feature
- CF tree, 144
- Chameleon, 203
- chebychev, 357
- Chernoff bound, 145, 255
- Chi-square statistic, *see* Statistic
- Chip manufacture, 323
- Chord distance, *see* Distance
- Cis-regulatory element, 325
- City block distance, *see* Distance
- cityblock, 357
- CLARANS, 31
- Class-preserving projection, *see* Projection
- Classification
 - likelihood, 227, 229, 233
 - unsupervised, 3
- CLINK, 141, 296
- Clipping, 288
- CLIQUE, 244, 249, 371
- CLM, 231
- CLOPE, 295
- CLTree, 244, 261, 371
- cluster, 355
- Cluster
 - analysis, 3
 - class, 6, 7
 - compact, 6
 - feature, 296
 - similarity measure, 306
 - validity, 299
- Clustering
 - crisp, 8, 151, 305
 - extended, 252
 - feature, 144

- fuzzy, 8, 183, 271, 305
gene-based, 325
hard, 7, 8, 151, 183, 305
model-based, 237
sample-based, 325
scheme, 304
soft, 8
subsequence, 287
whole, 287
- CluStream, 293
- Coefficient
asymmetrical, 78
cophenetic correlation, 101, 303
Czekanowski, 75
dice, 78
of divergence, 75
general distance, 80
general similarity, 79
Hamann's, 78
Jaccard, 78, 295, 303
Kulzinsky, 78, 79
Ochiai, 79
Pearson's, 78
Rogers-Tanimoto, 78
Russell-Rao, 78, 79
Simple matching, 78
Sokal-Michener, 78
Sokal-Sneath, 78
Sørensen, 79
symmetrical, 78
Yule, 78
- Coexpressed gene, *see* Gene
- Cofunction, 325
- Combinatorial optimization, 185
- Commutativity, 67
- Compactness, 265
of the fuzzy c -partition, 318
- Compactness-separation, 37
- Complement, 22
- complete, 358
- Complete
data, 235
link, 97, 116, 142, 326
- Computational complexity, 163
- Concentrated loglikelihood, 233
- Conditional
expectation, 237
information, 313
- Configuration, 206
- Conjugate gradient, 236
- Connectedness method, 118
- Contiguous, 34
partition, 148
- Contingency table, 101
- Continuous time series, *see* Time series
- Contrast set, 241
- Convergence, 206
- Convex hull, 174, 307, 314
- COOLCAT, 240
- Cophenetic
correlation coefficient, *see*
Coefficient
matrix, 303
- Coregulation, 325
- correlation, 357
- Correlation coefficient, 98
Pearson produce moment, 98
- Correspondence function, 88
- COSA, 244
- cosine, 357
- Cosine similarity measure, 93
- Cost function, 157, 294
- Covariance, 70, 285
matrix, 227, 230
- CPU time, 9
- Crisp clustering, *see* Clustering, 151, 305
- Criterion
Akárke information, 170
function, 207, 295
new Condorcet, 181
Schwarz, 170
sum of square, 38
validity, 37
- csvread, 344
- csvwrite, 344
- CUBN, 225
- CURE, 31, 144, 371
- Cut level, 61

- Data
 binary, 293
 clustering, 3
 file, 328
 market basket, 23
 matrix, 43
 mining, 4
 multimedia, 223
 parallelism, 258
 point, 5
 preprocessing, 324
 scale, 19
 stream, 289
 transaction, 23, 93
 transformation, *see* Transformation type, 19
- DBCLASD, 222
 DBCluC, 221
 DBSCAN, 219
 Decision
 node, 261
 tree, 261
 DENCLUE, 32, 223
 Dendrogram, 92, 109, 110, 138, 296, 358
 Density index, 212, 215, 219, 223
 Density-isoline figure, 214
 DFT, *see* Discrete Fourier transformation
 Diameter, 146, 174, 307
 DIANA, 138, 371
 Dichotomization, 25
 Dimension reduction, 243
 Dimensionality, 5
 Dimensionless, 43
 Discrete Fourier transformation (DFT), 91
 Discrete time series, *see* Time series
 Discrete wavelet transformation (DWT), 91
 Discriminability, 27
 Discriminant analysis, 27
 DISMEA, 147, 371
 Dispersion measure, 306
 Dissimilarity, 5, 6, 67, 68, 326
- Distance
 average, 74, 96
 between an object and a cluster, 94
 between partitions, 312
 between two clusters, 94
 chord, 74
 city block, 71
 dimension-oriented, 255
 Euclidean, 71, 118
 farthest neighbor, 95
 function, 67, 290
 generalized Mahalanobis, 73
 generalized Minkowski, 81, 82
 geodesic, 74
 index of association, 75
 interclass, 59
 intercluster, 37, 96, 307
 intraclass, 319
 intracluster, 37, 307
 Mahalanobis, 72
 Manhattan, 71
 Manhattan segmental, 72, 248
 maximum, 72
 mean character difference, 75
 measure, 271
 Minkowski, 72, 84
 nearest neighbor, 95, 118
 point symmetry, 169
 projected, 250
 simple matching, 76, 177, 265
 squared, 180
 statistical, 96
 sum of squared (SSD), 33, 147
- Divide-by-zero error, 271
 Divisive hierarchical, 9
 monothetic, 138
`d1mread`, 344
`d1mwrite`, 344
 DNA sequence, 323
 DNF expression, 244
 DOC, 244, 259, 371
 Domain, 94
 Dot-product, 250
 Dunn family of indices, 307
 DWT, *see* Discrete wavelet transformation

- Dynamic
piecewise, 88
programming, 34
system, 205
time warping, 87
- ECF, *see* Extended cluster feature
- Edit distance, 89
- Eigenvalue, 47, 252, 285
- Eigenvector, 47, 51, 252, 285
- EM, *see* Expectation-Maximization algorithm
- Empirical distribution, 26
- ENCLUS, 244, 253
- Entropy, 65, 175, 240, 253
- Equilibrium point, 282
- Erosion operation, 225
- Error function, 161
- Error sum-of-squares (ESS), 132
- ESS, *see* error sum-of-squares
- EST, *see* Expressed sequence tag
- Estimate
Andrew's wave, 44
Huber's, 44
Tukey's biweight, 44
- euclidean, 357
- Euclidean
distance, 326
norm, 175, 271
space, 144
sum of squares, 180
- Excited neuron, *see* Neuron
- Expectation, 39
- Expectation-maximization (EM) algorithm, 235
- Exponential histogram, 292
- Exponential-family, 236
- Expressed sequence tag (EST), 323
- Expression
level, 323, 324
matrix, 324
pattern, 324
profile, 324, 325
- Extended cluster feature (ECF), 249
- External criteria, 299
- Facility
clustering, 291
location problem, 290
- FastMap, 65
- fclose, 344
- FCLUST, 65
- Feature, 5, 325
extended, 249
selection, 243
space, 50, 55
tree structured, 81
vector, 78
- FINDIT, 244, 255
- Fisher's suboptimization lemma, 34
- Fit, 227
- Flat pattern, 324
- FLOC, 244
- Fluorescent intensity, 323
- Folkes and Mallows index, 303
- fopen, 343
- fprintf, 344
- fread, 344
- Free energy, 175
- Frequency, 76
estimation, 290
table, 20, 265
- FSC, *see* Fuzzy subspace clustering
- fscanf, 344
- FTS, *see* Fuzzy tabu search
- Function
density, 223
Gaussian, 57
Gaussian influence, 223
indicator, 153
local density, 224
square wave influence, 223
- Function M-file, *see* M-file
- Fuzzifier, 271, 316
- Fuzzy
c-means, 183
c-partition, 315
k-means, 183
k-partition, 8
clustering, *see* Clustering, 183, 270, 305
deviation, 318

