

Paolo Giordani  
Maria Brigida Ferraro  
Francesca Martella

# An Introduction to Clustering with R

# **Behaviormetrics: Quantitative Approaches to Human Behavior**

Volume 1

## **Series Editor**

Akinori Okada, Professor Emeritus, Rikkyo University, Tokyo, Japan

This series covers in their entirety the elements of behaviormetrics, a term that encompasses all quantitative approaches of research to disclose and understand human behavior in the broadest sense. The term includes the concept, theory, model, algorithm, method, and application of quantitative approaches from theoretical or conceptual studies to empirical or practical application studies to comprehend human behavior. The Behaviormetrics series deals with a wide range of topics of data analysis and of developing new models, algorithms, and methods to analyze these data.

The characteristics featured in the series have four aspects. The first is the variety of the methods utilized in data analysis and a newly developed method that includes not only standard or general statistical methods or psychometric methods traditionally used in data analysis, but also includes cluster analysis, multidimensional scaling, machine learning, corresponding analysis, biplot, network analysis and graph theory, conjoint measurement, biclustering, visualization, and data and web mining. The second aspect is the variety of types of data including ranking, categorical, preference, functional, angle, contextual, nominal, multi-mode multi-way, contextual, continuous, discrete, high-dimensional, and sparse data. The third comprises the varied procedures by which the data are collected: by survey, experiment, sensor devices, and purchase records, and other means. The fourth aspect of the Behaviormetrics series is the diversity of fields from which the data are derived, including marketing and consumer behavior, sociology, psychology, education, archaeology, medicine, economics, political and policy science, cognitive science, public administration, pharmacy, engineering, urban planning, agriculture and forestry science, and brain science.

In essence, the purpose of this series is to describe the new horizons opening up in behaviormetrics — approaches to understanding and disclosing human behaviors both in the analyses of diverse data by a wide range of methods and in the development of new methods to analyze these data.

### **Editor in Chief**

Akinori Okada (Rikkyo University)

### **Managing Editors**

Daniel Baier (University of Bayreuth)  
Giuseppe Bove (Roma Tre University)  
Takahiro Hoshino (Keio University)

More information about this series at <http://www.springer.com/series/16001>

Paolo Giordani · Maria Brigida Ferraro ·  
Francesca Martella

# An Introduction to Clustering with R



Springer

Paolo Giordani  
Dipartimento di Scienze Statistiche  
Sapienza Università di Roma  
Rome, Italy

Maria Brigida Ferraro  
Dipartimento di Scienze Statistiche  
Sapienza Università di Roma  
Rome, Italy

Francesca Martella  
Dipartimento di Scienze Statistiche  
Sapienza Università di Roma  
Rome, Italy

ISSN 2524-4027                   ISSN 2524-4035 (electronic)  
Behaviormetrics: Quantitative Approaches to Human Behavior  
ISBN 978-981-13-0552-8       ISBN 978-981-13-0553-5 (eBook)  
<https://doi.org/10.1007/978-981-13-0553-5>

© Springer Nature Singapore Pte Ltd. 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.  
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721,  
Singapore

*To the invisibles*

# Preface

“Yet another book on clustering” or “Yet another book on R”. These may be the first reactions when reading the title of this book. However, it presents several distinctive features with respect to the huge number of monographs devoted to these topics. In fact, the aim is to provide a practical guide to clustering through real-life examples and case studies by means of the open-source statistical software R, which is gradually introduced with a detailed explanation about the code. In contrast with books that are mainly involved in software, this book introduces the most common clustering methods from a theoretical point of view. In contrast with purely theoretical books on cluster analysis, this book carefully describes applications in practice by considering a wide variety of real-life examples from different fields of research to witness the practical effectiveness of clustering tools.

Such a mix between theoretical and practical aspects is motivated by the fact that clustering methods should not be considered as “black-box” tools producing an output given some input. In principle, clustering can be performed by running functions using default options and, in general, the obtained results may also be meaningful. For this reason, along the book, we will pay particular attention to the comprehension and interpretation of the results found by setting default options. However, results can be enhanced by introducing suitable modifications to such default options. To this purpose, clustering methods are briefly described from a theoretical point of view to help readers extract the most valuable output from the input data.

The book is primarily designed for practitioners who aim at analyzing data by using R for clustering purposes. Also, it can be used as a textbook for a (application-oriented) course on clustering. The prerequisites for reading this book do not go beyond some basic knowledge of matrices, statistics, and probability. Also, the reader should have some familiarity with computer programming in R.

Apart from a brief introduction to clustering, the book is divided into three main parts: standard, fuzzy, and model-based clustering methods. The techniques presented in the first part are usually referred to as hard, in contrast with the up-to-date soft clustering techniques presented in the remaining two parts.

The Part I on standard procedures focuses on hierarchical and non-hierarchical algorithms. It starts by recalling distance measures for quantitative (continuous or discrete), categorical, or mixed data. With hierarchical methods, attention is not only limited to the most common cause of quantitative data, but also to categorical and mixed data, since such cases frequently occur in practice. The chapter on non-hierarchical methods first involves the well-known  $k$ -Means algorithm and later moves to some extensions such as the  $k$ -Medoids algorithm and the divisive  $k$ -Means one, a compromise between hierarchical and non-hierarchical methods. Finally, an introductory discussion on clustering big data is reported.

The Part II of the book focuses on the fuzzy approach to clustering where cluster memberships are expressed in terms of fuzzy degrees with values between 0 and 1. Such a class of methods is non-hierarchical. As such, the fuzzy extensions of the  $k$ -Means and  $k$ -Medoids algorithms are illustrated and discussed. Furthermore, some alternative fuzzy clustering algorithms are investigated, such as robust variants to handle noisy data. All of these methods are especially tailored for quantitative data; nevertheless, categorical or mixed data can be analyzed by means of methods for relational data, provided that suitable distance measures are used.

In Part III, the field of model-based clustering is introduced. Differently from the previous ones, a probabilistic data generating process is assumed, as data are assumed to come from a finite mixture of distributions. The most common situation involves Gaussian distributions (continuous data). Attention is also paid to categorical or mixed data, where either distributions for such kinds of data, e.g., Multinomial, are used or observed variables are assumed to arise from underlying continuous latent variables following a Gaussian mixture model. Variants of model-based clustering methods for noisy data, based on mixtures of  $t$  distributions, skew, and high-dimensional data are also illustrated.

The website <https://www.dss.uniroma1.it/dipartimento/persone/giordani-paolo> contains the R script used in the book. All the datasets analyzed in the book are available in R packages. Most of them are included in the add-on package **datasetsICR** accompanying this book.

We are indebted to many people. We would like to thank Akinori Okada and Giuseppe Bove for stimulating us to write this book. We are thankful to all the scientists we shared time with to discuss clustering or R. These discussions, in our department as well as attending conferences, are the foundation of this book. We are most grateful to Marco Alfó who read the first draft of this book and provided very helpful feedback.

Rome, Italy  
May 2020

Paolo Giordani  
Maria Brigida Ferraro  
Francesca Martella

# Contents

## Part I Introduction

<b>1</b>	<b>Introduction to Clustering</b>	<b>3</b>
1.1	Basic Concepts	3
References		5

## Part II Standard Clustering

<b>2</b>	<b>Hierarchical Clustering</b>	<b>9</b>
2.1	Introduction	9
2.2	Distance Measures	9
2.3	Agglomerative Hierarchical Clustering	14
2.4	Divisive Hierarchical Clustering	17
2.5	<i>agnes</i> and <i>hclust</i>	18
2.5.1	Case Study with Quantitative Data	18
2.5.2	Case Studies with Mixed Data	33
2.5.3	One Further Case Study	44
2.6	Functions for Divisive Clustering	65
2.6.1	<i>diana</i>	65
2.6.2	<i>mona</i>	69
References		72
<b>3</b>	<b>Non-Hierarchical Clustering</b>	<b>75</b>
3.1	Introduction	75
3.2	<i>k</i> -Means	78
3.3	<i>k</i> -Medoids	80
3.4	<i>kmeans</i>	81
3.4.1	Alternative Functions	94

3.5	pam . . . . .	96
3.5.1	Plotting Cluster Solutions . . . . .	100
3.5.2	clara . . . . .	102
3.5.3	Alternative Functions . . . . .	105
3.6	Divisive $k$ -Means . . . . .	105
	References . . . . .	109
4	<b>Big Data and Clustering</b> . . . . .	111
4.1	Standard Methods . . . . .	111
	References . . . . .	120

### Part III Fuzzy Clustering

5	<b>Fuzzy Clustering</b> . . . . .	125
5.1	Introduction . . . . .	125
5.2	Fuzzy $k$ -Means . . . . .	126
5.2.1	Cluster Validity Indices . . . . .	128
5.2.2	FKM . . . . .	130
5.2.3	cmeans . . . . .	136
5.2.4	fcm . . . . .	138
5.2.5	fuzzy.CM . . . . .	140
5.3	Gustafson-Kessel Extensions of Fuzzy $k$ -Means . . . . .	143
5.3.1	FKM.gk . . . . .	145
5.3.2	FKM.gkb and fuzzy.GK . . . . .	152
5.4	Entropic Fuzzy $k$ -Means . . . . .	154
5.4.1	FKM.ent . . . . .	155
5.5	Fuzzy $k$ -Means with Polynomial Fuzzifier . . . . .	158
5.5.1	FKM.pf . . . . .	160
5.6	Fuzzy $k$ -Medoids . . . . .	164
5.6.1	FKM.med . . . . .	166
5.7	Fuzzy Clustering for Relational Data . . . . .	173
5.7.1	fanny . . . . .	176
5.7.2	NEFRC . . . . .	182
5.8	Fuzzy $k$ -Means with Noise Cluster . . . . .	189
5.8.1	FKM.noise . . . . .	190
5.9	Possibilistic $k$ -Means . . . . .	194
5.9.1	pcm . . . . .	195
5.10	Hybrid (Fuzzy/Possibilistic) Clustering Methods . . . . .	198
5.10.1	fpcm, mfpcm and pfcm . . . . .	201
	References . . . . .	208

**Part IV Model-Based Clustering**

<b>6 Model-Based Clustering</b> .....	215
6.1 Introduction .....	215
6.2 Finite Mixture Models .....	216
6.3 Parameter Estimation .....	217
6.3.1 EM Algorithm for Finite Mixture Models .....	218
6.3.2 EM-Type Algorithms .....	219
6.3.3 Initializing the EM Algorithm .....	221
6.3.4 Stopping Criteria for the EM Algorithm .....	222
6.4 On Identifiability of Finite Mixture Models .....	223
6.5 Model Selection .....	224
6.6 Gaussian Mixture Models for Continuous Data .....	228
6.7 Mixture Models for Categorical and Mixed Data .....	231
6.7.1 Latent Class Analysis Model .....	232
6.7.2 ClustMD Model .....	233
6.8 <b>mclust</b> , <b>Rmixmod</b> , and <b>clustMD</b> .....	235
6.8.1 Case Study with Continuous Data .....	236
6.8.2 Case Studies with Categorical and Mixed Data .....	256
References .....	285
<b>7 Issues in Gaussian Model-Based Clustering</b> .....	291
7.1 Introduction .....	291
7.2 High Dimensionality in Gaussian Mixture Models .....	291
7.2.1 Regularization and Tandem Analysis Strategies .....	292
7.2.2 Parsimonious Gaussian Mixture Models .....	292
7.2.3 pgmm .....	295
7.3 Departures from Gaussian Mixture Models .....	305
7.3.1 Mixtures of $t$ Distributions .....	306
7.3.2 Mixtures of Skew-Normal Distributions .....	307
7.3.3 Handling Kurtosis and Skewness via Finite Mixture Models .....	308
7.3.4 teigen and mixsmsn .....	309
7.4 Dealing with Outliers .....	324
7.5 Cluster or Component? .....	332
References .....	335

# Acronyms

AECM	Alternating Expectation Conditional Maximization
AIC	Akaike Information Criterion
ARI	Adjusted Rand Index
$B$	Between-cluster sum of squares
BIC	Bayesian Information Criterion
CEM	Classification Expectation-Maximization
DIANA	DIvisive ANAlysis clustering
DM	Deutsch Mark
dof	degrees of freedom
ECM	Expectation Conditional Maximization
ECME	Expectation Conditional Maximization Either
EF $k$ M	Entropic Fuzzy $k$ -Means
EM	Expectation-Maximization
EPGMM	Expanded Parsimonious Gaussian Mixture Model
FMM	Finite Mixture Model
F $k$ M	Fuzzy $k$ -Means
F $k$ Med	Fuzzy $k$ -Medoids
F $k$ MN	Fuzzy $k$ -Means with Noise cluster
F $k$ MPF	Fuzzy $k$ -Means with Polynomial Fuzzifier
FMSMSN	Finite Mixture of Scale Mixtures of Skew-Normals
FP $k$ M	Fuzzy Possibilistic $k$ -Means
FS	Fuzzy Silhouette
PF $k$ M	Possibilistic Fuzzy $k$ -Means
GEM	Generalized Expectation Maximization
GK-F $k$ M	Gustafson-Kessel Fuzzy $k$ -Means
GMM	Gaussian Mixture Model
MONA	MONothetic Analysis clustering of binary variables
HD-GMM	High-dimensional Gaussian Mixture Model
HMFA	Heteroscedastic Mixture Factor Analyzers
ICL	Integrated Completed Likelihood

LASSO	Least Absolute Shrinkage and Selection Operator
LCA	Latent Class Analysis
LRT	Likelihood Ratio Test
MAP	Maximum A Posteriori
MBHAC	Model-Based Hierarchical Agglomerative Clustering
MCA	Multiple Correspondence Analysis
MCFA	Mixture of Common Factor Analyzers
MCGFA	Mixture of Contaminated Gaussian Factor Analyzers
MCUFSAs	Mixture of Common Uncorrelated Factor Spherical-error Analyzers
MCEM	Monte Carlo Expectation Maximization
MFA	Mixtures of Factor Analyzers
MFPkM	Modified Fuzzy Possibilistic $k$ -Means
ML	Maximum Likelihood
MLE	Maximum Likelihood Estimate
MVN	MultiVariate Normal
MPC	Modified Partition Coefficient
MPPCA	Mixture of Probabilistic Principal Component Analyzers
NEC	Normalized Entropy Criterion
NEFRC	Non-Euclidean Fuzzy Relational Clustering
NNVE	Nearest Neighbor Variance Estimation
PAM	Partitioning Around Medoids
PCA	Principal Component Analysis
PC	Partition Coefficient
PCs	Principal Components
PE	Partition Entropy
PGMM	Parsimonious Gaussian Mixture Models
PkM	Possibilistic $k$ -Means
SEM	Stochastic Expectation Maximization
S	Silhouette
T	Total sum of squares
URV	Underlying Response Variable
W	Within-cluster sum of squares
XB	Xie and Beni index

## Packages

<b>advelust</b>	Object Oriented Advanced Clustering
<b>amap</b>	Another Multidimensional Analysis Package
<b>ape</b>	Analyses of Phylogenetics and Evolution
<b>BayesLCA</b>	Bayesian Latent Class Analysis
<b>bgmm</b>	Gaussian Mixture Modeling Algorithms and the Belief-Based Mixture Modeling

<b>cluster</b>	“Finding Groups in Data”: Cluster Analysis Extended Rousseeuw et al.
<b>ClusterR</b>	Gaussian Mixture Models, K-Means, Mini-Batch-Kmeans, K-Medoids and Affinity Propagation Clustering
<b>clustMD</b>	Model Based Clustering for Mixed Data
<b>clValid</b>	Validation of Clustering Results
<b>ContaminatedMixt</b>	Clustering and Classification with the Contaminated Normal
<b>covRobust</b>	Robust Covariance Estimation via Nearest Neighbor Cleaning
<b>datasets</b>	The R Datasets Package
<b>datasetsICR</b>	Datasets from the Book “An Introduction to Clustering with R”
<b>dclust</b>	Divisive Hierarchical Clustering
<b>dendextend</b>	Extending ‘dendrogram’ Functionality in R
<b>distances</b>	Tools for Distance Metrics
<b>e1071</b>	Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien
<b>EMCluster</b>	EM Algorithm for Model-Based Clustering of Finite Mixture Gaussian Distribution
<b>EMMIXmfa</b>	Mixture Models with Component-Wise Factor Analyzers
<b>EMMIXskew</b>	The EM Algorithm and Skew Mixture Distribution
<b>factoextra</b>	Extract and Visualize the Results of Multivariate Data Analyses
<b>fclust</b>	Fuzzy Clustering
<b>flexCWM</b>	Flexible Cluster-Weighted Modeling
<b>flexmix</b>	Flexible Mixture Modeling
<b>fpc</b>	Flexible Procedures for Clustering
<b>ggplot2</b>	Create Elegant Data Visualisations Using the Grammar of Graphics
<b>gridExtra</b>	Miscellaneous Functions for “Grid” Graphics
<b>lattice</b>	Trellis Graphics for R
<b>mcgfa</b>	Mixtures of Contaminated Gaussian Factor Analyzers
<b>MixAll</b>	Clustering and Classification using Model-Based Mixture Models
<b>MixGHD</b>	Model Based Clustering, Classification and Discriminant Analysis Using the Mixture of Generalized Hyperbolic Distributions
<b>MixSAL</b>	Mixtures of Multivariate Shifted Asymmetric Laplace (SAL) Distributions
<b>mixSPE</b>	Mixtures of Power Exponential and Skew Power Exponential Distributions for Use in Model-Based Clustering and Classification
<b>mixsmsn</b>	Fitting Finite Mixture of Scale Mixture of Skew-Normal Distributions
<b>mixtools</b>	Tools for Analyzing Finite Mixture Models

<b>mixture</b>	Finite Gaussian Mixture Models for Clustering and Classification
<b>mclust</b>	Gaussian Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation
<b>NbClust</b>	Determining the Best Number of Clusters in a Data Set
<b>parallel</b>	Support for Parallel computation in R
<b>PCAmixdata</b>	Multivariate Analysis of Mixed Data
<b>pgmm</b>	Parsimonious Gaussian Mixture Models
<b>philentropy</b>	Similarity and Distance Quantification Between Probability Functions
<b>poLCA</b>	Polytomous variable Latent Class Analysis
<b>ppclust</b>	Probabilistic and Possibilistic Cluster Analysis
<b>prabclus</b>	Functions for Clustering and Testing of Presence-Absence, Abundance and Multilocus Genetic Data
<b>RcppArmadillo</b>	‘Rcpp’ Integration for the ‘Armadillo’ Templated Linear Algebra Library
<b>Rmixmod</b>	Classification with Mixture Modelling
<b>smacof</b>	Multidimensional Scaling
<b>StatMatch</b>	Statistical Matching or Data Fusion
<b>stats</b>	The R Stats Package
<b>tclust</b>	Robust Trimmed Clustering
<b>teigen</b>	Model-Based Clustering and Classification with the Multivariate t Distribution
<b>usmap</b>	US Maps Including Alaska and Hawaii

## Datasets

banknote	(package <b>mclust</b> )
birds	(package <b>Rmixmod</b> )
butterfly	(package <b>datasetsICR</b> )
customers	(package <b>datasetsICR</b> )
Economics	(package <b>datasetsICR</b> )
Eurostat	(package <b>datasetsICR</b> )
FaceExp	(package <b>smacof</b> )
FaceScale	(package <b>smacof</b> )
faithful	(package <b>mclust</b> )
FIFA	(package <b>datasetsICR</b> )
flags	(package <b>datasetsICR</b> )
german	(package <b>datasetsICR</b> )
houseVotes	(package <b>fclust</b> )
iris	(package <b>datasets</b> )
gironde	(package <b>PCAmixdata</b> )
lasvegas.trip	(package <b>datasetsICR</b> )

mtcars	(package <b>datasets</b> )
NBA.48	(package <b>datasetsICR</b> )
NBA.efficiency	(package <b>datasetsICR</b> )
NBA.external	(package <b>datasetsICR</b> )
NBA.game	(package <b>datasetsICR</b> )
seeds	(package <b>datasetsICR</b> )
synt.data2	(package <b>fclust</b> )
USstate	(package <b>datasetsICR</b> )
wiki4HE	(package <b>datasetsICR</b> )
wine	(package <b>datasetsICR</b> )
x12	(package <b>ppclust</b> )

# **Part I**

## **Introduction**

# Chapter 1

## Introduction to Clustering



### 1.1 Basic Concepts

Clustering or cluster analysis [1–4] comprises a large class of methods aiming at discovering a limited number of *meaningful* groups (or clusters) in data. In particular, given a dataset with  $n$  units on which  $p$  variables are observed, the traditional goal of clustering is to identify  $k$  ( $<< n$ ) groups such that

1. Every group must contain at least one unit;
2. Units belong to one and only one group, i.e., the groups are disjoint.

This defines a *partition* of  $n$  units in  $k$  clusters. Since a huge number of partitions can be found, it remains to explain the meaning of the adjective *meaningful*. A group is meaningful if units in that group are similar to one another and different from units in other groups. The concept of similarity or dissimilarity among units is easy to understand and well acceptable: all of us are able to assess whether two units are similar or different. Nowadays, there is an increasing interest in clustering [5], and the need for finding groups in data is recognized in several fields of research, where clustering is widely applied. For example, in medicine, data can refer to variables on health status, diet, and habits observed on a set of patients. Clustering methods can discover groups of patients corresponding to different causes or levels of severity for a given disease allowing for tailoring of treatments. In genomics, when gene expression levels are measured on various experimental conditions (e.g., tissue samples, cellular states, or time points), clustering genes may offer a deeper understanding of the molecular variations allowing for elucidating genetic networks or some important biological processes. In marketing, suppose that a company collected information on its customers with respect to socio-demographic variables and past purchases. By clustering customers, the company aims at discovering buying behaviors to deploy targeted marketing campaigns. In this way, the company may maximize the probability that customers will purchase its products. Survey data can be fruitfully analyzed by clustering methods. Suppose that some college students are invited to answer questions of a web survey to gather students' feedback. Partitioning the students helps college managers making decisions to enhance students' learning experience. In economics, we may think about indicators on employment, social, and market

conditions, entrepreneurial capabilities observed in a set of municipalities. The use of clustering methods is fundamental for stakeholders to compare municipalities in order to target funds to those municipalities where they are more needed.

Although the scope of cluster analysis is very intuitive, its formalization is not straightforward. In fact, cluster analysis poses problems and challenges the user must deal with. A deeper discussion can be found in [6]. Here, we mention some of them. The first one is that there is no a priori knowledge of the existence and the number of groups. In fact, the clusters are rather naturally suggested by the data following the implicit assumption that some kind of homogeneity and separation is present in the dataset. As such, the clustering task presented in this book is *unsupervised* to distinguish it from a closely related large class of methods, known as supervised clustering or classification, where the groups are known a priori and the research problem is to predict the cluster membership of new units based on the past ones. Specifically, for the  $n$  units, a target variable identifying the class (we avoid the terms cluster or group) membership is available, and the other variables are used to build a rule for classifying units to the proper class. This rule is then used for classifying units for which all but the target variables are observed. See, for an overview, [7].

Another relevant issue involves how to assess the degrees of similarity or dissimilarity among units since it may strongly affect the obtained partition. This is connected with at least three points. The first one is the choice of the variables to be considered for clustering purposes. The second involves the preprocessing step, i.e., the possible standardization/normalization of the variables so that they can play the same role in the clustering task. The last point refers to the specification of the adopted measures for assessing the similarity or dissimilarity among units since an impressive number of alternatives are available. There are no golden rules for these choices because they depend on the specific research problem for the data under investigation.

Finally, one more relevant choice concerns the clustering method(s) to apply. As it occurs for the measures of similarity or dissimilarity, different clustering methods generate different partitions. Generally speaking, several cluster analyses can be conducted to see what results emerge and to assess their agreement.

The most famous clustering techniques according to the standard, fuzzy, and model-based approaches will be introduced in this book. Clustering methods belonging to the standard approach can be split into hierarchical and non-hierarchical methods. Non-hierarchical methods discover the *best* partition of  $n$  units in  $k$  clusters according to the minimization of a given cost function; hence, only one partition is determined. Hierarchical methods produce a series of partitions where, at each step, either two clusters are merged (*agglomerative* hierarchical clustering) or one cluster is divided into two clusters (*divisive* hierarchical clustering) according to a given criterion. This leads to a hierarchy of partitions represented by a (hierarchical) tree. Both hierarchical and non-hierarchical clustering methods are also referred to as *hard* clustering in the sense that each unit can be either assigned or not to a cluster. In contrast, fuzzy and model-based clustering methods are usually said to be *soft* because they relax the assignment of the units to clusters. More specifically, units belong to clusters according to the so-called (fuzzy) membership degrees, in

**Table 1.1** Differences among standard, fuzzy, and model-based approaches to clustering

Feature	Standard approach	Fuzzy approach	Model-based approach
Hard clustering	Yes	No	No
Soft clustering	No	Yes	Yes
Probabilistic assumptions	No	No	Yes

the fuzzy framework, or posterior probabilities, in the model-based framework. Both membership degrees and posterior probabilities take values in the interval  $[0, 1]$  and, for each unit, their sum over all clusters is equal to one. Membership degrees or posterior probabilities close to one identify units that are strongly assigned to the clusters. The main distinction between fuzzy and model-based approaches relies on the fact that the former identifies clusters by minimizing a cost function, while the latter by introducing probabilistic assumptions on the data. A rough summary of the differences among the three approaches is reported in Table 1.1.

Last but not least we observe that, obviously, the availability of software is a fundamental requirement for applying clustering methods. As we shall see, the open-source statistical software R [8] is rich in tools for performing cluster analysis as witnessed by the webpage <https://cran.r-project.org/web/views/Cluster.html>. The interested reader is invited to visit it frequently for up-to-date information on (new) R packages devoted to clustering.

## References

1. Everitt, B.S., Landau, S., Leese, M., Stahl, D.: *Cluster Analysis*. Wiley, Chichester (2011)
2. Hennig, C., Meila, M., Murtagh, F., Rocci, R. (eds.): *Handbook of Cluster Analysis*. Chapman and Hall/CRC, Boca Raton (2018)
3. Kaufman, L., Rousseeuw, P.J.: *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York (1990)
4. Kłopotek, M.A., Wierzchoń, S.T.: *Modern Algorithms of Cluster Analysis*. Springer, Cham (2018)
5. Murtagh, F., Kurtz, M.J.: The classification society's bibliography over four decades: history and content analysis. *J. Classif.* **33**, 6–29 (2016)
6. Milligan, G.W.: Clustering validation: results and implications for applied analyses. In: Arabie, P., Hubert, L.J., Soete, G.D. (eds.) *Clustering and Classification*, pp. 341–375. World Scientific, Singapore (1996)
7. James, G., Witten, D., Hastie, T., Tibshirani, R.: *An Introduction to Statistical Learning: With Applications in R*. Springer, New York (2013)
8. R Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna (2020). <https://www.R-project.org>

## **Part II**

# **Standard Clustering**

# Chapter 2

## Hierarchical Clustering



### 2.1 Introduction

Standard clustering methods can be approximately distinguished into two main classes, namely, hierarchical and non-hierarchical clustering procedures. This chapter focuses on the first class. The peculiarity of hierarchical clustering methods is that they do not lead to a single partition with a given number of clusters. Rather, they produce a series of partitions obtained in different steps. As such, hierarchical clustering methods are more user-friendly than non-hierarchical clustering methods that will be discussed in Chap. 3.

Hierarchical clustering methods require a dissimilarity/distance matrix (henceforth denoted as distance matrix for consistency with R [30], although several implemented dissimilarities are not distances in a strict sense). When the available information takes the form of a distance matrix, we sometimes refer to as relational data to highlight that the available information comes from the relation, for example, the distance, between each pair of units. Distance matrices are not always obtained from a unit-by-variable dataset. For instance, suppose that a consumer is invited to express her/his subjective levels of dissimilarity between pairs of items. In this case, distances are not computed according to the features (variables) of the items, but are available according to the perceived relations between items. Nevertheless, the most common case is when some variables are collected on a set of units and a distance matrix is built accordingly. This case will usually represent our starting point in the following sections.

### 2.2 Distance Measures

Let  $\mathbf{X}$  be the data matrix of order  $(n \times p)$  containing the values of  $p$  variables observed on  $n$  units to be clustered:

$$\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1j} & \cdots & x_{1p} \\ \vdots & & \vdots & & \vdots \\ x_{i1} & \cdots & x_{ij} & \cdots & x_{ip} \\ \vdots & & \vdots & & \vdots \\ x_{n1} & \cdots & x_{nj} & \cdots & x_{np} \end{pmatrix}. \quad (2.1)$$

The generic element of the matrix is  $x_{ij}$  expressing the value of variable  $j$  ( $j = 1, \dots, p$ ) observed on unit  $i$  ( $= 1, \dots, n$ ). The rows of  $\mathbf{X}$  refer to the units. For each unit  $i$  we observe a vector of values of length  $p$  denoted by  $\mathbf{x}_i = (x_{i1}, \dots, x_{ij}, \dots, x_{ip})$ . The variables can be quantitative (continuous or discrete) or categorical (measured on a nominal/ordinal scale). We will start by considering quantitative variables. Many distance measures are implemented in several R functions. The default option of all such functions is the well-known Euclidean distance. If  $\mathbf{x}_i$  and  $\mathbf{x}_{i'}$  denote the vectors of observed variables for units  $i$  and  $i'$ , respectively,  $i, i' = 1, \dots, n$ , the Euclidean distance is formalized as

$$d(\mathbf{x}_i, \mathbf{x}_{i'}) = \sqrt{\sum_{j=1}^p (x_{ij} - x_{i'j})^2}. \quad (2.2)$$

For simplicity, the square root can be omitted and the squared Euclidean distance can be obtained. The Euclidean distance is a particular case of a famous class of distances, the Minkowski distance of order  $q$  defined as

$$d_M^q(\mathbf{x}_i, \mathbf{x}_{i'}) = \sqrt[q]{\sum_{j=1}^p |x_{ij} - x_{i'j}|^q}. \quad (2.3)$$

It contains several widely used distances as special cases. The Euclidean distance is found when  $q = 2$ . Setting  $q = 1$ , the so-called Manhattan distance can be derived.

The previously recalled distances are implemented in several functions. The most common are `dist` of the package **stats** and `daisy` of the package **cluster** [23]. Both functions require an object of class `matrix` or `data.frame` as input argument `x`. The most relevant distances for quantitative variables available in `dist` and `daisy` are summarized in Table 2.1.

The function `daisy` offers the possibility of standardizing the data before computing the distance matrix. It can be done by setting `stand = TRUE`. This is not the case for `dist`. If one is interested in computing the distance matrix on standardized data by using `dist`, a possible way is to set `x = stand(dataset)` where `dataset` is the object containing the data at hand.

In the functions above the Mahalanobis distance [22] is not implemented. It compares two units taking into account the covariance structure of the variables and is formulated as

**Table 2.1** Distance measures and functions for quantitative variables

Function	Package	Option	Distance
dist	stats	method = "euclidean"	Euclidean
		method = "minkowski"	Minkowski
		method = "manhattan"	Minkowski with $q = 1$
		method = "maximum"	Minkowski with $q = +\infty$
daisy	cluster	metric = "euclidean"	Euclidean
		metric = "manhattan"	Minkowski with $q = 1$

$$d_M(\mathbf{x}_i, \mathbf{x}_{i'}) = \sqrt{(\mathbf{x}_i - \mathbf{x}_{i'})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \mathbf{x}_{i'})}, \quad (2.4)$$

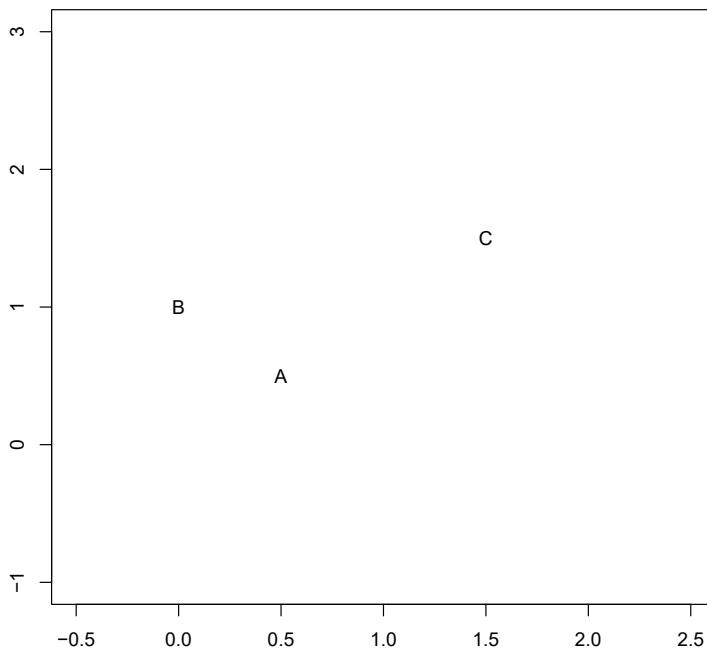
where  $\boldsymbol{\Sigma}$  denotes the covariance matrix of order  $(p \times p)$ . It is easy to show that the Mahalanobis distance reduces to the Euclidean one when  $\boldsymbol{\Sigma}$  is the identity matrix. The Mahalanobis distance can be calculated using several functions. The most common is probably the function `mahalanobis` of the package **stats**. The function computes the squared Mahalanobis distance between all the rows of a data matrix (option `x`) and the vector specified by the option `center` given the covariance matrix specified by the option `cov`. As such, it does not produce a distance matrix. To clarify it, let us consider the following artificial dataset with  $n = 3$  units and  $p = 2$  variables (see also Fig. 2.1).

```
> X <- matrix(c(0.5, 0.0, 1.5, 0.5, 1.0, 1.5),
+                 ncol = 2)
> plot(X, type = "n", xlim = c(-0.5, 2.5),
+       ylim = c(-1, 3), xlab = "", ylab = "")
> text(X[, 1], X[, 2], c("A", "B", "C"))
```

If the covariance matrix is equal to the identity matrix, the Mahalanobis distance matrix is equal to the Euclidean distance one. By using `dist`, we get the following output.

```
> dist(X, diag = TRUE)
      1           2           3
1 0.0000000
2 0.7071068 0.0000000
3 1.4142136 1.5811388 0.0000000
```

Instead, this output cannot be found by considering `mahalanobis`. For instance, setting `center = c(0.5, 0.5)` and `cov = diag(2)`, i.e., the identity matrix, we have the following.



**Fig. 2.1** X data: scatterplot

```
> mahalanobis(x = X, center = c(0.5, 0.5),
+               cov = diag(2))
[1] 0.0 0.5 2.0
```

Thus, the function `mahalanobis` is not designed to produce the distance matrix for all the units, but rather a vector of squared Mahalanobis distance values between units and the vector in `center`. In a sense, it produces the argument for the `exp` function in a multivariate Gaussian density with mean vector and covariance matrix specified in `center` and `cov`, respectively.

The computation of the Mahalanobis distance matrix can be obtained by the function `mahalanobis.dist` of the package **StatMatch** [5].

```
> library(StatMatch)
> mahalanobis.dist(data.x = X, vc = diag(2))
      [,1]      [,2]      [,3]
[1,] 0.0000000 0.7071068 1.414214
[2,] 0.7071068 0.0000000 1.581139
[3,] 1.4142136 1.5811388 0.000000
```

Since the covariance matrix coincides with the identity matrix, the same output as for the Euclidean case is found. Let us now compute the Mahalanobis distance matrix assuming a different covariance matrix taking into account the positive correlation between the two variables.

```
> Sigma <- matrix(c(0.3, 0.2, 0.2, 0.3), nrow = 2)
> mahalanobis.dist(data.x = x, vc = Sigma)
      [,1]      [,2]      [,3]
[1,] 0.000000 2.236068    2
[2,] 2.236068 0.000000    3
[3,] 2.000000 3.000000    0
```

An interesting result is that  $d(A, B) < d(A, C)$ , while  $d_M(A, B) > d_M(A, C)$ . This highlights that the chosen distance measures may substantially affect the clustering process. The study of the impact of different distance measures on cluster solutions is out of the scope of this book. The interested reader may refer to, e.g., [4, 24].

The package **distances** [31] contains the function **distances** that computes the Euclidean and Mahalanobis distances. The former is found by setting the option `normalize = NULL` or `normalize = "none"`. The latter can be obtained if `normalize = "mahalanobize"`. Note that any other positive semidefinite matrix can be specified in the option `normalize`. For the sake of completeness, we also mention the package **philentropy** [6] containing the function **distance** implementing a large number of distance and similarity measures.

All the abovementioned distance measures can be used when the variables are quantitative, i.e., by using the R terminology, of type `numeric` (continuous) or `integer` (discrete). If at least one variable is categorical, i.e., of type `factor`, alternative distance measures must be applied. This also holds for categorical variables entered into the computer using numeric codes that may be recognized of type `numeric` or `integer`. For instance, the variable `sex` has two labels, female and male, often coded by 0 and 1. It is recommended to convert the discrete quantitative variable to a factor by using the function `factor`.

A common choice is represented by the Gower distance [12] that represents a useful way to handle mixed data. The idea is to define a distance measure for each variable taking values in the interval  $[0, 1]$ .

For quantitative variables, one might distinguish between interval-scale and ratio-scale variables. The former follows a linear scale, it has no true zero and we can compute the difference between two values. The latter has a true zero and, in this case, the ratio between two values is meaningful. Temperature provides an example of an interval-scale variable. Suppose that the temperatures registered in two cities are 5 and 10°C. We can conclude that the difference in temperature is 5°C, but we cannot say that one is twice the other. In fact, if we convert the Celsius values to the Fahrenheit scale, the temperature is no longer twice the other. It also clarifies that a true zero point does not exist since 0°C is equivalent to 32°F. Height provides

an example of a ratio-scale variable. Obviously, 0 denotes a null measure. In this case, given two values equal to 10 and 5 cm, we can state that the former is twice the latter. Converting cm to in, we get 3.94 in and 1.97 in and the ratio is still equal to 2. In practice, the distinction between interval- and ratio-scale variables is often overlooked and, for the  $j$ -th quantitative variable, the distance between the values  $x_{ij}$  and  $x_{i'j}$  is computed as

$$\frac{|x_{ij} - x_{i'j}|}{\max_{i=1,\dots,n} x_{ij} - \min_{i=1,\dots,n} x_{ij}}. \quad (2.5)$$

With respect to categorical variables, in case of nominal categories, the simple matching criterion is applied. Namely, if two units present the same value, then the distance is equal to 0, otherwise it is set to 1. Obviously, this also holds for the binary case.

In case of ordinal variables, the concept of rank can be used. In particular, if the  $j$ -th ordinal variable has  $m_j$  levels (using the R terminology), the rank of unit  $i$ , say  $r_{ij}$ , is normalized as

$$\frac{r_{ij} - 1}{m_j - 1} \quad (2.6)$$

and (2.5) is then used.

The Gower distance is implemented in the function `daisy` of the package **cluster** [23] setting the option `metric = "gower"`. Note that the option `type` should be set to specify the types of the variables, i.e., when one is interested in distinguishing between interval- and ratio-scale variables.

## 2.3 Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering methods produce a series of partitions where the two most similar clusters are successively merged. In detail, given a distance matrix  $\mathbf{D}_n$  of order  $(n \times n)$ , they consist of the following steps. Note that, at the first step, the distance matrix provides the distance measures between  $n$  singleton clusters.

1. According to  $\mathbf{D}_n$ , merge the two units/clusters with the minimum distance in a new cluster. This leads to a new partition with  $n - 1$  clusters: the new cluster of size 2 and the remaining  $n - 2$  singleton clusters.
2. Compute the new distance matrix,  $\mathbf{D}_{n-1}$  of order  $((n - 1) \times (n - 1))$ . The distance measures between singleton clusters are inherited from the original matrix  $\mathbf{D}_n$ . There are several alternatives for computing the distances between the new cluster and the remaining ones. This choice distinguishes the existing agglomerative methods and it will be discussed below.
3. Merge together the clusters with minimum distance using  $\mathbf{D}_{n-1}$  leading to a partition with  $n - 2$  clusters.

4. Steps 2 and 3 are repeated until one cluster remains. In the last step,  $\mathbf{D}_2$  has order  $(2 \times 2)$  and contains the distance measures between two clusters that are merged together to obtain the final trivial partition with one cluster of  $n$  units.

The crucial point of an agglomerative clustering procedure relies in the method for computing distances between a cluster formed by the fusion of two clusters and the others. In general, different methods give different solutions. These methods can be defined according to the Lance-Williams formula [19]. Let  $C_1$  and  $C_2$  denote two clusters to be merged in cluster  $C_{1,2}$ , the distance between such a new cluster and cluster  $C_3$  can be expressed as follows:

$$d(C_{1,2}, C_3) = \alpha_1 d(C_1, C_3) + \alpha_2 d(C_2, C_3) + \beta d(C_1, C_2) + \gamma |d(C_1, C_3) - d(C_2, C_3)|. \quad (2.7)$$

Depending on the choice of the parameters  $\alpha_1, \alpha_2, \beta$ , and  $\gamma$ , the class of agglomerative methods is defined.

The functions `hclust` (Hierarchical Clustering) of the package `stats` and `agnes` (Agglomerative Nesting) of the package `cluster` are the two most popular tools implementing agglomerative hierarchical clustering. Both the functions offer the well-known single linkage method, average linkage method, complete linkage method, and Ward's method (see, for instance, the pioneeristic works in [11, 16, 34]) as well as other alternative methods by suitably specifying the option `method`. Let  $C_1$  and  $C_2$  denote two clusters. The single linkage method and the complete linkage method compute the distance between  $C_1$  and  $C_2$  by considering, respectively, the minimum and maximum distances between the units assigned to the two clusters. Thus, only one observed distance between units is used for calculating the distance between clusters. In this respect, a generalization is represented by the average linkage method which considers the average value of the distances between the units assigned to the two clusters. Finally, Ward's method measures the distance between  $C_1$  and  $C_2$  in terms of the distance between the cluster means of the two clusters. This highlights that Ward's method must be applied to quantitative variables. The distances used in these methods are summarized in Table 2.2. Table 2.3 contains the values of  $\alpha_1, \alpha_2, \beta$ , and  $\gamma$  in the Lance-Williams formula in order to obtain the four previously mentioned methods. Note that in Tables 2.3 and 2.2  $|C_1|$ ,  $|C_2|$ , and  $|C_3|$  denote the cardinality of the sets  $C_1$ ,  $C_2$ , and  $C_3$ , respectively, i.e., the number of units assigned to the corresponding clusters.

In Tables 2.4 and 2.5, we report all the available methods implemented in `hclust` and `agnes`, respectively. Before presenting such functions and their application in real-life case studies, we provide the following comments that may help clarify differences.

- Although most of the clustering methods are available in both functions, the default option differs: the complete linkage method (`method = "complete"`) for `hclust` and the average linkage method (`method = "average"`) for `agnes`.
- `agnes` is more general than `hclust` since setting `method = "flexible"` and specifying `par.method` allows to implement all the possible hierarchical methods according to the Lance-Williams formula.