- dimension weight, 271
 dimension weight matrix, 270, 271
 ISODATA, 183
 membership, 270
 number, 320
 relation, 142
 set, 152, 154
 subspace clustering (FSC), 270
 tabu search (FTS), *see* Tabu search
`fwrite`, 344
- Gain criterion, 261
 Gap statistic, 39
 Gaussian
 distribution, 227, 327
 kernel, 281
 mixture model, 230
 GDBSCAN, 221
 GDILC, 214
 Gene
 coexpressed, 324
 expression data, 3
 function, 324
 informative, 325
 irrelevant, 325
 regulation, 324
 Gene-based clustering, *see* Clustering
 General similarity coefficient, *see*
 Coefficient
 Generalized Mahalanobis distance, *see*
 Distance
 Generative algorithm, 288
 Genetic k -modes, *see* k -modes
 Geodesic distance, *see* Distance
 Gibbs distribution, 175
 Global standardization, *see*
 Standardization
 Goodness measure, 207
 GPP, *see* Graph partition problem
 Gradient, 224, 295
 vector, 55
 Graph, 140, 203
 k -nearest neighbor, 203
 directed, 69
 directed acyclic, 89
 partition problem (GPP), 200
- partitioning, 204
 proximity, 69
 similarity, 204
 sparse, 203
 undirected, 69
 weighted, 69
 Graphical representation, 60
 Greedy technique, 247
 Grid, 209, 212
 GRIDCLUS, 212
- H-BLOB, 65
 H -means, 192
 hamming, 357
 Hard c -partition, 151
 Hard clustering, *see* Clustering
 Hard k -partition, 8, 271
 Hard limiting, 288
 Harmonic
 average, 171
 mean, 171
 Hash table, 294
 Height, 97, 111
 Hessian matrix, *see* Matrix
 Heuristic, 185
 Hidden Markov model, 288
 Hierarchical, 9
 agglomerative, 109, 137
 divisive, 109, 137
 Hierarchy of clustering schemes, 303
 Hill climbing, 185
 Histogram, 258
 Hk -means, 192
 HPStream, 293
 Hubert's Γ statistic, 303, 305
 Hybridization, 323
 Hypercube, 224
 Hypergraph, 203, 205
 Hypersphere, 74
 Hypothesis, 238, 301
 random, 301
 random graph, 301
 random label, 301
 random position, 301

- Icicle plot, 115
Incomplete data, 235
Incremental, 240, 293, 296
Index
 Calinski-Harabasz (CH), 310
 CD, 310
 CH, *see* Calinski-Horabasz
 cutting, 35
 Davies-Bouldin, 305
 Dunn's, 307
 Fukuyama-Sugeno, 316
 fuzzy validity, 315
 partition coefficient, 316
 partition entropy, 316
 R-squared, 310
 Rand's, 311
 RMSSDT, 309
 S_Dbw validity, 308
 SD validity, 307
 semipartial R-squared, 310
Indexing, 83
Information
 retrieval (IR), 293
 theory, 312
Informative gene, *see* Gene
Inherent sparsity, 243
Initialization, 293
Inner product, 57, 93
Input layer, 56
Integrated likelihood, 238
Interactive, 296
Interactive hierarchical display, 61
Interclass distance, *see* Distance
Interclass scatter matrix, *see* Matrix
Intercluster density, 308
Internal criteria, 299
Interval scale, *see* Scale
Intraclass variance, 308
IR, *see* Information retrieval
Irrelevant gene, *see* Gene
Item, 5

J-means, 190, 192
jaccard, 357
Jackknife correlation, 327
Joint probability, 240

k-modes
 genetic, 195
Karhunen-Loëve transformation, *see*
 Transformation
kd-tree, 167, 375
 adaptive, 376
Kernel, 173
kmeans, 359
k-means, 147, 271, 275, 326
 genetic, 192
 global, 195
 greedy, 195
 parallel, 169
 trimmed, 169
k-median, 290
Kohonen layer, 56
k-plane, 182
Kuhn-Tucker problem, 280

Lagrange multiplier, 47, 175, 275
Lance-Williams formula, 96, 117
Landmark
 model, 91
 similarity, 91
LargeItem, 294
Lateral distance, 57
LCS, *see* Longest common subsequence
Leaf node, 261
Learning rate, 56
Least squares, 32, 34
Level, 138
Likelihood, 90
Linear
 scan, 289
 trend, 86
Link, 207
Link-based similarity measure, 93
linkage, 355
Location measure, 44
Log ratio data, 326
Loglikelihood, 91, 237
Logtransformation, 324
Longest common subsequence (LCS),
 88
Loop plot, 115
LSEARCH, 290