**Table 2.2** Distance between clusters  $C_1$  and  $C_2$ : some particular cases

Method	Formula
Single linkage	$\min_{i_{C_1}, i_{C_2}} d(i_{C_1}, i_{C_2}), i_{C_1} \in C_1, i_{C_2} \in C_2$
Complete linkage	$\max_{i_{C_1}, i_{C_2}} d(i_{C_1}, i_{C_2}), i_{C_1} \in C_1, i_{C_2} \in C_2$
Average linkage	$\frac{\sum_{i_{C_1} \in C_1} \sum_{i_{C_2} \in C_2} d(i_{C_1}, i_{C_2})}{ C_1  C_2 }$
Ward's method	$d(\bar{x}_{C_1}, \bar{x}_{C_2})$ , where $\bar{x}_{C_1} = \frac{\sum_{i_{C_1} \in C_1} x_i}{ C_1 }$ and $\bar{x}_{C_2} = \frac{\sum_{i_{C_2} \in C_2} x_i}{ C_2 }$

**Table 2.3** Lance-Williams formula: some particular cases

Method	$\alpha_1$	$\alpha_2$	$\beta$	$\gamma$
Single linkage	0.5	0.5	0	-0.5
Complete linkage	0.5	0.5	0	0.5
Average linkage	$\frac{ C_1 }{ C_1 + C_2 }$	$\frac{ C_2 }{ C_1 + C_2 }$	0	0
Ward's method	$\frac{ C_1 + C_3 }{ C_1 + C_2 + C_3 }$	$\frac{ C_2 + C_3 }{ C_1 + C_2 + C_3 }$	$-\frac{ C_3 }{ C_1 + C_2 + C_3 }$	0

**Table 2.4** Clustering methods implemented in `hclust`

method	Method
"single"	Single linkage method
"average"	Average linkage method
"complete"	Complete linkage method
"ward.D"	Variant of Ward's method
"ward.D2"	Ward's method
"mcquitty"	McQuitty method
"median"	Median method
"centroid"	Centroid method

**Table 2.5** Clustering methods implemented in `agnes`

method	Method
"single"	Single linkage method
"average"	Average linkage method
"complete"	Complete linkage method
"ward"	Ward's method
"weighted"	McQuitty method
"gaverage"	Variant of the average linkage

- `hclust` requires only a distance matrix as input (either produced by `dist` or `daisy`) while `agnes` needs a matrix, a data frame, or a distance matrix. In this respect, the logical flag `diss` should be set (TRUE for a distance matrix, FALSE otherwise).
- A different option in `hclust` and `agnes` should be specified for Ward's method. As one may expect, this is `metric = "ward"` for `agnes`. The proper one in `hclust` is `metric = "ward.D2"` to differentiate it from `metric = "ward.D"`. The latter was the only choice available in the previous releases of `hclust`, but it did not implement Ward's method [27].
- The option to standardize data is available in `agnes` (setting `stand = TRUE`), but not in `hclust`.

In the following sections, we illustrate the use of `hclust` and `agnes` by means of examples involving quantitative, categorical, and mixed data. Particular attention is paid to the choice of the optimal number of clusters. In this respect, additional functions from several packages can be used and commented.

## 2.4 Divisive Hierarchical Clustering

Hierarchical clustering methods are often recognized as agglomerative procedures moving from the trivial partition with  $n$  singleton clusters to the trivial partition with one cluster composed of  $n$  units. In such methods, at each step, two clusters are fused according to a certain criterion. Divisive clustering algorithms reverse the problem. Specifically, they produce a series of partitions moving from the trivial solution with one cluster containing all the units to  $n$  singleton clusters. At each step, a cluster is split into two clusters. Divisive clustering is less common than agglomerative clustering and related methods have been often ignored, especially in the past. The reason is merely computational. Since divisive clustering procedures subsequently split one of the existing clusters into two clusters, the problem is how to find the optimal splitting in practice. Specifically, given an optimality criterion, exploring all the possible splittings is computationally too expensive. For instance, in the first step, there exist  $2^{(n-1)} - 1$  possible ways to split the initial cluster into two clusters. It is therefore fundamental to find a simple way of splitting the clusters at each step of the procedure. A valuable solution is provided by the most famous divisive clustering procedure, the DIvisive ANAlysis Clustering (DIANA) method [18]. At the first step, to divide the initial cluster, DIANA seeks the unit with the highest average distance from all the remaining units. Such a unit is used to build the new cluster, called the splinter one. In fact, the algorithm looks for units with average distance to the other units assigned to the original group larger than the one with respect to the splinter group. If units are closer to the splinter group, then they are moved to such a group. This process is repeated until no units are moved. In this way, the original cluster is split into two clusters. During the following steps, a new splinter group is found and units are assigned to it, in exactly the same way as for

the first step. The only difference relies in the fact that there is more than one cluster. In order to decide which cluster to be split, DIANA selects the one with the largest diameter, i.e., the cluster with the largest distance between two of its units. Of course, a singleton cluster cannot be split (its diameter is equal to 0). The DIANA method is implemented by the function `diana` of the package `cluster`.

One more divisive clustering method for binary variables, called MONA [18], is also implemented (function `mona` of the package `cluster`).

## 2.5 `agnes` and `hclust`

In this section, some case studies for agglomerative hierarchical clustering are provided.

### 2.5.1 Case Study with Quantitative Data

The dataset `wine` [7], available in the package **datasetsICR** [10], includes  $n = 178$  Italian wines characterized by  $p = 13$  constituents. The dataset contains an additional variable, `class`, distinguishing the wines in three groups according to the cultivar. Assuming that the clusters are unknown, we try to recover them by means of hierarchical clustering methods. In particular, we study whether a solution with  $k = 3$  classes is suggested. For this purpose, the common class of agglomerative methods is applied. Note that, in all of these cases, the variable `class` is not considered for clustering purposes; rather, it will be used ex post for evaluating the obtained partition.

```
> library(datasetsICR)
> data("wine")
> str(wine)
'data.frame': 178 obs. of 14 variables:
 $ Class           : int 1 1 1 1 ...
 $ Alcohol         : num 14.2 13.2 ...
 $ Malic acid      : num 1.71 1.78 ...
 $ Ash             : num 2.43 2.14 ...
 $ Alcalinity of ash: num 15.6 11.2 ...
 $ Magnesium       : int 127 100 ...
 $ Total phenols   : num 2.8 2.65 ...
 $ Flavanoids      : num 3.06 2.76 ...
 $ Nonflavonoid phenols: num 0.28 0.26 ...
 $ Proanthocyanins: num 2.29 1.28 ...
 $ Color intensity: num 5.64 4.38 ...
 $ Hue             : num 1.04 1.05 ...
 $ OD280/OD315 of diluted wines: num 3.92 3.4 ...
 $ Proline         : int 1065 1050 ...
```

By inspecting the data, we can see that the units of measurement of the chemical variables are different and, therefore, we proceed to standardize data before running the clustering procedure. For this purpose, we create two new objects containing the standardized data matrix and the distance matrix based on standardized data. The former can be obtained by using the function `scale`, and the latter can be written in at least two ways by using the functions `dist` or `daisy` (in the last line of the code we check that the same output is obtained).

```
> wine.Z <- scale(wine[, 2:ncol(wine)],
+                     center = TRUE, scale = TRUE)
> D.wine <- dist(x = wine.Z, method = "euclidean")
> library(cluster)
> D.wine.bis <- daisy(x = wine.Z,
+                         metric = "euclidean")
> sum(abs(D.wine - D.wine.bis))
[1] 0
```

The standardization step requires some comments. In fact, several options for standardizing data can be chosen. By using the default options (`center = TRUE` and `scale = TRUE` could be omitted), the variables are standardized by subtracting the sample mean and dividing by the sample standard deviation. In other words, if `scale = TRUE`, then data are divided by `apply(x, 2, sd)` being `x` the data matrix at hand. If a user is interested in computing the standard deviation by dividing by `n`, (s)he can specify `scale = apply(x, 2, sd)*sqrt((nrow(x) - 1)/nrow(x))`. Therefore, `scale` is either a logical value (`FALSE` means no normalization) or a vector specifying how to normalize the variables. The same comment holds for `center`. If `center = FALSE`, no centering is done.

In the function `dist`, standardized data should be given in input to get a distance matrix based on standardized variables. The function `daisy` has the option `stand`, a logical flag equal to `TRUE` for standardizing the data before computing the distances and to `FALSE` otherwise. However, the user should be aware that, in this case, data are standardized by subtracting the variable means and by dividing by the mean absolute deviations. For this reason, we prefer to use the function `scale`, taking into account that the correction `sqrt((nrow(x) - 1)/nrow(x))` has a negligible impact especially when the number of units is large.

By using `wine` we discuss common and distinctive features of the functions `hclust` and `agnes`. First of all, a relevant difference concerns the input argument specifying the dataset to be clustered. In `hclust` it is denoted by `d`, and in `agnes` by `x`. Such a difference can be explained by noting that, as already observed, `hclust` requires only a distance matrix as input, while `agnes` also accepts a data matrix or data frame. Generally speaking, it is not necessary to specify `diss`. If `x` is not a distance matrix, `agnes` implicitly sets `diss = FALSE` (output `res.agnes`). If `diss = TRUE` is set, an error occurs. If `x` is a distance matrix, `agnes` implicitly

sets `diss = TRUE` (output `res.agnes.3`). Note that, in this case, if `diss = FALSE` is specified, a solution is found (output `res.agnes.2`) without warning or error messages. Such a solution is however meaningless.

The dataset to be clustered, `x` or `d`, is the only option that must be given in both functions. For `agnes`, we have the following code.

```
> res.agnes <- agnes(x = wine.Z, diss = FALSE)
> res.agnes.2 <- agnes(x = D.wine, diss = FALSE)
> res.agnes.3 <- agnes(x = D.wine, diss = TRUE)
```

The output in `res.agnes` and `res.agnes.3` is the same. This is the solution obtained by using the default method in `agnes`. In particular, if the option "method" is not specified, the average linkage method is applied. Other common options are "single" (single linkage method), "complete" (complete linkage method), and "ward" (Ward's method). In this book, we limit our analysis to such variants only, as they are the most common. However, as we already remarked, by specifying `par.methods`, other solutions according to the Lance-Williams formula can be found.

Let us now investigate in detail the object `res.agnes`.

```
> class(res.agnes)
[1] "agnes" "twins"
> names(res.agnes)
[1] "order" "height" "ac" "merge" "diss" "call"
[7] "method" "data"
```

The object is of class `agnes` and it is a list with eight components. The three most important ones are `merge`, `height`, and `order`. In `res.agnes$merge`, details on the obtained hierarchy are given. It is a matrix with  $n - 1$  rows and 2 columns with elements giving the clusters merged at each step. These clusters are indicated by integer numbers, where a negative value refers to a singleton cluster. To clarify, let us see the first 25 rows.

```
> t(res.agnes$merge[1:25,])
     [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
[1,] -10    -132   -12    -16    -93    -35    -1    -17   -165
[2,] -48    -134   -13    -54   -108   -38    -21   -18   -173
     [,10]  [,11]  [,12]  [,13]  [,14]  [,15]  [,16]  [,17]
[1,] -23    -41    -105   -24    -28    -141   -64    -27
[2,] -30    -57    -117   -25    -39    -143   -99    -58
     [,18]  [,19]  [,20]  [,21]  [,22]  [,23]  [,24]  [,25]
[1,] -112   -164   -149   -83    -91    -140   -11    12
[2,] -126   -171   -175   -88    -92    -163   -32   -107
```

All negative integers are visible by inspecting the first 24 steps. Therefore, during such steps, singleton clusters are merged. In the first step, units 10 and 48 are fused into a new cluster. The first positive integer appears at step 25. The two values,  $-107$  and 12, mean that, during this step, unit 107 is merged with the cluster obtained at step 12, thus identifying a cluster with three units (107, 105, and 117).

Further information is provided by `res.agnes$height`, a vector of length  $n - 1$  containing the merging distances at each step. In practice, the previously described components are usually not directly used. In fact, together with `res.agnes$order`, they are used for displaying the hierarchy by means of the corresponding dendrogram. The dendrogram is a tree, where one axis refers to the units and the other one to the merging distances, useful for a graphical analysis of the output of any hierarchical clustering method. In this respect, `res.agnes$order` avoids that the branches of the dendrogram cross the other ones. To plot the dendrogram, we simply need to apply the function `plot` to an object of class `agnes`. In this way, two plots are produced. To specify the dendrogram, the option `which.plots = 2` is set. Notice that the option `labels = FALSE` might be added in order to suppress leaf labels (by default the row names or row numbers are used).

```
> plot(res.agnes, which.plots = 2,
+       main = "Average")
```

The dendrogram is given in Fig. 2.2. At the bottom of the figure, the value of the agglomerative coefficient is reported (`res.agnes$ac`). The coefficient describes the strength of the clustering structure. However, its use appears to be of limited help since the value of the coefficient grows with  $n$ . Therefore, the coefficient should not be used to compare solutions from datasets of very different sizes. To avoid visualizing the value, it is sufficient to apply the function `plot` to `as.dendrogram(res.agnes)`.

Let us now repeat the analysis by applying `hclust`. As expected, the use of `hclust` leads to essentially the same results. The following code should be run.

```
> res.hclust <- hclust(d = D.wine)
```

As for `agnes`, the clustering method can be chosen by the option `method`. If `method` is omitted, `hclust` performs the complete linkage method. Although `res.hclust` is not an object of class `agnes`, the output of the two functions is closely related.

```
> class(res.hclust)
[1] "hclust"
> names(res.hclust)
[1] "merge" "height" "order" "labels" "method"
[6] "call"  "dist.method"
```

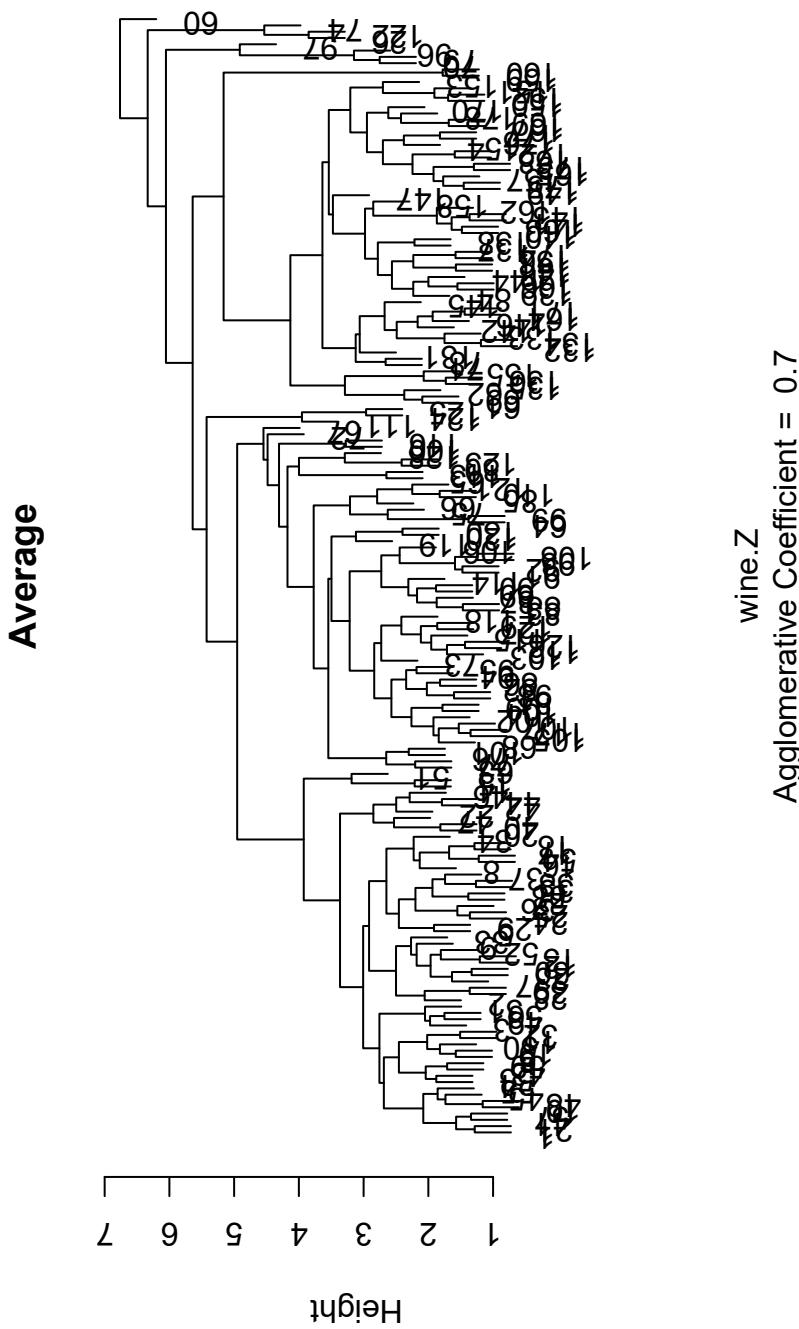


Fig. 2.2 Wine data: dendrogram using the average linkage method

The object `res.hclust` is of class `hclust` and, as before, it is a list with components quite similar to those in `res.agnes`. Specifically, `merge` and `height` have the same meanings. The role of `order` is similar to that of `agnes`, but it contains only integers because the labels of the units are available in `labels`. The dendrogram can be displayed by using the function `plot` (Fig. 2.3).

```
> plot(res.hclust, main = "Complete")
```

### 2.5.1.1 Variants of the Dendrogram

Different variants of a dendrogram can be plotted. These can be displayed by using the output of both functions `agnes` and `hclust`. For instance, after converting `res.hclust` to an object of class `dendrogram`, we can plot a dendrogram with triangular branches (Fig. 2.4).

```
> dend.res.hclust <- as.dendrogram(res.hclust)
> class(dend.res.hclust)
[1] "dendrogram"
> plot(dend.res.hclust, type = "triangle",
+       main = "Triangular")
```

An horizontal version is plotted in Fig. 2.5 by using the package **ape** [29].

```
> library(ape)
> phyl.res.hclust <- as.phylo(res.hclust)
> class(phyl.res.hclust)
[1] "phylo"
> plot(phyl.res.hclust, cex = 0.5, label.offset = 0.5,
+       main = "Horizontal")
```

The same package allows for representing the hierarchy by using a graph (see Fig. 2.6).

```
> plot(phyl.res.hclust, type = "unrooted", cex = 0.4,
+       main = "Graph")
```

These and other variants are also available for the output of `agnes`.

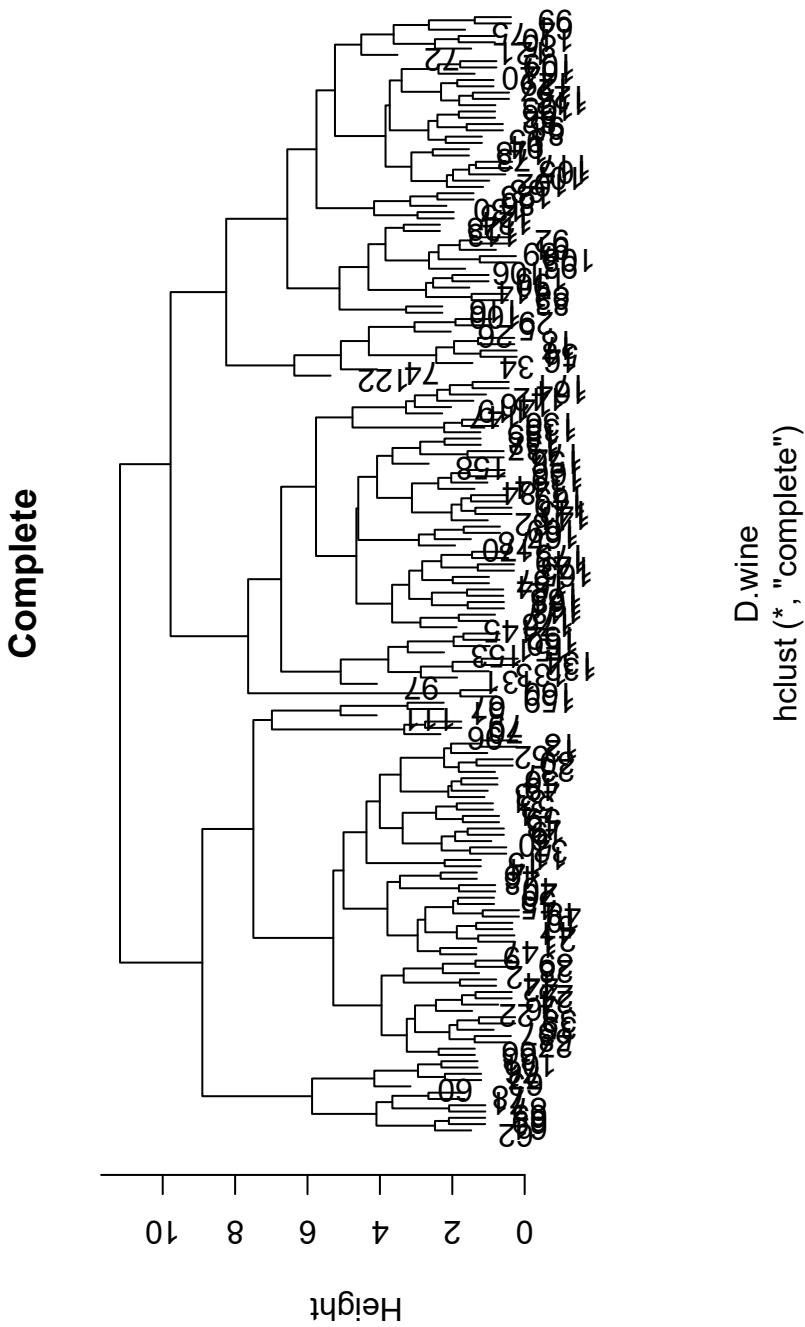


Fig. 2.3 wine data: dendrogram using the complete linkage method

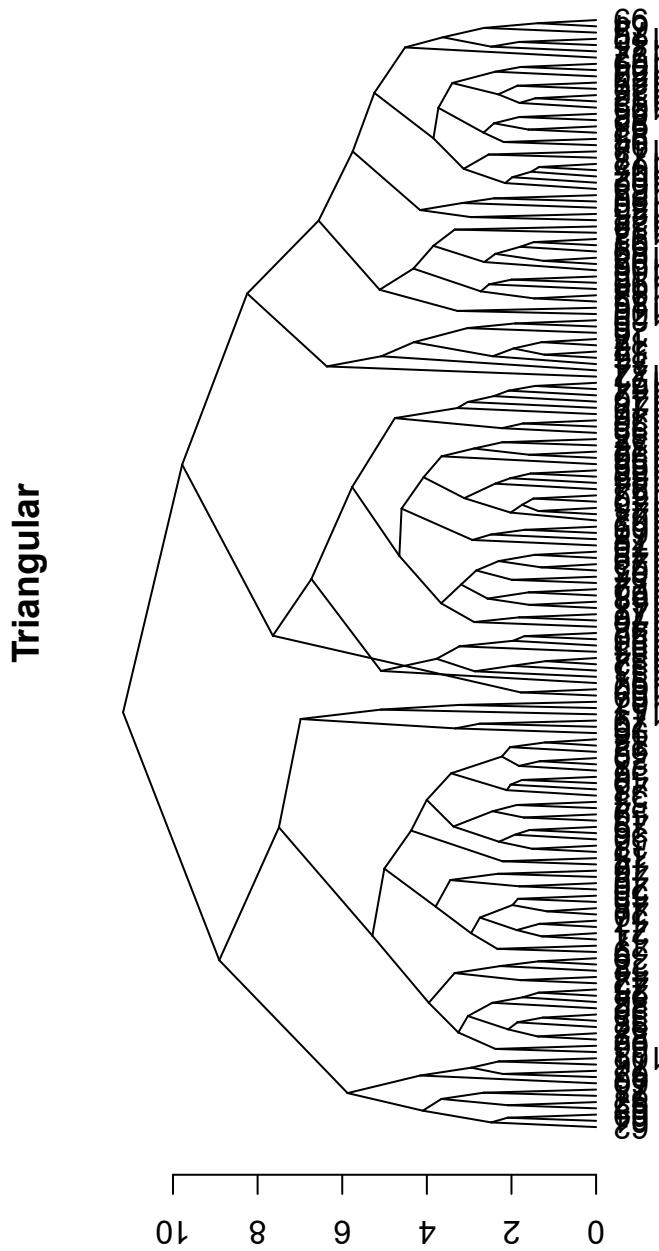
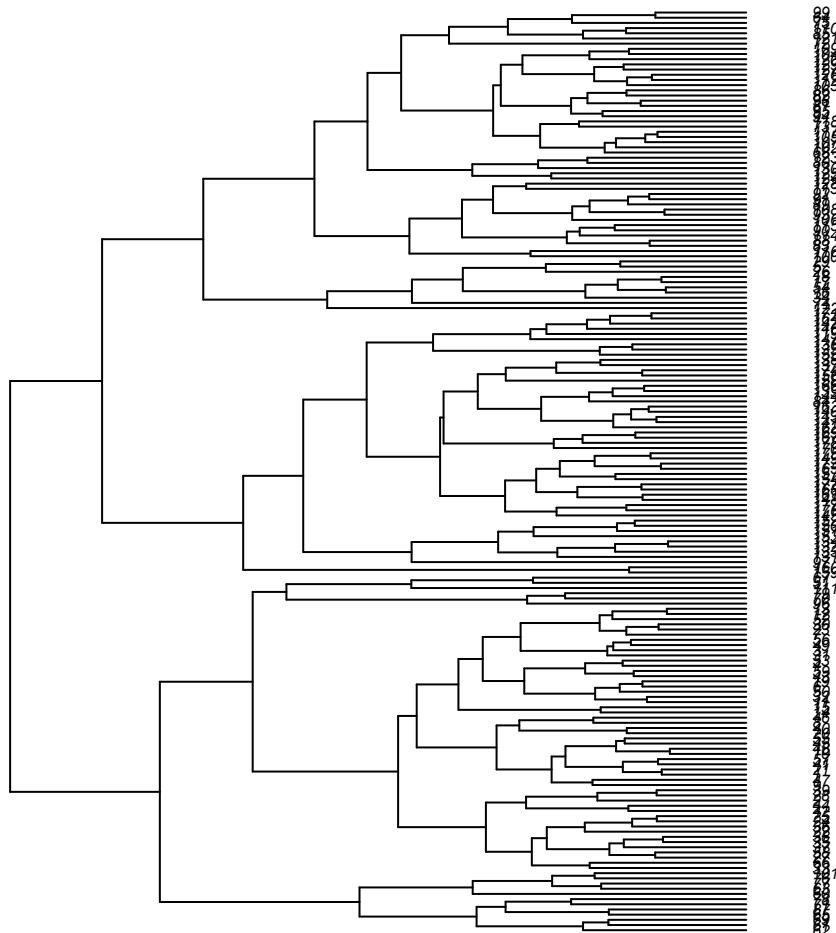


Fig. 2.4 wine data: triangular dendrogram using the complete linkage method

## Horizontal

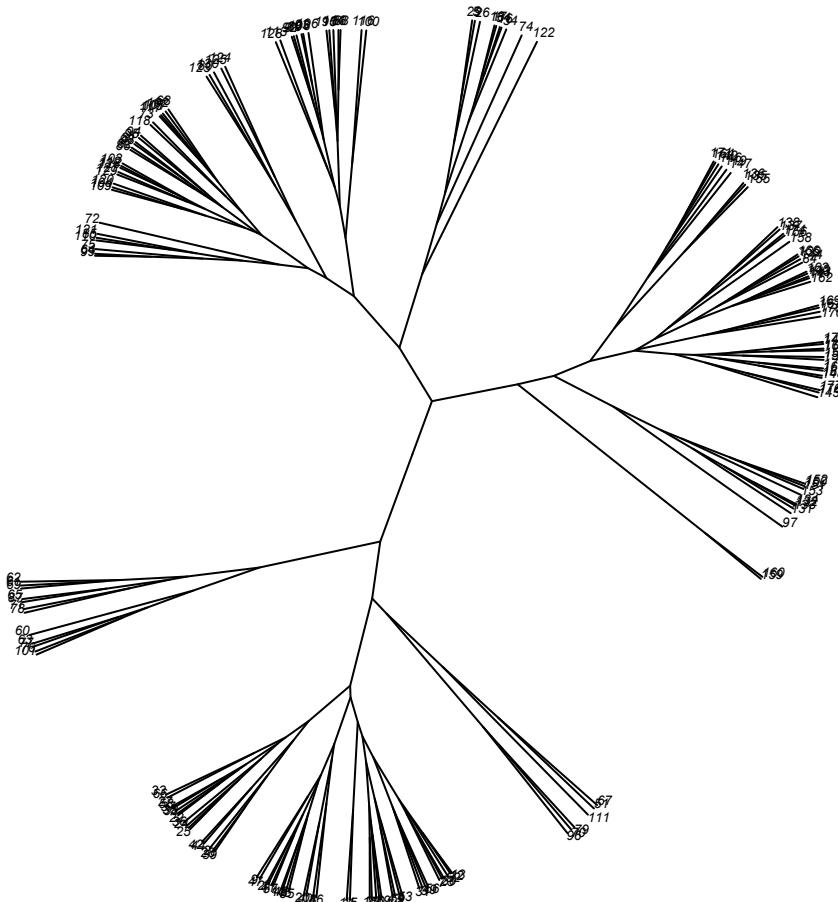


**Fig. 2.5** wine data: horizontal dendrogram using the complete linkage method

### **2.5.1.2 Choice of the Best Solution**

By inspecting the dendograms in Figs. 2.2 and 2.3 (using the average linkage method and the complete linkage method, respectively), we can observe rather different solutions. Therefore, we may question which is the best method to be used for

## Graph



**Fig. 2.6** wine data: graph dendrogram using the complete linkage method

clustering in this particular case. For the `wine` data, the true partition is known in advance. However, in real-life applications, the true number of clusters is usually unknown. In such cases, unless one knows that a clustering method is better than the other ones, it may be useful to study whether different clustering methods produce similar groups, at least to some extent.

The first point to address is how many clusters should be kept. By using the dendrogram, this is equivalent to assess where to cut it. This is a subjective choice and, obviously, cutting a dendrogram at a given level gives a partition, while, cutting at another level, a different partition is obtained. A general rule is to cut the dendrogram at the level where the difference between two consecutive fusions is the largest one. Unfortunately, this rule cannot always be applied because larger differences may not be always visible.

Although the dendrogram is often used to choose the number of clusters, we also note that such a choice can be data dependent in the sense that it can be related to the level of granularity the researcher is looking for. Let us consider a marketing example and suppose that a company is interested in distinguishing three classes of customers to offer three specifically tailored promotions. In such a case, the dendrogram should be cut so that three clusters are found. If the marketing team decides to add one more promotion, it is sufficient to separate a cluster into two clusters, i.e., to cut the dendrogram at the preceding step obtaining four clusters. Conversely, if only two tailored promotions are available, a different cut is chosen in such a way that two of the three clusters are merged to a single cluster.

The function `cutree` of the package `stats` allows for cutting a dendrogram. There are two input arguments to select. The first one is the obtained hierarchy (in our case, `res.agnes` or `res.hclust`). The latter one is either the number of clusters (option `k`) or the height where the dendrogram should be cut (option `h`). By looking at the dendograms, we can conclude that the complete linkage method seems to recover the three classes known in advance better than the average linkage method. To corroborate this claim we cut the dendrogram using either `k = 3` (object `cluster.hclust`) or `h = 9` (object `cluster.hclust.2`) and after observing that the two partitions coincide by using the Adjusted Rand Index (ARI) [14] implemented in the function `adjustedRandIndex` of the package `mclust` [32], we compare them with the variable `Class` to assess to what extent the cultivars are identified. As it may be known, the ARI index compares two partitions and values close to 1 indicate strong agreement (1 in the case of perfect agreement).

```
> cluster.hclust <- cutree(res.hclust, k = 3)
> cluster.hclust.2 <- cutree(res.hclust, h = 9)
> library(mclust)
> adjustedRandIndex(cluster.hclust, cluster.hclust.2)
> [1] 1
> table(cluster.hclust, wine$Class)
cluster.hclust 1 2 3
      1 51 18 0
      2  8 50 0
      3  0  3 48
> adjustedRandIndex(cluster.hclust, wine$Class)
[1] 0.5771436
```

We can see that the three clusters can be interpreted in terms of the cultivars, even if 29 out of 178 (about 16%) wines are misclassified.

We now analyze the solutions provided by some other clustering methods. In the following, we limit our attention to the function `hclust` bearing in mind that the same output can be found by using `agnes`. In particular, we apply the single linkage method and Ward's method.

```
> res.hclust.single <- hclust(d = D.wine,
+                                method = "single")
> plot(res.hclust.single, main = "Single")
> res.hclust.ward <- hclust(d = D.wine,
+                               method = "ward.D2")
> plot(res.hclust.ward, main = "Ward")
```

The solution obtained by the single linkage method (Fig. 2.7) is an example of the so-called chaining effect. Specifically, as for the complete linkage method, the single linkage method merges clusters by considering a specific distance between units belonging to different clusters. The merging criterion is based on the minimum distance and a chain of units can be generated producing straggling clusters.

Three well-separated clusters are found by using Ward's method (Fig. 2.8) and if we cut the dendrograms to obtain  $k = 3$  clusters we get the following partition.

```
> cluster.hclust.ward <- cutree(res.hclust.ward,
+                                   k = 3)
> table(cluster.hclust.ward, wine$Class)
cluster.hclust.ward  1   2   3
                  1 59  5  0
                  2  0 58  0
                  3  0  8 48
> adjustedRandIndex(cluster.hclust.ward, wine$Class)
[1] 0.7899332
```

The three clusters offer a better partition of wines with respect to the cultivars, with only 13 wines belonging to `Class = 2` that are assigned to different clusters and a value of ARI equal to 0.79. Thus, we focus our attention on such a solution.

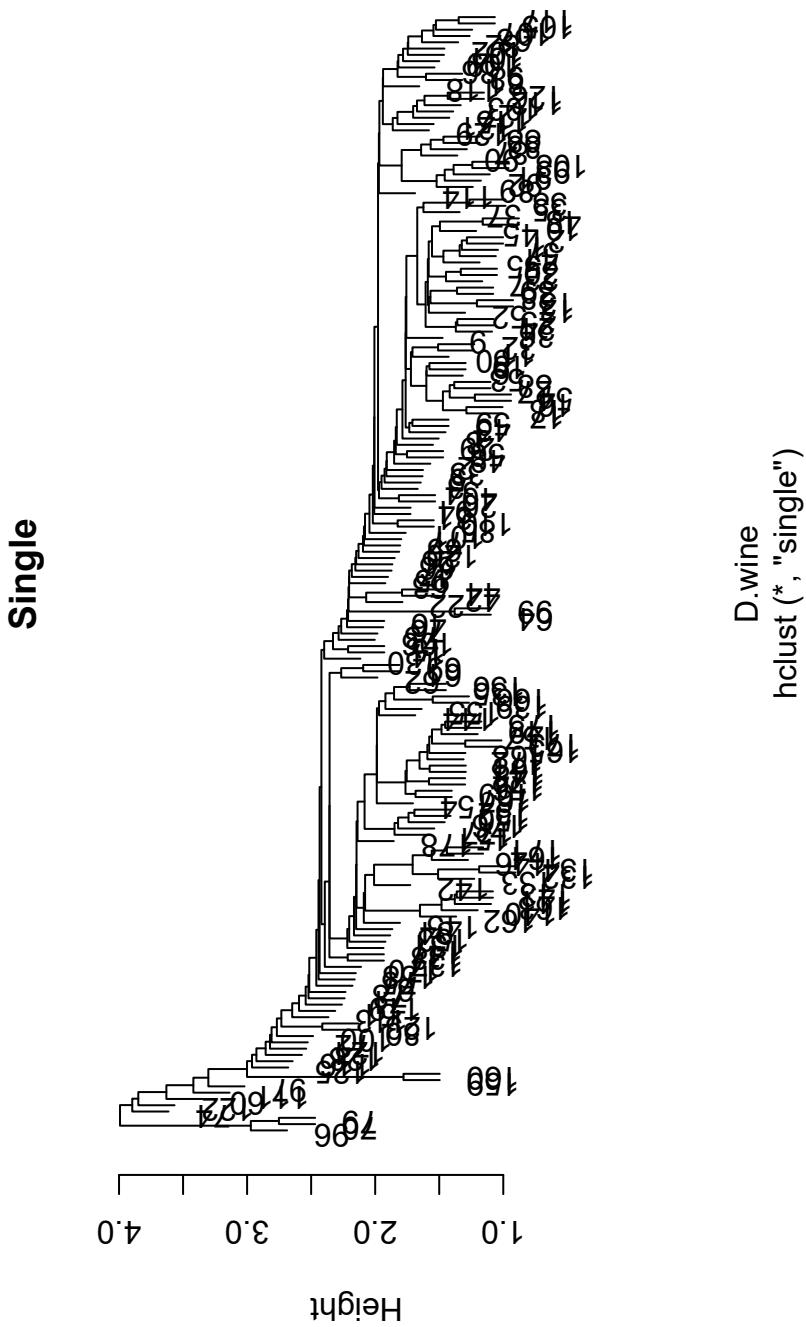


Fig. 2.7 wine data: dendrogram using the single linkage method



Fig. 2.8 wine data: dendrogram using Ward's method

To characterize the clusters and to distinguish the cultivars with respect to the chemical features, we compute the average values of the constituents for each cluster. We also report the overall mean values as a reference.

```
> k <- 3
> mean.cluster <- t(sapply(x = 1:k, FUN = function(nc)
+   apply(wine[cluster.hclust.ward == nc, 2:ncol(wine)],
+   2, mean)))
> round(mean.cluster, 2)
      Alcohol Malic acid Ash Alcalinity of ash
[1,]    13.67      1.97 2.46          17.53
[2,]    12.20      1.94 2.22          20.21
[3,]    13.06      3.17 2.41          21.00
      Magnesium Total phenols Flavanoids
[1,]     106.16      2.85  3.01
[2,]      92.55      2.26  2.09
[3,]     99.86      1.69  0.85
      Nonflavanoid phenols Proanthocyanins
[1,]                 0.29      1.91
[2,]                 0.36      1.69
[3,]                 0.45      1.13
      Color intensity Hue
[1,]           5.45 1.07
[2,]           2.90 1.06
[3,]           6.85 0.72
      OD280/OD315 of diluted wines Proline
[1,]                      3.16 1076.05
[2,]                      2.86 501.43
[3,]                      1.73 624.95
> mean.all <- apply(wine[, 2:ncol(wine)], 2, mean)
> round(mean.all, 2)
      Alcohol             Malic acid
            13.00              2.34
      Ash             Alcalinity of ash
            2.37              19.49
      Magnesium          Total phenols
            99.74              2.30
      Flavanoids          Nonflavanoid phenols
            2.03                0.36
      Proanthocyanins      Color intensity
            1.59                5.06
      Hue OD280/OD315 of diluted wines
            0.96                2.61
      Proline
            746.89
```

By comparing the mean values of the various clusters we are able to discover the features of the three clusters. For instance, the wines assigned to Cluster 1 have high values of Alcohol, Magnesium, Proanthocyanins, OD280/OD315 of diluted wines, and Proline and low values of Alcalinity of ash

and Nonflavanoid phenols. The distinctive features of the wines belonging to Cluster 2 are related to the values of Alcohol, Ash, Magnesium, and Color intensity. Finally, the wines in Cluster 3 are characterized by high values of Malic acid, Nonflavanoid phenols, and Color intensity and low values of Total phenols, Flavanoids, Proanthocyanins, Hue, and OD280/OD315 of diluted wines.

### 2.5.1.3 Alternative Functions

As far as we know, at least three other packages offer functions to perform agglomerative hierarchical clustering. These are the functions `NbClust` of the package **NbClust** [3], `hcluster` of the package **amap** [21], and `hcum` of the package **factoextra** [17]. The first one will be discussed in Sect. 2.5.3. Its peculiarity entails the implementation of several cluster validity indices for an optimal selection of the number of clusters. The second one performs some agglomerative clustering methods to be specified by the option `link` (with obvious notation, the most common ones can be set by writing "single", "complete", "average", and "ward") and implements various distance measures by means of the option `method`. The input dataset of the function is a matrix or data frame with quantitative data and, therefore, the function seems to work only for quantitative variables, although `method = "binary"` can be set. Note that the input can also be an object of class `exprSet`, useful for microarray data. The option `nbproc` can be used for specifying an integer value giving the number of parallel processes, an option that is currently available for Linux and Mac only. The distance measures available in `method` are implemented in the package function `Dist`. The package also contains the function `diss` computing a distance matrix from a dataset that contains only categorical variables. The third package computes any kind of hierarchical clustering cutting the dendrogram into the desired number of clusters and offers cluster validity indices, as **NbClust** does.

## 2.5.2 Case Studies with Mixed Data

In the previous section, we introduced the functions `agnes` and `hclust` and discussed their application to an example with quantitative variables. We now handle data that contain both quantitative and categorical variables. In such a situation, the use of `agnes` and `hclust` does not present particular difficulties, but it is fundamental to compute the distance matrix by means of a distance measure specifically designed for mixed data. In this context, just to give an example, the Euclidean distance cannot be longer used. Moreover, the availability of categorical data offers additional ways to interpret the clusters. All these points will be discussed by two case studies.

### 2.5.2.1 Case Study I

The dataset `lasvegas.trip` of the package **datasetsICR** refers to  $n = 21$  hotels in Las Vegas characterized by  $p = 9$  variables (own elaboration on data from [26]). Three of them are quantitative, while the remaining six are categorical, in particular, binary.

```
> data("lasvegas.trip")
> str(lasvegas.trip)
'data.frame': 21 obs. of 9 variables:
 $ Score      : num 4.21 4.12 3.21 4.54 3.71 ...
 $ Hotel.stars: num 5 5 3 5 3 3 3 4 4 ...
 $ Nr.rooms   : num 3933 3348 3773 2034 ...
 $ Pool        : Factor w/ 2 levels "NO", "YES": 2 ...
 $ Gym         : Factor w/ 2 levels "NO", "YES": 2 ...
 $ Tennis.court: Factor w/ 2 levels "NO", "YES": 1 ...
 $ Spa          : Factor w/ 2 levels "NO", "YES": 2 ...
 $ Casino      : Factor w/ 2 levels "NO", "YES": 2 ...
 $ Free.internet: Factor w/ 2 levels "NO", "YES": 2 ...
```

The continuous quantitative variable `Score` refers to the average TripAdvisor scores. The remaining two quantitative variables are discrete and concern the hotel stars and the number of rooms. The binary variables consider the presence of a pool, gym, tennis court, spa, casino, and free Internet connection. The number of rooms is not considered for clustering purposes because it does not reflect hotel quality.