- M-file, 349
 - function, 349
 - script, 349
- MAFIA, 244, 258
- mahalanobis, 357
- Main memory, 289
- Manhattan distance, *see* Distance
- Mapping
 - error, 54
 - nonprojective, 60
 - Sammon's, 53
- Market basket data, 93
- Market segmentation, 4
- Markov chain, 201, 288
- MAT-file, 349
- Matching criterion, 57
- Matching distance, 88
- MATLAB, 343
- Matrix
 - between-clusters scatter, 70, 96
 - binary indicator, 178
 - covariance, 47, 70, 252
 - dissimilarity, 118
 - distance, 69
 - fuzzy, 183
 - fuzzy covariance, 155
 - Hessian, 55
 - interclass scatter, 59
 - proximity, 68
 - relation, 154
 - sample covariance, 47, 71
 - scatter, 69
 - semidefinite, 59
 - similarity, 69
 - sparse, 293
 - sum of squares, 69
 - unitary, 48
 - within-cluster scatter, 70
- Maximum likelihood, 230
- Maximum-entropy clustering (MEC),
 - 175
- MDL, *see* Minimal description length
- MDS, *see* Multidimensional scaling
- Mean, 44, 81, 161, 181
- Mean shift, 173, 275
- Mean standardization, *see* Standardization
- Mean-based initialization, *see* Initialization
- Mean-square contingency, 102
- Measurement, 24
- MEC, *see* Maximum-entropy clustering
- Median, 44, 97, 290
 - method, 116
 - standardization, *see* Standardization
- median, 358
- Medoid, 247, 256
- Membership function, 152
- Memory space, 9
- Metric, 67, 68
- Metric space, 290
- MEX-file, 349, 351
- Microarray
 - experiment, 323
 - technology, 323
- Mini-max filter, 319
- Minimal description length (MDL), 245
- Minimal ultrametric relations (MUR),
 - 142
- Minimum distance/percentage principle, 91
- Minimum method, 118
- Minimum spanning tree (MST), 65
- Minimum sum of squares clustering (MSSC), 190, 275
- minkowski, 357
- Minkowski distance, *see* Distance
- Misclassification, 27
- Missing value, 10, 324
- Mixture likelihood, 227
- Mode, 6, 176, 181
- Model, 227
 - Autoregressive (AR), 289
 - polynomial mixture, 288
 - variable, *see* Variable
- Model-based clustering, *see* Clustering
- Monothetic, 138
- Monotonic hierarchy, 117
- Monotonicity, *see* Divisive hierarchical
- Monte Carlo, 39, 259, 301, 302

- Moving average, 86
MSSC, *see* Minimum sum of squares clustering
MST, *see* Minimum spanning tree
Multidimensional scaling (MDS), 54, 65 metric, 55
nonmetric, 55
Multivariate normal mixture, 229
MUR, *see* Minimal ultrametric relations

n-tree, 110
NCC, *see* Criterion
Nearest neighbor, 116, 118
Netlab toolbox, 343
Neural projection, 59
Neuron, 56, 57
Noise, 144, 223, 324
Nominal, 80
 attribute, 19
 scale, *see* Scale
Nonhierarchical, 161
Nonprojective mapping, *see* Mapping
Norm estimation, 290
Normalization, 19, 43, 206
Normalized Γ statistic, 303
Normalized point, 74
NP-hard, 138, 201
Null hypothesis, 301
Null model, 314

OAK, 296
Object, 5, 325, 326
 first-order, 23
 function, 180
 second-order, 23
Objective
 evaluation, 92
 function, 38, 271, 294
Observation, 5
Offset transformation, *see* Transformation
Oligo, *see* Oligonucleotide microarray
Oligonucleotide microarray (Oligo), 323
OptiGrid, 210
Optimization, 162
ORCLUS, 244, 249, 372

Order statistics, 290
Ordinal scale, *see* Scale
Orthonormal
 basis, 59
 transformation, *see* Transformation vector, 250

Packed representation, 113, 114
Parallel coordinates, 60
PART, 244, 262
Partition function, 175
Partitional, 9, 161
Pattern, 5
 recognition, 4
 standardization, 324
PCA, *see* Principle component analysis
PCR product, 323
pdist, 355
Peak, 327
Pearson's correlation coefficient, 326
Performance function, 171
Phenotype structure, 325
Pointer representation, 112, 138, 142, 296

Poisson
 model, 314
 process, 301
Polythetic, 138
Posterior probability, 238
Principal component, 46
Principal component analysis (PCA), 46, 243
Prior probability, 238
Probability
 density function, 301
 density-based, 230
 model, 43, 227
PROCLUS, 244, 246, 372
Profile, 173
Projected cluster, 243, 249, 252, 262
Projection, 174
 class-preserving, 59
Prototype, 165, 181
Proximal point, 236
Proximity, 19
 index, 68

- level, 303
- matrix, 302, 305
- relation, 54
- p*-values, 289
- Quadtree, 376
- Quantitative, 79
- Quantization, 19
- Quasi-Newton method, 236
- Query sequence, 90
- Quick sort, 32, 369
- Radius, 174
- Rand statistic, 303
- Random position hypothesis, 314
- Range, 44
- Range standardization, *see* Standardization
- Rank, 46
- Ratio scale, *see* Scale
- Raw data, 43
- Record name, 328
- Reference variable, *see* Variable
- Reflexivity, 67
- Regularity condition, 238
- Relation, 153
- Relative
 - closeness, 203
 - criteria, 299, 304
 - density, 261
 - distance plane, 65
 - interconnectivity, 203
- Relevance feedback, 85
- Replicate, 324
- Repulsion, 295
- ROCK, 207
- Sammon's mapping, *see* Mapping
- Sample mean, 281
- Sample space, 235
- Sample-based clustering, *see* Clustering
- SARS, *see* Simulated annealing using random sampling
- Scale, 19, 25
 - conversion, 25
 - interval, 25
- measure, 44
- nominal, 25
- ordinal, 25
- qualitative, 19
- quantitative, 19
- ratio, 25
- transformation, 324
- Scaled pattern, 326
- Scanning process, 323
- Scatter plot, 302
- Schwarz criterion, *see* Criterion
- Script M-file, *see* M-file
- Segmentation analysis, 3
- Self-organizing map (SOM), 56, 230, 326
 - architecture, 56
 - toolbox, 343
- Semidefinite, 51
- Separation, 265
 - of the fuzzy c -partition, 318
- Set-valued variable, *see* Variable
- euclidean, 357
- Shifting pattern, 326
- Signal reconstruction, 290
- Signal-to-noise ratio, 325
- Silhouette plots, 115
- Similarity, 5, 6
 - coefficient, 5, 67
 - dichotomy, 68
 - function, 68
 - measure, 5, 67, 326
 - trichotomy, 68
- Simulated annealing, 183, 201, 202
 - using random sampling (SARS), 201
- single, 358
- Single clustering scheme, 303
- Single link, 97, 116, 138, 325
- Singleton cluster, 233
- Singular value decomposition (SVD), 47, 48
- Skyline plot, 115
- SLINK, 138, 141, 296, 372
- Soft clustering, *see* Clustering
- Solution space, 183
- SOM, *see* Self-organizing map

- Soundness, 257
Sparse matrix, *see* Matrix
spearmen, 357
Spearman's rank correlation, 28
Spearman's rank-order correlation, 28,
 327
Spectral clustering, 208
Spot, 323
SSC, *see* Sum of squares criterion
SSD, *see* Sum of squared distance
Standard deviation, 44
Standard Template Library (STL), 363
Standard variance, 81
Standardization, 43
 global, 43
 mean, 44
 median, 44
 range, 44
 std, 44
 within-cluster, 43
Star coordinates, 61
State, 20, 180
Statistic
 chi-square, 101
Statistical scatter, 69
Statistics Toolbox, 355
std standardization, *see* Standardization
Steepest descent procedure, 53
STING, 209
STL, *see* Standard Template Library
STREAM, 293
Stream model, 289
STUCCO, 241
SUBCAD, 264, 373
Subcell type, 325
Subclusters, 204
Subjective evaluation, 92
Subspace
 clustering, 243, 264
 information, 271
Substitution, 26
Substructure, 325
Subtree, 61
Subtype, 324
Sum of squared distance (SSD), 33
Sum of squares, 147, 309
Sum of squares criterion (SSC), *see*
 Criterion
Summary, 204
Supervised
 analysis, 325
 learning, 261
 training, 59
Support, 94, 294
SVD, *see* Singular value decomposition
Symbol table, 20
Symbolic data, 23
Synopsis structure, 290
Systematic variation, 324
Tabu, 185, 186
Tabu search, 183, 185, 186
Task parallelism, 258
Taxonomy analysis, 3
Temporal data, 24
textread, 344
Time, 24
Time indexed, 290
Topological neighborhood, 57
Total separation, 307
Trace, 59, 69
Transaction, 23
Transformation, 43, 46
 amplitude, 85
 data, 46
 Karhunen-Loëve, 49
 offset, 85
 orthonormal, 50
Translation, 174
Tree, 140
 binary, 110
 dichotomous, 110
 map, 61
 minimum spanning (MST), 140
 nonranked, 110
 spanning, 140
 valued, 110
Trend, 24, 86
Triangle inequality, 67, 165
Tuple, 5

- Ultrametric, 111, 117
 - minimal, 142
 - relation, 142
- Uniform distribution, 26
- Uniformity hypothesis, 314
- Unimodal
 - function, 57
 - model, 314
- Unsupervised
 - analysis, 325
 - competitive learning, 59
 - processing, 299
- UPGMA, 122
- Validity
 - criteria, 299
 - criterion, *see* criterion
 - index, 8, 37
- Valley, 327
- Variable, 5
 - asymmetric binary, 21
 - categorical, 30
 - modal, 23
 - neighborhood search (VNS), 186
 - nominal, 180
 - numerical, 30
 - reference, 30
 - set-valued, 23
 - symmetric binary, 21
- Variance, 285
 - of a data set, 309
 - intraclass, 308
- Vector
 - feature, 50
 - field, 65
- Visualization, 60
 - categorical data, 62
- VNS, *see* Variable neighborhood search
- ward, 358
- Ward's method, 27, 97, 116
- Warping
 - cost, 87
 - path, 87
 - window, 87
- WaveCluster, 216
- Wavelet transform, 216
- Weight, 140, 206, 262, 271
 - vector, 57
- weighted, 358
- Weighted distance, 331
- WGSS, *see* Within-group sum of squares
- Winner-takes-all neuron, *see* Neuron
- Within-clusters standardization, *see* Standardization
- Within-group sum of squares (WGSS), 158
- x*-means, 170
- z*-score, 44

Author Index

A

Abbadi, A.E., 97
Achlioptas, D., 308
Agarwal, P.K., 177, 274, 303
Aggarwal, C., 311
Aggarwal, C.C., 76, 261, 264, 303
Agrawal, D., 50, 97
Agrawal, R., 90, 93, 257, 259, 264, 277
Ahmad, A., 172
Aho, A., 261
Alhoniemi, E., 363
Alippi, C., 194
Allison, L., 37
Alon, N., 308
Alpert, C.J., 220
Al-Sultan, K.S., 198, 201, 214
Altman, R.B., 344
Amir, A., 303
Anderberg, M.R., 27, 28, 54, 80
Andrews, D.F., 68
Andrews, H.C., 50
Andrienko, G., 63
Andrienko, N., 63
Anguelov, D., 98
Arabie, P., 57
Azoff, E.M., 25

B

Babcock, B., 308
Babu, G.J., 172
Babu, G.P., 172, 195
Back, B., 61
Bagnall, A.J., 306
Baker, F.B., 220

Bandyopadhyay, S., 194, 195, 329
Banfield, C.F., 118
Banfield, J.D., 242
Barbará, D., 12, 252, 307, 339
Barber, C.B., 156, 325
Barberá, H.M., 196
Bar-Joseph, Z., 306
Basford, K.E., 250
Batagelj, V., 123
Batistakis, I., 40, 326
Batistakis, Y., 40, 317, 328, 334
Baulieu, F.B., 81, 112
Baumgartner, R., 68
Bay, S., 254
Bay, S.D., 18
Belacel, N., 202
Bell, D.A., 214
Bellman, R., 159
Belongie, S., 220
Beltramo, M.A., 391
Beni, G., 159, 336
Bensmail, H., 255
Bentley, J.L., 397
Berndt, D., 92
Berry, M.J.A., 4
Beule, D., 344
Beyer, K.S., 257
Beygelzimer, A., 67
Bezdek, J.C., 162, 166, 167, 170, 195,
 334
Bhatnagar, V., 190
Binder, D.A., 239
Biswas, J., 167
Blum, A., 389, 392

- Bobisud, H.M., 116
Bobisud, L.E., 116
Bobrowski, L., 166, 170
Bock, H.H., 6, 333, 339
Bollobás, B., 25, 88
Borg, I., 57
Borisov, A., 221
Borodin, A., 309
Botstein, D., 3, 344
Bouldin, D.W., 40, 324
Boyles, R.A., 249
Bozdogan, H., 179
Bozkaya, T., 94
Bradley, P.S., 191, 282, 311
Brazma, A., 345
Broder, A.Z., 312
Brown, E.L., 343, 344
Brown, P., 344
Brown, P.O., 3, 343
Brown, S.J., 195, 205
Brunella, R., 68
Buhmann, J., 68
Buhmann, J.M., 9, 68
Buick, D., 4
Bumgarner, R.E., 3
Byrne, M.C., 343
- C**
- Cai, C., 339
Calinski, T., 329
Callaghan, L.O., 307
Campbell, M.J., 344, 348
Cantor, M., 344
Cao, Y., 278
Caraça-Valente, J.P., 89
Carmichael, J.W., 6
Carpenter, G.A., 278
Carroll, J.D., 57, 187, 401
Castelli, V., 51
Cattell, R.B., 106, 112
Cavalli-Sforza, L.L., 145, 155, 239
Celeux, G., 242
Ceżkiewski, J., 79
Chakrabarti, K., 88
Chan, K., 97
Chan, P.K., 220
- Chang, C., 65
Chang, J., 230, 303
Charikar, M., 308, 310
Chatterjee, S., 229
Chaturvedi, A., 187, 401
Chaudhuri, B.B., 195
Chaudhuri, S., 308
Chee, M.S., 343
Chen, A., 230
Chen, H., 183
Chen, J., 176
Chen, K., 195
Chen, M., 312, 316
Chen, N., 230
Cheng, C., 194, 268
Cheng, Y., 181, 299
Chernoff, H., 68
Ching, R., 176
Chiou, Y., 195
Chiu, B., 89
Cho, R.J., 344, 348
Chou, C., 177
Choudhary, A., 230, 273
Chow, T.W.S., 230
Chowdhury, N., 194
Chrétien, S., 249
Christopher, M., 4
Chu, F., 303
Chu, K.K.W., 93
Chu, S., 88
Chu, W.W., 98
Chuang, K., 312, 316
Chun, S., 90
Chung, C., 90
Chung, F., 220
Church, G.M., 344, 348
Clatworthy, J., 4
Clifford, J., 92
Cochran, W.G., 29
Cohen, P., 89, 306
Coller, H., 345
Comaniciu, D., 4, 181
Conover, W.J., 48
Constantine, A.G., 71
Constantinescu, P., 117
Conway, A., 348