In order to handle mixed variables, the function `daisy` can be applied. Note that the distance measures (option `distance`) available in `dist` can be used for continuous or nominal variables, but not for mixed data. The Gower distance is indeed used.

```
> D <- daisy(lasvegas.trip[, -3], metric = "gower")
```

To cluster hotels, we apply the complete linkage method. Obviously, the analysis can be equivalently carried out by using either `hclust` or `agnes`, and hence we decide to use the latter. Note that the single linkage method and the average linkage method could be run, while Ward's method should not be applied because the variables are not quantitative. Note that also `NbClust` cannot be used because distance measures for mixed variables are not implemented.

```
> res.agnes <- agnes(D, diss = TRUE,
+                      method = "complete")
```

It is important to note that, in the case of mixed variables, `agnes` must be run by using the distance matrix as input (option `diss = TRUE`). In fact, `agnes` allows the specification of a distance measure, but, as observed in the help file, the currently

available options are the Euclidean and Manhattan ones. The dendrogram is plotted (omitting the name of the hotels), see Fig. 2.9.

```
> res.agnes$order.lab <- rep("", 21)
> plot(res.agnes, hang = -0.1, which.plots = 2,
+       main = "")
```

By inspecting the dendrogram,  $k = 2$  clusters are easily visible.

```
> cluster.agnes <- cutree(res.agnes, k = 2)
> table(cluster.agnes)
cluster.agnes
 1  2
16   5
```

We can see that the solution distinguishes five hotels assigned to Cluster 2 from the remaining ones allocated to Cluster 1. To further inspect the clusters, we can use the observed variables. In particular, for the quantitative ones (including the variable concerning the number of rooms, used as external information), we can compute the average value distinguished by cluster and, for the categorical ones, we can plot the barplots and compute the  $p$ -values of the corresponding Pearson's  $\chi^2$  Test (Fig. 2.10).

```
> k <- 2
> lapply(X = 1:k, FUN = function(nc) apply
+           (lasvegas.trip[cluster.agnes == nc, 1:3],
+            2, mean))
[[1]]
  Score Hotel.stars Nr..rooms
4.151042    4.312500 2520.562500
[[2]]
  Score Hotel.stars Nr..rooms
4.033333    3.200000 1159.000000
>
> par(mfrow = c(3, 2))
> for (j in 4:dim(lasvegas.trip)[2])
+ {
+   counts <- table(lasvegas.trip[, j],
+                   cluster.agnes)
+   barplot(counts, main = names(lasvegas.trip)[j],
+           names.arg = paste("Clus.", 
+                             colnames(counts)),
+           xlab = round(chisq.test(counts)$p.value, 2),
+           col = c("darkblue", "red"),
+           legend = rownames(counts))
+ }
```

As we can easily notice, the two clusters are mainly distinguished in terms of the presence of a spa, the number of stars, and the size expressed in terms of the number of rooms. In comparison with the hotels assigned to Cluster 2, those belonging to Cluster 1 are bigger and with spa facilities.

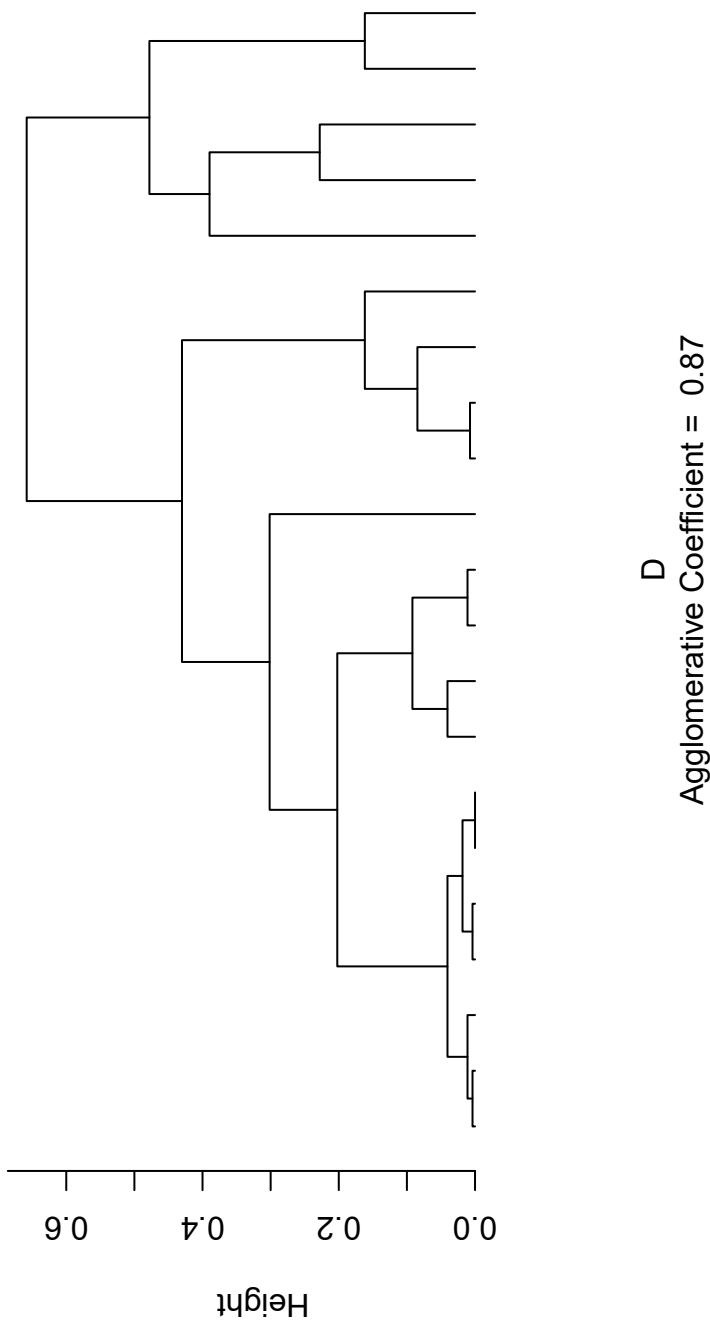
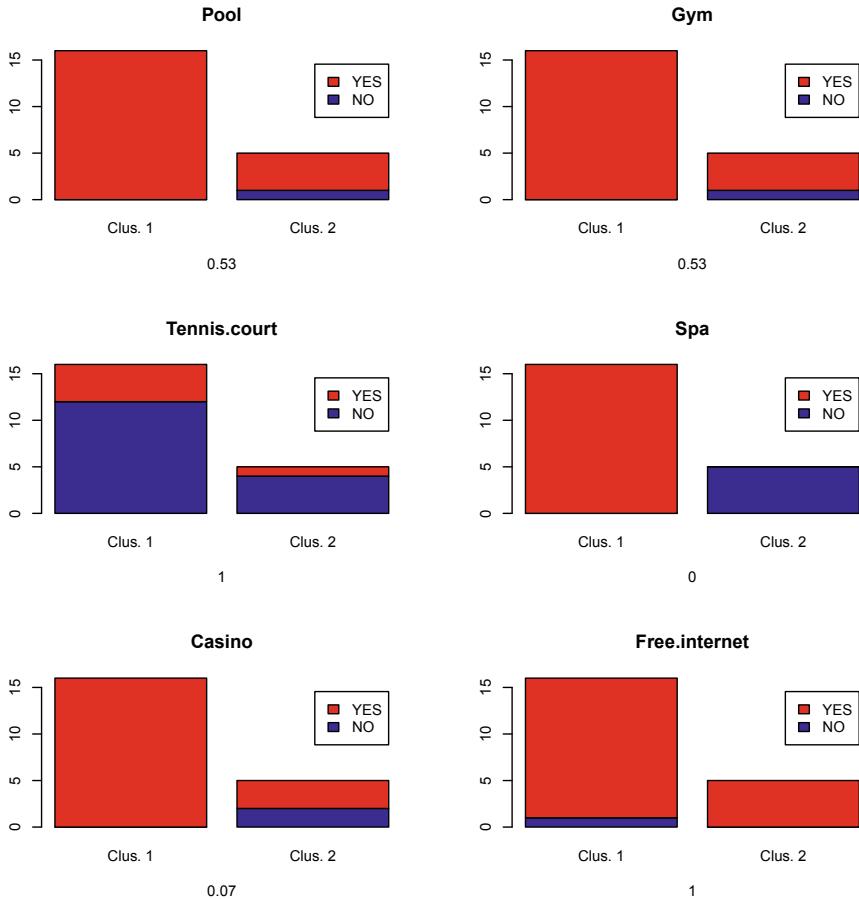


Fig. 2.9 `lasvegas.trip` data: dendrogram using the complete linkage method



**Fig. 2.10** lasvegas.trip data: barplots distinguished by cluster (complete linkage method,  $k = 2$  clusters)

### 2.5.2.2 Case Study II

In this section, we study a larger and more challenging dataset, the dataset `flags` [7] of the package **datasetsICR**, containing details on  $n = 194$  flags in terms of quantitative and categorical variables. The dataset also contains additional variables, such as `landmass`, `language`, and `religion`, that can be used for interpreting the clusters once they are found. The structure of `flags` is reported below.

```
> data("flags")
> str(flags)
'data.frame': 194 obs. of 29 variables:
$ landmass : Factor w/ 6 levels "N.America",...
$ zone : Factor w/ 4 levels "NE","SE","SW",...: 1 1 ...
$ area : int 648 29 2388 0 0 1247 0 0 2777 2777 ...
$ population: int 16 3 20 0 0 7 0 0 28 28 ...
$ language : Factor w/ 10 levels "English",...
$ religion : Factor w/ 8 levels "Catholic",..
$ bars : int 0 0 2 0 3 0 0 0 0 0 ...
$ stripes : int 3 0 0 0 0 2 1 1 3 3 ...
$ colours : int 5 3 3 5 3 3 3 5 2 3 ...
$ red : Factor w/ 2 levels "absent","present": 2 ...
$ green : Factor w/ 2 levels "absent","present": 2 ...
$ blue : Factor w/ 2 levels "absent","present": 1 ...
$ gold : Factor w/ 2 levels "absent","present": 2 ...
$ white : Factor w/ 2 levels "absent","present": 2 ...
$ black : Factor w/ 2 levels "absent","present": 2 ...
$ orange : Factor w/ 2 levels "absent","present": 1
...
$ mainhue : Factor w/ 8 levels "black","blue",...: 5
...
$ circles : int 0 0 0 0 0 0 0 0 0 0 ...
$ crosses : int 0 0 0 0 0 0 0 0 0 0 ...
$ saltires : int 0 0 0 0 0 0 0 0 0 0 ...
$ quarters : int 0 0 0 0 0 0 0 0 0 0 ...
$ sunstars : int 1 1 1 0 0 1 0 1 0 1 ...
$ crescent : Factor w/ 2 levels "absent","present":
...
$ triangle : Factor w/ 2 levels "absent","present":
...
$ icon : Factor w/ 2 levels "absent","present": 2 ...
$ animate : Factor w/ 2 levels "absent","present": ...
$ text : Factor w/ 2 levels "absent","present": 1 ...
$ topleft : Factor w/ 7 levels "black","blue",...: 1
...
$ botright : Factor w/ 8 levels "black","blue",...
...
```

In the data frame,  $p = 23$  variables describing the flags are located from column 7 (variable bars) to the last one. Eight of them are quantitative (discrete) and the remaining ones are categorical (factor) with two levels (absent and present) generally, but for the variables mainhue, topleft, and botright with seven or eight levels (referring to a set of colors).

The function `daisy` specifying the option `metric = "gower"` is used to calculate the distance matrix.

```
> D.flags <- daisy(flags[, 7:ncol(flags)],  
+                      metric = "gower")
```

By using `hclust` we consider the `single`, `average`, and `complete` linkage methods.

```
> res.single <- hclust(d = D.flags,
+                         method = "single")
> res.average <- hclust(d = D.flags,
+                         method = "average")
> res.complete <- hclust(d = D.flags,
+                         method = "complete")
```

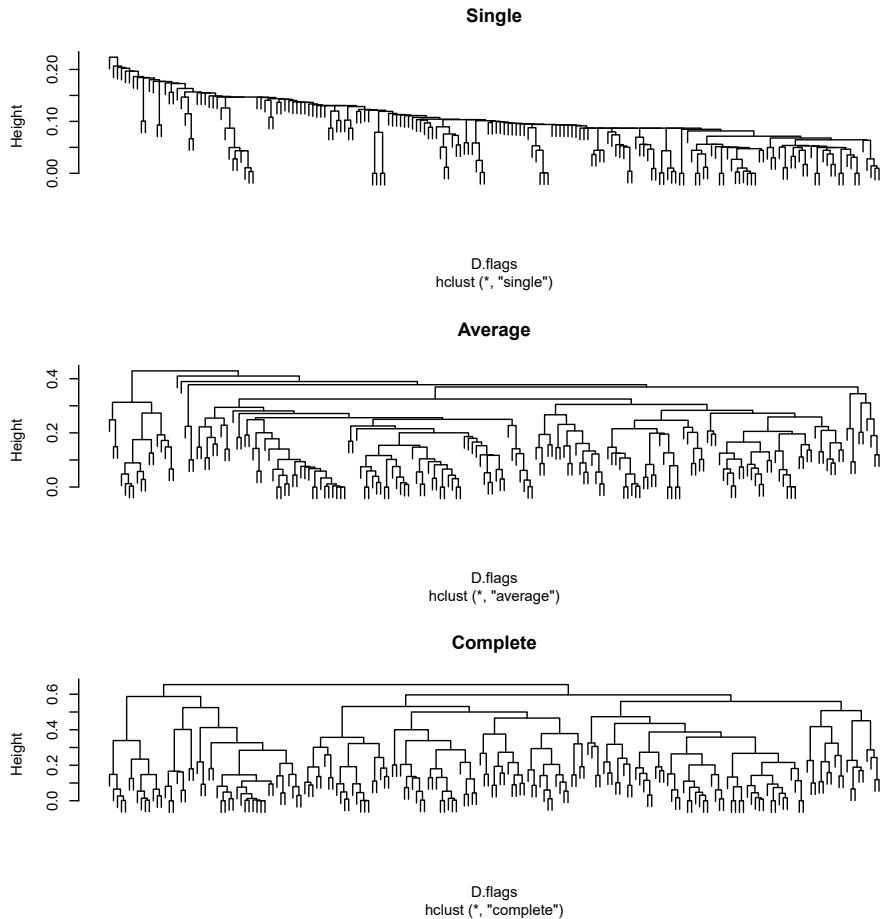
A first insight into the three results can be done by plotting the corresponding dendograms (Fig. 2.11).

```
> par(mfrow = c(3, 1))
> plot(res.single, labels = FALSE, main = "Single")
> plot(res.average, labels = FALSE, main = "Average")
> plot(res.complete, labels = FALSE, main = "Complete")
```

We can see that the single linkage method suffers from the chaining effect and, therefore, we focus our attention on the results obtained via the average linkage method and the complete linkage method. The average linkage method solution seems to discover three main clusters plus some clusters with small size. This can be found by cutting the dendrogram at height  $h = 0.32$ . Moreover, the small clusters can be merged so that a new cluster, labeled by 0, is obtained. Note that it is not a proper cluster because it does not contain homogeneous units.

```
> cl.ave <- cutree(res.average, h = 0.32)
> table(cl.ave)
cl.ave
  1   2   3   4   5   6   7   8
  78  85  17   6   3   3   1   1
> cl.ave[cl.ave > 3] <- 0
> table(cl.ave)
cl.ave
  0   1   2   3
 14  78  85  17
```

By inspecting the dendrogram resulting from the complete linkage method solution, three clusters can be observed. For this reason, we cut the dendrogram setting  $k = 3$ .



**Fig. 2.11** flags data: dendograms using the single (left), average (center), and complete (right) linkage methods

```
> cl.com <- cutree(res.complete, k = 3)
> table(cl.com)
cl.com
 1 2 3
71 74 49
```

To select the best solution, we compare the average linkage method and the complete linkage method solutions discovering that Clusters 1 and 2 are identified by both clustering methods. Such a comment does not hold for Cluster 3. Specifically,

Cluster 2 from the average linkage method is approximately a mix of Clusters 2 and 3 from the complete linkage method. This is well captured by the function `mapClass` of the package **mclust** that gives the best correspondence between clusters given two vectors viewed as alternative partitions of the same set of units.

```
> table(cl.com, cl.ave)
      cl.ave
cl.com 0   1   2   3
      1 2  64  2   3
      2 8  13  48  5
      3 4  1  35  9
> adjustedRandIndex(cl.com, cl.ave)
[1] 0.3726202
> mapClass(cl.com, cl.ave)
$ aTOb
$aTOb$ `1 `
[1] 1
$aTOb$ `2 `
[1] 2
$aTOb$ `3 `
[1] 2

$bTOa
$bTOa$ `0 `
[1] 2
$bTOa$ `1 `
[1] 1
$bTOa$ `2 `
[1] 2
$bTOa$ `3 `
[1] 3
```

The output confirms that there exists a certain agreement between the two solutions. However, the complete linkage method solution appears to be more valuable due to its higher level of simplicity. For this reason, we investigate whether the clusters found by using the complete linkage method have a meaningful interpretation. For this purpose, we consider the external information provided by the variables `landmass`, `language`, and `religion`.

```

> table(cl.com, flags$landmass)
cl.com N.America S.America Europe Africa Asia Oceania
  1           5        4       6      37    17      2
  2           8        9      23      9    16      9
  3          18       4       6      6    6      9
> chisq.test(cl.com, flags$landmass)$p.value
[1] 1.42892e-10

> table(cl.com, flags$language)
cl.com English Spanish French German Slavic
  1         13        2      10       0     1
  2         12        9       6      6     3
  3         18       10      1       0     0
cl.com Other Indo-European Chinese Arabic
  1                  12       1     13
  2                  10       2     5
  3                   8       1     1
cl.com Japanese/Turkish/Finnish/Magyar Others
  1                     1     18
  2                     2     19
  3                     1     9
> chisq.test(cl.com, flags$language)$p.value
[1] 0.0008972897

> table(cl.com, flags$religion)
cl.com Catholic Other Christian Muslim Buddhist Hindu
  1         7        13      23      2      2
  2        23        19      10      5      0
  3        10        28      3      1
cl.com Ethnic Marxist Others
  1        18        5      1
  2         7        8      2
  3         2        2      1
> chisq.test(cl.com, flags$religion)$p.value
[1] 1.557342e-06

```

The  $p$ -values of the  $\chi^2$  tests suggest to reject the null hypothesis of independence and if we look at the cross-tabulations some interesting comments can be done. Cluster 1 is mainly characterized by the levels Muslim and Ethnic for religion and its countries come from Africa and Asia. In this cluster, we observe the highest percentage of Arabic language countries. A lot of Asian countries are from the Middle East area except for China and Vietnam.

```
> row.names(flags)[flags$landmass == "Asia" &
+                   cl.com == 1]
[1] "Afghanistan"   "Bangladesh"    "China"
[4] "Iran"           "Iraq"          "Jordan"
[7] "Kampuchea"      "Kuwait"        "North-Yemen"
[10] "Oman"           "Pakistan"      "Saudi-Arabia"
[13] "Sri-Lanka"      "Syria"         "UAE"
[16] "USSR"           "Vietnam"
```

Countries from Europe and some others from Asia are assigned to Cluster 2. In particular, most of the Asian countries are from the Far East area.

```
> row.names(flags)[flags$landmass == "Asia" &
+                   cl.com == 2]
[1] "Bahrain"        "Bhutan"        "Brunei"
[4] "Burma"          "Japan"         "Laos"
[7] "Lebanon"         "Malaysia"      "Maldives-Islands"
[10] "Mongolia"       "Singapore"     "South-Korea"
[13] "South-Yemen"    "Taiwan"        "Thailand"
[16] "Turkey"
```

The religion is often Catholic or Other Christian with the highest percentage of Marxist. Finally, Cluster 3 contains countries from N. America with religion labeled Other Christian and a high percentage of English and, to a lesser extent, Spanish languages. The quite large frequency of Spanish countries can be explained by noting that the level N. America refers to countries from North and Central America.

```
> row.names(flags)[flags$landmass == "Asia" &
+                   cl.com == 3]
[1] "Hong-Kong"      "India"         "Israel"
[4] "Nepal"           "North-Korea"   "Qatar"
```

Since the three clusters have a meaningful interpretation, we believe that such a partition should be preferred to the one resulting from the average linkage method.

Finally, to further characterize the three clusters, we graphically discover some distinctive features for the flags of the countries assigned to these clusters by considering the flag colors.

```

> plot.data <- matrix(NA, nrow = 3, ncol = 7)
> rownames(plot.data) <-
+   c("Clus. 1", "Clus. 2", "Clus. 3")
> colnames(plot.data) <- rep(NA, 7)
> cn <- 0
> for (j in c(10:16)){
+   cn <- cn + 1
+   counts <- table(flags[, j], cl.com)
+   colnames(plot.data)[cn] <- names(flags)[j]
+   plot.data[, cn] <- prop.table(counts, 2)[2, ] * 100
+ }
> barplot(plot.data, col = gray.colors(3),
+           border = "white", beside = T,
+           legend = rownames(plot.data),
+           xlab = "colours", ylab = "percentage")

```

The plot is given in Fig. 2.12. Almost all the flags assigned to Cluster 3 are blue and white colored, while red and white are quite uncommon if compared to the other clusters. The green color characterizes Cluster 1, while all the flags in Cluster 2 contain red.

### 2.5.3 One Further Case Study

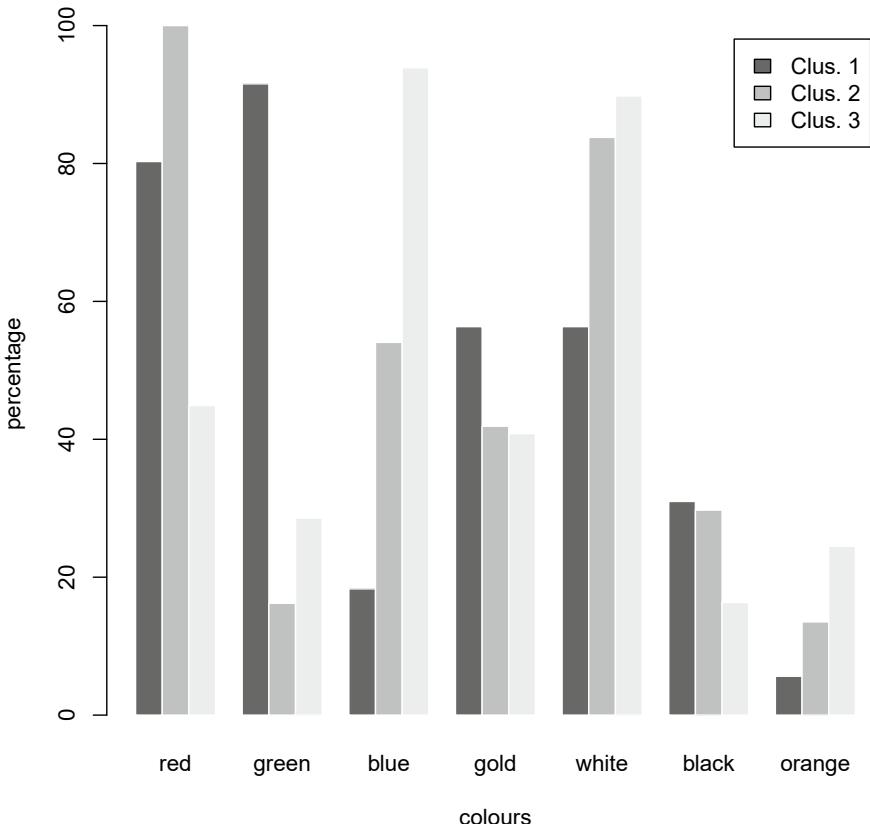
So far, we discussed the use of agglomerative clustering methods on three case studies. Apart from the use of a specific distance measure for the data at hand, no relevant differences emerged. In particular, the number of clusters was chosen by looking at the dendrogram and seeking the largest difference between two subsequent merging steps. This was possible because the size of data was quite small. In this section, we continue to investigate agglomerative hierarchical clustering methods by considering a bigger dataset. The size is still limited (the dataset contains 913 rows), but the example offers a deeper insight into the practical steps of an agglomerative hierarchical clustering analysis.

The analysis is conducted on the data frame `wiki4HE` [25] of the package **datasetsICR** containing the results of a survey on university faculty perceptions and practices of using Wikipedia as a teaching resource.

```
> data("wiki4HE")
> str(wiki4HE)
'data.frame': 913 obs. of 53 variables:
 $ AGE          : int  40 42 37 40 51 47 43 55 54 ...
 $ GENDER       : int  0 0 0 0 0 0 0 0 0 ...
 $ DOMAIN       : Factor w/ 7 levels "?","1","2",...
 $ PhD          : int  1 1 1 0 0 0 0 1 1 ...
 $ YEAREXP      : Factor w/ 37 levels "?","0","1",...
 $ UNIVERSITY   : int  1 1 1 1 1 1 1 1 1 ...
 $ UOC_POSITION : Factor w/ 7 levels "?","1","2",...
 $ OTHER_POSITION: Factor w/ 3 levels "?","1","2": ...
 $ OTHERSTATUS  : Factor w/ 8 levels "?","1","2",...
 $ USERWIKI     : Factor w/ 3 levels "?","0","1": ...
 $ PU1          : Factor w/ 6 levels "?","1","2",...
 $ PU2          : Factor w/ 6 levels "?","1","2",...
 $ PU3          : Factor w/ 6 levels "?","1","2",...
 $ PEU1         : Factor w/ 6 levels "?","1","2",...
 $ PEU2         : Factor w/ 6 levels "?","1","2",...
 $ PEU3         : Factor w/ 6 levels "?","1","2",...
 $ ENJ1         : Factor w/ 6 levels "?","1","2",...
 $ ENJ2         : Factor w/ 6 levels "?","1","2",...
 $ Qui          : Factor w/ 6 levels "?","1","2",...
 $ Qu2          : Factor w/ 6 levels "?","1","2",...
 $ Qu3          : Factor w/ 6 levels "?","1","2",...
 $ Qu4          : Factor w/ 6 levels "?","1","2",...
 $ Qu5          : Factor w/ 6 levels "?","1","2",...
 $ Vis1         : Factor w/ 6 levels "?","1","2",...
 $ Vis2         : Factor w/ 6 levels "?","1","2",...
 $ Vis3         : Factor w/ 6 levels "?","1","2",...
 $ Im1          : Factor w/ 6 levels "?","1","2",...
 $ Im2          : Factor w/ 6 levels "?","1","2",...
 $ Im3          : Factor w/ 6 levels "?","1","2",...
 $ SA1          : Factor w/ 6 levels "?","1","2",...
 $ SA2          : Factor w/ 6 levels "?","1","2",...
 $ SA3          : Factor w/ 6 levels "?","1","2",...
 $ Use1         : Factor w/ 6 levels "?","1","2",...
 $ Use2         : Factor w/ 6 levels "?","1","2",...
 $ Use3         : Factor w/ 6 levels "?","1","2",...
 $ Use4         : Factor w/ 6 levels "?","1","2",...
 $ Use5         : Factor w/ 6 levels "?","1","2",...
 $ Pf1          : Factor w/ 6 levels "?","1","2",...
 $ Pf2          : Factor w/ 6 levels "?","1","2",...
 $ Pf3          : Factor w/ 6 levels "?","1","2",...
 $ JR1          : Factor w/ 6 levels "?","1","2",...
 $ JR2          : Factor w/ 6 levels "?","1","2",...
 $ BI1          : Factor w/ 6 levels "?","1","2",...
 $ BI2          : Factor w/ 6 levels "?","1","2",...
 $ Inc1         : Factor w/ 6 levels "?","1","2",...
 $ Inc2         : Factor w/ 6 levels "?","1","2",...
 $ Inc3         : Factor w/ 6 levels "?","1","2",...
 $ Inc4         : Factor w/ 6 levels "?","1","2",...
 $ Exp1         : Factor w/ 6 levels "?","1","2",...
 $ Exp2         : Factor w/ 6 levels "?","1","2",...
```

```
$ Exp3      : Factor w/ 6 levels "?", "1", "2", ...
$ Exp4      : Factor w/ 6 levels "?", "1", "2", ...
$ Exp5      : Factor w/ 6 levels "?", "1", "2", ...
```

The first ten variables provide some information on the respondents and are not considered for clustering purposes. The clustering process is based on the remaining variables ( $p = 43$ ). All these variables take scores in the set  $\{1, 2, 3, 4, 5\}$ . Such values come from a Likert-type scale and the values range from strongly disagree/never (1) to strongly agree/always (5). Likert scale data are ordinal in nature, but are often treated as continuous in data analysis. Specifically, it is common to treat such scales as continuous variables under the assumption that there is an underlying measurement referring to a continuous variable. Therefore, Likert scale data are seen as ordinal approximations of continuous variables; for further insight, refer to, e.g., [15, 28, 33].



**Fig. 2.12** flags data: clusters by flag colors (complete linkage method,  $k = 3$  clusters)

In the current study, all the variables are managed as quantitative, and therefore we convert them from `factor` to numeric. This is done by creating the function `as.numeric.factor`. Before doing it, we observe that some missing data, denoted by "?", are present. We thus recode the label "?" with NA and then we exclude from the analysis the units having at least one missing value. All these pre-processing steps lead to a new data frame, called `wiki4HE.all` with  $n = 623$  rows.

```
> wiki4HE[wiki4HE == "?"] <- NA
> as.numeric.factor <- function(x)
+   {as.numeric(levels(x))[x]}
> for (nc in 11:53) {
+   wiki4HE[, nc] <-
+     as.numeric.factor(wiki4HE[, nc])
+ }
> wiki4HE.all <-
+   wiki4HE[complete.cases(wiki4HE[, 11:53]), 11:53]
> dim(wiki4HE.all)
[1] 623 43
```

For analysis purposes, we employed Ward's method and we compute the distance matrix by using the function `dist`. Then we run the function `hclust` and plot the obtained dendrogram in Fig. 2.13. Note that we do not standardize the data because the variables have all the same unit of measurement.

```
> D.wiki <- dist(wiki4HE.all)
> res.ward <- hclust(d = D.wiki, method = "ward.D2")
> plot(res.ward, labels = FALSE)
```

Apart from the overlapping labels of the units that make the units hard to be distinguished, the dendrogram offers some suggestions on the number of clusters. In particular, a cluster of units located in the left side of the dendrogram appears to be noticeably different from the remaining ones. The inspection of the right side of the dendrogram may suggest to consider a number of clusters varying from one to four. This choice may also depend on the desired level of segmentation. Generally speaking, the final choice should be based on both statistical and qualitative criteria. To this purpose, once a dendrogram is plotted, it might be helpful highlighting some clusters and/or focusing on a subset of clusters. For this purpose, some functions are available. In the following, a few examples are given in order to illustrate the corresponding code. Note, however, that such findings will not be used for selecting the optimal partition in the current case study. For instance, the function `rect.hclust` of the package `stats` draws rectangles around the branches of a dendrogram. Suppose that  $k = 3$  clusters are sought. The following code can be used to produce Fig. 2.14.

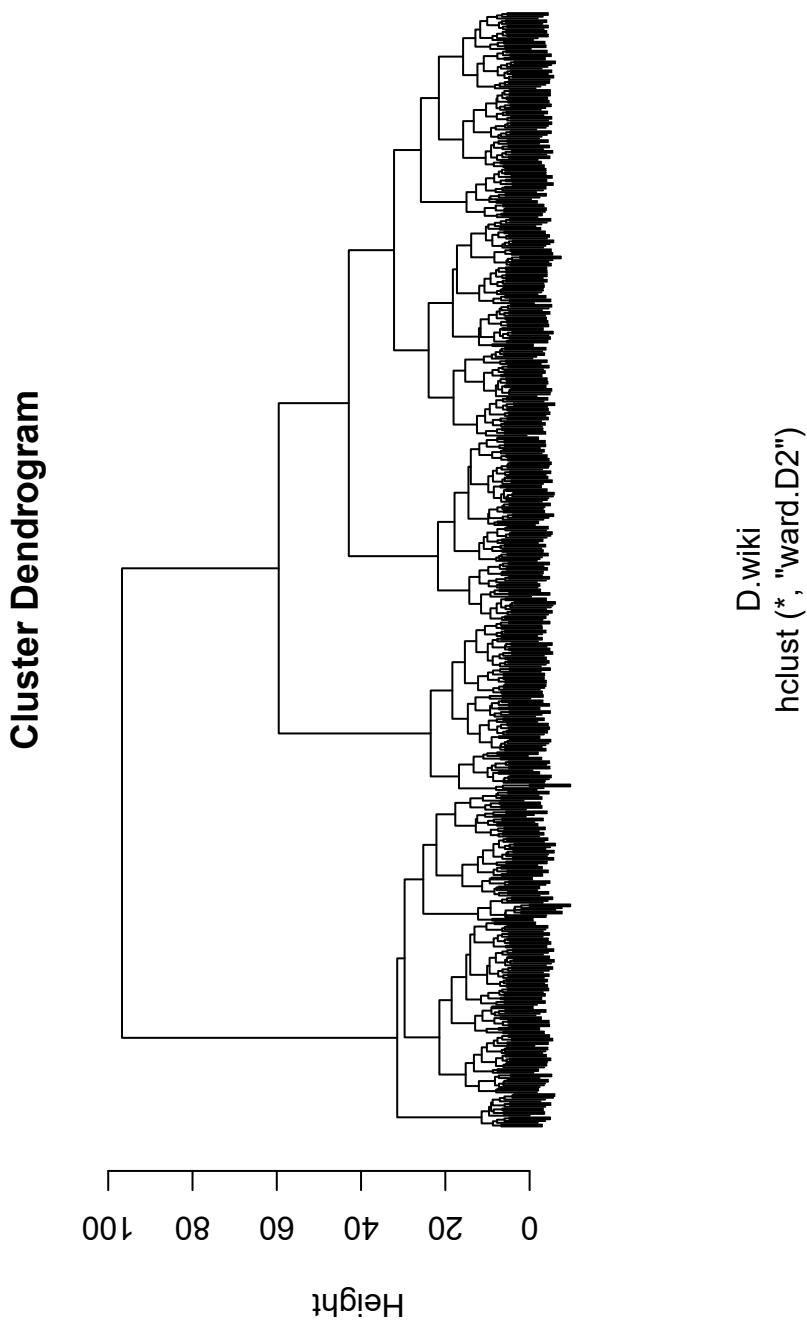


Fig. 2.13 `wiki14HE` data: dendrogram using Ward's method

```
> plot(res.ward, labels = FALSE)
> rect.hclust(res.ward, k = 3)
```

Suppose now that a partition with  $k = 4$  clusters is found and that, starting from the left of the dendrogram, Clusters 2 and 4 should be highlighted by green and blue boxes, respectively. This can be done by specifying the options `border` and `which` (Fig. 2.15).

```
> plot(res.ward, labels = FALSE)
> rect.hclust(res.ward, k = 4,
+               border = c("green", "blue"),
+               which = c(2, 4))
```

The selection of some branches of a dendrogram can be done by combining the functions `plot` and `cut` provided that the output of `hclust` is converted to an object of class `dendrogram`.

```
> dendr.hclust <- as.dendrogram(res.ward)
> class(dendr.hclust)
[1] "dendrogram"
```

By setting `h = 40` we obtain a summary of the dendrogram with  $k = 3$  clusters (the labels of the branches are located according to the first left unit assigned to every cluster). See Fig. 2.16.

```
> plot(cut(dendr.hclust, h = 40)$upper,
+       main = "Upper dendrogram of cut at height h = 40")
```

To inspect a specific cluster, for instance, Cluster 2, we can zoom on such a branch. The resulting plot is displayed in Fig. 2.17.

```
> plot(cut(dendr.hclust, h = 40)$lower[[2]],
+       main = "2nd branch of lower dendrogram with cut
+               at height h = 40")
```

Let us now discuss how to select the optimal solution. So far, some preliminary conclusions on the number of clusters have been found by looking at the dendrogram. In order to reach a final decision, one may be interested in using some cluster validity indices. Several measures are implemented in the function `NbClust` of the package **NbClust**. Note that the function offers an alternative way to perform agglomerative

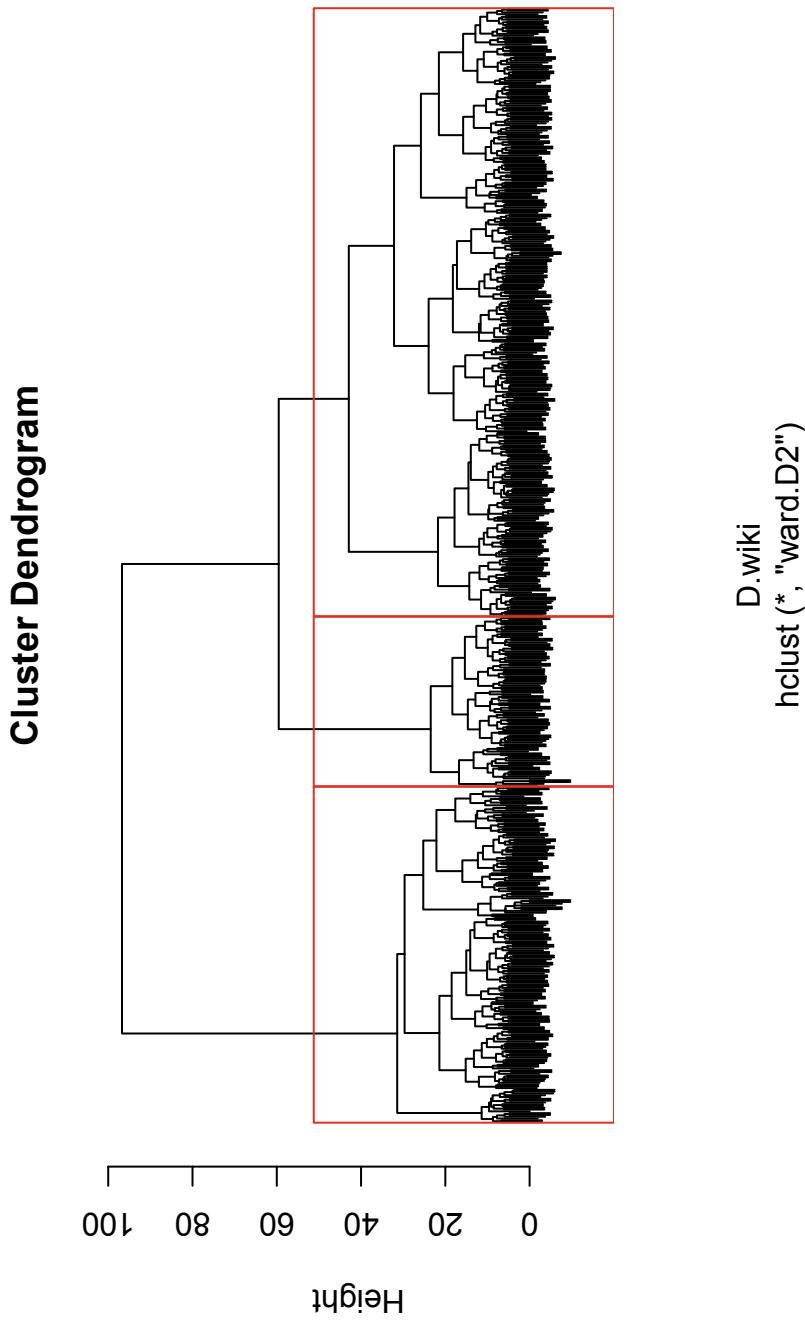


Fig. 2.14 `wiki14HE` data: dendrogram using Ward's method ( $k = 3$  clusters denoted by red rectangles)

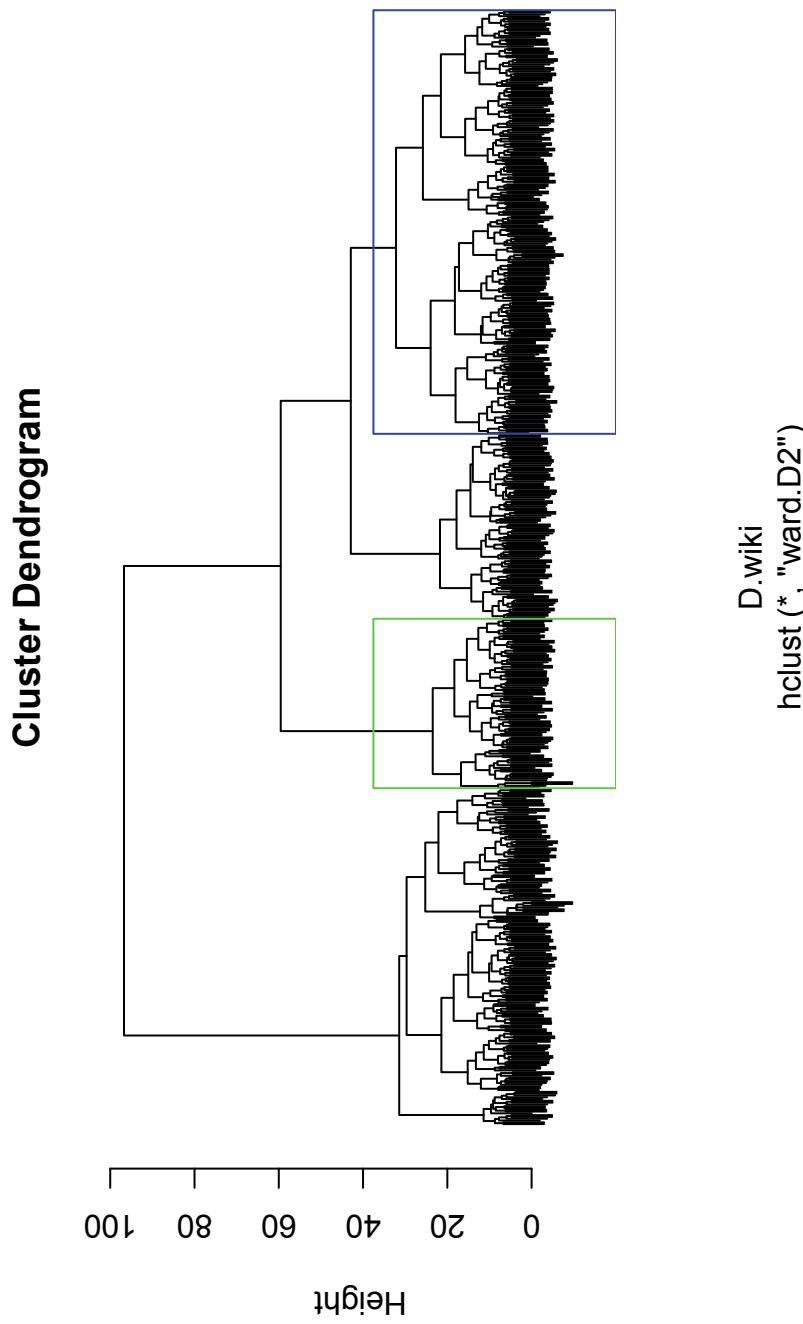


Fig. 2.15 `wiki4HE` data: dendrogram using Ward's method ( $k = 4$  clusters, Clusters 2 and 4 denoted by green and blue rectangles, respectively)

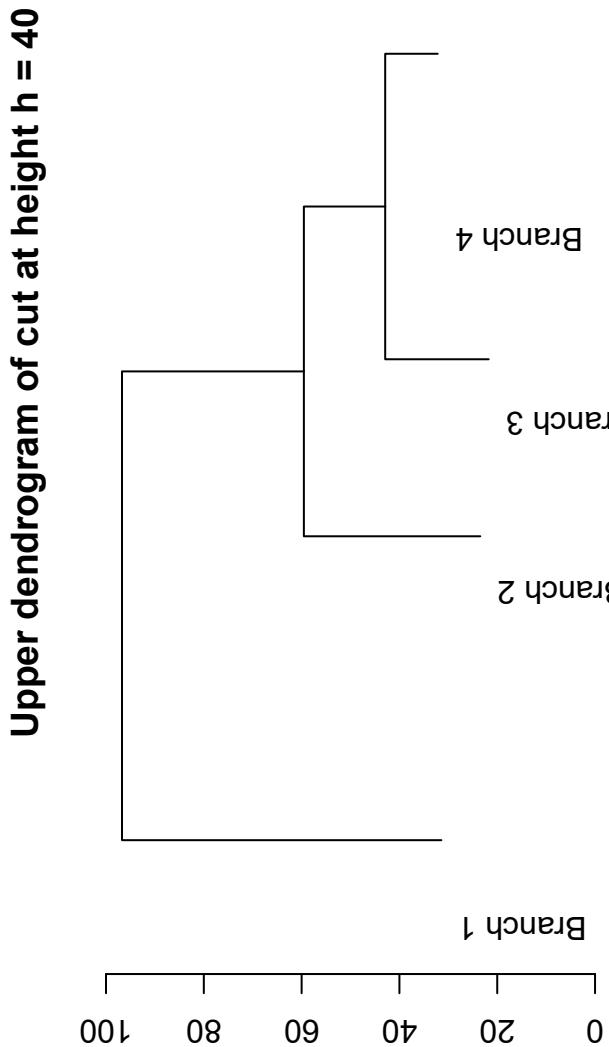
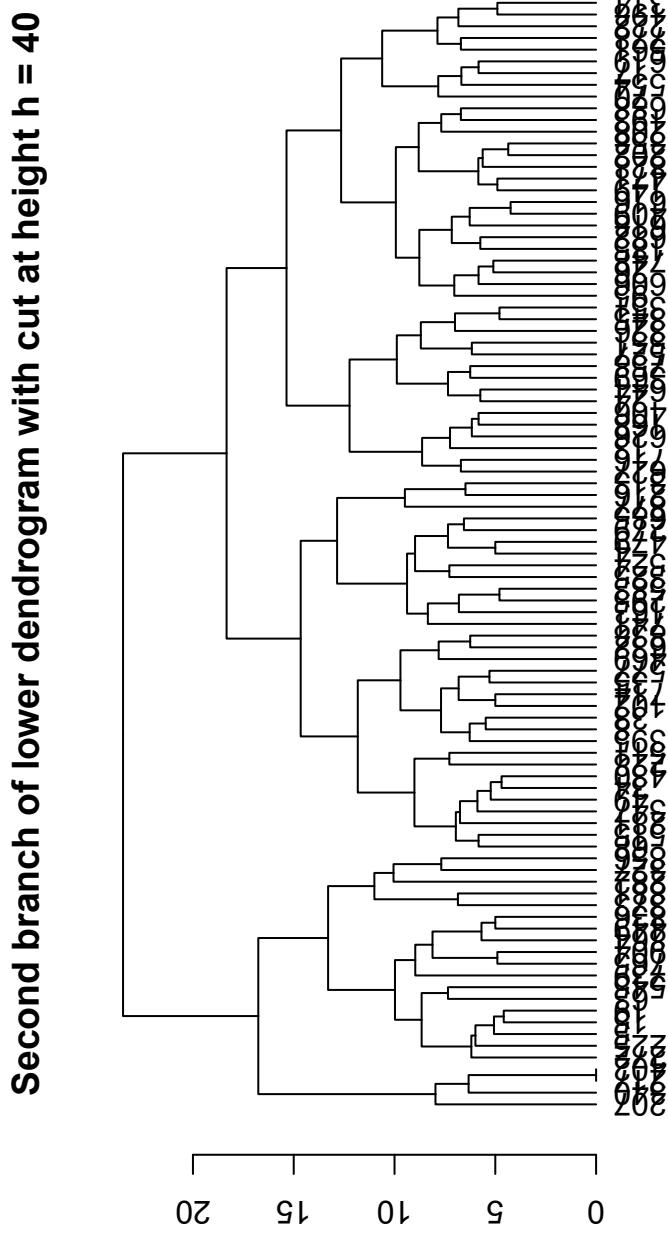


Fig. 2.16 wiki4HE data: upper dendrogram using Ward's method (cut at height  $h = 40$ )



**Fig.2.17** wiki4HE data: branch 2 of the dendrogram using Ward's method (cut at height  $h = 40$ )

hierarchical clustering in R. Specifically, the input of `NbClust` is neither the output of `hclust` nor that of `agnes`. `NbClust` requires either the dataset (option `data`) and the distance measure (option `distance`) or the distance matrix (option `diss`), the clustering method (option `method`), and some additional information on the minimal and maximal acceptable number of clusters (options `min.nc` and `max.nc`, respectively) and on the cluster validity indices to be computed (option `index`). About 30 indices are available and, if one is interested in calculating all the indices, `index = "all"` can be specified. This is the default option that we obviously omit in the script we run. Note that the required indices are computed for a number of clusters varying from `min.nc` to `max.nc`. Therefore, `NbClust` does not produce a hierarchy in a strict sense as it is for `agnes` and `hclust`.