- Cook, D., 68
Cooper, M., 63
Cooper, M.C., 46
Cormack, R.M., 18
Cormen, T.H., 94
Corter, J., 339
Costa, I.G., 346
Cotofrei, P., 305
Couch, A.L., 177
Couto, J., 252, 339
Cowgill, M.C., 195
Cox, M.A.A., 57
Cox, T.F., 57
Cuesta-Albertos, J.A., 177
Cunningham, K.M., 339
- D**
D'haeseleer, P., 347
Dale, M.B., 145
Damond, M., 348
Das, G., 24, 25, 88, 89, 95
Dasgupta, A., 242
Dash, M., 233
Datar, M., 308
Dave, R.N., 334
Davies, D.L., 40, 324
Davis, L., 183
Davis, R.W., 343, 348
Day, N.E., 239
Day, W.H.E., 157
Dayal, U., 180
De Backer, S., 56, 57
de Carvalho, F.A., 346
De Moor, B., 347
De Smet, F., 347
de Souto, M.C., 346
Debregeas, A., 98
Defays, D., 149, 315
Delattre, M., 145
Delgado, M., 196
DeMenthon, D., 183
Demko, A., 68
Dempster, A.P., 248
Deng, K., 397
Deng, Y., 195, 205
Dhillon, I.S., 62
- Diaz-Uriarte, R., 344
Diday, E., 24
Dimsdale, B., 62
Dinesh, M.S., 24
Ding, C., 49
Ding, Z., 65
Dittenbach, M., 68
Dobkin, D.P., 156, 325
Dolenko, B., 68
Domeniconi, C., 303
Dong, H., 343
Dopazo, J., 344
Dowe, D.L., 37
Drineas, P., 51, 308
Dubes, R.C., 5, 19, 45, 116, 129, 169,
205, 214, 317, 333
- DuBien, J.L., 102
Dunham, H., 312
Dunham, M.H., 23, 98, 315
Dunn, J.C., 40, 325
Duraiswami, R., 183
Duran, B.S., 101
- E**
Edelsbrunner, H., 157
Edwards, A.W.F., 145, 155, 239
Efron, B., 321, 347
Egan, M.A., 68
Eickhoff, H., 344
Eisen, M.B., 3, 343
Eklund, T., 61
Elkan, C., 172
El-Sonbaty, Y., 231
Engelman, L., 38
Erhart, M., 226
Esposito, F., 23
Estabrook, G.F., 84, 220
Ester, M., 231, 233, 234
Estivill-Castro, V., 5
Everitt, B.S., 6, 121, 138, 145
- F**
Faber, V., 34, 173
Faloutsos, C., 68, 90, 97
Fan, W., 303
Farmer, E.E., 348

- Farouk, M., 231
Fashing, M., 183
Fayyad, U.M., 282, 311
Feder, T., 309
Fedjki, C.A., 201
Feigenbaum, J., 308
Felsenstein, J., 334
Filho, J.L.R., 194
Firoiu, L., 306
Fisher, L., 144
Fisher, W.D., 11, 34, 35, 156
Fitzgibbon, L.J., 37
Florek, K., 124
Flynn, P.J., 5, 18
Follettie, M.T., 343
Fotouchi, F., 195
Fotouhi, F., 205
Fowlkes, C., 220
Fox, G.C., 182, 183, 292, 293
Fraley, C., 5, 25, 240, 242, 249, 255
Frank, R.E., 4
Freg, C.P., 303
Friedman, H.P., 11, 41
Friedman, J.P., 121
Frieze, A., 51, 308
Fu, A., 339
Fu, A.W., 268
Fu, W., 97
Fua, Y., 63
Fuhrman, S., 348
Fujikawa, Y., 11
Fukunaga, K., 51, 181
Fukuyama, Y., 335
- G**
Gaber, M.M., 307
Gabrielian, A.E., 348
Gada, D., 306
Gaede, V., 398
Gallo, M.V., 343
Gan, G., 188, 195, 207, 280, 287, 292
Ganti, V., 99, 216, 277, 308
Garai, G., 195
García-Escudero, L.A., 178
Gassenbeek, M., 345
Gath, I., 163
- Gavrilov, M., 98
Ge, X., 96
Gehrke, J., 99, 216, 257, 259, 264, 308
Gehrke, J.E., 277
Gen, M., 195
George, J.A., 6
Georgescu, B., 183
Gerber, G., 306
Geurts, P., 98
Geva, A.B., 163
Ghosh, J., 306
Gibbons, F.D., 346
Gibbons, P.B., 308
Gibson, D., 217, 339
Gifford, D.K., 306
Gilbert, A.C., 308
Gionis, A., 308
Gioviale, V., 23
Glassman, S.C., 312
Gluck, A., 339
Goil, S., 230, 273
Goldberg, D.E., 194
Goldin, D.Q., 91
Goldstein, J., 257
Golub, G.H., 344
Golub, T.R., 345
Gong, G., 321
Gonzalez, T.F., 261
Goodall, D.W., 112
Goodman, L.A., 109, 188
Gordaliza, A., 177, 178
Gordon, A.D., 6, 18, 116, 117, 157
Gotlieb, C.C., 220
Govaert, G., 242
Gowda, K.C., 24
Gower, J.C., 71, 81, 84, 104, 118, 138,
148, 156
Grabusts, P., 221
Green, P.E., 4, 112, 187, 401
Greene, D., 309
Greene, W.A., 194
Greenwald, M., 308
Groenen, P., 57
Gross, M.H., 68
Grossberg, S., 278
Guan, X., 23, 312