In the following, we apply the function `NbClust` to `wiki4HE.all`. In other words, we repeat the previously described analyses varying the number of clusters from `min.nc = 2` to `max.nc = 15`. Consistently with the previous analyses, the Euclidean distance and Ward's method are selected. Note that, to specify Ward's method, the same notation of `hclust`, i.e., "`ward.D2`", should be adopted.

```
> library(NbClust)
> res.nbclust <- NbClust(data = wiki4HE.all,
+                           distance = "euclidean",
+                           method = "ward.D2",
+                           min.nc = 2, max.nc = 15)

*****
* Among all indices:
* 8 proposed 2 as the best number of clusters
* 8 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 2 proposed 9 as the best number of clusters
* 1 proposed 10 as the best number of clusters
* 2 proposed 15 as the best number of clusters

***** Conclusion *****

* According to the majority rule, the best number of
clusters is 2

*****
```

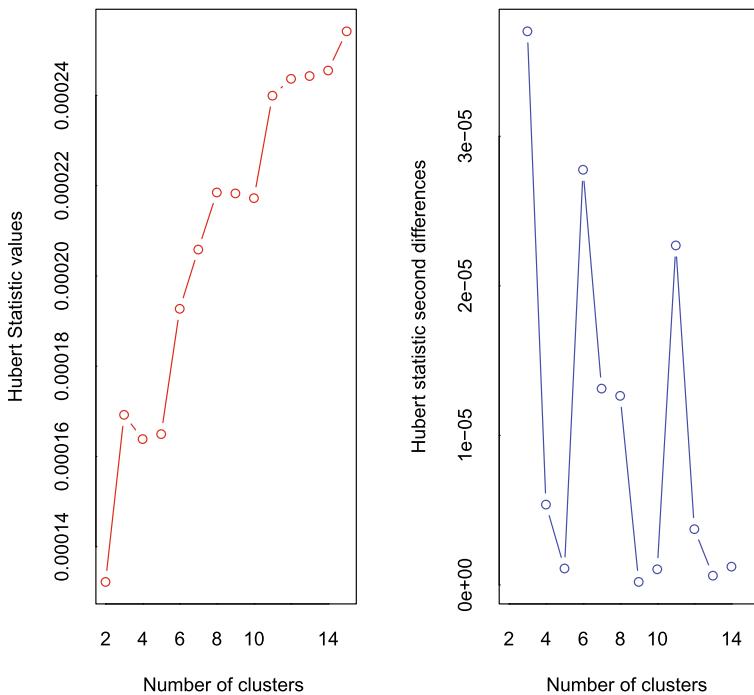
A summary of the NbClust output is reported. It offers the number of times in which a certain number of clusters is recommended and a conclusion based on the majority rule. Note that in this case, an ex-equo is registered and the majority rule should give  $k = 2$  or  $k = 3$ . For the sake of parsimony,  $k = 2$  is recommended. Furthermore, a graphical output referring to the indices labeled "hubert" [14] and "dindex" [20] is obtained (see Figs. 2.18 and 2.19, respectively).

Details on the computed indices can be found by inspecting the components of the list `res.nbclust`.

```
> names(res.nbclust)
[1] "All.index"           "All.CriticalValues"
[3] "Best.nc"             "Best.partition"
> res.nbclust$Best.nc
          KL        CH Hartigan       CCC
Number_clusters 2.000    2.0000 3.0000 2.0000
Value_Index      2.802   118.1267 22.1709 36.4757
                  Scott     Marriot   TrCovW
Number_clusters 3.0000 4.000000e+00 3.0
Value_Index      386.7282 5.257584e+109 100183.3
                  TraceW Friedman Rubin Cindex
Number_clusters 3.0000 3.0000 3.0000 10.0000
Value_Index      851.8762 9.7866 -0.4089 0.488
                  DB Silhouette Duda PseudoT2
Number_clusters 2.0000 2.0000 9.0000 9.0000
Value_Index      2.1048 0.1507 0.9247 9.7736
                  Beale Ratkowsky Ball PtBiserial
Number_clusters NA 2.0000 3.000 3.0000
Value_Index      NA 0.2667 4689.593 0.3927
                  Frey McClain Dunn Hubert SDindex
Number_clusters 1 2.0000 15.0000 0 2.0000
Value_Index      NA 0.6115 0.1871 0 1.0881
                  Dindex SDbw
Number_clusters 0 15.0000
Value_Index      0 0.6651
```

The component `Best.nc` gives the optimal values for all the indices and the corresponding suggested numbers of clusters. The values of the indices for all the possible numbers of clusters from `min.nc` to `max.nc` are reported in `All.index`. In our study, we investigate the solutions with  $k = 2$  and  $k = 3$ . The recommendation is to inspect a subset of "good" solutions according to the cluster validity indices instead of only the one that is automatically suggested by the function. The optimal partition according to the majority rule is found in `Best.partition`. As already observed, this is the solution with  $k = 2$ . First of all, we see the cluster sizes.

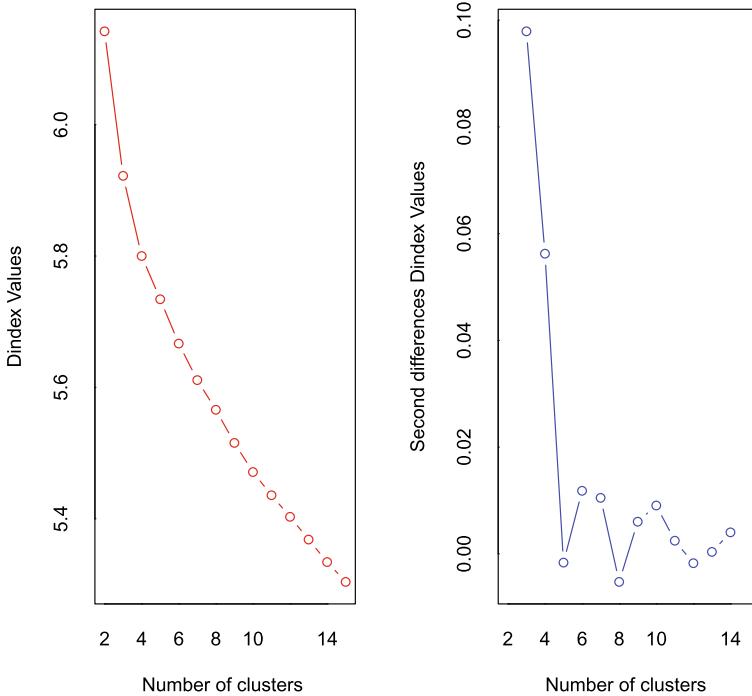
```
> table(res.nbclust$Best.partition)
 1   2
435 188
```



**Fig. 2.18** wiki4HE data: results of the index labeled "hubert" using Ward's method

We get an extremely large size cluster with more than two-thirds of the units. From a qualitative point of view, this is often unsatisfactory and the solution involved is often discarded. However, before analyzing the solution with  $k = 3$  clusters, we study the cluster mean values in Fig. 2.20.

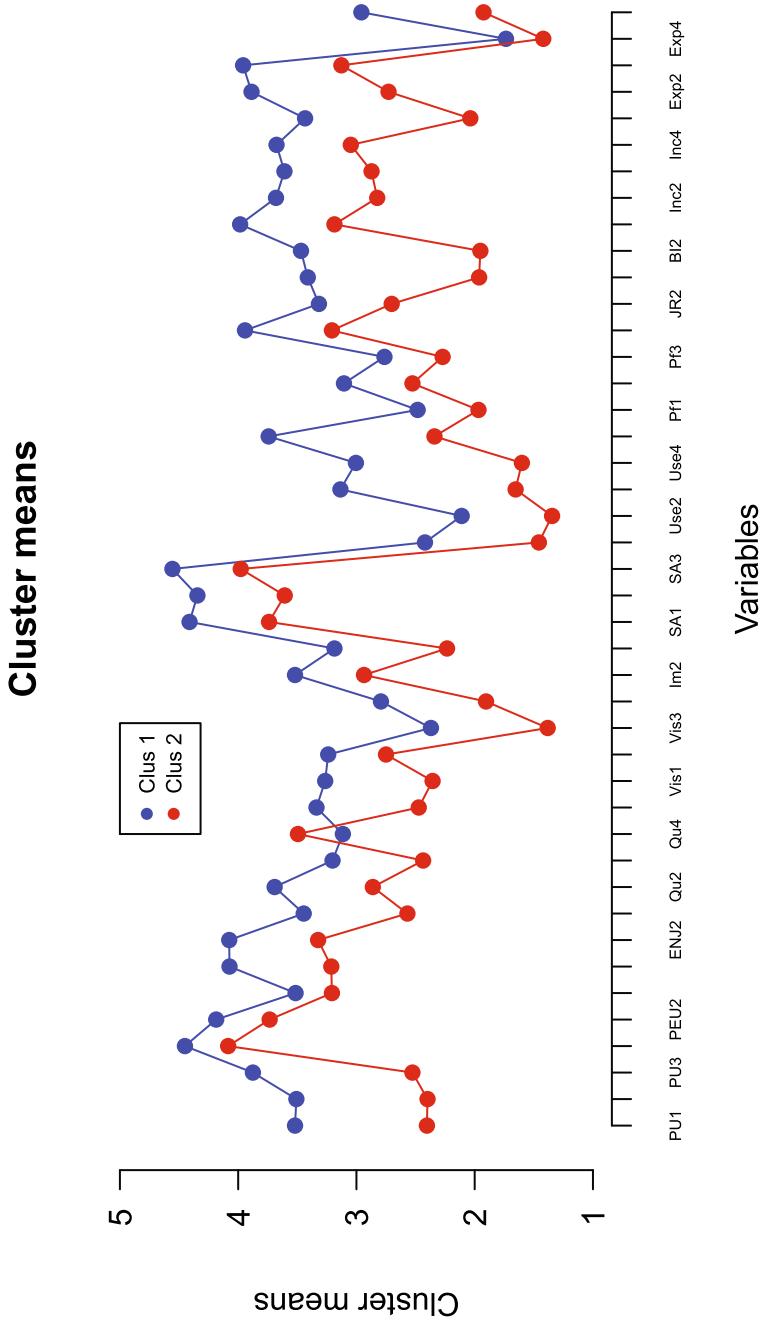
```
> k <- 2
> mean.cluster.2 <- t(sapply(X = 1:k,
+     FUN = function(nc)
+       apply(wiki4HE.all$res.nbclust$Best.partition
+             == nc, 1, 2, mean)))
> plot(mean.cluster.2[1, 1], type = "o", pch = 19,
+       xlab = "Variables", ylab = "Cluster means",
+       ylim = c(1, 5), axes = FALSE, col = "blue",
+       main = "Cluster means")
> lines(mean.cluster.2[2, 1], type = "o", pch = 19,
+       col = "red")
> axis(1, at = 1:ncol(mean.cluster.2),
+       lab = colnames(mean.cluster.2), cex.axis = 0.5)
> axis(2, at = 1:5, las = 1)
> legend(12, 5, c("Clus 1", "Clus 2"), pch = 19,
+       cex = 0.7, col = c("blue", "red"))
```



**Fig. 2.19** wiki4HE data: results of the index labeled "dindex" using Ward's method

We can deduce that, except for one variable (Qu4), the solution seems to distinguish the respondents who agree with the questions and those who disagree. Note that, for all the items, high values in the Likert scale, i.e., values equal to 4 and 5, mean positive perceptions on Wikipedia. The opposite comment holds for Qu4 corresponding to the item “In my area of expertise, Wikipedia has a lower quality than other educational resources.” Therefore, the first level of segmentation (the solution with  $k = 2$  clusters) discriminates the respondents with respect to the overall agreement on the use of Wikipedia as a teaching resource. This result may appear too simplistic and confirms that it may be convenient to investigate the next level of the hierarchy, i.e., the solution with  $k = 3$  clusters.

Since such a solution does not coincide with the one obtained according to the majority rule, there is no direct way to analyze the corresponding partition or, better, there are several ways to do it. One of them consists in running `hclust` or `agnes` and cutting the dendrogram so that the partition with the expected number of clusters is obtained. Nevertheless, if  $n$  is large, the strategy can be very time-consuming. An alternative strategy is based on `NbClust`, which does not allow that `min.nc` is equal to `max.nc` ( $= k$ ). However, we may use an index which is known from previous results to favor the solution with  $k = 3$ .



**Fig. 2.20** `wiki4HE` data: cluster means (Ward's method,  $k = 2$  clusters)

```
> res.nbclust.3 <- NbClust(data = wiki4HE.all,
+                               distance = "euclidean",
+                               method = "ward.D2",
+                               min.nc = 2, max.nc = 4,
+                               index = "hartigan")
> res.nbclust.3$Best.nc
Number_clusters      Value_Index
            3.0000          22.1709
```

To this purpose, we run `NbClust` varying the number of clusters in a smallest interval (of length 2), setting `index = "hartigan"` because we already know that such an index recommends  $k = 3$  clusters. The remaining options are kept fixed. We get the plot displayed in Fig. 2.21.

```
> table(res.nbclust.3$Best.partition)
  1   2   3
340 188  95
> k <- 3
> mean.cluster.3 <- t(sapply(X = 1:k,
+                               FUN = function(nc)
+                                 apply(wiki4HE.all[res.nbclust.3$Best.partition
+                                     == nc, ], 2, mean)))
> plot(mean.cluster.3[1, ], type = "o", pch = 19,
+       xlab = "Variables", ylab = "Cluster means",
+       ylim = c(1, 5), axes = FALSE, col = "blue",
+       main = "Cluster means")
> lines(mean.cluster.3[2, ], type = "o", pch = 19,
+        col = "red")
> lines(mean.cluster.3[3, ], type = "o", pch = 19,
+        col = "darkgreen")
> axis(1, at = 1:ncol(mean.cluster.3),
+       lab = colnames(mean.cluster.3), cex.axis = 0.5)
> axis(2, at = 1:5, las = 1)
> legend(12, 5, c("Clus 1", "Clus 2", "Clus 3"),
+         pch = 19, cex = 0.7,
+         col = c("blue", "red", "darkgreen"))
```

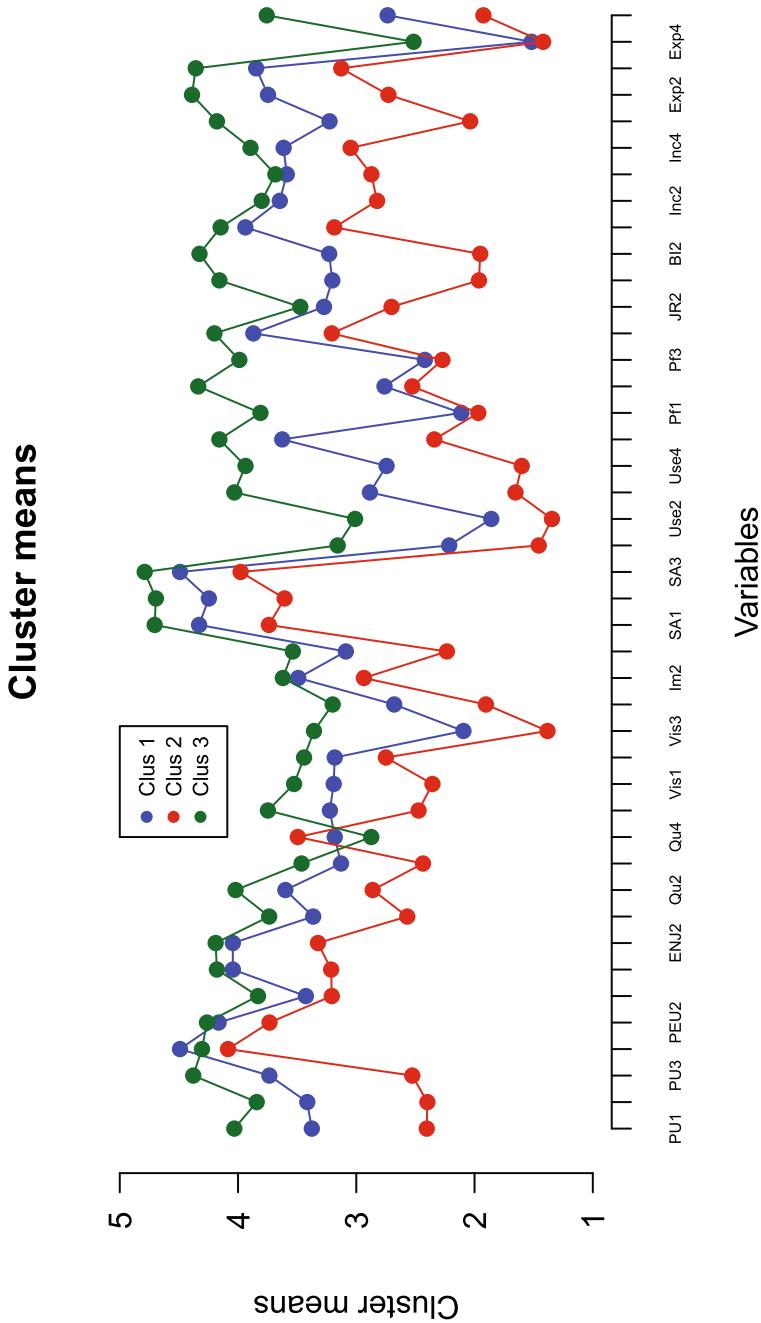
The big size cluster is split into two clusters of size 340 and 95, respectively. In Fig. 2.21, they are represented by blue and green colors, respectively. The green-colored cluster identifies respondents who are strongly in favor of using Wikipedia as a teaching resource. Such respondents have the highest mean values for all the variables, except for Qu4 and PEU1 ("Wikipedia is user-friendly"). The information gain of the three-cluster solution with respect to the two-cluster one is represented by the blue-colored cluster. It refers to respondents who are generally in favor of using Wikipedia as a teaching resource. However, such respondents appear to have much lower scores when compared to those of the green-colored cluster, especially for a few items. These are PEU3, Vis3, Use1, Use2, Pf1, Pf2, Pf3, and Exp4. In particular, it is interesting to see that Pf1 ("I contribute to blogs"), Pf2 ("I

actively participate in social networks”), and Pf3 (“I publish academic content in open platforms”) belong to the same group of survey items and concern the active role of Wikipedia users. Thus, this level of the hierarchy seems to distinguish those respondents who are in favor of using Wikipedia as a teaching resource, but do not play an active role in the Internet community.

We are satisfied with the solution with  $k = 3$  clusters. Nevertheless, we briefly look at the more complex solution with  $k = 4$  clusters. For this purpose, we can specify the option `index = "marriot"`.

```
> res.nbclust.4 <- NbClust(data = wiki4HE.all,
+                               distance = "euclidean",
+                               method = "ward.D2",
+                               min.nc = 3, max.nc = 5,
+                               index = "marriot")
> table(res.nbclust.4$Best.partition)
  1   2   3   4
237 188  95 103
> k <- 4
> mean.cluster.4 <- t(sapply(X = 1:k,
+                                FUN = function(nc)
+                                  apply(wiki4HE.all[res.nbclust.4$Best.partition
+                                     == nc, ], 2, mean)))
> plot(mean.cluster.4[1, ], type = "o", pch = 19,
+       xlab = "Variables", ylab = "Cluster means",
+       ylim = c(1, 5), axes = FALSE, col = "blue",
+       main = "Cluster means")
> lines(mean.cluster.4[2, ], type = "o", pch = 19,
+        col = "red")
> lines(mean.cluster.4[3, ], type = "o", pch = 19,
+        col = "darkgreen")
> lines(mean.cluster.4[4, ], type = "o", pch = 19,
+        col = "orange")
> axis(1, at = 1:ncol(mean.cluster.4),
+       lab = colnames(mean.cluster.4), cex.axis = 0.5)
> axis(2, at = 1:5, las = 1)
> legend(3, 2.2,
+         c("Clus 1", "Clus 2", "Clus 3", "Clus 4"),
+         pch = 19, cex = 0.7,
+         col = c("blue", "red", "darkgreen", "orange"))
+     )
```

By inspecting the cluster means in Fig. 2.22, we see that the new cluster (orange colored) detects some respondents with medium level of agreement for the items. Taking into account the suggestions made by the cluster validity indices and the cluster interpretation, the solution with  $k = 3$  clusters should be preferred and we consider it as the optimal one.



**Fig. 2.21** `wiki4HE` data: cluster means (Ward's method choosing  $k = 3$  clusters)

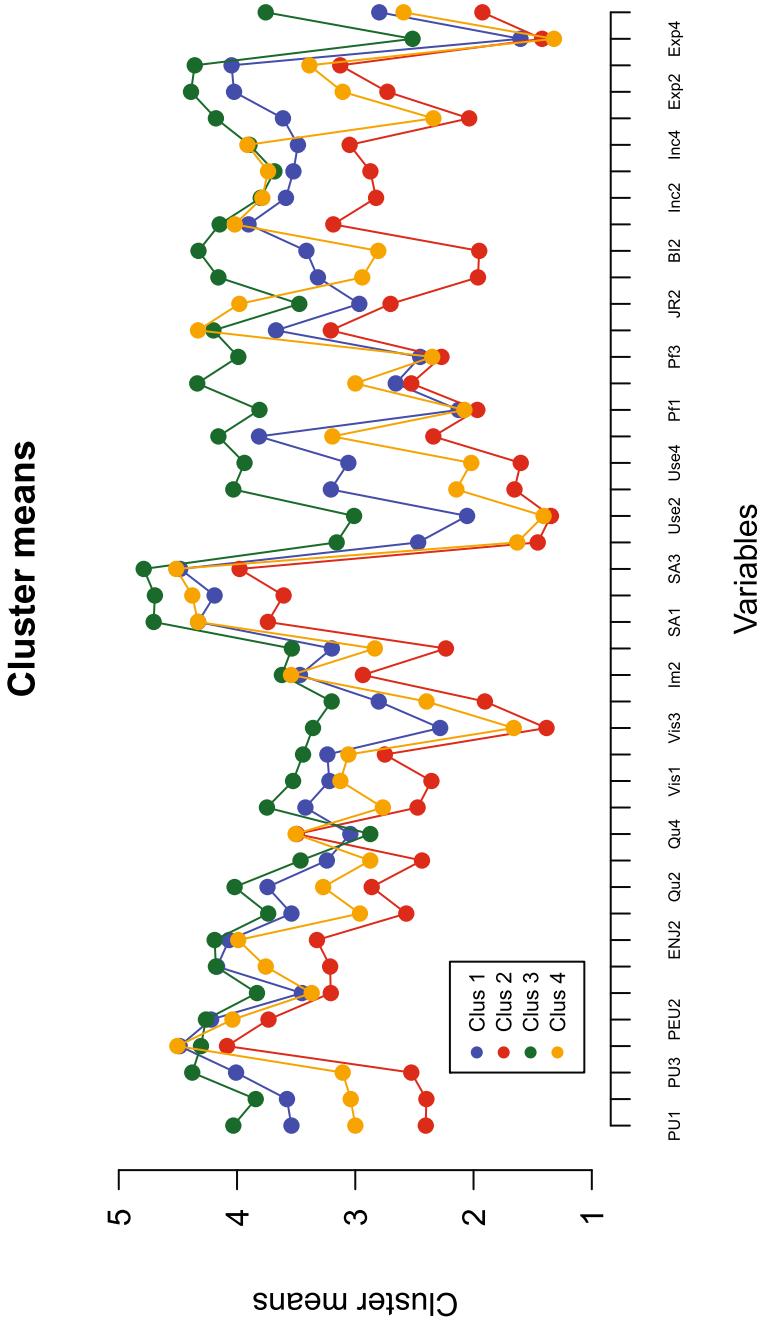


Fig. 2.22 *wiki4HE* data: cluster means (Ward's method choosing  $k = 4$  clusters)

The previous analysis is based on the package **NbClust**. It is worth to mention that interesting functions for computing cluster validity indices are also available in the packages **fpc** [13], **factoextra**, and **clValid** [2].

### 2.5.3.1 Missing Data

Once the optimal number of clusters has been chosen on the basis of the units that have no missing values, the obtained partition can be used as a simple tool to impute missing data on the remaining units. There exist a lot of alternative methods of imputation that perform better than the one we are going to implement, but they are out of the scope of this book. In this section, we show how to impute missing information by means of a clustering method. The idea is to estimate missing values for incomplete units by looking at those units that are complete and more similar. In this respect, we can assign the incomplete units to clusters with the minimum distance and then replace the missing values with the corresponding cluster means. We use this strategy to impute the missing values of `wiki4HE` by considering the solution of Ward's method with  $k = 3$  clusters, i.e., by considering `res.nbclust.3` and the cluster means `mean.cluster.3`.

To assign the units with missing values to clusters, we compute the distances between such units (contained in the data frame `wiki4HE.missing`) and the cluster means. This step is performed by using the Euclidean distance, already used to derive the distance matrix `D.wiki`.

```
> wiki4HE.missing <- wiki4HE[ !complete.cases(
+                                     wiki4HE[, 11:53]), 11:53]
> n.missing <- nrow(wiki4HE.missing)
> diss.cl <- rep(NA, length = nrow(mean.cluster.3))
> cluster.missing <- rep(NA, length = n.missing)
> for (i in 1:n.missing)
+ {
+   unit <- wiki4HE.missing[i, ]
+   SqEuclDist <- (unit[col(mean.cluster.3)] -
+                   mean.cluster.3)^2
+   diss.cl <- sqrt(apply(SqEuclDist, 1, sum,
+                         na.rm = TRUE))
+   cluster.missing[i] <- which(diss.cl == min(diss.cl))
+ }
```

The object `cluster.missing` is a vector containing the cluster assignment for the units with missing values. Such units belong to the closer clusters, in terms of cluster means, computed by using non-missing information. Then, for each unit, we can impute the missing values by using the cluster mean information.

```
> for (i in 1:n.missing)
+ {
+   for (j in 1:ncol(wiki4HE.missing))
+   {
+     if (is.na(wiki4HE.missing[i, j]))
+     {
+       wiki4HE.missing[i, j] <-
+         mean.cluster.3[cluster.missing[i], j]
+     }
+   }
+ }
```

To clarify the goal of the script, we inspect row 290 of `wiki4HE.missing`. Such a row had several missing values, one of them refers to the column `PEU3` (by writing `wiki4HE.missing[290, "PEU3"]` we got NA). The cluster means for variable `PEU3` are the following ones.

```
> mean.cluster.3[, "PEU3"]
[1] 3.426471 3.207447 3.831579
```

By computing the Euclidean distance using the non-missing values between the unit in row 290 and the three centers, we get the following result.

```
> diss.cl
[1] 5.157957 5.947080 8.436657
```

The closest mean is the one of Cluster 1. Thus, by using the previous for-loop, the missing value is replaced by the mean value of `PEU3` for such a cluster.

```
> wiki4HE.missing[290, "PEU3"]
[1] 3.426471
```

Obviously, the previous code provided (single) imputation for all the missing values.

```
> all(complete.cases(wiki4HE.missing))
[1] TRUE
```

## 2.6 Functions for Divisive Clustering

This section presents some case studies for two divisive hierarchical clustering methods. The first method is called DIANA [18] and it is probably the most famous divisive procedure. It is a peculiarity of R since divisive clustering methods are implemented in statistical software only rarely. The second method is MONA [18]. As we are going to see, MONA is specially tuned for binary data.

### 2.6.1 *diana*

In this section, we perform divisive hierarchical clustering by means of the DIANA method, implemented in the function `diana` of the package `cluster`. For this purpose, we consider the dataset `customers` [1] of the package `datasetsICR` on a sample of customers characterized by  $p = 6$  continuous variables, giving the annual spending related to different types of goods. Two more variables are categorical and provide information on the customer channel (two levels: Horeca, i.e., Hotel/Restaurant/-Café, Retail) and region (three levels: Lisbon, Oporto, Other). The categorical variables do not play an active role in the clustering process, but they will be used ex post to aid cluster interpretation. Some preprocessing is required in order to recode the categorical variables as objects of class factor.

```
> library(datasetsICR)
> data("customers")
> names(customers)
[1] "Channel"           "Region"              "Fresh"
[4] "Milk"               "Grocery"             "Frozen"
[7] "Detergents_Paper"  "Delicassen"
> customers$Channel <- as.factor(customers$Channel)
> levels(customers$Channel) <- c("Horeca", "Retail")
> customers$Region <- as.factor(customers$Region)
> levels(customers$Region) <- c("Lisbon", "Oporto",
+                                "Other")
> str(customers)
'data.frame':   440 obs. of  8 variables:
 $ Channel       : Factor w/ 2 levels "Horeca",...
 $ Region        : Factor w/ 3 levels "Lisbon",...
 $ Fresh          : int  12669 7057 6353 13265 ...
 $ Milk           : int  9656 9810 8808 1196 5410 ...
 $ Grocery        : int  7561 9568 7684 4221 7198 ...
 $ Frozen         : int  214 1762 2405 6404 3915 ...
 $ Detergents_Paper: int  2674 3293 3516 507 1777 ...
 $ Delicassen     : int  1338 1776 7844 1788 5185 ...
```

The function `diana` can be used by specifying either a distance matrix or a data matrix (or a data frame). This should be specified as the argument of the option `x` provided that `diss` is equal to `TRUE` or `FALSE` (default), respectively. The syntax of `diana` is similar to that of other functions, for instance, `agnes`. Note that `diana` can be applied to cluster categorical or mixed data. In this case, a distance matrix should be given by means of, e.g., the function `daisy` setting `metric = "gower"`. In fact, when `diss = FALSE`, `diana` computes the distances in terms of the Euclidean or Manhattan distances, as the Gower distance is not implemented.

We now apply `diana` to `customers`, limiting our attention to the quantitative variables. We decide not to standardize the data. Since the variables refer to annual spendings, this choice implies that the larger spendings play a more relevant role in the clustering process.

```
> apply(customers[, 3:8], 2, mean)
      Fresh           Milk          Grocery
    12000.298      5796.266     7951.277
      Frozen Detergents_Paper       Delicassen
    3071.932        2881.493     1524.870
```

Therefore, we can see that the partition mainly depends on `Fresh` and, to a lesser extent, `Grocery` and `Milk`.

```
> library(cluster)
> res.diana <- diana(x = customers[, 3:8],
+                      diss = FALSE)
```

The object `res.diana` is of class `diana`. Such a class inherits from the class `twins` allowing to represent the obtained hierarchy by a dendrogram. The three most relevant components of `res.diana` are `height`, a vector containing the diameters of the clusters before splitting; `merge`, a matrix describing (by row) the different splits at the various steps; and `order`, a vector useful for graphical purposes. It is easy to see that such components are the same as for `agnes`. It follows that the analysis of the `diana` solution is similar to the one of any hierarchical clustering function. For instance, the dendrogram can be displayed. To this purpose, two plots can be performed by applying the function `plot` to an object of class `diana`. To get the dendrogram (see Fig. 2.23), the option `which.plots = 2` should be given.

```
> plot(res.diana, main = "DIANA",
+       labels = FALSE, which.plots = 2)
```

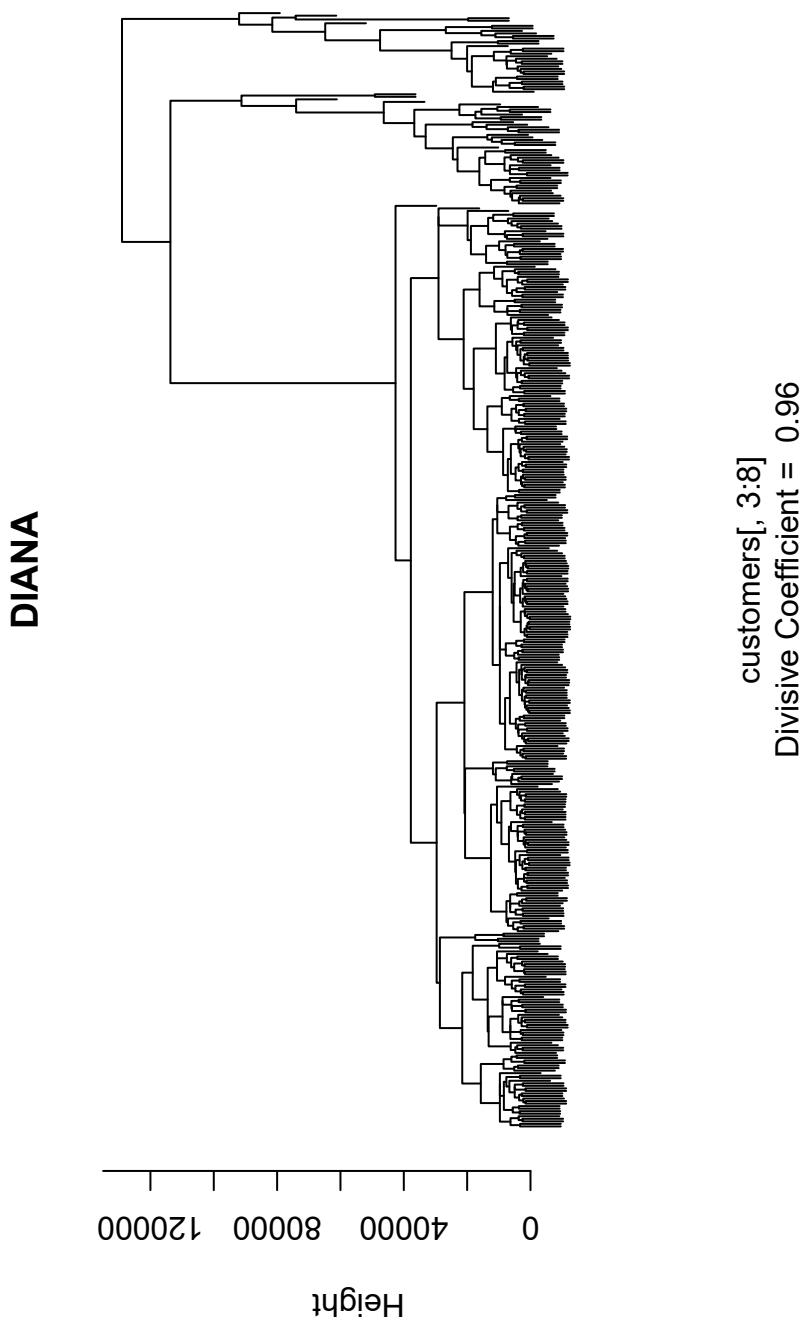


Fig. 2.23 `customers` data: Dendrogram using DIANA

By the visual inspection of the dendrogram, it is reasonable to consider  $k = 3$  clusters.

```
> cluster.diana <- cutree(res.diana, k = 3)
> table(cluster.diana)
cluster.diana
  1   2   3
364  44  32
> mean.cluster <- t(sapply(X = 1:3,
+     FUN = function(nc)
+       apply(customers[cluster.diana == nc, 3:8],
+             2, mean)))
> rownames(mean.cluster) <- paste("Clus.", 1:3)
> round(mean.cluster, 2)
      Fresh    Milk  Grocery  Frozen
Clus. 1  9779.56  4108.77  5525.34 2721.40
Clus. 2  6329.89 17463.43 27204.80 1662.14
Clus. 3  45058.03  8949.22  9072.69 8997.69
      Detergents_Paper Delicassen
Clus. 1            1740.50    1235.02
Clus. 2            12626.20   2116.95
Clus. 3            2461.28    4007.84
> table(cluster.diana, customers$Channel)
cluster.diana Horeca Retail
      1     270     94
      2      0     44
      3     28      4
> chisq.test(cluster.diana, customers$Channel)$p.value
[1] 1.593597e-23
> table(cluster.diana, customers$Region)
cluster.diana Lisbon Oporto Other
      1     64     38    262
      2      8      8    28
      3      5      1    26
> chisq.test(cluster.diana, customers$Region)$p.value
[1] 0.2942478
```

Cluster 1 has the largest size and identifies customers characterized by a medium profile in terms of annual spending. In fact, the mean values for such a cluster are very close to the overall mean. The peculiarity of customers assigned to Cluster 2 is the extremely high annual spending for a subset of types of goods, namely, Milk, Grocery, and Detergents\_Paper. Finally, Cluster 3 includes customers with a uniformly high annual spending for all the types of goods. Further comments can be made by considering the categorical variables, with particular reference to Channel. The  $p$ -value of the  $\chi^2$  test suggests to reject the null hypothesis of independence

between the obtained partition and this variable. In particular, by looking at the cross-tabulation, we can discover that all the units assigned to Cluster 2 present the level Retail. On the contrary, a large frequency can be observed for the level Horeca in Cluster 1 and, especially, in Cluster 3.

### 2.6.2 **mona**

The package **cluster** offers one more function to perform a divisive cluster analysis. Such a function is called **mona**, the name of which comes from the first two words of MONothetic Analysis clustering of binary variables. The function **mona** can be applied only when all the variables are binary.

The MONA procedure is quite different from the hierarchical methods we have previously discussed. In fact, the main idea of MONA is to select one of the binary variables and to divide the units accordingly. For instance, let us suppose that the variable  $X$  taking values  $x_1$  and  $x_2$  is selected for splitting the units: the units with  $X = x_1$  belong to one cluster and those such that  $X = x_2$  to the other one. Starting from one cluster with  $n$  units, this process is repeated until  $n$  singleton clusters are obtained. Therefore, at each step, only one variable is used for splitting, and hence the adjective monothetic. It is important to note that, at each step, the chosen variable splits all the clusters. In order to choose the splitting variables, the most centrally located variable is sought, i.e., the variable such that the sum of similarities to all the other variables is as large as possible. See, for further details, [18].

Note that, if all the variables are binary, MONA is not the only one possible choice. In fact, we can compute the distance matrix by means of **daisy** setting **metric** = "gower" and then apply any other function. For this reason, in the following, we apply **mona** comparing its solution with that of **agnes**.

To this purpose, the congressional voting records dataset available in the package **fclust** [8, 9] is used. It refers to the 1984 voting records for 475 U.S. House of Representative congressmen on  $p = 16$  key votes. The key votes are objects of class **factor** with three levels, namely, **y** referring to the types of votes "voted for", "paired for", and "announced for"; **n** to "voted against", "paired against", and "announced against"; **yn** to "voted present", "voted present to avoid conflict of interest", and "did not vote or otherwise make a position known". Therefore, **yn** is related to unknown preferences for some key votes. We thus remove all the records containing at least one **yn** value obtaining a dataset with  $n = 435$  units. One more variable, not used for clustering purposes, splits the congressmen into Democrats and Republicans (**class**).

```
> library(fclust)
> data("houseVotes")
> names(houseVotes)
[1] "class"
[2] "handicapped-infants"
[3] "water-project-cost-sharing"
[4] "adoption-of-the-budget-resolution"
[5] "physician-fee-freeze"
[6] "el-salvador-aid"
[7] "religious-groups-in-schools"
[8] "anti-satellite-test-ban"
[9] "aid-to-nicaraguan-contras"
[10] "mx-missile"
[11] "immigration"
[12] "synfuels-corporation-cutback"
[13] "education-spending"
[14] "superfund-right-to-sue"
[15] "crime"
[16] "duty-free-exports"
[17] "export-administration-act-south-africa"
> level.drop <- droplevels(houseVotes, exclude = "yn")
> houseVotesComplete <-
+   level.drop[complete.cases(level.drop), ]
> X.houseVotesComplete <- houseVotesComplete[, -1]
> dim(X.houseVotesComplete)
[1] 232 16
```

We can now apply MONA.

```
> res.mona <- mona(x = X.houseVotesComplete)
```

The output, called `res.mona`, is an object of class `mona`. It is a list with five components. From such components, it is not obvious how to know the cluster memberships. If a researcher is interested in analyzing the obtained clusters, things are less natural when compared to the previous analyses. Let `mona.obj` and `n.spl` be an object of class `mona` and the number of splits performed before obtaining the desired solution, respectively. The following code may help.<sup>1</sup>

---

<sup>1</sup>Own elaboration from:

[https://www.reddit.com/r/rstats/comments/1rj7fr/clustering\\_using\\_mona\\_from\\_cluster\\_package/](https://www.reddit.com/r/rstats/comments/1rj7fr/clustering_using_mona_from_cluster_package/).

```

> cluster.mona <- function(mona.obj, n.spl){
+   indices <- c()
+   results.vector <- c()
+   for(i in 1:n.spl){
+     indices <- append(indices,
+                        which(mona.obj$step == i))
+   }
+   sorted.indices <- sort(indices)
+   sorted.indices <- append(sorted.indices,
+                            length(mona.obj$order))
+   low <- 1
+   for(j in 1:2^n.spl){
+     high <- sorted.indices[j]
+     cluster.size <-
+       length(mona.obj$order[low:high])
+     results.vector <- append(results.vector,
+                               rep(j, cluster.size))
+     print(paste("Cluster = ", j, ", count = ",
+                cluster.size))
+     low <- high +1
+   }
+   df <- data.frame(order = mona.obj$order,
+                     results.vector = results.vector)
+   df <- df[order(df$order), ]
+   return(df$results.vector)
+ }
```

In our study, the aim is to investigate whether the method distinguishes the Democrats and the Republicans, i.e., we are interested in  $k = 2$  clusters, and therefore we run the function `clus.mona` setting `n.spl = 1`.

```

> clus.mona <- cluster.mona(res.mona, 1)
[1] "Cluster = 1 , count = 128"
[1] "Cluster = 2 , count = 104"
> table(clus.mona, houseVotesComplete$class)
clus.mona democrat republican
      1          25         103
      2          99          5
> library(mclust)
> adjustedRandIndex(clus.mona,
+                     houseVotesComplete$class)
[1] 0.5476924
```

We discover that, to some extent, Cluster 1 and Cluster 2 refer to the Republicans and Democrats, respectively.