- Guha, S., 34, 99, 152, 219, 271, 307
Gunopulos, D., 24, 25, 88, 95, 257, 259,
 264, 303
Günther, O., 398
Güntzer, U., 312, 316
Gupta, S.K., 190
Gurel, A., 97
Gurewitz, E., 182, 183, 292, 293
- H**
- Haas, P.J., 308
Hadjieleftheriou, M., 93
Halkidi, M., 40, 317, 326, 328, 334
Hall, L.O., 195
Hamerly, G., 172
Hampel, F.R., 178
Han, E., 215
Han, J., 34, 311
Hand, D.J., 252
Handl, J., 346
Hankins, M., 4
Hansen, P., 145, 197, 201, 202
Haque, E., 303
Harabasz, J., 329
Harding, E.F., 157
Harel, D., 68
Har-Peled, S., 177, 303
Hartigan, J.A., 5, 34, 38, 72, 112, 117,
 169, 333
Harvey, R.J., 195
Hastie, T., 42, 344
Hathaway, R.J., 167
Haykin, S., 59
He, X., 49
Hebrail, G., 98
Heng, P., 339
Henzinger, M., 307
Herniter, M.E., 363
Hero III, A.O., 249
Herrero, J., 344
Hertz, J., 61
Herzel, H., 344
Hetland, M.L., 88
Hettich, S., 18
Heyer, L.J., 347
Higuchi, S., 162
- Hill, A.A., 344
Himberg, J., 363
Hinneburg, A., 34, 172, 223, 235
Hinterberger, H., 225
Hipp, J., 312, 316
Ho, T., 11
Hoare, C.A.R., 391
Hodges, K., 4
Hodgson, J., 343
Hodson, F.R., 157
Hofmann, T., 68
Holland, J.H., 194
Holman, E.W., 145
Hook, M.E., 140
Hopcroft, J., 261
Hopkins, C.E., 29
Höppner, F., 167
Horne, R., 4
Hostetler, L., 181
Howard, R., 202
Hsu, C., 67, 98
Hsu, M., 180
Hu, Y., 232
Hua, K.A., 212, 214
Huang, Y., 307
Huang, Z., 7, 185, 190, 391, 401
Huard, C., 345
Hubálek, Z., 81, 112
Hubert, L.J., 220
Hubert, M., 363
Hughes, J.D., 344, 348
Huhdanpaa, H., 156, 325
Hunter, C.P., 344
Hussein, N., 206
- I**
- Ibaraki, T., 261
Ichino, M., 86
Ihm, P., 125
Ikehata, Y., 68
Iman, R.L., 48
Indyk, P., 98, 308
Inselberg, A., 62
Ismail, M.A., 170, 231
Itoh, T., 68

J

Jaakkola, T.S., 306
 Jagadish, H.V., 97
 Jäger, J., 233
 Jain, A.K., 5, 18, 19, 45, 76, 116, 129,
 169, 317, 333
 Jambu, M., 102
 Jamshidian, M., 249
 Janacek, G.J., 306
 Jardine, J., 11
 Jardine, N., 220
 Jeffreys, H., 250
 Jennrich, R.I., 249
 Jern, M., 63
 Jiang, D., 3, 343, 347
 Jiang, T., 194
 Jin, D., 230, 303
 Jin, H.W., 196
 Johansson, J., 63
 Johnson, S.C., 117, 124
 Jolliffe, I.T., 49, 75
 Jones, D.R., 391
 Jones, M., 274
 Jordan, M.I., 220
 Julius, R.S., 6

K

Kahveci, T., 97
 Kailing, K., 303
 Kajinaga, Y., 68
 Kalaba, R., 159
 Kalpakis, K., 306
 Kanade, T., 303
 Kandogan, E., 63
 Kanellakis, P.C., 91
 Kannan, R., 51, 220, 308
 Kannan, S., 308
 Kantabutra, S., 177
 Kanth, K.V.R., 50
 Kanungo, T., 176
 Karypis, G., 215
 Kasetty, S., 89
 Kashi, R., 303
 Kass, R.E., 179, 250
 Katoh, N., 261

Kaufman, L., 11, 80, 106, 119, 146, 153,
 190, 261
 Ke, Q., 303
 Keim, D., 172, 223, 303
 Keim, D.A., 34, 69, 235
 Kell, D.B., 346
 Kendall, S.M., 24
 Keogh, E., 88, 89, 305
 Khan, S.S., 172
 Khanna, S., 308
 Kim, D., 90
 Kim, H., 344
 Kim, S., 68, 98
 Kiviluoto, K., 363
 Klawonn, F., 167
 Klein, R.W., 205, 214
 Kleinberg, J., 339
 Kleinberg, J.M., 217
 Klett, C.J., 46
 Klock, H., 68
 Knowles, J., 346
 Kobayashi, M., 343
 Kohonen, T., 58, 355
 Konig, A., 61
 Koren, Y., 68
 Kotidis, Y., 308
 Koudas, N., 98, 308
 Koutroubas, K., 317
 Kriegel, H., 69, 231, 233, 234, 303
 Krishna, K., 195, 203
 Krishnamoorthy, M., 68
 Krishnan, T., 250
 Krishnapuram, R., 303
 Krishnaswamy, S., 307
 Kroeger, P., 303
 Krogh, A., 61
 Kruglyak, S., 347
 Kruse, R., 167
 Kruskal, J.B., 56, 57, 121, 148
 Kruskal, W.H., 109
 Krzanowski, W.J., 41
 Kuiper, F.K., 144
 Kumar, S., 220
 Kumar, V., 215, 312, 316
 Kuncicky, D.C., 372
 Kwon, S., 68

L

- Lai, Y.T., 41
Laird, N.M., 248
Lambert, J.M., 100
Lan, L.W., 195
Lance, G.N., 102, 138, 157
Landau, S., 6
Landsman, D., 348
Landwehr, J.M., 121
Lang, S.D., 212, 214
Lanyon, S.M., 333
Larrañaga, P., 172
Lee, C., 233
Lee, D., 98
Lee, J., 90, 270
Lee, R.C.T., 12
Lee, S., 90
Lee, W., 194
Lee, W.K., 212, 214
Lee, W.S., 230
Leese, M., 6
Legendre, L., 6, 83
Legendre, P., 6, 81, 83, 112
Lehrach, H., 344
Leiserson, C.E., 94
Li, C., 196
Li, C.S., 51
Li, Y., 252, 339
Liao, X., 195
Likas, A., 206
Lin, J., 305
Lin, K., 68, 89, 93
Lin, Y., 176
Lindsay, B.G., 308
Linoff, G.S., 4
Liu, B., 23, 276, 312
Liu, H., 233, 303
Liu, Y., 195
Livny, M., 34, 151, 264, 267, 311
Ljung, P., 63
Lloyd, S.P., 173
Lockhart, D.J., 343, 348
Loh, M.L., 345
Loh, W., 277
Lonardi, S., 89
López-Chavarrías, I., 89

Lorr, M., 9

- Lozano, J.A., 172
Lu, S., 195, 205
Lu, Y., 195, 205, 307
Lukaszewicz, J., 124
- M**
- Mántaras, R.L., 331
Ma, C., 287
Ma, E.W.M., 230
Ma, S., 67, 194, 232, 303
Macnaughton-Smith, P., 145
Macqueen, J.B., 34, 154, 169, 242
Macy, R.B., 389, 392
Maharaj, E.A., 306
Malerba, D., 23
Malik, A., 344
Malik, J., 220
Manasse, M.S., 312
Mandelzweig, M., 68
Mangasarian, O.L., 191
Manku, G.S., 308
Mannila, H., 25, 88, 89, 95
Manolopoulos, Y., 97
Mántaras, R.L., 12, 111
Mao, J., 76
Marchal, K., 347
Marriott, F.H.C., 41
Marshall, A., 343
Martinez, A.R., 241
Martinez, W.L., 241
Mathys, J., 347
Matias, Y., 308
Matoušek, J., 177
Matrán, C., 177
Maulik, U., 194, 195, 329
Mazumdar, S., 177
McClean, S.I., 214
McErlan, F.J., 214
McGill, M.J., 98
McLachlan, G.J., 250
McMorris, F.R., 116
McQuitty, L.L., 104, 124
McSherry, F., 308
Medvedovic, M., 3
Meer, P., 4, 181

- Mehrotra, S., 88
 Mehta, M., 277
 Mendelzon, A., 92, 97
 Mendelzon, A.O., 97
 Meng, X., 248
 Meronk, D.B., 116
 Mesirov, J.P., 345
 Messatfa, H., 11
 Mettu, R.R., 309
 Meyerson, A., 307
 Michalewics, Z., 194
 Michaud, P., 7, 190
 Miller, J.J., 63
 Milligan, G.W., 46, 106, 122
 Milo, T., 97
 Mirkin, B., 14
 Mishra, N., 307, 309
 Mittmann, M., 343
 Mladenović, N., 197, 201, 202
 Mockett, L.G., 145
 Modha, D.S., 62
 Mohanty, S., 333
 Mong, C.T., 12
 Moore, A., 179, 282
 Moore, A.W., 175, 397, 398
 Moreau, Y., 347
 Morgan, B.J.T., 11
 Morrison, D.G., 77
 Motwani, R., 98, 152, 271, 307–309
 Mount, D.M., 176
 Munro, J.I., 308
 Muntz, R.r., 222
 Murali, T.M., 274
 Murtagh, F., 11, 18, 117, 122, 146, 151, 157
 Murthy, C.A., 194
 Murty, M.N., 5, 18, 172, 195
 Murua, A., 25, 255
 Mustafa, N.H., 177, 303
 Muthukrishnan, S., 98, 308
- N**
 Nabney, I.T., 363
 Nagesh, H., 230
 Nagesh, H.S., 273
 Nakhaeizadeh, G., 312, 316
- Narahashi, M., 303
 Narasayya, V., 308
 Narasimha, M.M., 195, 203
 Naud, A., 56, 57
 Naughton, J.F., 308
 Netanyahu, N.S., 176, 303
 Neumann, D.A., 116
 Ng, A.Y., 220
 Ng, M.K., 199
 Ng, R.T., 34
 Nicholls, P., 121
 Nievergelt, J., 225
 Nikulin, A.E., 68
 Norton, H., 343
- O**
 Oates, T., 306
 O’Callaghan, L., 309
 Odell, P.L., 101
 Ogilvie, J.C., 339
 Oh, K.W., 338
 Ord, J.K., 24
 Ordonez, C., 311
 Orlóci, L., 78
 Ostrovsky, R., 309
 Overall, J.E., 46
 Özsoyoglu, M., 94
 Ozsoyoglu, Z.M., 93
 Özyurt, I.B., 195
- P**
 Pal, N.R., 167
 Pal, S.K., 194
 Palmer, R.G., 61
 Pandya, A.S., 389, 392
 Panigrahy, R., 311
 Papadopoulos, D., 303
 Parhankangas, J., 363
 Park, H., 344
 Park, J.S., 76, 261
 Park, N.H., 230
 Park, S., 98
 Parker, D.S., 96
 Parsons, L., 303
 Paterson, M.S., 308
 Patterson, C.L., 50

- Payne, T.R., 90
Pazzani, M., 88, 254
Pazzani, M.J., 92
Pei, J., 303, 336, 347
Pelleg, D., 175, 179, 282
Peña, J.M., 172
Perng, C., 67, 96
Phillips, S.J., 171, 173
Piatko, C.D., 176
Pizzi, N.J., 68
Plaxton, C.G., 309
Podani, J., 105
Pollard, D., 172
Pölzlauer, G., 68
Posse, C., 157
Preparata, F.P., 397
Prim, R.C., 148
Procopiuc, C., 76, 261
Procopiuc, C.M., 274
Puttagunta, V., 306
- Q**
Qu, Y., 98
Quinlan, J.R., 277
- R**
R.J. Rummel, 74
Rabani, Y., 309
Rafiei, D., 92, 97
Rafsky, L.C., 121
Raftery, A.E., 5, 25, 179, 240, 242, 249, 250, 255
Raghavan, P., 152, 217, 257, 259, 264, 271, 307, 339
Rajagopalan, S., 307, 308
Rajan, K., 68
Ramakrishnan, R., 34, 99, 151, 216, 257, 264, 267, 277, 308, 311
Ramanathan, M., 347
Ramoní, M., 89, 306
Ramsay, G., 343
Rand, W.M., 11, 330
Ranganathan, M., 97
Rao, H., 389, 392
Rao, K.S., 190
Rao, V., 389, 392
- Rao, V.R., 112
Rastogi, R., 34, 99, 152, 219, 271
Ratanamahatana, C.A., 93
Rauber, A., 68
Ravi, T.V., 24
Ray, S., 40
Reina, C., 311
Renganathan, G., 89
Reymond, P., 348
Rhee, H.S., 338
Rivest, R.L., 94
Robert, C.P., 255
Rogers, D.J., 81, 84, 220
Rohlf, F.J., 124, 157
Rohlf, R.J., 118
Rose, K., 182, 183, 292, 293
Ross, G.J.S., 118, 148
Roth, F.P., 346
Rousseeuw, P.J., 11, 80, 106, 119, 121, 146, 153, 261, 363
Rubin, D.B., 248
Rubin, J., 11, 41
Rundensteiner, E.A., 63
Runkler, T., 167
Ruspini, E.H., 159
Ruzzo, W.L., 25, 50, 255
- S**
Sabin, M.J., 167
Salton, G., 98
Sammon, J.W., Jr., 55
Sander, J., 231, 233, 234
Sarafis, I.A., 303
Saunders, J.A., 4
Sawhney, H.S., 93
Scheibler, D., 106
Schena, M., 343
Scheunders, P., 56, 57
Schikuta, E., 224, 226
Schlag, M., 220
Schneider, W., 106
Schuchhardt, J., 344
Schwarz, G., 179
Scott, A.J., 41, 156, 239
Sebastiani, P., 89, 306
Sedgewick, R., 391