We briefly compare the partition with  $k = 2$  clusters found by means of MONA with the one resulting from standard hierarchical clustering. We thus run `agnes` using the default options (hence considering the average linkage method).

```

> D.houseVotes <- daisy(x = X.houseVotesComplete,
+                         metric = "gower")
> res.agnes <- agnes(D.houseVotes, diss = TRUE)
> clus.agnes <- cutree(res.agnes, k = 2)
> table(clus.mona, clus.agnes)
      clus.agnes
clus.mona 1 2
  1 122 6
  2 2 102
> adjustedRandIndex(clus.mona, clus.agnes)
[1] 0.8662483
> adjustedRandIndex(clus.agnes,
+                     houseVotesComplete$class)
[1] 0.627407

```

We may notice that the two partitions are closely related and the one obtained by agnes performs slightly better than the one obtained by mona in distinguishing the two parties.

## References

1. Abreu, N.: Analise do perfil do cliente Recheio e desenvolvimento de um sistema promocional. Mestrado em Marketing, ISCTE-IUL, Lisbon (2011)
2. Brock, G., Pihur, V., Datta, S., Datta, S.: clValid: An R package for cluster validation. *J. Stat. Softw.* **25**, 1–22 (2008). <http://www.jstatsoft.org/v25/i04/>
3. Charrad, M., Ghazzali, N., Boiteau, V., Niknafs, A.: NbClust: An R package for determining the relevant number of clusters in a data set. *J. Stat. Softw.* **61**, 1–36 (2014). <https://www.jstatsoft.org/v061/i06>
4. Chou, C.-H., Su, M.-C.: A modified version of the  $K$ -means algorithm with a distance based on cluster symmetry. *IEEE Trans. Pattern Anal. Mach. Intell.* **23**, 674–680 (2001)
5. D’Orazio, M.: StatMatch: statistical matching or data fusion. R package version 1.3.0 (2019). <https://CRAN.R-project.org/package=StatMatch>
6. Drost, H.G.: Philentropy: information theory and distance quantification with R. *J. Open Source Softw.* **3**, 765 (2018)
7. Dua, D., Graff, C.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine (2019). <http://archive.ics.uci.edu/ml>
8. Ferraro, M.B., Giordani, P.: A toolbox for fuzzy clustering using the R programming language. *Fuzzy Sets Syst.* **279**, 1–16 (2015)
9. Ferraro, M.B., Giordani, P., Serafini, A.: fclust: an R package for fuzzy clustering. *R J.* **11**(1), 205–233 (2019)
10. Giordani, P., Ferraro, M.B., Martella, F.: datasetsICR: Datasets from the Book “An Introduction to Clustering with R”, R package version 1.0 (2020). <https://CRAN.R-project.org/package=datasetsICR>
11. Gower, J.C.: A comparison of some methods of cluster analysis. *Biometrics* **23**, 623–637 (1967)
12. Gower, J.C.: A general coefficient of similarity and some of its properties. *Biometrics* **27**, 857–874 (1971)
13. Hennig, C.: fpc: Flexible Procedures for Clustering. R package version 2.2-5 (2020). <https://CRAN.R-project.org/package=fpc>
14. Hubert, L., Arabie, P.: Comparing partitions. *J. Classif.* **2**, 193–218 (1985)

15. Jamieson, S.: Likert scales: How to (ab)use them. *Med. Educ.* **38**, 1212–1218 (2004)
16. Johnson, S.C.: Hierarchical clustering schemes. *Psychometrika* **32**, 241–254 (1967)
17. Kassambara, A., Mundt, F.: Factoextra: extract and visualize the results of multivariate data analyses. R Package Version 1.0.7 (2020). <https://CRAN.R-project.org/package=factoextra>
18. Kaufman, L., Rousseeuw, P.J.: *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York (1990)
19. Lance, G.N., Williams, W.T.: A general theory of classificatory sorting strategies. I. *Hierar. Syst. Comput. J.* **9**, 373–380 (1967)
20. Lebart, L., Morineau, A., Piron, M.: *Statistique Exploratoire Multidimensionnelle*. Dunod, Paris (2000)
21. Lucas, A.: Amap: another multidimensional analysis package. R Package Version 0.8-18 (2019). <https://CRAN.R-project.org/package=amap>
22. Mahalanobis, P.C.: On the generalised distance in statistics. *Proc. Natl. Inst. Sci. India* **2**, 49–55 (1936)
23. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.: Cluster: cluster analysis basics and extensions. R Package Version 2.1.0 (2019). <https://CRAN.R-project.org/package=cluster>
24. Melnykov, I., Melnykov, V.: On  $k$ -means algorithm with the use of Mahalanobis distances. *Stat. Probabil. Lett.* **84**, 88–95 (2014)
25. Meseguer, A., Aibar, E., Lladós, J., Minguillón, J., Lerga, M.: Factors that influence the teaching use of Wikipedia in higher education. *J. Assoc. Inf. Sci. Tech.* **67**, 1224–1232 (2015)
26. Moro, S., Rita, P., Coelho, J.: Stripping customers' feedback on hotels through data mining: the case of Las Vegas strip. *Tourism Manage. Persp.* **23**, 41–52 (2017)
27. Murtagh, F., Legendre, P.: Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion? *J. Classif.* **31**, 274–295 (2014)
28. Norman, G.: Likert scales, levels of measurement and the "laws" of statistics. *Adv. Health Sci. Educ.* **15**, 625–632 (2010)
29. Paradis E., Schliep, K.: Ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics* **35**, 526–528 (2018)
30. R Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna (2020). <https://www.R-project.org>
31. Savje, F.: Distances: tools for distance metrics. R package version 0.1.8 (2019). <https://CRAN.R-project.org/package=distances>
32. Scrucca, L., Fop, M., Murphy, T.B., Raftery, A.E.: mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *R J.* **8**(1), 205–233 (2016)
33. Sullivan, G., Artino Jr., A.R.: Analyzing and interpreting data from Likert-type scales. *J. Grad. Med. Educ.* **5**, 541–542 (2013)
34. Ward, J.H.: Hierarchical grouping to optimize an objective function. *J. Am. Stat. Assoc.* **58**, 236–244 (1963)

# Chapter 3

## Non-Hierarchical Clustering



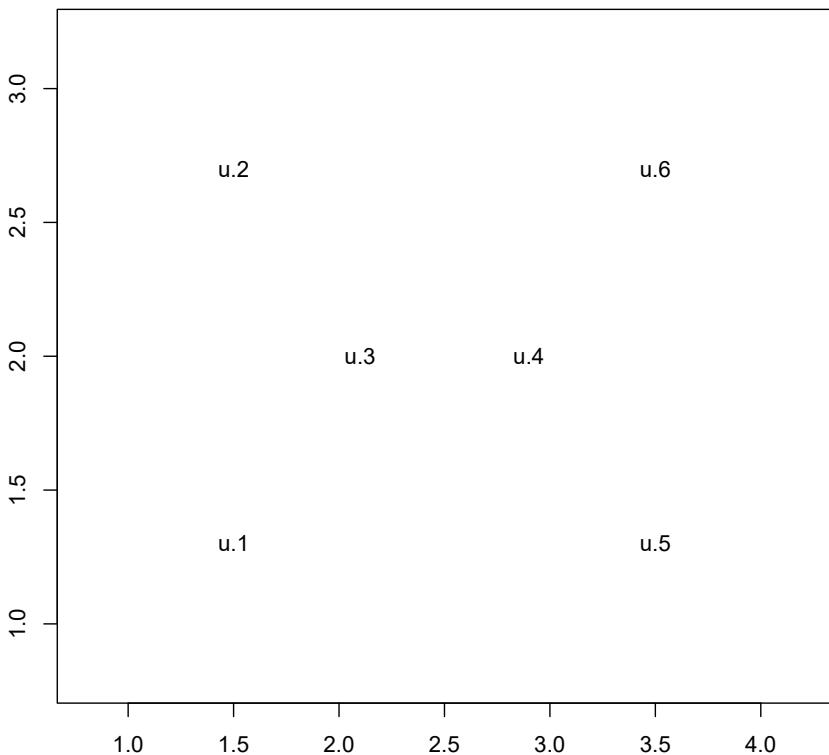
### 3.1 Introduction

In the previous chapter, we focused on hierarchical clustering methods. They are very intuitive and easy to implement, as the number of clusters should not be set in advance. In this respect, the class of non-hierarchical clustering methods we are going to present is more complex and counter-intuitive, since the number of clusters must be given before performing the analysis. This might be a very hard choice especially when no preliminary knowledge on the data is available. The above difficulty may prevent the use of non-hierarchical methods. Nevertheless, such methods are widely applied in practice. At least two valuable properties justify the need for non-hierarchical methods.

- Non-hierarchical methods do not require the computation of a distance matrix. This is particularly relevant for moderately large datasets for which the computation of a distance matrix of order  $(n \times n)$  is extremely time- and memory-consuming.
- In agglomerative hierarchical clustering, once two units are merged in a cluster, they cannot be split during the following steps. Similarly, in divisive hierarchical clustering, if two units belong to different clusters, they cannot be joined together. In other words, hierarchical clustering procedures lead to a hierarchy of partitions such that the partition at a given step depends on the partitions obtained at the previous steps. In some cases, this may produce a solution in contrast with the standard human thinking.

Before going on, we consider the following artificial example to clarify the previously mentioned possible limitations of hierarchical methods. It refers to a simple dataset, with  $n = 6$  units and  $p = 2$  variables, plotted in Fig. 3.1.

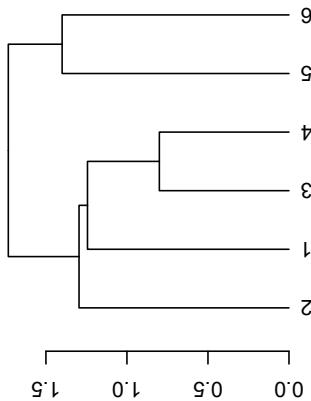
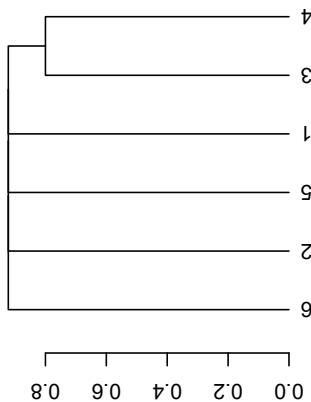
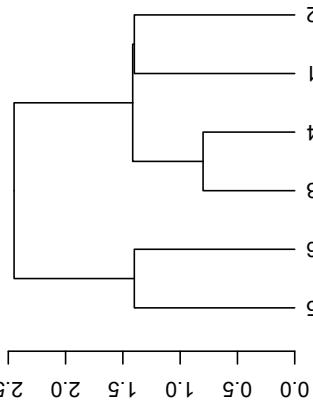
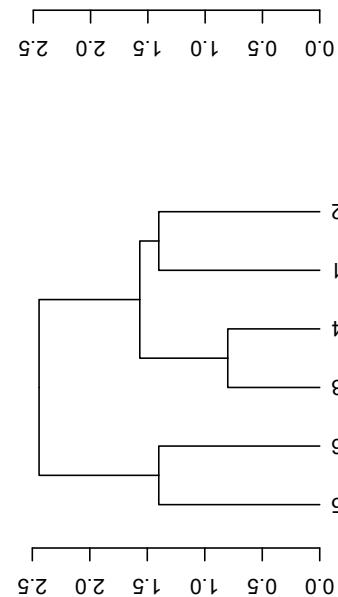
```
> X <- matrix(c(1.5, 1.5, 2.1, 2.9, 3.5, 3.5, 1.3,
+               2.7, 2.0, 2.0, 1.3, 2.7), ncol = 2)
> plot(X, type = "n", xlim = c(0.8, 4.2),
+       ylim = c(0.8, 3.2), xlab = "", ylab = "")
> text(X[, 1], X[, 2], paste("u.", 1:6, sep = " "))
```



**Fig. 3.1** x data: scatterplot

Suppose now that we are interested in detecting a partition with  $k = 2$  clusters. By looking at Fig. 3.1, it is reasonable to assume that one cluster is formed by units n.1–n.3 and the other one by units n.4–n.6. However, the most common agglomerative clustering methods fail to discover such a partition (Fig. 3.2).

```
> D <- dist(X)
> par(mfrow = c(2, 2))
> res <- hclust(D, method = "single")
> plot(as.dendrogram(res), hang = -0.1,
+       main = "Single linkage")
> res <- hclust(D, method = "average")
> plot(as.dendrogram(res), hang = -0.1,
+       main = "Average linkage")
> res <- hclust(D, method = "complete")
> plot(as.dendrogram(res), hang = -0.1,
+       main = "Complete linkage")
> res <- hclust(D, method = "ward.D2")
> plot(as.dendrogram(res), hang = -0.1,
+       main = "Ward's method")
```

**Average linkage****Single linkage****Ward's method****Complete linkage**

**Fig. 3.2** Synthetic data: dendograms using four agglomerative hierarchical methods

The results reported in Fig. 3.2 can be explained by noting that, during a previous step, units n.3 and n.4 are merged and, therefore, still remain in the same cluster in the next steps.

### 3.2 $k$ -Means

The  $k$ -Means algorithm [8, 15] is likely the most famous non-hierarchical clustering method. It can be applied when all observed variables are quantitative. It looks for the best partition of  $n$  units in  $k$  clusters. In order to define the concept of best partition, it is useful to decompose the Total sum of squares ( $T$ ) into the sum of two terms, namely, the Within-cluster sum of squares ( $W$ ) and the Between-cluster sum of squares ( $B$ ). We have

$$T = W + B, \quad (3.1)$$

where

- $T = \sum_{i=1}^n \sum_{j=1}^p (x_{ij} - \bar{x}_j)^2$ ,
- $W = \sum_{g=1}^k W_g$  with  $W_g = \sum_{i=1}^{n_g} \sum_{j=1}^p (x_{ij} - \bar{x}_{gj})^2$ ,
- $B = \sum_{g=1}^k n_g (\bar{x}_{gj} - \bar{x}_j)^2$ ,

being  $n_g$  the number of units belonging to cluster  $g$ ,  $\bar{x}_j = \frac{\sum_{i=1}^n x_{ij}}{n}$  the overall mean value of the  $j$ -th variable, and  $\bar{x}_{gj} = \frac{\sum_{i=1}^{n_g} x_{ij}}{n_g}$  the mean value for the  $j$ -th variable in the  $g$ -th cluster.

The quantity  $W$  allows for evaluating the quality of a partition. In fact, it is expressed as the sum of  $W_g$ , where  $W_g$  gives the Within-cluster sum of squares for cluster  $g$ . When all the units take the same values, that is,  $x_{ij} = \bar{x}_{gj}$ , we have  $W_g = 0$ . It follows that the best partition of  $n$  units in  $k$  clusters can be defined as

$$\min W \quad (3.2)$$

or, equivalently,  $\max B$ . The introduction of  $W$  helps us to compare the  $k$ -Means algorithm with Ward's method. In fact, it can be shown that, at each step, Ward's method merges the pair of clusters such that the smallest increase of  $W$  is obtained. Since the partitions at the various steps are nested, it is not guaranteed that the partition with  $k$  clusters resulting from Ward's method satisfies (3.2). In other words, the  $k$ -Means solution should be preferred to that obtained by Ward's method in terms of  $W$  because the latter method does not always produce a value of  $W$  as small as the former one.

In principle, the partition obtained by fulfilling (3.2) should be found by calculating  $W$  for all the possible partitions of  $n$  units into  $k$  clusters. In practice, this is unfeasible since the number of possible partitions is intractable (for instance, if  $n = 20$  and  $k = 4$ , there are about 45 billions of partitions). The best partition can be found by solving the following constrained minimization problem:

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{H}} & \sum_{i=1}^n \sum_{g=1}^k \sum_{j=1}^p u_{ig} (x_{ij} - h_{gj})^2 = \sum_{i=1}^n \sum_{g=1}^k u_{ig} d^2(\mathbf{x}_i, \mathbf{h}_g), \\ \text{s.t. } & u_{ig} \in \{0, 1\}, \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ & \sum_{g=1}^k u_{ig} = 1, \quad i = 1, \dots, n. \end{aligned} \quad (3.3)$$

The loss function in (3.3) is an equivalent formulation of  $W$  expressed in terms of the so-called allocation matrix  $\mathbf{U}$  of order  $(n \times k)$ . It is a binary matrix such that each row corresponds to a unit and it contains only one element equal to 1. This element denotes the cluster membership of the unit taking into account that the columns refer to clusters. Furthermore,  $\mathbf{H}$  is the prototype (centroid) matrix of order  $(k \times p)$ , with rows  $\mathbf{h}_g = (h_{g1}, \dots, h_{gp})$ ,  $g = 1, \dots, k$ . The rows of  $\mathbf{H}$  contain the cluster prototypes serving to characterize the clusters.

The optimal solution can be found by means of the following iterative algorithm:

1. Rationally or randomly choose  $k$  initial centroids, i.e., the centroid matrix  $\mathbf{H}$ .
2. Given  $\mathbf{H}$ , assign each unit to the cluster such that its distance from the centroid is minimum:

$$u_{ig} = \begin{cases} 1, & \text{if } g = \arg \min_{g'=1, \dots, k} d^2(\mathbf{x}_i, \mathbf{h}_{g'}), \\ 0, & \text{otherwise,} \end{cases} \quad (3.4)$$

for  $i = 1, \dots, n$ , and  $g = 1, \dots, k$ .

3. Given  $\mathbf{U}$ , compute the centroids:

$$\mathbf{h}_g = \frac{\sum_{i=1}^n u_{ig} \mathbf{x}_i}{\sum_{i=1}^n u_{ig}}, \quad g = 1, \dots, k. \quad (3.5)$$

4. Repeat steps 2 and 3 until there are no more changes in two consecutive iterations.

From (3.5), we can see that the rows  $\mathbf{h}_g$  contain the cluster-specific mean values of the  $p$  variables, and hence the term ‘‘centroid’’. The convergence criterion can be relaxed by fixing a maximum number of iterations. At each iteration, the loss function value does not increase, and hence an equally well or better partition in terms of  $W$  is found. However, it is not guaranteed that the global optimum is attained. In other words, upon convergence, it is not guaranteed that the partition with the lowest  $W$  is discovered. For this reason, it is customary to run the algorithm more than one time

considering different (random) starts; at the end of such a process, the solution with the lowest loss function value upon convergence is retained.

The most famous function implementing the  $k$ -Means clustering algorithm is the function `kmeans` of the package `stats`.

### 3.3 $k$ -Medoids

Another famous non-hierarchical clustering algorithm is the  $k$ -Medoids [12, 13]. When compared to the  $k$ -Means algorithm, at least two relevant differences may be noticed. The first and more important difference is related to the concept of medoid as opposed to that of centroid. In fact, the clusters are now interpreted in terms of medoids, a subset of observed units characterizing the clusters of size  $k \leq n$ . Therefore, the cluster prototypes are no longer fictitious entities, the centroids, computed as means of units assigned to the clusters, but real observed entities. This allows for a more intuitive cluster interpretation. The second difference is related to the first one and involves computational aspects, as computing the medoids is computationally heavier than computing the centroids.

The  $k$ -Medoids clustering algorithm can be formulated in terms of the following constrained minimization problem:

$$\begin{aligned} & \min_{\mathbf{U}, \mathbf{H}} \sum_{i=1}^n \sum_{g=1}^k u_{ig} d^2(\mathbf{x}_i, \mathbf{h}_g), \\ \text{s.t. } & u_{ig} \in \{0, 1\}, \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ & \sum_{g=1}^k u_{ig} = 1, \quad i = 1, \dots, n, \\ & \{\mathbf{h}_1, \dots, \mathbf{h}_g, \dots, \mathbf{h}_k\} \subseteq \{\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_n\}. \end{aligned} \tag{3.6}$$

By comparing (3.3) and (3.6) we discover that the only difference can be found in the latter constraint on the medoids, while the loss functions and the constraints on the allocation matrix  $\mathbf{U}$  remain the same. Note that last row in (3.6) implies that  $\mathbf{H}$  is no longer the centroid matrix, but the medoid matrix of order  $(k \times p)$  containing, as rows, the medoids  $\mathbf{h}_g$ ,  $g = 1, \dots, k$ .

An iterative algorithm similar to  $k$ -Means can be implemented for solving the problem in (3.6). It consists of the following steps:

1. Rationally or randomly choose  $k$  initial medoids, i.e., the medoid matrix  $\mathbf{H}$ .
2. Given  $\mathbf{H}$ , assign each unit to the cluster such that its distance from the medoid is minimum:

$$u_{ig} = \begin{cases} 1, & \text{if } g = \arg \min_{g'=1,\dots,k} d^2(\mathbf{x}_i, \mathbf{h}_{g'}), \\ 0, & \text{otherwise,} \end{cases} \tag{3.7}$$

for  $i = 1, \dots, n$ , and  $g = 1, \dots, k$ .

3. Given  $\mathbf{U}$ , compute the medoids:

$$\mathbf{h}_g = \arg \min_{i=1, \dots, n} \sum_{i'=1}^n u_{ig} d^2(\mathbf{x}_i, \mathbf{x}_{i'}), \quad g = 1, \dots, k. \quad (3.8)$$

4. Repeat steps 2 and 3 until there are no more changes in two consecutive iterations.

The update of the medoids differs from the one of the centroids. In particular, for each cluster, the medoid is the observed unit that minimizes the sum of (squared) distances between that unit and all the units belonging to the cluster. This clarifies the reason why the  $k$ -Medoids algorithm is computationally cumbersome. At each iteration, it essentially requires the computation of the distances among all the units. Thus, its use may be prevented for increasing values of  $n$ . As for  $k$ -Means, local optima may occur and more than one (random) start may be recommended.

The iterative algorithm is usually described by mentioning the so-called BUILD and SWAP phases. The units that are tentatively defined as medoids are placed into the set  $S$  of the selected units. Letting  $O$  denote the set of all the observed units, the set  $U = O \setminus S$  contains the unselected units. In order to find the  $k$ -Medoids partition, in the BUILD phase, a collection of  $k$  units are selected for the initial set  $S$ . Then, in the SWAP phase, one tries to improve the quality of the clustering by exchanging selected with unselected units passing from  $S$  to  $U$  and vice versa.

The  $k$ -Medoids clustering algorithm is also known as the Partitioning Around Medoids (PAM) algorithm [13]. Such a name is used in the most common function implementing the  $k$ -Medoids algorithm: the function `pam` of the package `cluster` [16].

### 3.4 `kmeans`

This section is devoted to the application of the  $k$ -Means clustering algorithm by the function `kmeans` of the package `stats`. In particular, we analyze data on the NBA players with respect to their statistics for the regular season 2018–19 [18] available in the data frame `NBA.48`, contained in the package `datasetsICR` [7]. Suppose that a team manager is interested in acquiring new players for the forthcoming season and want to reach a decision on the basis of the player statistics registered during the current year. By clustering, (s)he aim to discover a cluster with the best players according to certain criteria.

```
> library(datasetsICR)
> data("NBA.48")
> names(NBA.48)
[1] "PLAYER"   "TEAM"     "AGE"      "GP"       "W"
[6] "L"         "MIN"     "PTS"      "FGM"      "FGA"
[11] "FG . ."   "X3 PM"   "X3 PA"    "X3 P . ." "FTM"
[16] "FTA"      "FT . ."  "OREB"     "DREB"     "REB"
[21] "AST"      "TOV"     "STL"      "BLK"      "PF"
[26] "FP"       "DD2"    "TD3"     "X . . ."
```

Note that NBA.48 contains statistics normalized per 48 min. This is useful in order to detect clusters of players with similar performance from a qualitative point of view. For clustering purposes, we consider a subset of the quantitative variables related to the players' performance. These are points (PTS), field goals attempted (FGA), field goals percentage (FG.), three-point field goals attempted (X3PA), three-point field goals percentage (X3P.), free throws attempted (FTA), free throws percentage (FT.), offensive rebounds (OREB), defensive rebounds (DREB), assists (AST), turnovers (TOV), steals (STL), and blocks (BLK). Moreover, we exclude from the analysis those players who played a limited number of minutes during the regular season, using the threshold of 12 min per match on average. This choice is motivated by the fact that such players can be outliers with a limited playing time and excellent statistics normalized per 48 min. Note that such an information is available in the data frame NBA.game of the package **datasetsICR**. The data frame NBA.game contains the same statistics as for NBA.48, but in this case these are not normalized. Thus, we obtain the following dataset:

```
> NBA <- NBA.48[, c(1, 8, 10, 11, 13, 14, 16, 17, 18,
+                      19, 21, 22, 23, 24)]
> data("NBA.game")
> NBA <- NBA[NBA.game$MIN >= 12, ]
> row.names(NBA) <- NBA[, 1]
> NBA <- NBA[, -1]
> str(NBA)
'data.frame': 403 obs. of 13 variables:
 $ PTS : num 22.7 21.9 22.5 15.9 19.6 13.5 20.7 ...
 $ FGA : num 19.1 19.5 17.6 12.4 16.5 12.8 15.6 ...
 $ FG. : num 44.9 40.1 53.5 43.3 40.5 35.7 44.5 ...
 $ X3PA: num 6.3 9.4 4.9 5.9 5.9 10.4 4.5 6.3 3.6 ...
 $ X3P.: num 34.9 33.9 36 34.3 36.3 32.3 48 36.3 ...
 $ FTA : num 4.6 3.7 2.3 3.6 4.9 1.1 5.8 6.7 4.6 ...
 $ FT. : num 73.1 82 82.1 86.7 82.3 92.3 79.7 64.8
 ...
 $ OREB: num 2.4 0.4 2.9 2.3 1 0.4 1.8 4.9 4.6 3.9
 ...
 $ DREB: num 8.1 4.6 8.2 10.4 7.2 3.5 4.2 8.3 7.4 ...
 $ AST : num 5.3 6.5 6.9 2.2 4.5 1.6 7 2.7 2.7 1.5
 ...
 $ TOV : num 3 3 2.5 1.5 2.3 1.1 3.8 3 2 1.4 ...
 $ STL : num 1 1.6 1.4 1.4 1.4 1.4 2.2 0.8 0.6 0.9
 ...
 $ BLK : num 1 1 2.1 0.7 0.7 0.5 0.8 2.1 1.6 0.7 ...
```

Since the variables have different units of measurement, these should be standardized before applying the  $k$ -Means algorithm. The function `kmeans` does not allow to standardize the data. This can be done by using the function `scale`.

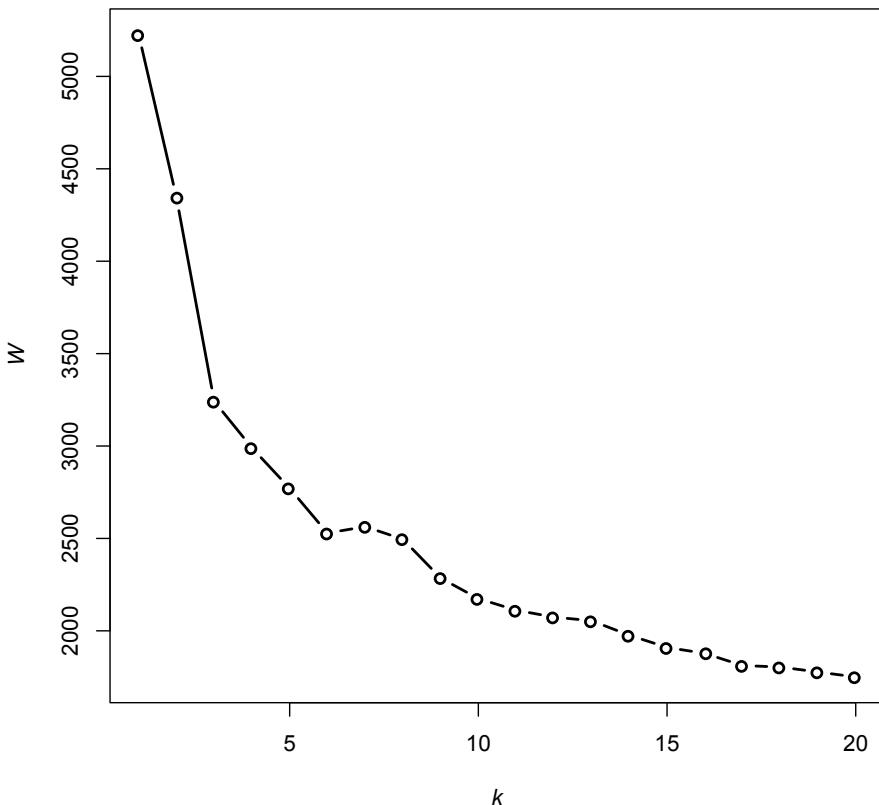
```
NBA.Z <- scale(NBA, center = TRUE, scale = TRUE)
```

We do not know in advance the number of clusters. For this reason, we look for different  $k$ -Means solutions by applying `kmeans` to `NBA.Z` and varying the number of clusters  $k$  from 2 to 20. This can be set by means of the option `centers`. In order to find a subset of reasonably good solutions, we inspect the  $W$  values in Fig. 3.3. Later, we will describe all the components of the function `kmeans`. For the time being, it is important to know that  $W$  can be found in the output component `tot.withinss`.

```
> n <- dim(NBA.Z) [1]
> k.min <- 2
> k.max <- 20
> wss <- numeric()
> wss[1] <- (n - 1)*sum(apply(NBA.Z, 2, var))
> for(k in k.min:k.max) {
+ wss[k] <-
+   kmeans(NBA.Z, centers = k)$tot.withinss
+ }
> plot(1:k.max, wss,
+       xlab = expression(italic(k)),
+       ylab = expression(italic(W)),
+       type = "b", lwd = 2)
> diff(wss)
[1] -881.169150 -1102.667248 -251.827311
[4] -219.142866 -242.191991  34.734789
[7] -67.847526 -210.408384 -111.054578
[10] -62.882312 -37.043878 -19.731489
[13] -78.517493 -66.010441 -32.156429
[16] -65.695989 -8.197986 -26.825519
[19] -27.083279
```

Figure 3.3 presents an anomaly. Namely, passing from  $k = 6$  to  $k = 7$  clusters,  $W$  increases. Since such a quantity is equal to the  $k$ -Means loss function, we discover that at least one local optimum is attained because the loss function value upon convergence of the more parsimonious solution with  $k = 6$  clusters is lower than that with one more cluster. This can also be highlighted by observing the positive element of `diff(wss)` taking into account that `wss` is a vector containing the variations in  $W$  passing from  $k$  to  $k + 1$  clusters. Note that running the same code does not guarantee to obtain exactly the same output. To obtain results that are reproducible, especially for simulation studies, we recommend to set the seed of the random number generated by using either the function `set.seed` or, if available, the specific input option of the clustering function to be executed.

To limit the risk of hitting local optima, the number of iterations and, especially, the number of random starts should be increased. These values can be specified by setting the options `iter.max` (default 10) and `nstart` (default 1). Note that when `nstart` is higher than 1, `nstart` random sets of centroids are used as the starting points in the iterative algorithm. A rational start can also be chosen by setting `centers` equal to a specified set of initial centroids. A good rational starting point (e.g., the cluster means of the partition with  $k$  clusters found by a hierarchical method) usually limits the risk of hitting local optima.

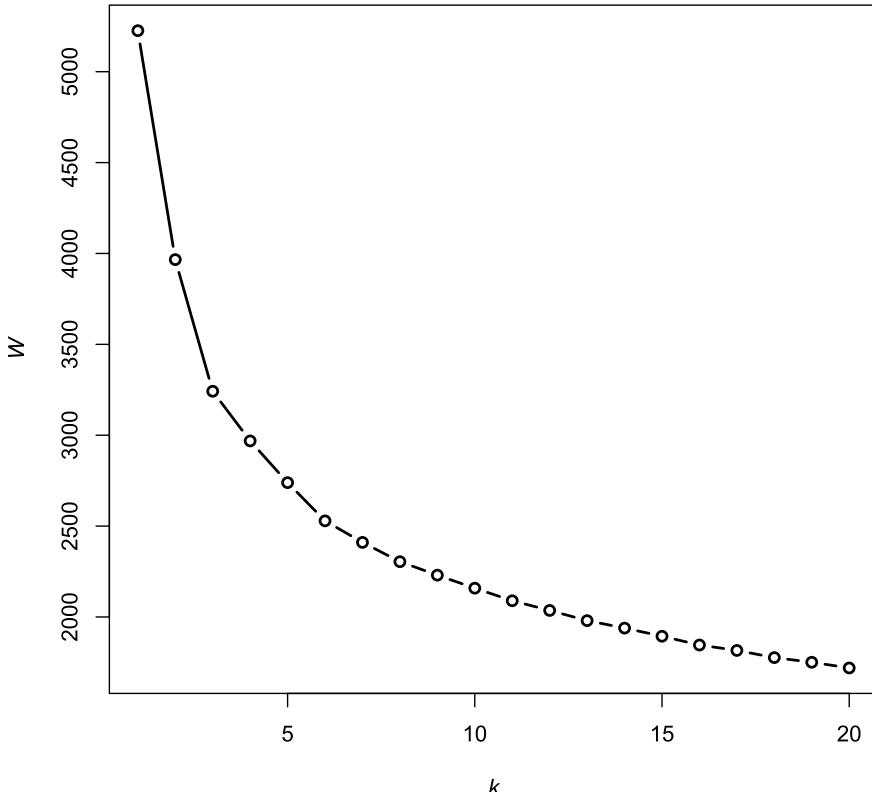


**Fig. 3.3** NBA.48 data:  $W$  values for different choices of  $k$  using the  $k$ -Means algorithm (one starting value)

```

> wss <- numeric()
> wss[1] <- (n - 1)*sum(apply(NBA.Z, 2, var))
> for(k in k.min:k.max) {
+   wss[k] <- kmeans(NBA.Z, k, nstart = 50,
+                      iter.max = 1000)$tot.withinss
+ }
> plot(1:k.max, wss,
+       xlab = expression(italic(k)),
+       ylab = expression(italic(W)),
+       type = "b", lwd = 2)
> diff(wss)
 [1] -1259.56508  -724.36690  -273.32528  -229.41215
 [5] -210.38554  -118.53340  -106.08272  -74.07274
 [9] -71.71824   -69.21948   -53.27570  -56.12123
[13] -40.89597   -44.87829   -48.36577  -29.85702
[17] -39.34973   -25.68572   -31.21280

```



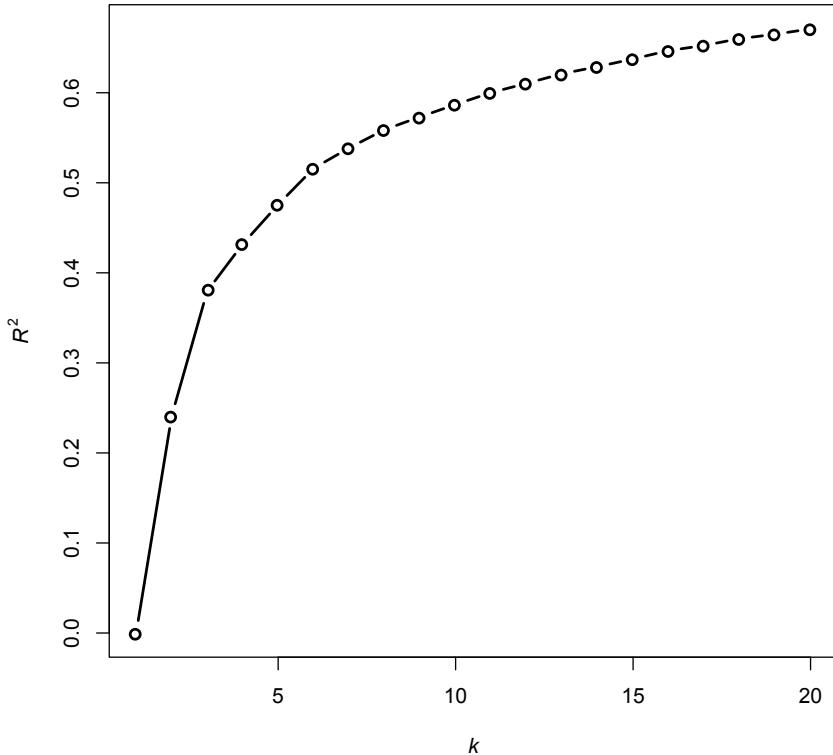
**Fig. 3.4** NBA . 48 data:  $W$  values for different choices of  $k$  using the  $k$ -Means algorithm (50 starting values)

By comparing the resulting  $W$  values displayed in Fig. 3.4 with those of Fig. 3.3, we can see that the bumps disappear and the  $W$  values are decreasing with respect to the number of clusters. The number of clusters can be chosen in connection with an elbow, which denotes a number of clusters  $k$  such that passing from  $k - 1$  to  $k$  clusters produces a large decrease of  $W$ , while passing from  $k$  to  $k + 1$  slightly reduces it. In the current plot, an elbow may be found at  $k = 3$ . We therefore investigate such a solution and the two closest ones with  $k = 2$  and  $k = 4$ .

Note that a plot with the same meaning (Fig. 3.5) can be done by means of the  $R^2$  measure. It is

$$R^2 = 1 - \frac{W}{T} = \frac{B}{T}. \quad (3.9)$$

$R^2$  takes values in the interval  $[0, 1]$ . In this case, the higher the  $R^2$ , the more the clusters are formed by homogeneous units.



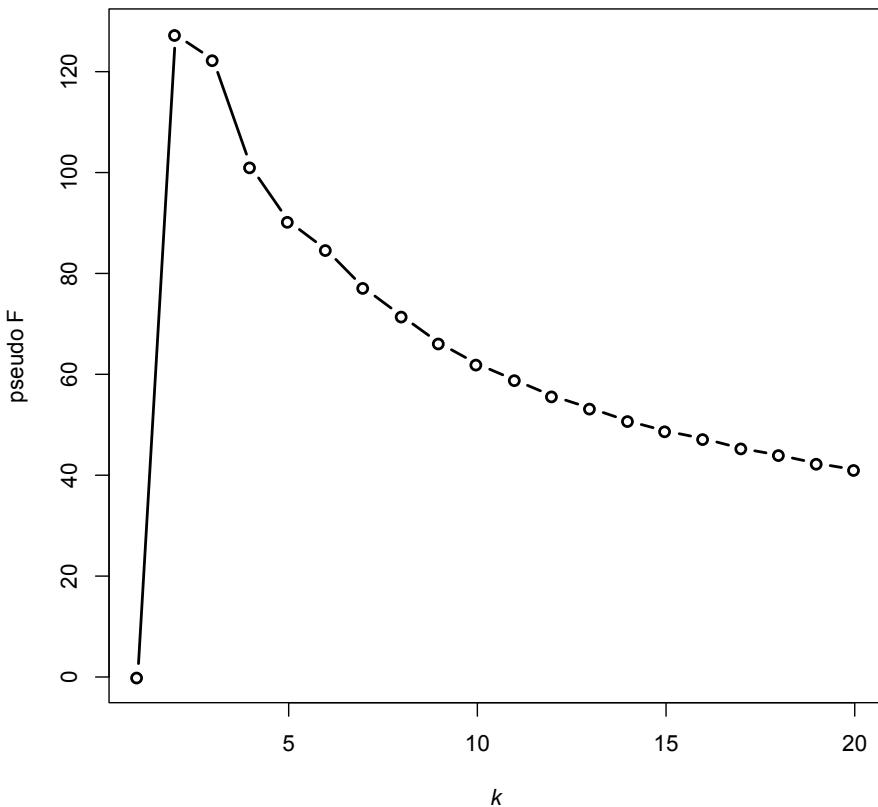
**Fig. 3.5** NBA .48 data:  $R^2$  values for different choices of  $k$  using the  $k$ -Means algorithm

```
> R2 <- 1 - wss/wss[1]
> plot(1:k.max, R2,
+       xlab = expression(italic(k)),
+       ylab = expression(italic(R^2)),
+       type = "b", lwd = 2)
```

From Fig. 3.5, we can see that, as expected, the  $R^2$  values increase for increasing values of  $k$ . However, the information provided by the figure is exactly the same as Fig. 3.4 since the number of clusters can be determined by seeking an elbow which is obviously visible for  $k = 3$ .

A closely related cluster validity index is the so-called pseudo F measure, also known as the Calinski and Harabasz index [1]. It is defined as

$$\frac{B}{k-1} \Bigg/ \frac{W}{n-k}. \quad (3.10)$$



**Fig. 3.6** NBA .48 data: Pseudo F values for different choices of  $k$  using the  $k$ -Means algorithm

It compares the  $B$  and  $W$  values. A peak in the pseudo F values indicates the number of clusters. In fact, large pseudo F values identify partitions with well-separated clusters (large  $B$ ) and homogeneous clusters (small  $W$ ). In the current example, we use the following code yielding Fig. 3.6.

```
> bss <- wss[1] - wss
> pseudoF <- numeric()
> pseudoF[1] <- 0
> for(k in k.min:k.max) {
+   pseudoF[k] <- (bss[k]/(k - 1))/(wss[k]/(n - k))
+ }
> plot(1:k.max, pseudoF,
+       xlab = expression(italic(k)),
+       ylab = "pseudo F",
+       type = "b", lwd = 2)
```

By inspecting Fig. 3.6, we can see peaks for  $k = 2$  and  $k = 3$ . We can thus conclude that there is some consensus among the various cluster validity indices confirming that it is useful to analyze in detail the solutions with  $k$  ranging from 2 to 4.