- Selim, S.Z., 170, 214
Serinko, R.J., 172
Seshadri, S., 308
Sevcik, K.C., 225
Shadbolt, J., 25
Shafer, J.C., 277
Shaft, U., 257
Shahabi, C., 88
Shalon, D., 343
Shamir, R., 347
Shamos, M.I., 397
Sharan, R., 347
Sharma, S.C., 328
Sheikholeslami, G., 229
Shepard, R.N., 57
Sherlock, G., 344
Shim, K., 34, 93, 99, 152, 219, 271, 308
Shimshoni, I., 183
Shneiderman, B., 64
Sibson, R., 11, 117, 119, 120, 146, 150,
 220, 315
Silverman, R., 176
Simon, I., 306
Singh, A., 50, 97
Skármata, A.G., 196
Slagle, J.R., 12
Slonim, D.K., 344, 345
Smyth, P., 89, 95
Sneath, P.H.A., 6, 83, 104, 112, 121,
 124, 333, 339
Sokal, R.R., 6, 83, 112, 121
Somogyi, R., 348
Somorjai, R.L., 68
Song, J., 227
Spangler, W.S., 62
Späth, H., 145, 154
Spellman, P.T., 3
Spielmat, D.A., 220
Sprenger, T.C., 68
Srihari, S.N., 71
Stein, C., 94
Steinhaus, H., 124
Steinmetz, L., 348
Stewart, P.M., 214
Stoffel, K., 305
Stokes, L., 308
Strauss, M., 308
Struyf, A., 363
Stute, W., 177
Su, M., 177
Su, Z., 232
Sugeno, M., 335
Sung, C.S., 196
Sung, S.Y., 312, 316
Suzuki, E., 303
Swami, A.N., 90
Sycara, K.P., 90
Symons, M.J., 41, 156, 239
Szegedy, M., 308
- T**
- Tamayo, P., 345
Tamma, V., 23
Tamura, S., 162
Tanaka, K., 162
Tang, C., 3, 343, 347
Tarjan, R., 261
Tarsitano, A., 172
Tavazoie, S., 344, 348
Taylor, J.G., 25
Teng, S., 220
Thaper, N., 308
Theodoridis, S., 317
Thijs, G., 347
Thomasian, A., 51
Thorup, M., 309
Tibshirani, R., 42, 344
Tomasi, C., 183
Trauwaert, E., 334
Treleaven, P.C., 194
Trinder, P.W., 303
Troyanskaya, O., 344
Truppel, W., 305
Tsay, R.S., 25
Tseng, L.Y., 195
Tsujimura, Y., 195
Tsumoto, S., 89
Tubbs, J.D., 82
Tucker, W., 167
Tucker-Kellogg, G., 344
Turi, R.H., 40

U

Ullman, J., 261

V

van Dyk, D., 248

van Groenewoud, H., 125

van Rijsbergen, C.J., 118

Vanharanta, H., 61

Varadarajan, K., 303

Vazirgiannis, M., 40, 317, 326, 328, 334

Vempala, S., 51, 220, 308

Verbeek, J.J., 206

Vesanto, J., 68, 363

Vetta, A., 220

Vilo, J.M., 345

Vinay, V., 51, 308

Visa, A., 61

Viswanathan, M., 308

Vlachos, M., 93

Vlassis, N., 206

W

Walther, G., 42

Wang, C., 98, 343

Wang, H., 96, 196, 303, 347

Wang, J., 311

Wang, K., 23, 312

Wang, L., 237

Wang, W., 222, 303, 347

Wang, X.S., 98

Wang, Z., 237

Ward, J.H., 104

Ward Jr., J.H., 140, 242

Ward, M.O., 63

Warde, W.D., 102

Watson, L.T., 195

Wawryniuk, M., 303

Weber, H., 348

Wegman, E.J., 63

Weinman, J., 4

Weiss, Y., 220

Wen, X., 348

Whitley, M.Z., 344

Whittaker, R.H., 79

Wilks, S.S., 73

Willett, P., 145, 146

Williams, W.T., 100, 102, 138, 145, 157

Wills, G.J., 64

Wimmer, M., 233

Winzeler, E.A., 348

Wirth, M., 220

Wishart, D., 104, 140, 284

Wodicka, L., 348

Wolf, J.L., 76, 261

Wolfe, J.H., 239

Wolfsberg, T.G., 348

Wolski, E., 344

Wong, J.C., 199

Wong, K., 194

Wong, M.A., 34, 169

Wong, M.H., 93

Woo, K., 270

Wotring, J., 4

Wu, A.Y., 176

Wu, C.F.J., 249, 347

Wu, J., 188, 195, 207, 278, 287

Wu, K.L., 335, 337

Wu, L., 90

Wu, X., 12

Wu, Y., 97

Wunsch II, D., 13

X

Xia, Y., 276

Xiao, Y., 23, 98, 312, 315

Xie, X., 159, 336

Xiong, H., 312, 316

Xiong, Y., 306

Xu, C., 23, 312

Xu, H., 196

Xu, J., 312, 316

Xu, R., 13

Xu, X., 231–234

Y

Yaguchi, H., 86

Yamaguchi, Y., 68

Yang, C., 183

Yang, J., 63, 222, 303, 347

Yang, K., 88

Yang, M.S., 164, 335, 337

Yang, Q., 232

- Yang, S.B., 195
Yang, X., 336
Yang, Y., 23, 312
Yang, Z., 188, 195, 207
Yao, S., 220
Yazdani, N., 93, 94
Yeung, D., 306
Yeung, K.Y., 3, 25, 50, 255
Yi, B., 90
Yoon, J., 98
Yooseph, S., 347
You, J., 23, 312
Yu, P., 303
Yu, P.S., 76, 261, 264, 276, 303, 311, 347
Yun, C., 312, 316
- Z
Zadeh, L.A., 159, 160, 162
Zahn, C.T., 220
Zaiane, O.R., 233
Zait, M., 11
- Zalzala, A.M.S., 303
Zaslavsky, A., 307
Zeng, G., 333
Zhang, A., 3, 229, 343, 347
Zhang, B., 71, 180
Zhang, H., 232
Zhang, L., 347
Zhang, M., 306
Zhang, S.R., 96
Zhang, T., 34, 151, 264, 267, 311
Zhang, W., 195
Zhang, Y., 268, 339
Zhao, L., 195
Zhao, Y., 227
Zhong, S., 306
- Zhou, L., 230
Zhu, L.X., 177
Zien, J.Y., 220
Zubrzycki, S., 124
Zweig, G., 312