We start by considering the more parsimonious solution with  $k = 2$  clusters. Note that, in the following script, we set the seed in order to fix the cluster label switching.

```
> set.seed(2345)
> res.kmeans.2 <- kmeans(NBA.Z,
+                           centers = 2,
+                           nstart = 50,
+                           iter.max = 1000)
> names(res.kmeans.2)
[1] "cluster"        "centers"        "totss"
[4] "withinss"       "tot.withinss"   "betweenss"
[7] "size"           "iter"           "ifault"
```

The details on the decomposition of  $T$  are given in `totss` (Total sum of squares,  $T$ ), `betweenss` (Between-cluster sum of squares,  $B$ ), `tot.withinss` (Within-cluster sum of squares,  $W$ ), and `withinss` (Within-cluster sum of squares distinguished by cluster,  $W_g$ ). In particular, the latter information is helpful in order to assess the level of homogeneity in each cluster. The obtained partition is reported in `size`, a vector giving the cluster sizes; `cluster`, a vector giving the cluster membership of all the units (integers from 1 to  $k$  denote the clusters); and `centers`, a matrix with the optimal centroids in its rows. Finally, some computational aspects are summarized in `iter` and `ifault`. The former gives the number of iterations needed for convergence and the latter is an integer indicating computational problems (0 means that no problem occurs).

```
> res.kmeans.2$size
[1] 90 313
> res.kmeans.2$withinss
[1] 1100.263 2866.172
> round(res.kmeans.2$centers, 2)
    PTS    FGA    FG.   X3PA   X3P.    FTA    FT.    OREB
1  0.18 -0.22  1.30 -1.26 -1.11  0.66 -0.58  1.53
2 -0.05  0.06 -0.37  0.36  0.32 -0.19  0.17 -0.44
      DREB    AST    TOV    STL    BLK
1  1.18 -0.41  0.19 -0.18  1.24
2 -0.34  0.12 -0.05  0.05 -0.36
```

The solution with  $k = 2$  clusters offers a too simplistic representation of the NBA players with a big size cluster, containing about  $\frac{3}{4}$  of players, with a quite high value of  $W_g$ . A quick view of the centroids shows that different kinds of players are identified. For instance, Cluster 1 is characterized by a higher number of points, rebounds, turnovers, and blocks, while Cluster 2 detects players with a larger number of assists and, especially, three-point field goals attempted and percentage.

```

> set.seed(2345)
> res.kmeans.3 <- kmeans(NBA.Z,
+                           centers = 3,
+                           nstart = 50,
+                           iter.max = 1000)
> res.kmeans.3$size
[1] 226 97 80
> res.kmeans.3$withinss
[1] 1509.7538 875.9919 856.3223
> round(res.kmeans.3$centers, 2)
    PTS   FGA   FG.   X3PA   X3P.   FTA   FT.   OREB
1 -0.51 -0.38 -0.46  0.30  0.30 -0.59  0.06 -0.42
2  1.21  1.24 -0.06  0.44  0.35  0.98  0.43 -0.33
3 -0.03 -0.42  1.36 -1.37 -1.27  0.49 -0.68  1.60
    DREB   AST   TOV   STL   BLK
1 -0.39 -0.18 -0.46 -0.01 -0.33
2  0.00  0.83  1.04  0.21 -0.24
3  1.10 -0.51  0.04 -0.23  1.23

```

The output for the solution with  $k = 3$  clusters reveals that the cluster sizes are approximately  $\frac{1}{2}$ ,  $\frac{1}{4}$ , and  $\frac{1}{4}$ . The biggest size cluster is the most heterogeneous one, characterized by the highest value of  $W_g$ , which is however one half of the maximum for the two-cluster solution.

The centroids allow for distinguishing the players with respect to their characteristics. However, they are standardized and their meaning is quite complex. It would be more useful for interpretative purposes to express them with respect to the original units of measurement. This can be done by computing the mean values of the original variables distinguished by cluster membership. Note that running the following script overwrites the output component `centers`.

```

> k <- 3
> res.kmeans.3$centers <- t(sapply(X = 1:k,
+                                     FUN = function(nc) apply(
+                                         NBA[res.kmeans.3$cluster == nc, ],
+                                         2, mean)))
> rownames(res.kmeans.3$centers) <- 1:k
> round(res.kmeans.3$centers, 2)
    PTS   FGA   FG.   X3PA   X3P.   FTA   FT.   OREB   DREB
1 17.50 15.15 42.37 7.15 34.30 2.85 75.79 1.39 5.72
2 28.35 22.57 45.16 7.66 34.91 6.56 79.75 1.54 6.83
3 20.55 14.97 55.13 1.43 16.85 5.41 68.10 4.86 9.99
    AST   TOV   STL   BLK
1 4.14  2.06  1.49  0.71
2 6.87  3.59  1.61  0.78
3 3.24  2.57  1.35  2.01

```

Also note that, instead of using the function `sapply`, the same output can be found by means of the function `aggregate`.

```
> round.aggregate(. ~ res.kmeans.3$cluster,
+                  data = NBA, FUN = mean) [, -1], 2)
```

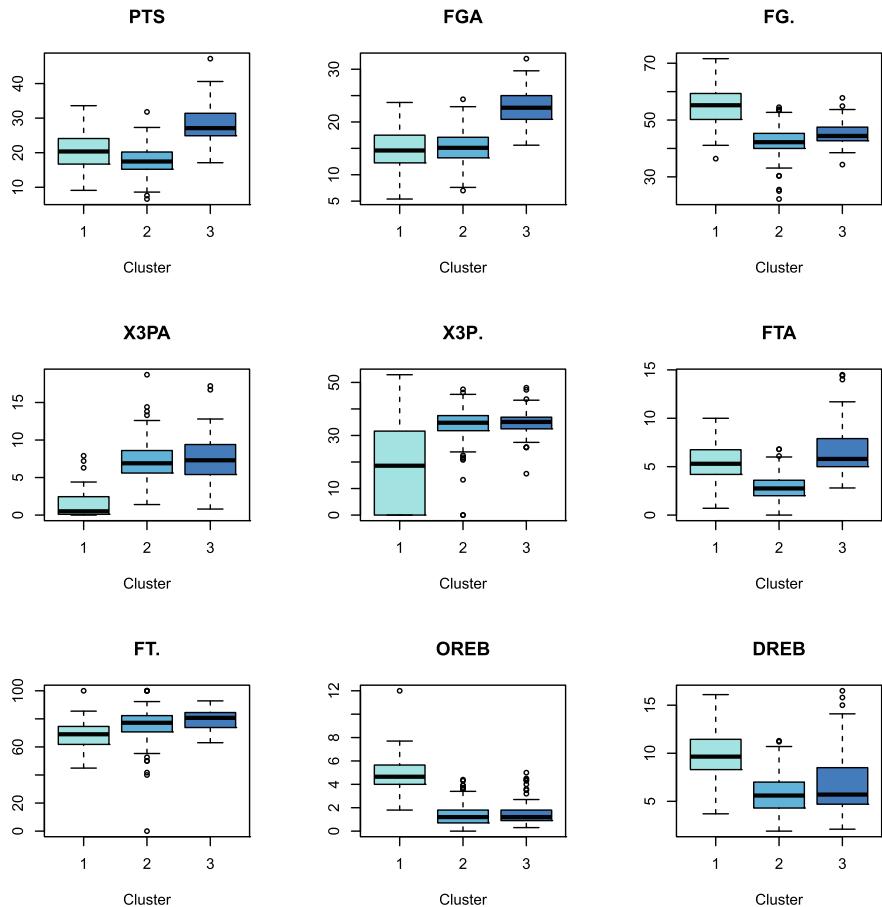
The script consists in splitting the rows of the data frame `NBA`, specified by the argument `data`, with respect to the cluster membership available in `res.kmeans.3$cluster` (the grouping element after the symbol `~`) and in computing the mean values (option `FUN = mean`) for all the variables (as indicated by using the symbol `.` before the symbol `~`) of `NBA`. If one is interested in a subset of variables, the symbol `.` should be replaced by the corresponding variables taken by means of the function `cbind`. For instance, in order to display the centroids for `PTS` and `FGA`, the symbol `.` must be replaced by `cbind(PTS, FGA)`. If the interest relies on only one variable, say `PTS`, it is sufficient to write `PTS` in place of the symbol `..`.

To improve the cluster interpretation and to have an insight into the cluster variability, we display the boxplots for all the variables stratified by cluster in Figs. 3.7 and 3.8.

```
> par(mfrow = c(5, 3))
> for (j in 1:ncol(NBA)){
+   boxplot(NBA[, j] ~ res.kmeans.3$cluster,
+           xlab = "Cluster", ylab = "",
+           col = colors()[c(110, 121, 130)],
+           main = names(NBA)[j])
+ }
> barplot(table(res.kmeans.3$cluster),
+           xlab = "Cluster", ylab = "",
+           col = colors()[c(110, 121, 130)],
+           main = "Cluster size")
> barplot(res.kmeans.3$withinss,
+           xlab = "Cluster", names.arg = 1:3,
+           col = colors()[c(110, 121, 130)],
+           main = expression(italic(W[g])))
```

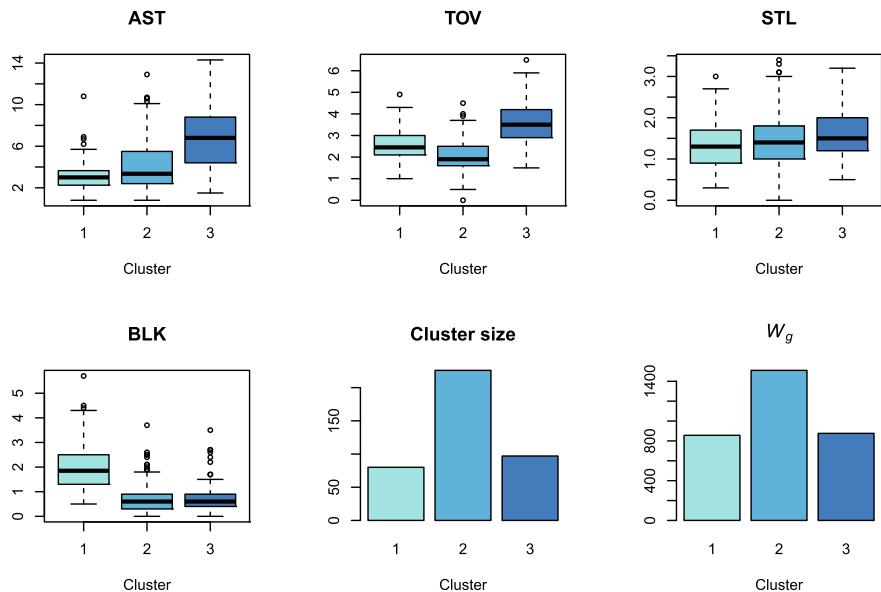
From Figs. 3.7 and 3.8, we can see that Cluster 1 includes players specialized in perimeter shots and with low values for points, field goals, rebounds, and turnovers. The players assigned to Cluster 2 are top scorers, characterized by a large number of attempts and high percentages, a lot of assists and steals, and a quite low number of rebounds and blocks. Finally, the players belonging to Cluster 3 have, in general, low shooting percentages, except for the free throws and large numbers of rebounds and blocks.

To further interpret the clusters, we consider additional information available in the data frames `NBA.external` and `NBA.efficiency` [18] available in the package **datasetsICR**. In particular, we focus our attention on the position and the efficiency. Note that the efficiency is the most commonly used index for comparing the overall value of players. It is derived by considering a set of basic individual statistics. Note also that only positive values of efficiency are available, and hence the rows of the clustered dataset and those of the efficiency dataset differ.



**Fig. 3.7** NBA . 48 data: Boxplots of the first nine variables by cluster ( $k$ -Means solution with  $k = 3$  clusters)

```
> data("NBA.external")
> str(NBA.external)
'data.frame':   530 obs. of  10 variables:
$ PLAYER     : Factor w/ 530 levels "Aaron Gordon",...
$ FORWARD    : Factor w/ 2 levels "No", "Yes": 2 1 2 ...
$ CENTER     : Factor w/ 2 levels "No", "Yes": 1 1 1 ...
$ GUARD      : Factor w/ 2 levels "No", "Yes": 1 2 1 ...
$ ROOKIE     : Factor w/ 2 levels "No", "Yes": 1 2 1 ...
$ SOPHOMORE  : Factor w/ 2 levels "No", "Yes": 1 1 2 ...
$ VETERAN    : Factor w/ 2 levels "No", "Yes": 2 1 1 ...
$ 1ST ROUND  : Factor w/ 2 levels "No", "Yes": 2 2 1 ...
$ 2ND ROUND  : Factor w/ 2 levels "No", "Yes": 1 1 2 ...
$ UNDRAFTED : Factor w/ 2 levels "No", "Yes": 1 1 1 ...
> for (nc in 2:4) {
+   print(names(NBA.external)[nc])
```



**Fig. 3.8** NBA .48 data: Boxplots of the last four variables by cluster and barplots of cluster sizes and  $W_g$  values ( $k$ -Means solution with  $k = 3$  clusters)

```

+      print(table(res.kmeans.3$cluster,
+                  NBA.external[NBA.game$MIN >= 12, nc]))
+
[1] " FORWARD "
  No Yes
1 101 125
2 65  32
3 40  40
[1] " CENTER "
  No Yes
1 220   6
2 86   11
3 20   60
[1] " GUARD "
  No Yes
1 93 133
2 34  63
3 77  3
> data("NBA.efficiency")
> str(NBA.efficiency)
'data.frame': 258 obs. of 2 variables:
 $ Player: Factor w/ 258 levels "Aaron Gordon",...
 $ EFF   : num  35.3 33.1 32.2 30.4 29 28.9 28 27.9 ...
...
> NBA.efficiency$Cluster1 <- 0
> NBA.efficiency$Cluster2 <- 0

```

```

> NBA.efficiency$Cluster3 <- 0
> for (nr in 1:dim(NBA.efficiency)[1]){
+   if (NBA.efficiency$Player[nr] %in%
+       names(which(res.kmeans.3$cluster == 1)) ){
+     NBA.efficiency$Cluster1[nr] <- 1
+   }
+   if (NBA.efficiency$Player[nr] %in%
+       names(which(res.kmeans.3$cluster == 2)) ){
+     NBA.efficiency$Cluster2[nr] <- 1
+   }
+   if (NBA.efficiency$Player[nr] %in%
+       names(which(res.kmeans.3$cluster == 3)) ){
+     NBA.efficiency$Cluster3[nr] <- 1
+   }
+ }
> round(mean(NBA.efficiency$EFF), 2)
[1] 13.39
> round(100*sum(NBA.efficiency$Cluster1) /
+         res.kmeans.3$size[1], 2)
[1] 57.08
> round(mean(NBA.efficiency$EFF[
+           NBA.efficiency$Cluster1 == 1]), 2)
[1] 10.1
> round(100*sum(NBA.efficiency$Cluster2) /
+         res.kmeans.3$size[2], 2)
[1] 73.2
> round(mean(NBA.efficiency$EFF[
+           NBA.efficiency$Cluster2 == 1]), 2)
[1] 18.77
> round(100*sum(NBA.efficiency$Cluster3) /
+         res.kmeans.3$size[3], 2)
[1] 63.75
> round(mean(NBA.efficiency$EFF[
+           NBA.efficiency$Cluster3 == 1]), 2)
[1] 15.41

```

We can observe that Cluster 1 is a mix of guard and forward players (only 6 out of 77 center players belong to the cluster). The average positive efficiency value for such a cluster is the lowest one (10.10). Only 57% of players assigned to the cluster have positive efficiency values and there are no players with efficiency values larger than 20. Cluster 2, mainly composed of guard players (63 out of 106 players assigned to the cluster), is characterized by the top players in terms of efficiency. 73% of players have positive efficiency values with an average positive efficiency value equal to 18.77. Cluster 2 contains 28 players with efficiency values larger than 20 and the first nine best players in terms of efficiency. Cluster 3 includes a lot of center players (60 out of 77) together with some forward players (and only 3 guard players). In terms of efficiency, Cluster 3 is in between Clusters 1 and 2: 64% of players have positive efficiency values, the average value is 15.41 and 10 players have efficiency values larger than 20.

The solution with  $k = 3$  clusters allows us to detect the cluster of the best players. In particular, it helps us to identify the basic statistics leading to the largest efficiency values. For instance, a manager interested in a top scorer should look at players assigned to Cluster 2. Instead, high-quality center players can be found in Cluster 3.

For the sake of completeness, we briefly inspect the solution with  $k = 4$  clusters.

```
> set.seed(2345)
> res.kmeans.4 <- kmeans(NBA.Z,
+                           centers = 4,
+                           nstart = 50,
+                           iter.max = 1000)
> res.kmeans.4$size
[1] 135 64 74 130
> res.kmeans.4$withinss
[1] 697.7570 598.3199 770.8335 901.8323
> round(res.kmeans.4$centers, 2)
    PTS     FGA     FG.   X3PA   X3P.    FTA    FT.    OREB
1  0.15  0.27 -0.33  0.70  0.47 -0.29  0.55 -0.54
2  1.41  1.40 -0.06  0.40  0.29  1.32  0.34 -0.33
3  0.08 -0.34  1.44 -1.41 -1.32  0.59 -0.62  1.66
4 -0.89 -0.78 -0.45 -0.13  0.12 -0.69 -0.39 -0.23
    DREB     AST     TOV     STL     BLK
1 -0.39 -0.26 -0.36 -0.40 -0.45
2  0.04  1.27  1.47  0.49 -0.22
3  1.14 -0.55  0.08 -0.31  1.28
4 -0.26 -0.05 -0.39  0.35 -0.16
> table(res.kmeans.3$cluster,
+        res.kmeans.4$cluster)
    1    2    3    4
1 105   0   0 121
2  30   64   1   2
3   0    0  73   7
```

To some extent the solution resembles the one with  $k = 3$  clusters. In particular, the biggest size cluster contains a subset of players assigned to the previously described Clusters 1 and 2. By looking at the (standardized) centroids, the value added by such a more complex solution appears to be negligible and hence we prefer the solution with  $k = 3$  clusters.

### 3.4.1 Alternative Functions

The function `kmeans` of the package **stats** is the most famous implementing the  $k$ -Means algorithm. However, for the sake of completeness, we note that some other packages offer functions for running the  $k$ -Means algorithm. Among them, we mention the function `Kmeans` of the package **amap** [14], where different distance measures can be set by means of the option `method` (the other input values remain the same as for `kmeans`).

In the package **ppclust** [2], we can find the functions `hcm` and `ekm`. The first is an extension of `kmeans` with more input arguments and output values. As for `Kmeans`, alternative distance measures can be used (option `dmetric`). Some new input arguments are the convergence criterion (option `con.val`), a logical flag to standardize the data (option `stand`), and the seeding number for the random starts of the algorithm (option `numseed`). In the output, useful additional information are provided. For instance, details on computational aspects can be found in `comp.time` (execution time for all the starts), `func.val` (function value upon convergence for all the starts), and `best.start` (index of the start reaching the minimum function value upon convergence). Obviously, the same information provided by the function `kmeans` is available, even if sometimes with different labels (for instance, the number of units in the clusters can be found in `csize` instead of `size`). The function `ekm` is a slightly different version of `hcm`.

The package **ClusterR** [17] offers two functions, called `KMeans_arma` and `KMeans_rcpp`, implementing the  $k$ -Means algorithm by exploiting the potentialities of the package **RcppArmadillo** [5] to speed up the computationally intensive parts of the algorithm.

The function `NbClust` of the package **NbClust** [3] implements the  $k$ -Means algorithm by setting `method = "kmeans"` and the cluster validity indices available in the function can be used for selecting the number of clusters. By analyzing the NBA data, we get the following output (omitting the details and the plot for the Hubert and D indices).

```
> library(NbClust)
> res.kmeans.NbClust <- NbClust(NBA.Z,
+                                     distance = "euclidean",
+                                     min.nc = 2, max.nc = 20,
+                                     method = "kmeans")

*****
* Among all indices:
* 8 proposed 2 as the best number of clusters
* 8 proposed 3 as the best number of clusters
* 3 proposed 4 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 1 proposed 13 as the best number of clusters
* 2 proposed 18 as the best number of clusters
* 1 proposed 20 as the best number of clusters

***** Conclusion *****

* According to the majority rule, the best number of
clusters is 2

*****
```

We can observe a large consensus toward the solutions with  $k = 2$  and  $k = 3$  clusters.

### 3.5 pam

In this section, we apply the  $k$ -Medoids clustering algorithm and we consider the dataset NBA.48 of the package **datasetsICR**, to discuss the differences between the  $k$ -Means and  $k$ -Medoids solutions. The  $k$ -Means solution with  $k = 3$  clusters has been chosen. To interpret the clusters the centroids have been computed. As these are fictitious units, obtained as average values of the units assigned to the clusters, their use may confuse non-expert users trying to interpret the clustering solution. For this purpose, in the  $k$ -Medoids solution, the clusters are interpreted by using real units, i.e., the medoids.

We apply the same preprocessing as for the  $k$ -Means analysis, i.e., we use the same subset of variables and exclude from the analysis those players who played less than 12 min on average; we run the function **pam** of the package **cluster** on the resulting dataset.

```
> library(datasetsICR)
> data("NBA.48")
> NBA <- NBA.48[, c(1, 8, 10, 11, 13, 14, 16, 17, 18,
+                   19, 21, 22, 23, 24)]
> data("NBA.game")
> NBA <- NBA[NBA.game$MIN >= 12, ]
> row.names(NBA) <- NBA[, 1]
> NBA <- NBA[, -1]
```

Before applying **pam** the data should be standardized. Differently from **kmeans**, the function **pam** allows for standardizing the data by setting **stand = TRUE**. The input data can be either a distance matrix or a standard data matrix or a data frame as **NBA** is. In the latter case (the default option), the adopted metric for comparing units should be set. The options are **metric = "euclidean"** (default) or **metric = "manhattan"**. With respect to the function **kmeans**, note that the number of clusters is specified by setting **k** in place of **centers**. Initial medoids can be given by using the option **medoids**.

```
> library(cluster)
> res.pam.3 <- pam(x = NBA, k = 3, stand = TRUE)
```

The output of **pam** is an object of class **pam**, a list with components that are rather different with respect to the **kmeans** output.

```
> names(res.pam.3)
[1] "medoids"      "id.med"       "clustering"
[4] "objective"    "isolation"   "clusinfo"
[7] "silinfo"      "diss"        "call"
[10] "data"
```

The most important components are `medoids` giving the medoids (`id.med` is a vector containing the corresponding row numbers), `clustering`, a vector containing the cluster assignments similar to `cluster` for `kmeans`. Moreover, `objective` is a two-dimensional vector, the second element of which is the loss function after the SWAP phase. The quality of the obtained clusters is no longer expressed in terms of  $W_g$ , although it might be done if `diss = FALSE`. It is based on `max_diss` and `av_diss` (for every cluster, the maximum and average distances between the units and the medoid, respectively), `diameter` (for every cluster, the maximum distance between units assigned to the same cluster), and `separation` (for every cluster, the minimum distance between a unit assigned to the cluster and one belonging to a different cluster). This information, together with the cluster sizes, can be found in `clusinfo`.

The value of the loss function upon convergence in `objective[2]` can be used to assess whether local optima are attained. Differently from `kmeans`, an option to specify the number of random starts is not available. The following code can be used.

```
> nstart <- 20
> mlfv <- Inf
> lfv <- numeric()
> for (ns in 1:nstart) {
+   res.pam.3.ns <- pam(x = NBA, k = 3,
+                         stand = TRUE)
+   lfv[ns] <- res.pam.3.ns$objective[2]
+   if (lfv[ns] < mlfv) {
+     mlfv <- lfv[ns]
+     res.pam.3 <- res.pam.3.ns
+   }
+ }
> print(lfv)
[1] 3.724711 3.724711 3.724711 3.724711 3.724711
[6] 3.724711 3.724711 3.724711 3.724711 3.724711
[11] 3.724711 3.724711 3.724711 3.724711 3.724711
[16] 3.724711 3.724711 3.724711 3.724711 3.724711
```

We start by setting `mlfv = Inf`. For each random start, if the loss function value `objective[2]` is lower than `mlfv`, then `mlfv` is updated using the current loss function value and the solution is marked as the optimal one. At the end, `mlfv` contains the minimum loss function value using `nstart = 20` random starts and `res.pam.3` is the corresponding solution. All the loss function values are stored in the vector `lfv`. This information is useful to assess whether additional starting points are needed. In our case, the same value is always attained. Likely, this is the global minimum and more random starts are not necessary.

```
> res.pam.3$medoids
          PTS   FGA   FG. X3PA X3P. FTA   FT.
Cedi Osman    19.5 16.6 42.7   7.3 34.8 3.6 77.9
Cody Zeller    19.2 13.3 55.1   0.8 27.3 5.4 78.7
Emmanuel Mudiay 26.1 22.1 44.6   6.3 32.9 5.6 77.4
          OREB DREB AST TOV STL BLK
Cedi Osman     0.9  6.1 3.8 2.2 1.2 0.2
Cody Zeller     4.2  8.6 3.9 2.4 1.5 1.6
Emmanuel Mudiay 1.0  4.9 6.8 4.2 1.3 0.6
> res.pam.3$clusinfo
      size max_diss av_diss diameter separation
[1,] 226 11.359254 3.300690 14.46731 1.265893
[2,]  95 9.187339 4.623588 12.39399 1.309079
[3,]  82 9.330624 3.851973 12.54901 1.265893
```

Cluster 1 is the biggest one (226 players). Despite the size, the average distance is the smallest one, but the diameter and the maximum distance are the largest. This implies that at least one player has been assigned to the cluster even if it is very far from the medoid, even if farther with respect to the medoids of the remaining two clusters. The medoid is Cedi Osman (forward, efficiency = 12.8). Cluster 2 has size equal to 95. Its peculiarity is the lowest diameter and maximum distance, but the highest average distance. This means that Cluster 2 contains players not very far from the medoid, but it is quite heterogeneous. The representative player is Cody Zeller (center). Finally, the medoid of Cluster 3 is Emmanuel Mudiay (guard, efficiency = 13.0). He characterizes the smallest cluster (82 players) having medium levels of diameter, maximum distance, and average distance. The separation values for the three clusters are quite similar (the same for Clusters 1 and 3) highlighting that they are more or less equally far from each other.

It can be interesting to compare the pam solution with the one obtained by kmeans to assess to which extent the clusters overlap. If so, it is reasonable to conclude that the true partition underlying the data has been discovered. For this purpose, we use the object `res.kmeans.3` we have previously obtained.

```
> table(res.pam.3$clustering,
+        res.kmeans.3$cluster)
      1   2   3
1 210 16  0
2 10   5  80
3  6  76  0
> library(mclust)
> adjustedRandIndex(res.pam.3$clustering,
+                     res.kmeans.3$cluster)
[1] 0.7407294
```

The Adjusted Rand Index (ARI) [10] value is quite high (0.74) and a one-to-one relation between clusters approximately holds. Clusters 1, 2, and 3 from pam correspond to Cluster 1, 3, and 2 from kmeans, respectively. Therefore, the top player cluster is now Cluster 3. Only 37 players (out of 407) have been assigned to different clusters by the two algorithms.

Generally speaking, in real-life applications, the number of clusters is unknown. If `diss = FALSE`, the same strategy adopted in the `kmeans` case, based on  $W$ , can be followed (obviously, provided that the centroids are replaced by the medoids). Nevertheless, in `pam`, the number of clusters can be selected according to the Silhouette ( $S$ ) index [13, 19]. The Silhouette value for the generic  $i$ -th unit is defined as

$$s_i = \frac{b_i - a_i}{\max(b_i, a_i)}, \quad i = 1, \dots, n, \quad (3.11)$$

where  $a_i$  and  $b_i$  denote, respectively, the average distance between that unit and all the units belonging to the same cluster and the lowest average distance of  $i$  to any other cluster  $i$  does not belong to. The cluster used for calculating  $b_i$  is the second best fit cluster for unit  $i$  and is usually referred to as the neighboring cluster. The Silhouette value  $s_i$  takes values in the interval  $[-1, 1]$ . Values close to 1 mean that the unit is well assigned to the cluster. On the contrary, values close to  $-1$  imply a wrong assignment of the unit to the cluster. If so, the unit should be assigned to the neighboring cluster. An overall measure of the goodness of a partition can be found by considering the  $S$  index, i.e., the average Silhouette values of all the units:

$$S = \frac{\sum_{i=1}^n s_i}{n}. \quad (3.12)$$

To determine the number of clusters, we can compute the  $S$  index for different values of  $k$  and look for peaks. Note that  $s_i$  can be computed for any clustering solution and any kind of distance measure.

Details on the Silhouette are available in the component `silinfo`. It is a list with three components: `widths`, `clus.avg.widths`, and `avg.width`.

```
> names(res.pam.3$silinfo)
[1] "widths"                  "clus.avg.widths"
[3] "avg.width"
> res.pam.3$silinfo$avg.width
[1] 0.2313821
> res.pam.3$silinfo$clus.avg.widths
[1] 0.2902806 0.1479204 0.1657453
> res.pam.3$silinfo$widths[1:10, ]
            cluster neighbor sil_width
Justin Jackson           1         3 0.4669362
Garrett Temple             1         3 0.4669037
Rodney McGruder           1         3 0.4645819
Josh Hart                 1         3 0.4598478
Timothe Luwawu-Cabarrot   1         3 0.4593672
Terrance Ferguson          1         3 0.4565596
Jared Dudley               1         2 0.4528593
Shake Milton               1         3 0.4519996
Glenn Robinson III         1         3 0.4515082
Wesley Johnson              1         2 0.4479433
```

The information provided by `res.pam.3$silinfo$avg.width` is the S index, while `res.pam.3$silinfo$clus.avg.widths` reports the S index distinguished by cluster. We can see that Cluster 1 is the best one according to S. Finally, `res.pam.3$silinfo$widths` is a matrix with the Silhouette values  $s_i$  in the last column. The first two columns give the assigned cluster and the neighboring cluster used for computing the Silhouette value, respectively. Note that the rows are ordered according to the cluster membership and the Silhouette value. For this reason, the first 10 rows displayed refer to the 10 players assigned to Cluster 1 with the maximum Silhouette values.

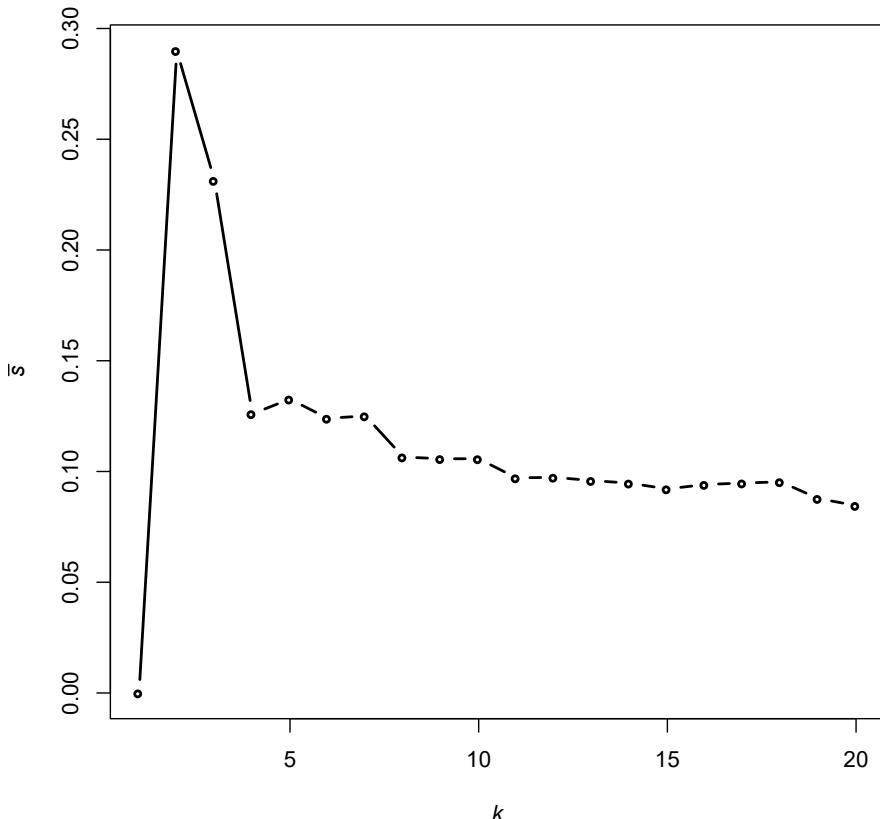
We use S to check whether the choice of  $k = 3$  clusters is reasonable. For this purpose, we vary the number of clusters and select the optimal number of clusters that corresponds to the maximum S index. It should be clear that, differently from other criteria such as that based on  $W$ , S represents an objective measure to select the number of clusters. However, as already observed, our recommendation is to inspect more than one solution with good fit values instead of passively selecting only the one with the highest value of the S index.

```
> k.min <- 2
> k.max <- 20
> asw <- numeric(20)
> asw[1] <- 0
> for(k in k.min:k.max) {
+   asw[k] <- pam(x = NBA, k = k,
+                 stand = TRUE)$silinfo$avg.width
+ }
> plot(1:k.max, asw,
+       xlab = expression(italic(k)),
+       ylab = expression(bar(italic(s))),
+       type = "b", lwd = 2)
```

The plot in Fig. 3.9 suggests a solution with  $k = 2$  (maximum value of the S index) or  $k = 3$  (quite high value of the S index) clusters. As for the  $k$ -Means case, the pam solution with  $k = 2$  tends to oversimplify the data structure and, therefore, we prefer the previously described solution with  $k = 3$  clusters.

### 3.5.1 Plotting Cluster Solutions

Several functions and packages can be used for plotting cluster solutions. A famous function of the package **cluster** is `clusplot`. The function can be applied to the output of `pam`. It uses Principal Component Analysis (PCA) in order to produce a scatterplot of the data onto the low-dimensional space spanned by the first two Principal Components (PCs), i.e., the so-called principal plane. In the plot, different symbols are used to identify units belonging to different clusters and, for every cluster, an ellipse containing all the assigned units is displayed. The plot is an intuitive tool to evaluate the cluster structure in terms of separation, diameter, and size. For the NBA data, the plot of Fig. 3.10 is obtained.

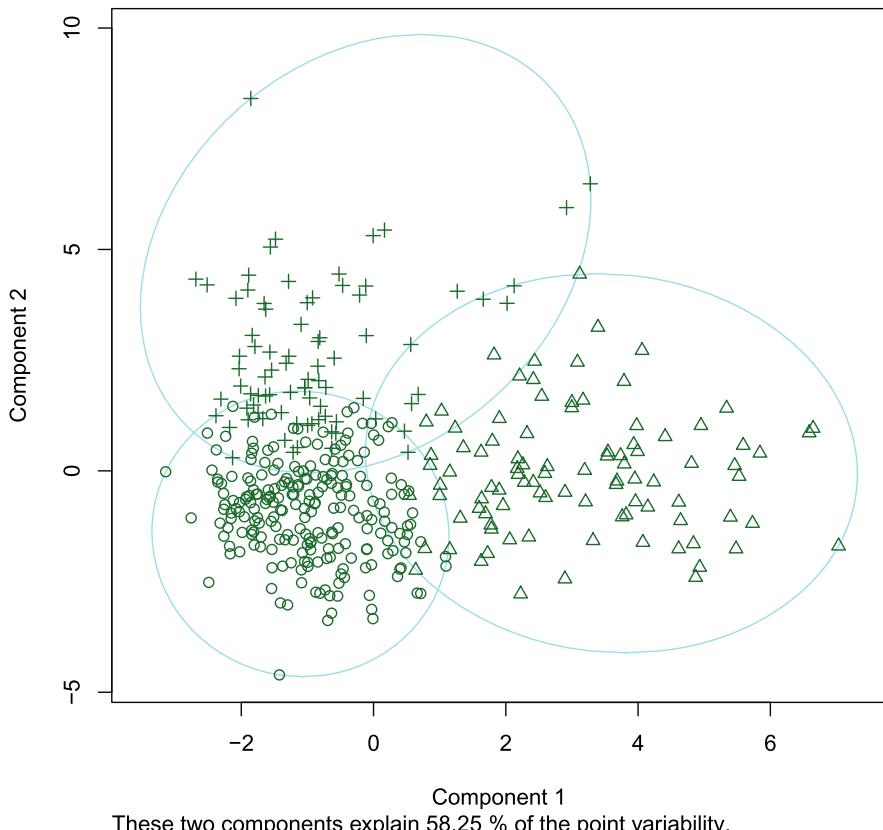


**Fig. 3.9** NBA.48 data: values of the S index for different choices of  $k$  using the  $k$ -Medoids algorithm

```
> clusplot(res.pam.3,
+           main = "low dimensional plot of the pam
+                   solution (principal components)")
```

The function `plotcluster` of the package **fpc** [9] produces a scatterplot of the data by using the solution of Linear Discriminant Analysis (LDA). In particular, data are plotted on the plane spanned by the first two discriminant variables. The obtained scatterplot is given in Fig. 3.11.

```
> library(fpc)
> plotcluster(x = NBA,
+             clvcd = res.pam.3$clustering,
+             main = "low dimensional plot of the pam solution
+                     (discriminant coordinates)")
```



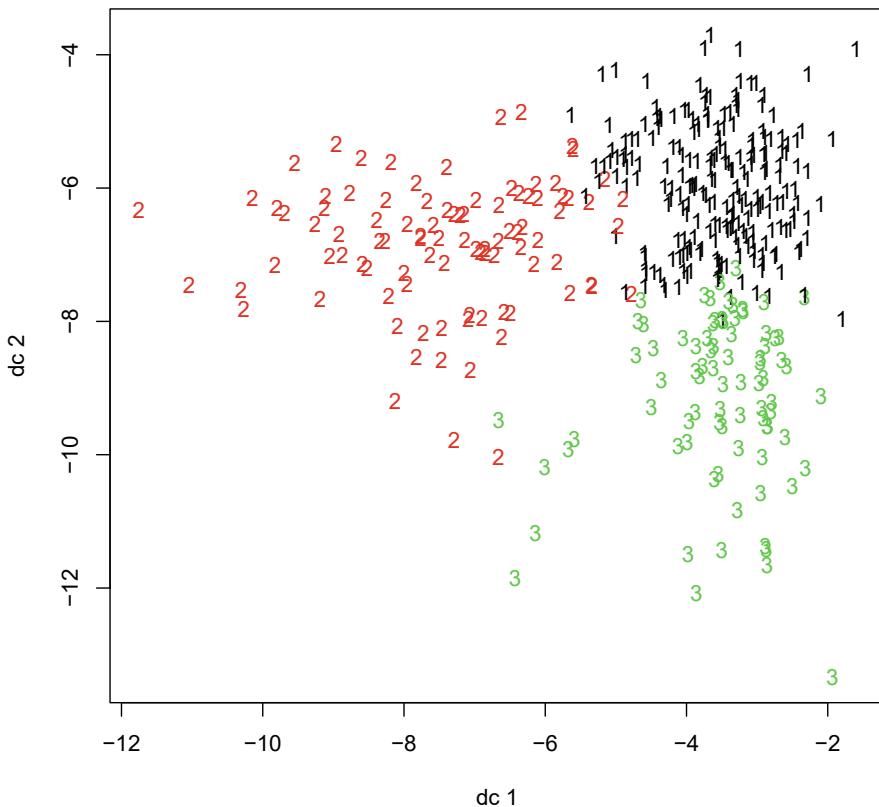
**Fig. 3.10** NBA . 48 data: scatterplot using the first two PCs ( $k$ -Medoids solution with  $k = 3$  clusters)

Both PCA and LDA are data reduction techniques. The main difference is that principal components are found to best explain the variance in the data, while discriminant variables are found to maximize class (cluster) separability. Therefore, it is not surprising that the plot in Fig. 3.10 is more cluttered than that in Fig. 3.11.

Further valuable options for plotting clustering solutions are also available in the package **factoextra** [11].

### 3.5.2 **clara**

CLARA (Clustering LARge Applications) [13] is a variant of PAM specifically designed for data where  $n$  is large. It is implemented by the function **clara** of the package **cluster**. As noted in the help page of **clara**, the definition of “large”



**Fig. 3.11** NBA . 48 data: scatterplot using the first two discriminant coordinates ( $k$ -Medoids solution with  $k = 3$  clusters)

is year dependent. In 1990, the use of `clara` was recommended when  $n > 100$ . Nowadays, `pam` works well for several thousand units.

To detect medoids for large size data, the idea underlying `clara` is to apply `pam` to samples. The number of samples can be specified by setting the option `samples` (default 5). The size of the samples can be set by means of `sampsize`, the default value of which is  $\min(n, 40 + 2k)$ . Of course, `sampsize` should be large, but small enough so that `pam` can be run. In fact, for each sample, `pam` is run and the solution is found. Once the medoids are determined, the out-of-sample units are assigned to the cluster of the nearest medoid. In this way, a solution is obtained without computing the full distance matrix of order  $n$ . This process is repeated for all the samples and the best solution reported in the output of `clara` is the one for which the sum of the distances of the units to their closest medoid is minimum. The output is an object of class `clara`, which is essentially equivalent to an object of class `pam`.

It is not guaranteed that the partition from `clara` is the optimal one that could be found by running `pam`. To increase the chance of attaining the optimal solution, `samples` and/or `sampsize` should be increased as much as possible. To support

this claim, for illustrative purposes, we apply `clara` to the NBA data to show the impact of the options `samples` and `sampsize` on the obtained partition. First of all, we recall the row numbers of the medoids obtained by running `pam`.

```
> print(res.pam.3$id.med)
[1] 48 55 119
```

Obviously, in large size applications, such values are unknown because `pam` cannot be run. However, in the current example, we use them to assess whether the `clara` solutions coincide with the one obtained by `pam`. Such solutions are found by varying `samples` and `sampsize`. Note that the default value of `sampsize` is  $46 = \min(403, 40 + 2 * 3)$ .

```
> set.seed(2345)
> res.med <- matrix(0, nrow = 6, ncol = 5)
> nr <- 0
> rownames(res.med) <- paste("samples =", 
+                               c(1, 5, 10, 20, 40, 80))
> colnames(res.med) <- paste("sampsize =", 
+                               c(23, 46, 92, 184, 368))
> for (sam.value in c(1, 5, 10, 20, 40, 80)){
+   nr <- nr + 1
+   nc <- 0
+   for (size.value in c(23, 46, 92, 184, 368)){
+     nc <- nc + 1
+     res.clara.3 <- clara(x = NBA, k = 3,
+                           stand = TRUE,
+                           samples = sam.value,
+                           sampsize = size.value)
+     res.med[nr, nc] <-
+       length(intersect(res.clara.3$med,
+                         res.pam.3$id.med)) / 3 * 100
+   }
+ }
> round(res.med, 2)
           sampsize = 23 sampsize = 46
samples = 1          0.00          0.00
samples = 5          0.00          0.00
samples = 10         0.00        33.33
samples = 20         33.33        33.33
samples = 40         33.33        33.33
samples = 80         33.33        66.67
           sampsize = 92 sampsize = 184
samples = 1          0.00          0.00
samples = 5          33.33        66.67
samples = 10         33.33        100.00
samples = 20         33.33        33.33
samples = 40          0.00        33.33
samples = 80         33.33        33.33
           sampsize = 368
samples = 1          0.00
samples = 5          66.67
```

```

samples = 10          66.67
samples = 20          100.00
samples = 40          100.00
samples = 80          100.00

```

The object `res.med` is a matrix with elements giving the percentage of medoids correctly discovered by `clara`, i.e., the percentage of medoids from `clara` coinciding with those from `pam`. This can be assessed by means of the function `intersect`. Although the output of `pam` and `clara` is essentially the same, we should note that in `clara` the row numbers of the medoids are given by `i.med` (`id.med` for `pam`). We can see that, for the current example, the default values of `samples` and `sampsiz`e are not adequate. Therefore, we can state that a general rule to follow is to increase both values, especially `sampsiz`e, as much as possible according to the computational constraints in terms of memory and time.

### 3.5.3 Alternative Functions

Functions similar to `pam` and `clara` are also implemented in the package **ClusterR**. These are called `Cluster_Medoids` and `Clara_Medoids`, respectively. The input and output arguments are quite close to those of the original functions. An interesting peculiarity of `Cluster_Medoids` and `Clara_Medoids` is the input option `threads` allowing for parallel computing in order to speed up the computation time. It is an integer value specifying the number of cores to run in parallel (default 1).

## 3.6 Divisive $k$ -Means

This section presents the so-called divisive  $k$ -Means clustering algorithm, also known as the bisecting  $k$ -Means clustering algorithm [20]. It exploits the potentialities of both hierarchical and non-hierarchical clustering. It is a divisive clustering procedure based on a non-hierarchical clustering method, i.e., the  $k$ -Means algorithm. The idea is very simple and intuitive. Starting from the trivial partition with one cluster composed of  $n$  units, split the data by applying the  $k$ -Means algorithm setting  $k = 2$  clusters. This process is repeated until  $n$  singleton clusters are obtained. At each step, the cluster to be divided into two parts is the one with the highest value of  $W_g$ . Therefore, the divisive  $k$ -Means algorithm consists in applying  $n - 1$  times the  $k$ -Means algorithm to a subset of units assigned to one cluster (except for the first step, when the full dataset is clustered).

It should be clear that, differently from any hierarchical clustering procedure, the divisive  $k$ -Means algorithm does not require the computation of distance matrices as it is for the  $k$ -Means algorithm. Moreover, differently from DIANA, the splitting choice does not depend on a single unit. In fact, we recall that, at each step, DIANA splits the cluster with the maximum diameter. This represents a limitation when, for

instance, a cluster is composed of all but one very similar units. If so, the diameter for such a cluster is very large and therefore DIANA splits it. This is not the case for the divisive  $k$ -Means because that cluster would be characterized by a limited value of  $W_g$ .

As for  $k$ -Means, the divisive  $k$ -Means algorithm can be applied when all the variables are quantitative. In contrast to  $k$ -Means, the number of clusters  $k$  is not chosen in advance. The dendrogram can be used ex post to select the best number of clusters as it is the case for any hierarchical procedure.

We apply the divisive  $k$ -Means algorithm to the data frame `seeds` [4] of the package **datasetsICR**.

```
> library(datasetsICR)
> data("seeds")
> str(seeds)
'data.frame': 210 obs. of 8 variables:
 $ area           : num  15.3 14.9 14.3 ...
 $ perimeter      : num  14.8 14.6 14.1 ...
 $ compactness    : num  0.871 0.881 ...
 $ length.of.kernel: num  5.76 5.55 5.29 ...
 $ width.of.kernel: num  3.31 3.33 3.34 ...
 $ asymmetry.coefficient: num  2.22 1.02 2.7 ...
 $ length.of.kernel.groove: num  5.22 4.96 4.83 ...
 $ variety        : Factor w/ 3 levels ...
```

The data have been collected by the Institute of Agrophysics of the Polish Academy of Sciences in Lublin (Poland), which is gratefully acknowledged. The dataset is about  $n = 210$  wheat grains belonging to three different varieties. Such an information is available in the factor `variety` with three levels (Karma, Rosa, and Canadian). The factor `variety` is not used for clustering purposes and the divisive  $k$ -Means is applied using the remaining  $p = 7$  variables that are quantitative and concern details on the internal kernel structure of the wheat grains detected by using a soft X-ray technique. Since the variables have different units of measurement, these are standardized before running the divisive  $k$ -Means clustering algorithm.

```
> seeds.Z <- scale(seeds[, -8])
> row.names(seeds.Z) <- seeds$variety
```

The standardized data are available in the new data frame `seeds.Z`. We also specify the names of the rows of `seeds.Z` by using the information on the variety. This is useful for graphically evaluating whether the varieties are correctly identified by the clustering method.

The divisive  $k$ -Means algorithm is implemented by the function `dclust` of the package **dclust** [21].

```
> library(dclust)
> res.hkmeans <- dclust(seeds.Z)
```

In order to interpret the clusters and to assess whether the varieties are correctly recognized, we use the package **dendextend** [6]. The package offers some interesting functions for an improved and user-friendly visualization of dendograms of hierarchical clustering methods. We provide an example of the effectiveness of the package by considering `res.hkmeans`. In the following code, first we convert the output of `res.hkmeans` to an object of class `dendrogram`. Then we offer a more intuitive view of the dendrogram by cutting it into three branches/clusters with different colors and coloring the labels in different ways according to the corresponding varieties. The latter steps are done by some functions of the package **dendextend**. The dendrogram is given in Fig. 3.12.

```
> library(dendextend)
> dend <- as.dendrogram(res.hkmeans)
> dend <- color_branches(dend, k = 3)
> labels_colors(dend) <-
+  rainbow(3)[sort_levels_values(
+  as.numeric(seeds$variety)[order.dendrogram(dend))]]
> dend <- set(dend, "labels_cex", 0.5)
> plot(dend)
```

By inspecting the figure, we can see that the proper number of clusters is correctly identified. In fact, the choice of  $k = 3$  arises quite naturally. Furthermore, by looking at the color labels, we discover that the clusters can be interpreted in terms of the varieties. In fact, except from a few misclassifications, from the left to the right, the clusters seem to clearly distinguish the varieties Rosa, Canadian, and Kama, respectively. Further details can be found by the following cross-tabulation where we compare the partition with  $k = 3$  clusters (in `res.hkmeans.3`) and the varieties.

```
> res.hkmeans.3 <- cutree(dend, k = 3)
> table(seeds[, 8], res.hkmeans.3)

      res.hkmeans.3
      1   2   3
Kama     60   9   1
Rosa      2  68   0
Canadian  8   0  62
```

We can see that only 20 out of 210 cases are misclassified.

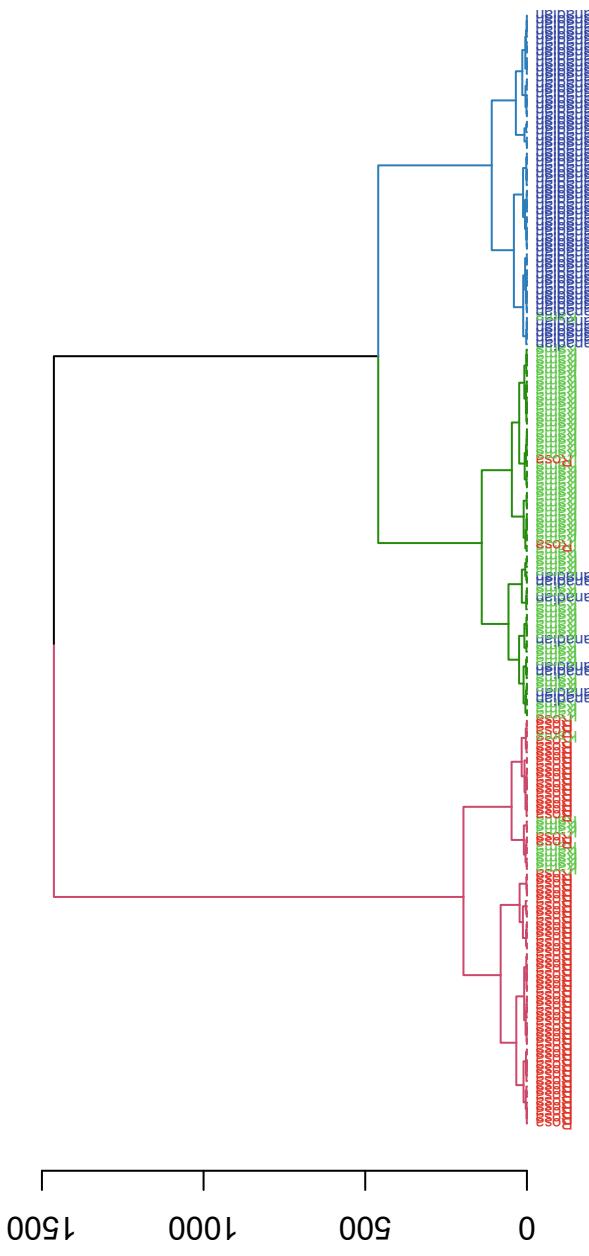


Fig. 3.12 seeds data: dendrogram using the divisive  $k$ -Means algorithm (the labels give the true varieties)

## References

1. Calinski, T., Harabasz, J.: A dendrite method for cluster analysis. *Commun. Stat. Theor. M.* **3**, 1–27 (1974)
2. Cebeci, Z.: Comparison of internal validity indices for fuzzy clustering. *J. Agr. Inform.* **10**, 1–14 (2019)
3. Charrad, M., Ghazzali, N., Boiteau, V., Niknafs, A.: NbClust: an R package for determining the relevant number of clusters in a data set. *J. Stat. Softw.* **61**, 1–36 (2014). <https://www.jstatsoft.org/v061/i06>
4. Dua, D., Graff, C.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine (2019). <http://archive.ics.uci.edu/ml>
5. Eddelbuettel, D., Sanderson, C.: RcppArmadillo: accelerating R with high-performance C++ linear algebra. *Comput. Stat. Data Anal.* **71**, 1054–1063 (2014)
6. Galili, T.: dendextend: an R package for visualizing, adjusting, and comparing trees of hierarchical clustering. *Bioinformatics* **31**, 3718–3720 (2015)
7. Giordani, P., Ferraro, M.B., Martella, F.: datasetsICR: Datasets from the Book “An Introduction to Clustering with R”, R package version 1.0 (2020). <https://CRAN.R-project.org/package=datasetsICR>
8. Hartigan, J.A., Wong, M.A.: A  $K$ -means clustering algorithm. *J. R. Stat. Soc. C-Appl.* **28**, 100–108 (1979)
9. Hennig, C.: fpc: Flexible Procedures for Clustering. R package version 2.2-5 (2020). <https://CRAN.R-project.org/package=fpc>
10. Hubert, L., Arabie, P.: Comparing partitions. *J. Classif.* **2**, 193–218 (1985)
11. Kassambara, A., Mundt, F.: factoextra: Extract and Visualize the Results of Multivariate Data Analyses. R package version 1.0.7 (2020). <https://CRAN.R-project.org/package=factoextra>
12. Kaufman, L., Rousseeuw, P.J.: Clustering by means of medoids. In: Dodge, Y. (ed.) *Statistical Data Analysis Based on the  $L_1$ -Norm and Related Methods*, pp. 405–416. North-Holland, Amsterdam (1987)
13. Kaufman, L., Rousseeuw, P.J.: *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York (1990)
14. Lucas, A.: amap: Another Multidimensional Analysis Package. R package version 0.8-18 (2019). <https://CRAN.R-project.org/package=amap>
15. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, vol. 1, pp. 281–298 (1967)
16. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.: cluster: Cluster Analysis Basics and Extensions. R package version 2.1.0 (2019)
17. Mouselimis, L.: ClusterR: Gaussian Mixture Models,  $K$ -Means, Mini-Batch-Kmeans,  $K$ -medoids and Affinity Propagation Clustering. R package version 1.2.1 (2019). <https://CRAN.R-project.org/package=ClusterR>
18. NBA Advanced Stats (2019). <https://stats.nba.com/>
19. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* **20**, 53–65 (1987)
20. Steinbach, M., Karypis, G., Kumar, V.: A comparison of document clustering techniques. *Proceedings of the KDD-2000 Workshop on Text Mining* (2000)
21. Wilkinson, S., Giordani, P.: dclust: Divisive Hierarchical Clustering. R package version 0.1.0 (2019). <https://CRAN.R-project.org/package=dclust>

# Chapter 4

## Big Data and Clustering



**Abstract** This chapter is devoted to an introductory discussion on clustering big data. In particular, we briefly reconsider all the most common clustering methods, presented in the previous chapters, and evaluate how they work when the data size increases. The analyses are not based on big data in a strict sense. In fact, the data we investigate do not pose memory/storage problems. Nevertheless, some preliminary relevant conclusions can be drawn even handling “non-small” size data.

### 4.1 Standard Methods

In order to briefly address the problem of clustering large data by means of standard methods, we study the data frame `FIFA` [1] contained in the package **datasetsICR** [2]. The data refer to 18,207 football players and their attributes registered in the database of the videogame FIFA 19. The clustering of players is based on the ratings on different features concerning the player ability in a given position and with respect to a certain skill. Goalkeepers have specific characteristics and are thus removed from the analysis. This leads to the new data frame `FIFA.clus`.

```
> library(datasetsICR)
> data("FIFA")
> FIFA.clus <- FIFA[FIFA$Position != "GK",
+                           c(1, 20:79)]
```

In `FIFA.clus`, a few missing values are found. They are either `NA` or empty values. Therefore, we replace empty values with `NA` and remove all rows containing missing values.

```
> FIFA.clus <- as.data.frame(apply(FIFA.clus, 2,
+                                    function(x) gsub("^$|^ $", NA, x)))
> FIFA.clus <- FIFA.clus[complete.cases(FIFA.clus), ]
> dim(FIFA.clus)
[1] 16122      61
```

We obtain a dataset with  $n = 16,122$  units and  $p = 60$  ability variables plus the variable Name. Obviously, the dataset cannot be considered large. Nonetheless, it allows us to offer some assessment on the existing standard clustering functions with respect to their ability to handle large size data.

Before applying the functions, some further preprocessing is necessary because the variables are recognized as factors. This depends on the use of the function gsub and to the fact that some scores are expressed as the sum of two numbers. The first number is the player value when the game was launched, while the second is the possible improvement during the season. We mend the problem by converting the entries in numbers and ignoring the possible improvement.

```
> for (nc in 2:27) {
+   FIFA.clus[, nc] <- as.numeric(substr(x =
+     FIFA.clus[, nc], start = 1,
+     stop = regexpr('[+]',
+       FIFA.clus[, nc])[1] - 1)))
}
> for (nc in 28:61) {
+   FIFA.clus[, nc] <- as.numeric(FIFA.clus[, nc]))
}
> str(FIFA.clus)
'data.frame': 16122 obs. of 61 variables:
 $ Name           : Factor w/ 15347 levels
 ...- attr(*, "names")= chr "1" "2" "3" "5" ...
 $ LS             : num  88 91 84 82 83 77 87 73 87 ...
 $ ST             : num  88 91 84 82 83 77 87 73 87 ...
 $ RS             : num  88 91 84 82 83 77 87 73 87 ...
 $ LW             : num  92 89 89 87 89 85 86 70 83 ...
 $ LF             : num  93 90 89 87 88 84 87 71 86 ...
 $ CF             : num  93 90 89 87 88 84 87 71 86 ...
 $ RF             : num  93 90 89 87 88 84 87 71 86 ...
 $ RW             : num  92 89 89 87 89 85 86 70 83 ...
 $ LAM            : num  93 88 89 88 89 87 85 71 83 ...
 $ CAM            : num  93 88 89 88 89 87 85 71 83 ...
 $ RAM            : num  93 88 89 88 89 87 85 71 83 ...
 $ LM             : num  91 88 88 88 89 86 84 72 81 ...
 $ LCM            : num  84 81 81 87 82 88 79 75 77 ...
 $ CM             : num  84 81 81 87 82 88 79 75 77 ...
 $ RCM            : num  84 81 81 87 82 88 79 75 77 ...
 $ RM             : num  91 88 88 88 89 86 84 72 81 ...
 $ LWB            : num  64 65 65 77 66 82 69 81 61 ...
 $ LDM            : num  61 61 60 77 63 81 68 84 62 ...
 $ CDM            : num  61 61 60 77 63 81 68 84 62 ...
 $ RDM            : num  61 61 60 77 63 81 68 84 62 ...
 $ RWB            : num  64 65 65 77 66 82 69 81 61 ...
 $ LB              : num  59 61 60 73 60 79 66 84 58 ...
 $ LCB            : num  47 53 47 66 49 71 63 87 57 ...
 $ CB              : num  47 53 47 66 49 71 63 87 57 ...
 $ RCB            : num  47 53 47 66 49 71 63 87 57 ...
 $ RB              : num  59 61 60 73 60 79 66 84 58 ...
 $ Crossing       : num  74 74 69 83 71 76 67 56 52 ...
 $ Finishing      : num  86 85 78 73 75 63 84 51 82 ...
```

```
$ HeadingAccuracy : num  54 73 46 39 45 39 61 75 69
...
$ ShortPassing   : num  69 60 63 71 68 72 61 57 62 ...
$ Volleys        : num  77 78 75 73 71 67 79 57 80 ...
$ Dribbling       : num  83 74 82 72 81 76 73 49 71 ...
$ Curve          : num  83 71 78 75 73 75 76 64 67 ...
$ FKAccuracy     : num  84 67 78 74 70 69 75 63 77 ...
$ LongPassing    : num  69 59 60 73 65 70 46 59 47 ...
$ BallControl     : num  73 71 72 68 71 70 67 61 66 ...
$ Acceleration   : num  69 67 72 56 72 58 64 54 55 ...
$ SprintSpeed    : num  61 66 65 51 63 47 50 50 53 ...
$ Agility         : num  67 63 72 55 71 69 58 54 54 ...
$ Reactions        : num  66 67 65 62 61 61 63 56 61 ...
$ Balance          : num  72 47 61 54 71 71 60 43 55 ...
$ ShotPower        : num  73 83 68 79 70 67 74 67 76 ...
$ Jumping          : num  41 68 34 36 29 41 42 66 57 ...
$ Stamina          : num  46 62 55 64 57 63 64 58 52 ...
$ Strength          : num  35 55 25 51 42 34 59 59 60 ...
$ LongShots        : num  84 83 72 81 70 72 75 49 74 ...
$ Aggression       : num  33 48 41 61 39 47 72 73 65 ...
$ Interceptions   : num  13 20 27 52 32 74 32 81 30 ...
$ Positioning      : num  84 85 79 77 77 69 82 50 81 ...
$ Vision           : num  83 71 76 83 78 81 73 52 66 ...
$ Penalties         : num  64 74 70 68 75 71 74 64 77 ...
$ Composure        : num  69 68 67 61 64 57 58 55 59 ...
$ Marking          : num  24 19 18 59 25 51 53 78 25 ...
$ StandingTackle   : num  19 22 15 49 18 67 36 83 33
...
$ SlidingTackle   : num  17 14 24 42 13 64 29 82 10
...
$ GKDiving         : num  6 7 9 15 11 13 24 11 15 10 ...
$ GKHandling       : num  11 11 9 13 12 9 21 8 6 11 ...
$ GKKicking         : num  15 15 15 5 6 7 26 9 12 13 ...
$ GKPositioning    : num  14 14 15 10 8 14 26 7 8 7 ...
$ GKReflexes       : num  8 11 11 13 8 9 27 11 10 10 ...
```

All the ratings range in the interval [0, 100]. For this reason, we decide not to standardize the data. We now apply different functions implementing the  $k$ -Means algorithm [3, 4] varying the number of clusters  $k$  in the set {2, 3, 4, 5}. To this purpose, different functions were used. These are kmeans of the package **stats**, Kmeans of the package **amap** [5], ekm and hcm of the package **ppclust** [6], KMeans\_arma and KMeans\_rcpp of the package **ClusterR** [7], and NbClust of the package **NbClust** [8]. In the latter case, we set method = "kmeans", min.nc = 2 (default option), and max.nc = 5. Three cluster validity indices are chosen. These are the average silhouette index [9, 10] (option index = "silhouette"), recalled in (3.12), the pseudo F measure or Calinski and Harabasz index [11] (option index = "ch"), recalled in (3.10), and the Hartigan index [12] (option index = "hartigan"), which is based on the values of  $W$  for two consecutive solutions with  $k$  and  $k + 1$  clusters (denoted by  $W_k$  and  $W_{k+1}$ , respectively). It is defined as

$$H = \left( \frac{W_k}{W_{k+1}} - 1 \right) (n - k - 1). \quad (4.1)$$

When an option for the number of iterations is available, a large value ( $1e+9$ ) is selected in order to guarantee that convergence is attained. Moreover, the functions `hcm` and `KMeans_rcpp` allow for specifying the convergence criterion (we set the default option  $1e-9$  or  $1e-12$ ). We thus use two levels to assess whether and how such a choice affects the computation time.

The analysis is not limited to the  $k$ -Means algorithm. In fact, we also consider the  $k$ -Medoids algorithm [9, 13] by applying the function `pam` of the package `cluster` [14] and some agglomerative hierarchical clustering procedures (developed in [15–17]), by means of the functions `hclust` of the package `stats` and `agnes` of the package `cluster`, with different options for `method` ("complete", "average", "single", and "ward" for `agnes` or "ward.D2" for `hclust`). The Euclidean distance is used for the  $k$ -Medoids algorithm and the agglomerative hierarchical clustering procedures.

We omit the details of the script we run and concentrate our attention on the results. In brief, it produces the output of all the functions and the vector `CPU.TIME` containing the computation time in seconds by means of the function `system.time`. Letting `FUN` and `OUTPUT.FUN` a generic function and the corresponding output, respectively, the following code is run (`num.met` denotes the control variable).

```
> CPU.TIME[num.met] <- system.time({ OUTPUT.FUN <-
+   FUN(x=FIFA.clus[, -1], ...)) [3]
```

For the function `hclust`, the option `x = FIFA.clus[, -1]` is replaced by `d = dist(FIFA.clus[, -1])`. Further details on `FUN` and `OUTPUT.FUN` are provided in Table 4.1.

```
> sum(CPU.TIME < 1)
[1] 18
> round(CPU.TIME/min(CPU.TIME), 1)
      nbclust.h      nbclust.ch      nbclust.s
      1281.5        1294.2        2159.1
      kmeans.2       kmeans.3       kmeans.4
      1.0            1.0           1.2
      kmeans.5       hclust.c       hclust.a
      2.2            919.3         910.0
      hclust.s       hclust.w       agnes.c
      941.6          916.7         149302.3
      agnes.a        agnes.s       agnes.w
      114667.5       116449.4       114685.9
      Kmeans.2        Kmeans.3       Kmeans.4
      6.8            9.1            19.9
      Kmeans.5        ekm.2          ekm.3
      60.4           14.4           18.1
      ekm.4          ekm.5          hcm.2.09
```

**Table 4.1** Details on FUN, OUTPUT.FUN, and related options (default options are not reported)

FUN	N. of Cl.	OUTPUT.FUN	Additional arguments to FUN
NbClust	max.nc = 5	nbclust.h	index = "hartigan"
NbClust	max.nc = 5	nbclust.ch	index = "ch"
NbClust	max.nc = 5	nbclust.s	index = "silhouette"
kmeans	centers = 2	kmeans.2	iter.max = 1e+9
kmeans	centers = 3	kmeans.3	iter.max = 1e+9
kmeans	centers = 4	kmeans.4	iter.max = 1e+9
kmeans	centers = 5	kmeans.5	iter.max = 1e+9
hclust		hclust.c	
hclust		hclust.a	method = "average"
hclust		hclust.s	method = "single"
hclust		hclust.w	method = "ward.D2"
agnes		agnes.c	method = "complete"
agnes		agnes.a	
agnes		agnes.s	method = "single"
agnes		agnes.w	method = "ward"
Kmeans	centers = 2	Kmeans.2	iter.max = 1e+9
Kmeans	centers = 3	Kmeans.3	iter.max = 1e+9
Kmeans	centers = 4	Kmeans.4	iter.max = 1e+9
Kmeans	centers = 5	Kmeans.5	iter.max = 1e+9
ekm	centers = 2	ekm.2	iter.max = 1e+9
ekm	centers = 3	ekm.3	iter.max = 1e+9
ekm	centers = 4	ekm.4	iter.max = 1e+9
ekm	centers = 5	ekm.5	iter.max = 1e+9
hcm	centers = 2	hcm.2.09	iter.max = 1e+9
hcm	centers = 3	hcm.3.09	iter.max = 1e+9
hcm	centers = 4	hcm.4.09	iter.max = 1e+9
hcm	centers = 5	hcm.5.09	iter.max = 1e+9
hcm	centers = 2	hcm.2.12	iter.max = 1e+9, con.val = 1e-12

(continued)

**Table 4.1** (continued)

FUN	N. of Cl.	OUTPUT.FUN	Additional arguments to FUN
hcm	centers = 3	hcm.3.12	iter.max = 1e+9, con.val = 1e-12
hcm	centers = 4	hcm.4.12	iter.max = 1e+9, con.val = 1e-12
hcm	centers = 5	hcm.5.12	iter.max = 1e+9, con.val = 1e-12
KMeans_arma	clusters = 2	KMeans_arma.2	n_iter = 1e+9
KMeans_arma	clusters = 3	KMeans_arma.3	n_iter = 1e+9
KMeans_arma	clusters = 4	KMeans_arma.4	n_iter = 1e+9
KMeans_arma	clusters = 5	KMeans_arma.5	n_iter = 1e+9
KMeans_rcpp	clusters = 2	KMeans_rcpp.2.09	max_iters = 1e+9, tol = 1e-09
KMeans_rcpp	clusters = 3	KMeans_rcpp.3.09	max_iters = 1e+9, tol = 1e-09
KMeans_rcpp	clusters = 4	KMeans_rcpp.4.09	max_iters = 1e+9, tol = 1e-09
KMeans_rcpp	clusters = 5	KMeans_rcpp.5.09	max_iters = 1e+9, tol = 1e-09
KMeans_rcpp	clusters = 2	KMeans_rcpp.2.12	max_iters = 1e+9, tol = 1e-12
KMeans_rcpp	clusters = 3	KMeans_rcpp.3.12	max_iters = 1e+9, tol = 1e-12
KMeans_rcpp	clusters = 4	KMeans_rcpp.4.12	max_iters = 1e+9, tol = 1e-12
KMeans_rcpp	clusters = 5	KMeans_rcpp.5.12	max_iters = 1e+9, tol = 1e-12
pam	k = 2	pam.2	
pam	k = 3	pam.3	
pam	k = 4	pam.4	
pam	k = 5	pam.5	

21.5	25.9	135.1
hcm.3.09	hcm.4.09	hcm.5.09
334.4	473.8	1035.1
hcm.2.12	hcm.3.12	hcm.4.12
440.5	200.0	473.6
hcm.5.12	KMeans_arma.2	KMeans_arma.3
725.8	2.3	1.4
KMeans_arma.4	KMeans_arma.5	KMeans_rcpp.2.09
2.7	1.8	4.5
KMeans_rcpp.3.09	KMeans_rcpp.4.09	KMeans_rcpp.5.09
2.5	5.9	3.8
KMeans_rcpp.2.12	KMeans_rcpp.3.12	KMeans_rcpp.4.12
4.0	2.1	5.9
KMeans_rcpp.5.12	pam.2	pam.3
3.8	651.8	562.5
pam.4	pam.5	
934.6	1546.4	

We can see that 18 times out of 47, the clustering functions take less than 1 s. On the one hand, this shows that several functions are very efficient (note that the comparison is carried out by using a standard performance laptop). However, on the other hand, more relevant conclusions can be drawn by analyzing the computation time ratio computed with respect to the lowest registered corresponding to the function `kmeans` of the package **stats**. The computation time increases with respect to  $k$  as it occurs for all the other non-hierarchical methods. Very good performances are also observed for `KMeans_arma` and `Kmeans_rcpp` of the package **ClusterR**. For the latter function, the convergence criterion does not affect the results. With respect to the packages **amap** and **ppclust**, the functions, ordered with respect to the best computation time, are `Kmeans`, `ekm`, and `hcm` for which different computation times are observed by varying the convergence criterion. The computation time of `NbClust` from the homonymous package is quite high, especially for the average silhouette case. However, it should be noted that such computation times refer to the  $k$ -Means algorithms ranging  $k$  from 2 to 5, while those for the previously mentioned functions refer to a single value of  $k$ . As expected, the computation time of `pam` of the package **cluster** is higher than those of the functions implementing the  $k$ -Means algorithm.

With respect to hierarchical clustering, we observe relevant differences. In particular, regardless of the clustering method, `hclust` of the package **stats** is much more efficient than `agnes` of the package **cluster**. In particular, the order of magnitude for `hclust` appears to be quite similar to the one of `pam`.

We conclude our analysis by briefly describing the obtained clusters. First of all, we observe that all the three cluster validity indices considered in `NbClust` recommend a partition with  $k = 3$  clusters.

```
> nbclust.ch$Best.nc
Number_clusters      Value_Index
            3.000        7614.852
```

**Table 4.2** Output component containing the cluster memberships

Function	Package	Output's component
kmeans	<b>stats</b>	cluster
ekm	<b>ppclust</b>	cluster
hcm	<b>ppclust</b>	cluster
KMeans_rcpp	<b>ClusterR</b>	clusters
NbClust	<b>NbClust</b>	Best.partition
Kmeans	<b>amap</b>	cluster
pam	<b>cluster</b>	clustering

```
> nbclust.h$Best.nc
Number_clusters      Value_Index
            3.000      3851.922
> nbclust.s$Best.nc
Number_clusters      Value_Index
            3.0000     0.2805
```

In the following, we consider the solution obtained by running `kmeans`, i.e., our comments are based on the vector `cluster` providing the cluster memberships. Of course, the interpretation could be based on the output of the alternative functions. Note that the labels of the output components providing the cluster memberships by using the other functions are not always called `cluster`. With respect to the non-hierarchical methods, these are summarized in Table 4.2. It does not contain information on `KMeans_arma` because its output only gives the centroid matrix. Similarly, `agnes` and `hclust` are not mentioned in the table because their partitions can be found by means of `cutree`.

For the sake of completeness, we also note that the various functions implementing  $k$ -Means lead to slightly different cluster memberships. Specifically, some units (less than 10 by considering all the possible pairs of cluster membership vectors) are assigned to different clusters. For instance, by comparing the `kmeans` and `KMeans_rcpp` solutions, we can see that five units belong to different clusters.

```
> table(kmeans.3$cluster, KMeans_rcpp.3.09$clusters)
      1    2    3
  1  0  2 5611
  2  0 4861  0
  3 5645  0  3
```

These differences are likely related to local optimum solutions. Regardless of such minor differences, the three clusters have approximately equal size. In order to interpret them, we graphically inspect the centroids plotted in Fig. 4.1.

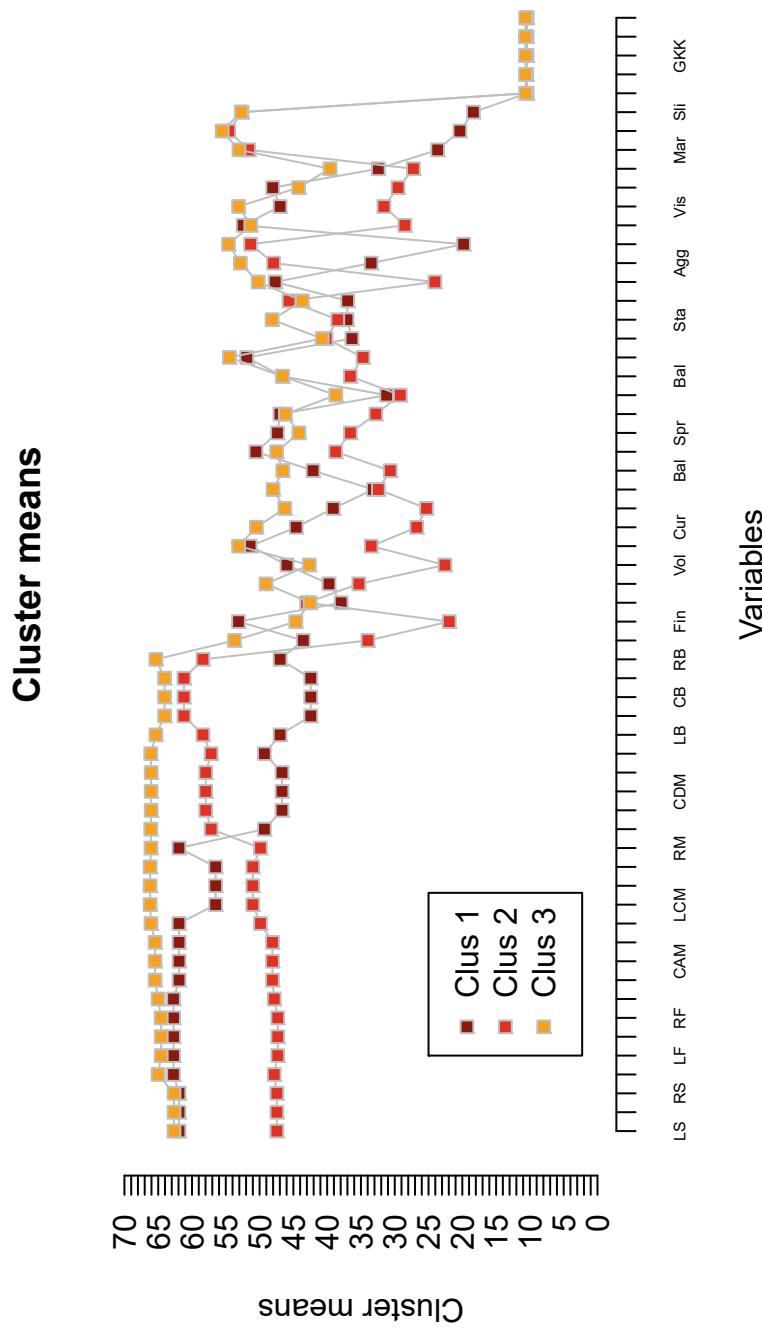


Fig. 4.1 FIFA data: cluster means for the solution with  $k = 3$  clusters using the  $k$ -Means algorithm for the FIFA data

```
> plot(kmeans.3$centers[1, ], type = "o", pch = 22,
+       col = "gray", bg = "darkred", ylim = c(0, 70),
+       xlab = "Variables", ylab = "Cluster means",
+       axes = FALSE, main = "Cluster means")
> lines(kmeans.3$centers[2, ], type = "o", pch = 22,
+       col = "gray", bg = "red")
> lines(kmeans.3$centers[3, ], type = "o", pch = 22,
+       col = "gray", bg = "orange")
> axis(1, at = 1:ncol(kmeans.3$centers),
+       lab = substr(colnames(kmeans.3$centers), 1, 3),
+       cex.axis = 0.5)
> axis(2, at = 0:70, las = 1)
> legend(5, 25, c("Clus 1", "Clus 2", "Clus 3"),
+         pch = 22, col = "gray",
+         pt.bg = c("darkred", "red", "orange"))
```

From the figure, we can see that the last five variables, involving the goalkeeper attitudes, do not play a relevant role in clustering the players. Clusters 2 and 3 seem to distinguish the players with respect to their roles. In particular, Cluster 2 includes midfielders and forwards, while Cluster 3 identifies the defenders. This clearly emerges by looking at the first set of variables (those in capital letters ending with “B”, “M”, and “F” refer to defenders, midfielders, and forwards, respectively). The second set of variables further characterizes the clusters by considering ratings for specific attributes. Cluster 1 mainly contains forwards. The players assigned to such a cluster seem to be the best ones. In fact, the cluster means are almost always the highest ones. Moreover, we can see that the two most valuable players, Ronaldo and Messi, belong to Cluster 1.

```
> FIFA.clus <- cbind(FIFA.clus, kmeans.3$cluster)
> FIFA.clus[FIFA.clus>Name == "Cristiano Ronaldo",
+             "kmeans.3$cluster"]
[1] 1
> FIFA.clus[FIFA.clus>Name == "L. Messi",
+             "kmeans.3$cluster"]
[1] 1
```

## References

1. Gadiya, K.: FIFA 19 complete player dataset (2018). <https://www.kaggle.com/karangadiya/fifa19>
2. Giordani, P., Ferraro, M.B., Martella, F.: datasetsICR: Datasets from the Book “An Introduction to Clustering with R”, R package version 1.0 (2020). <https://CRAN.R-project.org/package=datasetsICR>
3. Hartigan, J.A., Wong, M.A.: A  $K$ -means clustering algorithm. J. R. Stat. Soc. C-Appl. **28**, 100–108 (1979)

4. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematics, Statistics and Probability, vol. 1, pp. 281–298 (1967)
5. Lucas, A.: amap: Another Multidimensional Analysis Package. R package version 0.8-18 (2019). <https://CRAN.R-project.org/package=amap>
6. Cebeci, Z.: Comparison of internal validity indices for fuzzy clustering. *J. Agr. Inform.* **10**, 1–14 (2019)
7. Mouselimis, L.: ClusterR: Gaussian Mixture Models,  $K$ -Means, Mini-Batch-Kmeans,  $K$ -Medoids and Affinity Propagation Clustering. R package version 1.2.1 (2019). <https://CRAN.R-project.org/package=ClusterR>
8. Charrad, M., Ghazzali, N., Boiteau, V., Niknafs, A.: NbClust: An R package for determining the relevant number of clusters in a data set. *J. Stat. Softw.* **61**, 1–36 (2014). <https://www.jstatsoft.org/v061/i06>
9. Kaufman, L., Rousseeuw, P.J.: *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York (1990)
10. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* **20**, 53–65 (1987)
11. Calinski, T., Harabasz, J.: A dendrite method for cluster analysis. *Commun. Stat. Theor. M.* **3**, 1–27 (1974)
12. Hartigan, J.A.: *Clustering Algorithms*. Wiley, New York (1975)
13. Kaufman, L., Rousseeuw, P.J.: Clustering by means of medoids. In: Dodge, Y. (ed.), *Statistical Data Analysis Based on the  $L_1$ -Norm and Related Methods*, pp. 405–416. North-Holland, Amsterdam (1987)
14. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.: Cluster: Cluster Analysis Basics and Extensions. R package version 2.1.0 (2019)
15. Gower, J.C.: A comparison of some methods of cluster analysis. *Biometrics* **23**, 623–637 (1967)
16. Johnson, S.C.: Hierarchical clustering schemes. *Psychometrika* **32**, 241–254 (1967)
17. Ward, J.H.: Hierarchical grouping to optimize an objective function. *J. Am. Stat. Assoc.* **58**, 236–244 (1963)

## **Part III**

# **Fuzzy Clustering**

# Chapter 5

## Fuzzy Clustering

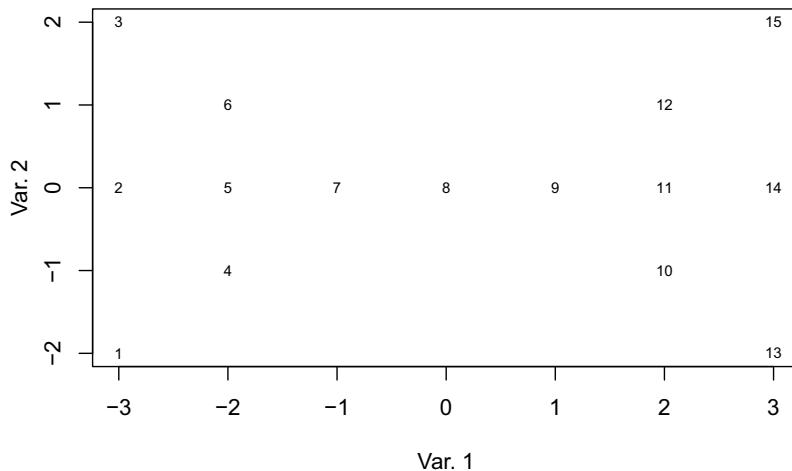


### 5.1 Introduction

In the previous chapters, we focused on standard clustering methods. The main idea of those methods is to produce a hard (crisp) partition of the units; that is, each unit is assigned to one and only one cluster. In many practical situations, this assignment is unrealistic as units may have features that are common to more than one cluster. This is very clear in Fig. 5.1 where the well-known dataset `butterfly` [45], available in the package `datasetsICR` [25], is shown.

```
> library(datasetsICR)
> data("butterfly")
> plot(butterfly, type = "n")
> text(butterfly, labels = 1:15)
```

As we can notice in Fig. 5.1, unit n. 8 is between two clusters, and hence it seems to have “intermediate” values. If we use a standard clustering algorithm, the unit will be assigned to only one cluster. The fuzzy approach to clustering can be introduced to overcome this limit. It consists in assigning each unit to clusters with (fuzzy) membership degrees taking values in the interval  $[0, 1]$ . Starting from the fuzzy extension of the classical  $k$ -Means algorithm, referred to as the Fuzzy  $k$ -Means (FkM) [6, 7], several new fuzzy clustering algorithms have been introduced: the Gustafson-Kessel Fuzzy  $k$ -Means (GK-FkM) [26], the Entropic Fuzzy  $k$ -Means (EFkM) [37, 38], the Fuzzy  $k$ -Means with Polynomial Fuzzifier (FkMPF) [31, 53], the Fuzzy  $k$ -Medoids (FkMed) [33], two fuzzy clustering algorithms for relational data (FANNY and NEFRC) [16, 29], and the Fuzzy  $k$ -Means with Noise cluster (FkMN) [14], just to mention a few. In many cases, there is not a standard (hard/crisp) counterpart. Furthermore, in [32] the authors have shown that fuzzy clustering is more than just fuzzification of a standard method; the use of membership degrees may be fundamental for the computational behavior of the clustering algorithm. All of the above fuzzy clustering methods can be used with quantitative data, but fuzzy



**Fig. 5.1** butterfly data: scatterplot

clustering methods for relational data that admit as input argument a distance matrix can also be run for categorical or mixed data.

In this chapter, a further approach, close to the fuzzy one, will be addressed: the probabilistic approach. In comparison with the fuzzy approach, it is more robust to outliers. The most known clustering algorithm is, in this context, the extension of the  $k$ -Means, the Possibilistic  $k$ -Means (PkM) [34, 35].

Finally, some hybrid algorithms will be described: the Fuzzy Possibilistic  $k$ -Means (FPkM) [42], the Possibilistic Fuzzy  $k$ -Means (PFkM) [43], and the Modified Fuzzy and Possibilistic  $k$ -Means (MFPkM) [46].

All of these clustering algorithms will be briefly detailed and then applied to case studies by using fuzzy and probabilistic clustering algorithms implemented in the packages **fclust** [20, 21], **e1071** [41], **ppclust** [10], **advcclust** [4], and **cluster** [40]. The implemented fuzzy and probabilistic clustering algorithms are reported in Table 5.1.

## 5.2 Fuzzy $k$ -Means

In a fuzzy context, the most known and used clustering algorithm is probably the Fuzzy  $k$ -Means (FkM) [7], providing a generalization of the  $k$ -Means algorithm introduced in Chap. 3. The aim of the FkM algorithm is to look for the best fuzzy partition of  $n$  units into  $k$  clusters by solving the following minimization problem:

**Table 5.1** Fuzzy clustering algorithm implementations

Algorithm	<b>fclust</b>	<b>e1071</b>	<b>ppclust</b>	<b>advclust</b>	<b>cluster</b>
FkM	FKM	cmeans	fcm	fuzzy.CM	
GK-FkM	FKM.gk		gk		
GK-FkMb	FKM.gkb			fuzzy.GK	
EFkM	FKM.ent				
FkMPF	FKM.pf				
FkMed	FKM.med				
FANNY					fanny
NEFRC	NEFRC				
FkMN	FKM.noise				
PkM			pcm		
FPkM			fpcm		
MFPkM			mfpcom		
PFkM			pfcn		

$$\begin{aligned}
 \min_{\mathbf{U}, \mathbf{H}} J_{\text{FkM}} &= \sum_{i=1}^n \sum_{g=1}^k u_{ig}^m d^2(\mathbf{x}_i, \mathbf{h}_g), \\
 \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\
 \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n,
 \end{aligned} \tag{5.1}$$

where the term  $u_{ig}$  represents the membership degree of unit  $i$  to cluster  $g$  taking values in the interval  $[0, 1]$ . It expresses the degree for unit  $i$  to be a member of cluster  $g$ , and it is stored in the membership degree matrix  $\mathbf{U}$  of order  $(n \times k)$ . The row-wise sums of  $\mathbf{U}$  are equal to 1. Like the  $k$ -Means algorithm, the prototypes (centroids)  $\mathbf{h}_g$  are stored in the centroid matrix  $\mathbf{H}$  of order  $(k \times p)$ . Finally, the parameter  $m$  is used to tune the fuzziness of the obtained partition ( $m > 1$ ).

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible membership degree matrix  $\mathbf{U}$ .
2. Given  $\mathbf{U}$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n u_{ig}^m \mathbf{x}_i}{\sum_{i=1}^n u_{ig}^m}, \quad g = 1, \dots, k. \tag{5.2}$$

3. Given  $\mathbf{H}$ , update the membership degree matrix  $\mathbf{U}$ :

$$u_{ig} = \frac{1}{\sum_{g'=1}^k \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_{g'})}{d^2(\mathbf{x}_i, \mathbf{h}_g)} \right)^{\frac{1}{m-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k. \quad (5.3)$$

4. Repeat steps 2 and 3 until convergence is reached.

The above updates are got by means of the Lagrange multiplier method. Notice that only the latter constraint in (5.1) is explicitly considered because the former on non-negative membership degrees is automatically satisfied.

Different convergence conditions can be used. In particular, we have at least three options.

- Compare the membership degree matrices for two consecutive iterations,  $r$  and  $r + 1$ ,

$$\|\mathbf{U}^{(r+1)} - \mathbf{U}^{(r)}\| < \varepsilon,$$

where  $\|\cdot\|$  denotes the Frobenius norm of matrices and  $\varepsilon$  is a fixed value, e.g.,  $10^{-9}$ .

- Compare the centroid matrices for two consecutive iterations

$$\|\mathbf{H}^{(r+1)} - \mathbf{H}^{(r)}\| < \varepsilon.$$

- Compare the values of the objective function of two consecutive iterations

$$|J_{\text{FKM}}^{(r+1)} - J_{\text{FKM}}^{(r)}| < \varepsilon.$$

Like standard non-hierarchical methods, presented in Chap. 3, the number  $k$  of clusters must be specified in advance, and a single partition with  $k$  clusters is obtained. In order to properly choose the number of clusters and to evaluate the quality of the obtained partition, cluster validity indices can be used. As we are going to see, they differ with respect to those available in the hard clustering framework.

### 5.2.1 Cluster Validity Indices

To investigate the clustering performance of a fuzzy algorithm, different measures of partition quality can be used. There exist, at least, two types of measures: fuzziness measures and compactness/separation measures. The former are used to evaluate the partition fuzziness, by summarizing the information contained in the membership degree matrix in one value. Hence, the accuracy of the assignments is checked. Among the fuzziness indices the most known are the Partition Coefficient (PC) and the Partition Entropy (PE) [6]. The PC index is defined as

$$\text{PC} = \sum_{i=1}^n \sum_{g=1}^k \frac{(u_{ig})^2}{n}. \quad (5.4)$$

Its values range from  $1/k$  to 1. The optimal number of cluster  $k$  is achieved when the index value is maximized over different values of  $k$ .

PE is defined as

$$\text{PE} = - \sum_{i=1}^n \sum_{g=1}^k \frac{u_{ig} \log(u_{ig})}{n} \quad (5.5)$$

and takes value in  $[0, \log k]$ . The optimal value of  $k$  corresponds to the minimum of the index over a range of values of  $k$ .

A modification of PC has been proposed by Davé [15] to eliminate dependency on  $k$ :

$$\text{MPC} = 1 - \frac{k}{k-1}(1 - \text{PC}). \quad (5.6)$$

MPC ranges from 0 to 1 and, as for PC, the optimal values are obtained by maximizing the index.

In the category of measures based on compactness/separation, we can find the Xie and Beni (XB) index [54] and the Fuzzy Silhouette (FS) [9].

The first one is defined as the ratio between a compactness and a separation measure:

$$\text{XB} = \frac{\sum_{i=1}^n \sum_{g=1}^k u_{ig}^2 d^2(\mathbf{x}_i, \mathbf{h}_g)}{n \min_{g,g'(g \neq g')} d^2(\mathbf{h}_g, \mathbf{h}_{g'})}. \quad (5.7)$$

The value of XB does not depend on the algorithm used to obtain the membership degrees and the centroids. A smaller value of the numerator indicates more compact clusters, while a larger value of the denominator denotes more separated clusters. Hence, the optimal value of  $k$  is obtained by minimizing the index, that is, by maximizing the compactness (minimizing the numerator) and maximizing the separation between clusters (maximizing the denominator).

A further measure based on the idea of compactness/separation is the fuzzy extension of the Silhouette index S (see Sect. 3.5), the Fuzzy Silhouette FS. This is more appropriate to evaluate a fuzzy partition since it uses the membership degree values. It is defined as

$$\text{FS} = \frac{\sum_{i=1}^n (u_{ig} - u_{ig'})^\alpha s_i}{\sum_{i=1}^n (u_{ig} - u_{ig'})^\alpha}, \quad (5.8)$$

where  $s_i$ , defined in 3.11, is the Silhouette index for unit  $i$ ,  $u_{ig}$  and  $u_{ig'}$  are the first and the second largest elements of the  $i$ -th row of  $\mathbf{U}$ . The parameter  $\alpha$  is a weighting coefficient (usually  $\alpha = 1$ ). As for the Silhouette index, the optimal value of  $k$  corresponds to the maximum of the index.

### 5.2.2 **FKM**

The function **FKM**, contained in the package **fclust**, provides the implementation of the FKM algorithm. In the last version of the package, the number of clusters  $k$  is no longer required as input argument (however, it can be specified by the option `k`) but only the data (in `X`) as an object of class `matrix` or `data.frame`. The optimal number of clusters is automatically chosen by optimizing one of the available fuzzy cluster validity indices to be specified in the option `index`: "PC" (Partition Coefficient), "PE" (Partition Entropy), "MPC" (Modified Partition Coefficient), "SIL" (Silhouette), "SIL.F" (Fuzzy Silhouette), "XB" (Xie and Beni). The default index argument is "SIL.F", and the default possible number of clusters is in the set `2 : 6`, unless a different integer value or vector is selected in `k`.

The user does not need to specify many other arguments due to existing default values. In detail, `m` is the parameter of fuzziness (default: 2); `RS` is the number of (random) starts (default: 1); `stand` is the standardization (default: 0, no standardization); `startU` is the rational start for the membership degree matrix `U` (default: 0, no rational start); `conv` is the convergence criterion based on the objective function (default: `1e-9`); `maxit` is the maximum number of iterations (default: `1e+6`); and `seed` is the seed value for the random number generation (default: `NULL`).

For illustrative purposes, we apply FKM to the dataset `wine` [13] contained in the package **datasetsICR**, already analyzed in Sect. 2.5.1. We use all the default values except that we standardize data (`stand = 1`), we opt for 10 (random) starts and a specific seed value.

```
> library(datasetsICR)
> data("wine")
> Class <- wine$Class
> wine <- wine[, -1]
> library(fclust)
> wine.FKM <- FKM(X = wine, stand = 1, RS = 10,
+                     seed = 264)
The default value k=2:6 has been set
The default index SIL.F has been set
The default value alpha=1 has been set for computing
    SIL.F
```

The output, `wine.FKM`, is an object of class `fclust`, a list with components describing the resulting partition. The optimal value for the number of clusters, automatically obtained as output of the function `FKM`, is  $k = 4$ .

```
> wine.FKM$k
Number of clusters
        4
```

Since the default index is used, it corresponds to the maximum value of the FS index (0.5638547). The FS values for  $k=2:6$  are reported in the output argument criterion.

```
> wine.FKM$criterion
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.4996776 0.5466120 0.5638547 0.5061594 0.2432936
```

We now run the function FKM by choosing a different cluster validity measure, i.e., the Xie and Beni index.

```
> wine.FKM <- FKM(X = wine, stand = 1, index = "XB",
+                      RS = 10, seed = 264)
The default value k=2:6 has been set
```

In this case, the optimal number of clusters, obtained by minimizing the index, is equal to 3.

```
> wine.FKM$k
Number of clusters
            3
> wine.FKM$criterion
      XB k=2           XB k=3           XB k=4           XB k=5
6.634215e-01 4.688938e-01 5.175954e+18 8.925704e+30
      XB k=6
5.017882e+31
```

The optimal value is different from the one suggested by FS, but, since the value of FS for  $k = 3$  (0.5466120) is just a bit lower than the one obtained for  $k = 4$ , we decided to focus on the solution with  $k = 3$ . Note that the extremely large values of XB for  $k > 3$  can be explained by the presence of at least two clusters with virtually the same centroids, suggesting a too high number of clusters. Considering the variable Class, we know that  $k = 3$  is indeed the true number of clusters.

```
> wine.FKM.3 <- FKM(X = wine, k = 3, stand = 1,
+                      RS = 10, seed = 264)
```

The assignments of all the units and the corresponding membership degrees are contained in the output argument clus, a matrix with  $n$  rows and two columns, labeled cluster and Membership degree, respectively.

The cluster size can be obtained by means of the function `cl.size`. It produces the sizes of the clusters and needs as input argument the membership degree matrix. Each unit is assigned to a cluster according to the maximum membership degree getting the so-called closest hard clustering partition. The three clusters contain 51, 62, and 65 units, respectively.

```
> cl.size(wine.FKM.3$U)
Clus 1  Clus 2  Clus 3
      51       62       65
```

In order to interpret and characterize the obtained clusters, the centroid features are reported.

```
> wine.FKM.3$H
          Alcohol Malic acid           Ash
Clus 1  0.05402085  0.7713744  0.1332914
Clus 2  0.71389339 -0.3457861  0.2064307
Clus 3 -0.67440645 -0.3212858 -0.3584025
          Alcalinity of ash   Magnesium Total phenols
Clus 1            0.4549067 -0.09915824 -0.85185725
Clus 2            -0.5808811  0.41184057  0.79086090
Clus 3            0.1533944 -0.42058030 -0.06714076
          Flavanoids Nonflavanoid phenols Proanthocyanins
Clus 1 -1.01555063             0.66111527 -0.66624469
Clus 2  0.83846846             -0.54259971  0.54561841
Clus 3  0.02930926             -0.03543114 -0.01153538
          Color intensity        Hue
Clus 1            0.6552466 -0.8975154
Clus 2            0.1719347  0.4159182
Clus 3            -0.6692050  0.3171835
          OD280/OD315 of diluted wines    Proline
Clus 1                  -1.0157960 -0.3644820
Clus 2                  0.6564268  0.9437575
Clus 3                  0.2568837 -0.5655673
```

These are standardized since in the function `FKM` the standardization option has been set equal to 1. To better interpret them, it is useful to express them with respect to the original units of measurement. In the package `fclust`, the function `Hraw` transforms the standardized centroids using the original units of measurement of  $X$ .

```
> Hraw(wine.FKM.3$X, wine.FKM.3$H)
          Alcohol Malic acid           Ash Alkalinity of ash
Clus 1 13.04447     3.198086 2.403085            21.01413
Clus 2 13.58018     1.950055 2.423150            17.55505
Clus 3 12.45312     1.977425 2.268191            20.00721
          Magnesium Total phenols Flavanoids
Clus 1   98.32535      1.761977  1.014878
Clus 2 105.62368      2.790073  2.866781
Clus 3  93.73464      2.253092  2.058545
          Nonflavanoid phenols Proanthocyanins
Clus 1           0.4441319    1.209568
Clus 2           0.2943256    1.903188
Clus 3           0.3574444    1.584297
          Color intensity       Hue
Clus 1           6.577139  0.7523029
Clus 2           5.456684  1.0525165
Clus 3           3.506681  1.0299486
          OD280/OD315 of diluted wines   Proline
Clus 1           1.890480  632.1152
Clus 2           3.077742 1044.0895
Clus 3           2.794070  568.7919
```

Cluster 1 is characterized by the highest values of malic acid, nonflavanoid phenols, color intensity, and the lowest values of total phenols, flavanoids, proanthocyanins, hue, and OD280/OD315 of diluted wines. Cluster 2 contains wines with the highest values of alcohol, magnesium, total phenols, flavanoids, proanthocyanins, OD280/OD315 of diluted wines, and proline, while wines in Cluster 3 are mainly characterized by the lowest values of magnesium, color intensity, and proline. Notice that these centroids are consistent with the cluster means obtained by applying Ward's method (see Sect. 2.5).

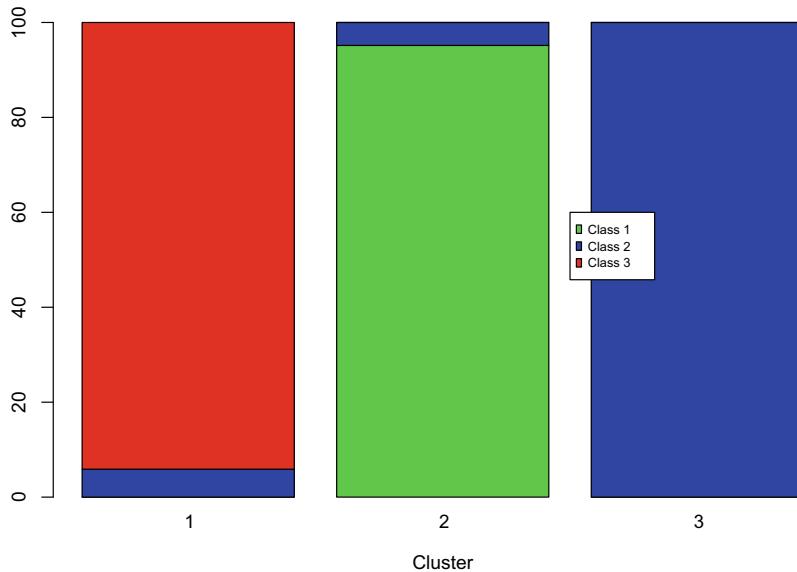
We compare the obtained partition with the known classification (in Class).

```
> table(Class, wine.FKM.3$cclus[, 1])

Class   1   2   3
  1    0  59   0
  2    3   3  65
  3   48   0   0
```

It results that units of Class 1 and Class 3 are all assigned to Cluster 2 and Cluster 1, respectively. Instead, units of Class 2 are mainly assigned to Cluster 3 but for three units assigned to Cluster 1 and three to Cluster 2. Hence, the three clusters can be interpreted in terms of the cultivars, and only six out of 178 wines are misclassified. Notice that this number is lower than the one observed for Ward's method (13 misclassified wines).

The relation between the FKM partition and the cultivars can be graphically represented by the barplot shown in Fig. 5.2. As we can notice, there is an almost perfect correspondence between the partition and the known classification.



**Fig. 5.2** wine data: barplot of the FkM solution ( $k = 3$  clusters) by the variable Class

```
> perc.wine <- table(Class, wine.FKM.3$clus[, 1]) /
+   rbind(colsums(table(Class, wine.FKM.3$clus[, 1])),
+         colsums(table(Class, wine.FKM.3$clus[, 1])),
+         colsums(table(Class, wine.FKM.3$clus[, 1])),
+         ) * 100
> barplot(perc.wine,
+           beside = FALSE,
+           col = c("green", "blue", "red"),
+           cex.names = 0.7,
+           sub = "Cluster")
> legend(2.5, 60, c("Class 1", "Class 2", "Class 3"),
+         fill = c("green", "blue", "red"),
+         bg = "white", cex = 0.7, text.width = 0.3)
```

We now analyze the misclassified units. The three units of Class 2 assigned to Cluster 1 are characterized by the following membership degrees.

```
> i_mc1 <- which(Class == 2 & wine.FKM.3$clus[, 1]
+                   == 1)
> i_mc1
Obj 62  Obj 84  Obj 119
       62      84      119
> wine.FKM.3$clus[i_mc1, ]
  Cluster Membership degree
Obj 62            1            0.4398723
Obj 84            1            0.6149427
```

```
Obj 119           1           0.4726816
> wine.FKM.3$U[i_mc1, ]
      Clus 1     Clus 2     Clus 3
Obj 62   0.4398723 0.1811209 0.3790068
Obj 84   0.6149427 0.1225309 0.2625264
Obj 119  0.4726816 0.1576827 0.3696357
```

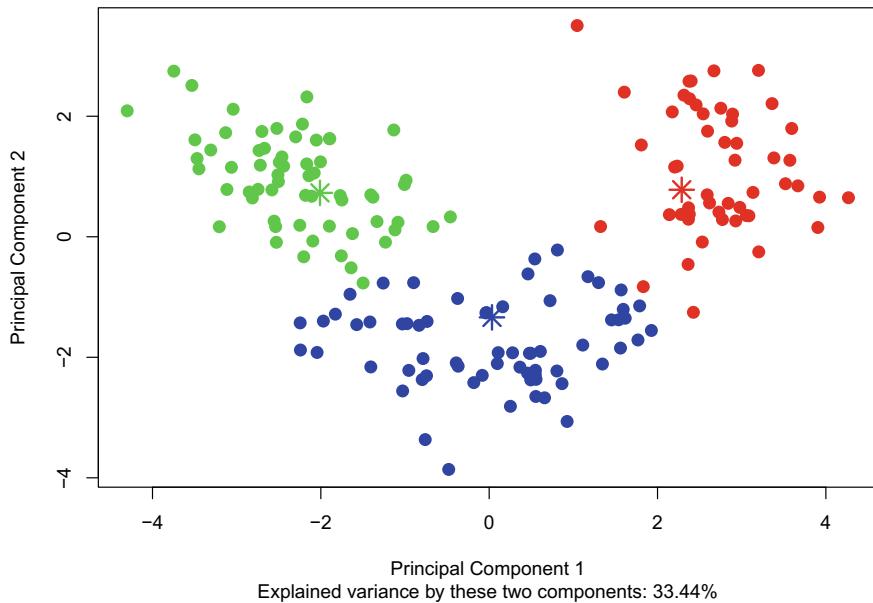
As we can notice, these units are assigned to Cluster 1 with low membership degrees and for two of them the assignment is not “clear” (we refer to an unclear assignment when the maximum membership degree of a unit to a cluster is lower than 0.5). The same also holds for three units of Class 2 assigned to Cluster 2.

```
> i_mc2 <- which(Class == 2 & wine.FKM.3$clus[, 1]
+                   == 2)
> i_mc2
Obj 74   Obj 96  Obj 122
    74       96      122
> wine.FKM.3$clus[i_mc2, ]
      Cluster Membership degree
Obj 74           2           0.4354254
Obj 96           2           0.4255220
Obj 122          2           0.3649285
> wine.FKM.3$U[i_mc2, ]
      Clus 1     Clus 2     Clus 3
Obj 74   0.2228015 0.4354254 0.3417731
Obj 96   0.2366013 0.4255220 0.3378767
Obj 122  0.2732887 0.3649285 0.3617828
```

All the above units have intermediate characteristics, and this is also suggested by the values of the membership degrees. In all these cases, the membership degrees are lower than 0.5.

To visualize the fuzzy partition, we can use the function `plot.fclust` of the package **fclust**. The function creates a scatterplot visualizing the cluster structure. The units are represented by points in the plot using observed variables or principal components. The input arguments are `x` (object of class `fclust`), `v1v2` (vector with two elements specifying the numbers of the variables, or of the principal components, to be plotted), `colclus` (vector specifying the color palette for the clusters), `umin` (lowest maximum membership degree such that a unit is considered to be assigned to a cluster), `ucex` (logical value specifying if the points are magnified according to the maximum membership degree), and `pca` (logical value specifying if the units are represented by principal components). Additional arguments are those of the basic function `plot`. A plot of the data onto the low-dimensional space spanned by the first two principal components is displayed in Fig. 5.3 (except for `pca`, default options are used).

```
> plot.fclust(x = wine.FKM.3, pca = TRUE)
```



**Fig. 5.3** wine data: scatterplot of the FkM solution ( $k = 3$  clusters) using the first two principal components (blue points: cluster 1, red points: cluster 2, green points: cluster 3)

### 5.2.3 **cmeans**

An alternative implementation of the FkM algorithm is provided by the function **cmeans** contained in the package **e1071**. This function requires as input arguments the data matrix, **x**, as well the number of clusters, **centers**. Unlike the function **FKM**, neither the standardization option nor the possibility to choose the number of random starts is available, while, also in this case, the default value for the parameter of fuzziness is 2.

To standardize the data (as already seen in the previous chapters), the function **scale** can be used.

```
> wine.Z <- scale(wine, center = TRUE, scale = TRUE)
```

Also in this case we focus on the solution with  $k = 3$  clusters.

```
> set.seed(264)
> library(e1071)
> wine.Z.cmeans.3 <- cmeans(wine.Z, 3)
```

The assignments are contained in the output component cluster.

Details on the cluster size are obtained by the output argument `size`.

```
> wine.Z.cmeans.3$size  
[1] 51 65 62
```

By comparing the results of cmeans with those obtained by FKM, via a cross-tabulation, we can notice that the two partitions coincide.

```
> table(wine.FKM.3$clus[, 1], wine.Z.cmeans.3$cluster)

      1   2   3
1 51  0  0
2  0  0 62
3  0 65  0
```

In the vector `centers`, the standardized centroids are contained. They obviously coincide with those previously obtained.

```

> wine.Z.cmeans.3$centers
          Alcohol Malic acid           Ash Alkalinity of ash
1  0.05400681  0.7713678  0.1332822      0.4549055
2 -0.67432853 -0.3212848 -0.3583906      0.1533307
3  0.71382303 -0.3457733  0.2064298     -0.5808272
          Magnesium Total phenols Flavanoids
1 -0.09918577   -0.85186711 -1.01555704
2 -0.42051362   -0.06705617  0.02937869
3  0.41181160    0.79080300  0.83841450
Nonflavanoid phenols Proanthocyanins Color intensity
1            0.66114490   -0.66625516      0.6552214
2           -0.03551205   -0.01147738     -0.6691533
3           -0.54256572    0.54557695      0.1718908
          Hue OD280 / OD315 of diluted wines Proline
1 -0.8975058                  -1.0157916   -0.3644891
2  0.3171955                  0.2569219   -0.5654881
3  0.4159107                  0.6563952   0.9436799

```

Also in this case, in order to obtain the centroids in the original units of measurement, we can use the function `Hraw` of the package **fclust**.

Finally, the membership degrees are stored in the output argument `membership`.

### 5.2.4 **fcm**

The third function, called `fcm`, that partitions a dataset of quantitative variables by using the FkM clustering algorithm is included in the package **ppclust**. It requires the data in `x` (`matrix` or `data.frame`) and the number of clusters, `centers`, to be specified. There is an input argument related to the type of distance, `dmetric`, that is equal to "squeuclidean" (squared Euclidean) by default, but it can also be set equal to "minkowski". In addition, as for the function `FKM`, the number of random starts, `nstart`, can be chosen and the data can be standardized by the option `stand = TRUE`. We fix the number of clusters equal to  $k = 3$ .

```
> library(ppclust)
> wine.fcm.3 <- fcm(wine, centers = 3,
+                      stand = TRUE, numseed = 264)
```

The output arguments are the following.

```
> names(wine.fcm.3)
[1] "u"          "v"          "v0"
[4] "d"          "x"          "cluster"
[7] "csiz"       "sumsqrs"    "k"
[10] "m"          "iter"       "best.start"
[13] "func.val"   "comp.time"  "inpargs"
[16] "algorithm"  "call"
```

In detail, the obtained partition is contained in the vector `cluster` where the assignments of the units to the clusters are stored.

```
> wine.fcm.3$cluster
 1  2  3  4  5  6  7  8  9 10 11 12 13
 2  2  2  2  2  2  2  2  2  2  2  2  2
14 15 16 17 18 19 20 21 22 23 24 25 26
 2  2  2  2  2  2  2  2  2  2  2  2  2
27 28 29 30 31 32 33 34 35 36 37 38 39
 2  2  2  2  2  2  2  2  2  2  2  2  2
40 41 42 43 44 45 46 47 48 49 50 51 52
 2  2  2  2  2  2  2  2  2  2  2  2  2
53 54 55 56 57 58 59 60 61 62 63 64 65
 2  2  2  2  2  2  2  3  3  1  3  3  3
```

66	67	68	69	70	71	72	73	74	75	76	77	78
3	3	3	3	3	3	3	3	2	3	3	3	3
79	80	81	82	83	84	85	86	87	88	89	90	91
3	3	3	3	3	1	3	3	3	3	3	3	3
92	93	94	95	96	97	98	99	100	101	102	103	104
3	3	3	3	2	3	3	3	3	3	3	3	3
105	106	107	108	109	110	111	112	113	114	115	116	117
3	3	3	3	3	3	3	3	3	3	3	3	3
118	119	120	121	122	123	124	125	126	127	128	129	130
3	1	3	3	2	3	3	3	3	3	3	3	3
131	132	133	134	135	136	137	138	139	140	141	142	143
1	1	1	1	1	1	1	1	1	1	1	1	1
144	145	146	147	148	149	150	151	152	153	154	155	156
1	1	1	1	1	1	1	1	1	1	1	1	1
157	158	159	160	161	162	163	164	165	166	167	168	169
1	1	1	1	1	1	1	1	1	1	1	1	1
170	171	172	173	174	175	176	177	178				
1	1	1	1	1	1	1	1	1				

The cluster size is provided by the output called `csize`.

```
> wine.fcm.3$csize
 1  2   3
51 62 65
```

The resulting partition is exactly the same as those obtained with the functions described in the previous sections, as reported in the following cross-tabulations.

```
> table(wine.fcm.3$cluster, wine.FKM.3$clus[, 1])
      1   2   3
 1 51  0  0
 2  0 62  0
 3  0  0 65
> table(wine.fcm.3$cluster, wine.Z.cmeans.3$cluster)
      1   2   3
 1 51  0  0
 2  0  0 62
 3  0 65  0
```

Since data are standardized, the values contained in `v` correspond to the standardized centroids.

```
> wine.fcm.3$v
          Alcohol  Malic acid           Ash
Cluster 1  0.05402085  0.7713744  0.1332914
```

```

Cluster 2  0.71389339 -0.3457861  0.2064307
Cluster 3 -0.67440645 -0.3212858 -0.3584025
          Alcalinity of ash   Magnesium
Cluster 1           0.4549067 -0.09915824
Cluster 2           -0.5808811  0.41184057
Cluster 3           0.1533944 -0.42058030
          Total phenols   Flavanoids
Cluster 1         -0.85185725 -1.01555063
Cluster 2         0.79086090  0.83846846
Cluster 3         -0.06714076  0.02930926
          Nonflavanoid phenols Proanthocyanins
Cluster 1         0.66111527      -0.66624469
Cluster 2         -0.54259971      0.54561841
Cluster 3         -0.03543114      -0.01153538
          Color intensity       Hue
Cluster 1         0.6552466 -0.8975154
Cluster 2         0.1719347  0.4159182
Cluster 3         -0.6692050  0.3171835
          OD280/OD315 of diluted wines   Proline
Cluster 1           -1.0157960 -0.3644820
Cluster 2           0.6564268  0.9437575
Cluster 3           0.2568837 -0.5655673

```

The membership degrees of the units are contained in the matrix  $u$  (the output is omitted for the sake of brevity).

### 5.2.5 **fuzzy.CM**

The last implementation of the FkM algorithm is included in the package **advclust** via the function **fuzzy.CM**. The main input arguments are the dataset,  $X$  (matrix or `data.frame`), the number of clusters,  $K$ , the parameter of fuzziness,  $m$ , the maximum number of iterations for convergence, `max.iteration`, and the convergence criteria, `threshold`.

Since the standardization option is not provided by the function **fuzzy.CM**, we use the standardized wine dataset, `wine.Z`.

```

> library(advclust)
> wine.Z.fuzzy.CM.3 <- fuzzy.CM(wine.Z, K = 3, m = 2,
+                                     max.iteration = 100,
+                                     threshold = 1e-5,
+                                     RandomNumber = 264)

Membership initialized randomly

iteration:      29
Finish :)

```

The above-printed output informs the user that the membership degree matrix has been initialized randomly and the convergence has been reached after 29 iterations. The structure of the output is as follows.

```
> str(wine.Z.fuzzy.CM.3)
Formal class 'fuzzycluster' [package "advclust"] with
 8 slots
..@ centroid      : num [1:3, 1:13] -0.674 0.714 0.054
   -0.321 -0.346 ...
.. ..- attr(*, "dimnames")=List of 2
.. .. .$. : NULL
.. .. .$. : chr [1:13] "Alcohol" "Malic acid" "Ash"
   "Alcalinity of ash" ...
..@ distance      : num [1:178, 1:3] 21.9 13.2 15.4
   32.8 10.9 ...
..@ func.obj      : num 717
..@ call.func     : chr [1:2] "fuzzy.CM(X = wine.Z, K
  = 3, m = 2, max.iteration = 100, threshold = 1e
  -05, " " RandomNumber = 264)"
..@ fuzzyfier     : num 2
..@ method.fuzzy: chr "Fuzzy C-Means"
..@ member        : num [1:178, 1:3] 0.171 0.292 0.194
   0.184 0.31 ...
..@ hard.label    : int [1:178] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
...
...
```

The hard assignment is contained in the output argument `hard.label`. Notice the use of @ (instead of \$) to display `hard.label` because the output of `fuzzy.CM` is an object of the so-called S4 class (instead of the S3 class as it is for all the output objects discussed so far).

```
> wine.Z.fuzzy.CM.3@hard.label
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[24] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[47] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[70] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[93] 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[116] 1 1 1 3 1 1 2 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[139] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[162] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

The cluster sizes can be visualized by the following code.

```
> table(wine.Z.fuzzy.CM.3@hard.label)

 1   2   3
65 62 51
```

The obtained partition is the same of the functions FKM, cmeans and fcm.

```
> table(wine.Z.fuzzy.CM.3@hard.label,
+        wine.FKM.3$clus[, 1])

 1 2 3
1 0 65
2 0 62
3 51 0

> table(wine.Z.fuzzy.CM.3@hard.label,
+        wine.Z.cmeans.3$cluster)

 1 2 3
1 0 65 0
2 0 62
3 51 0

> table(wine.Z.fuzzy.CM.3@hard.label,
+        wine.fcm.3$cluster)

 1 2 3
1 0 0 65
2 0 62 0
3 51 0 0
```

The centroids and the membership degrees of the function `fuzzy.CM` are contained in the output components (called slots for the S4 class) `centroid` and `member`, respectively.

In the package **advclust**, as for **fclust**, the function `validation.index` can be used to compute cluster validity indices for a given number of clusters. The function needs as input argument the output of `fuzzy.CM`.

```
> validation.index(wine.Z.fuzzy.CM.3)
Validation Index result:
|                               | Value |
|:-----|-----|
| Partition Coefficient          | 0.476 |
| Modified Partition Coefficient | 0.214 |
| Classification Entropy        | 0.894 |
| Xie Beni                      | 0.469 |
| Separation                     | 0.469 |
| Kwon                           | 86.322 |
| Tang                           | 84.894 |
```

The printed values are related to the following indices: PC [6], MPC [15], PE [6], XB [54], Kwon [36], and Tang [48]. See, for more details on those new indices, [51]. Obviously, the values of PC, MPC, PE (indicated above as classification entropy), and XB are exactly the same provided by the function `Fclust.index` of the package **fclust**.

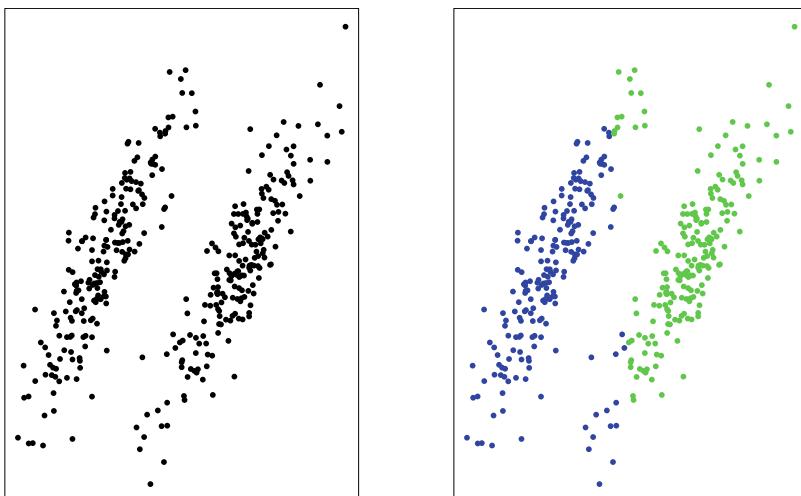
```
> Fclust.index(wine.FKM.3)
The default value alpha=1 has been set for computing
SIL.F
      PC          PE          MPC          SIL          SIL.F
0.4761498  0.8944189  0.2142247  0.4508372  0.5466120
      XB
0.4688938
```

### 5.3 Gustafson-Kessel Extensions of Fuzzy $k$ -Means

In the FkM algorithm, as in standard  $k$ -Means, the (squared) Euclidean distance is used. This leads to spherical clusters. In the presence of non-spherical clusters, the FkM may fail to properly recognize the clusters (see, for example, Fig. 5.4).

To overcome this limitation, Gustafson and Kessel [26] propose to extend the FkM algorithm by replacing the Euclidean distance by a cluster-specific Mahalanobis distance:

$$d_M^2(\mathbf{x}_i, \mathbf{h}_g) = (\mathbf{x}_i - \mathbf{h}_g)' \mathbf{M}_g (\mathbf{x}_i - \mathbf{h}_g), \quad (5.9)$$



**Fig. 5.4** An example of non-spherical clusters: data (on the left) and FkM solution with  $k = 2$  clusters where the different colors identify the clusters (on the right)

where  $\mathbf{M}_g$  is a symmetric and positive-definite matrix. Notice that the distance in (5.9) is equal to the Euclidean when  $\mathbf{M}_g$  is the identity matrix.

The optimization problem of the Gustafson-Kessel-type Fuzzy  $k$ -Means (GK-FkM) can be written as follows:

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{H}, \mathbf{M}_1, \dots, \mathbf{M}_k} J_{\text{GK-FkM}} &= \sum_{i=1}^n \sum_{g=1}^k u_{ig}^m d_M^2(\mathbf{x}_i, \mathbf{h}_g), \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n, \\ |\mathbf{M}_g| &= \rho_g > 0, \quad g = 1, \dots, k. \end{aligned} \quad (5.10)$$

Unlike the FkM algorithm, an additional constraint on  $\mathbf{M}_g$  is added. Usually  $\rho_g = 1$  is set. This avoids the trivial solution  $\mathbf{M}_g = \mathbf{0}$ , which would be determined since  $J_{\text{GK-FkM}}$  is linear in  $\mathbf{M}_g$ .

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible membership degree matrix  $\mathbf{U}$ .
2. Given  $\mathbf{U}$  and  $\mathbf{M}_g$ ,  $g = 1 \dots, k$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n u_{ig} \mathbf{x}_i}{\sum_{i=1}^n u_{ig}}, \quad g = 1, \dots, k. \quad (5.11)$$

3. Given  $\mathbf{U}$  and  $\mathbf{H}$ , update the matrix  $\mathbf{M}_g$ ,  $g = 1, \dots, k$ ,

$$\mathbf{M}_g = (|\Sigma_g|)^{\frac{1}{p}} \Sigma_g^{-1}, \quad (5.12)$$

where

$$\Sigma_g = \frac{\sum_{i=1}^n u_{ig}^m (\mathbf{x}_i - \mathbf{h}_g)(\mathbf{x}_i - \mathbf{h}_g)'}{\sum_{i=1}^n u_{ig}^m} \quad (5.13)$$

is the fuzzy covariance matrix of the  $g$ -th cluster.

4. Given  $\mathbf{H}$  and  $\mathbf{M}_g$ ,  $g = 1 \dots, k$ , update the membership degree matrix  $\mathbf{U}$ :

$$u_{ig} = \frac{1}{\sum_{g'=1}^k \left( \frac{d_M^2(\mathbf{x}_i, \mathbf{h}_{g'})}{d_M^2(\mathbf{x}_i, \mathbf{h}_g)} \right)^{\frac{1}{m-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k. \quad (5.14)$$

5. Repeat steps 2, 3, and 4 until convergence is reached.

The eigenvalues and eigenvectors of  $\Sigma_g$  describe the shape and orientation of the  $g$ -th cluster. Unfortunately, when an eigenvalue is equal to 0 or when the ratio between the maximum and the minimal eigenvalues is very large, the matrix is nearly singular; hence,  $\Sigma_g^{-1}$  cannot be calculated. The condition  $|\Sigma_g| = \rho_g$  cannot overcome this drawback, as the determinant becomes 0. Babuska et al. [3] propose to avoid these

numerical problems by constraining the ratio between the maximum and minimal eigenvalues to be smaller than a predefined threshold (hereinafter, GKB-FkM algorithm). When this ratio exceeds the threshold, the covariance matrix is reconstructed by  $\mathbf{D}_g \mathbf{E}_g \mathbf{D}_g^{-1}$ , where  $\mathbf{E}_g$  and  $\mathbf{D}_g$  are, respectively, a diagonal matrix containing the modified eigenvalues and a matrix containing the corresponding eigenvectors as columns of  $\boldsymbol{\Sigma}_g$ . Note that the eigenvalues are modified by increasing the smallest ones in such a way that the additional constraint is satisfied. Unfortunately, this can lead to overfit the data. A way to avoid it is to apply the eigendecomposition to

$$(1 - \gamma) \boldsymbol{\Sigma}_g + \gamma |\boldsymbol{\Sigma}|^{1/p} \mathbf{I}, \quad (5.15)$$

where  $\gamma \in [0, 1]$  and  $\boldsymbol{\Sigma}$  is the covariance matrix of the whole dataset.

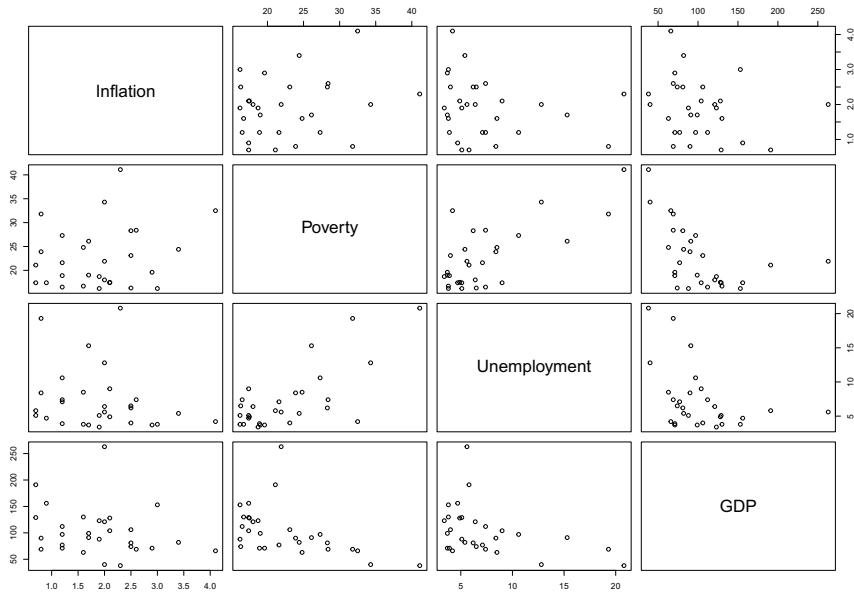
The GK-FkM algorithm is implemented in the following packages: **fclust** (function `FKM.gk`) and **ppclust** (function `gk`). The GKB-FkM algorithm is implemented in **fclust** (function `FKM.gkb`) and **advclust** (function `fuzzy.GK`).

### 5.3.1 ***FKM.gk***

This section describes the implementation of the GK-FkM clustering algorithm of the package **fclust**: the function `FKM.gk`. The performance of the function `FKM.gk` is evaluated by using the dataset `Eurostat` included in the package **datasetsICR**. The dataset contains four economic indicators observed in some European countries in 2018: annual HICP inflation rate (in percentage) (`Inflation`), people at risk of poverty or social exclusion (`Poverty`), total unemployment rate (`Unemployment`), and GDP per capita in PPS (`GDP`).

Taking a look at the matrix of scatterplots in Fig. 5.5, we wonder whether non-spherical clusters exist.

```
> library(datasetsICR)
> data("Eurostat")
> str(Eurostat)
'data.frame':   29 obs. of  4 variables:
 $ Inflation    : num  0.7 1.9 3.4 0.7 0.8 1.7 2.1 ...
 $ Poverty       : num  17.4 18.7 24.4 21.1 31.8 ...
 $ Unemployment : num  5.1 3.4 5.4 5.8 19.3 15.3 9 ...
 $ GDP          : int  129 123 82 191 69 91 104 63 ...
> row.names(Eurostat)
[1] "Denmark"           "Germany"          "Estonia"
[4] "Ireland"            "Greece"            "Spain"
[7] "France"             "Croatia"           "Italy"
[10] "Cyprus"             "Latvia"            "Lithuania"
[13] "Luxembourg"         "Hungary"           "Malta"
[16] "Netherlands"        "Austria"           "Poland"
[19] "Portugal"            "Romania"           "Slovenia"
[22] "Slovakia"           "Finland"           "Sweden"
```



**Fig. 5.5** Eurostat data: matrix of scatterplots

```
[25] "UK"           "Norway"        "Switzerland"
[28] "NorthMacedonia" "Serbia"
> plot(Eurostat)
```

The input arguments of the function `FKM.gk` are the same as for the function `FKM` except for the volume parameter, `vp` (default: `rep(1, k)`), i.e.,  $\rho_g$ . We run the function for  $k=2:5$ , by using the `XB` index and 10 random starts, to prevent local optima.

```
> library(fclust)
> EURO.FKM.gk <- FKM.gk(Eurostat, k = 2:5,
+                           index = "XB",
+                           RS = 10, seed = 123)
```

The optimal number of clusters, corresponding to the minimum of the `XB` index (0.3764435), is reached for  $k = 2$ .

```
> EURO.FKM.gk$k
Number of clusters
                2
> EURO.FKM.gk$criterion
      XB k=2       XB k=3       XB k=4       XB k=5
    0.3764435   1.7745923  143.0723763  25.0965283
```

The values of the objective function for the 10 random starts are contained in the output argument `value`.

```
> EURO.FKM.gk$value
 Start 1   Start 2   Start 3   Start 4   Start 5
972.3388 972.3388 972.3388 972.3388 972.3388
 Start 6   Start 7   Start 8   Start 9   Start 10
972.3388 972.3388 972.3388 972.3388 972.3388
```

The above values suggest that the solution corresponds to the global optimum because the ten runs of the algorithm, starting from a different initial guess, always converge to the same value.

Unlike the other clustering algorithms in the package, `FKM.gk` is also characterized by the output argument `F`, an array containing the covariance matrices of all the clusters, i.e., the matrices  $\Sigma_g$ .

```
> EURO.FKM.gk$F[, , 1]
          Inflation      Poverty Unemployment
Inflation  0.05064247 -0.02889397 -0.011231866
Poverty    -0.02889397  0.45503358  0.039743457
Unemployment -0.01123187  0.03974346  0.212594322
GDP        -0.73023981  5.32481504 -0.009343862
          GDP
Inflation  -0.730239811
Poverty    5.324815044
Unemployment -0.009343862
GDP        283.811655226
> EURO.FKM.gk$F[, , 2]
          Inflation      Poverty Unemployment
Inflation  0.06738350  0.07949867 -0.1933109
Poverty    0.07949867  2.60860384  1.6182626
Unemployment -0.19331086  1.61826261  2.4197288
GDP        -0.27700550 -7.95766843 -4.5140534
          GDP
Inflation  -0.2770055
Poverty    -7.9576684
Unemployment -4.5140534
GDP        36.0426017
```

We can notice that the correlation structure is different for the two clusters (`corr1` and `corr2`) and quite different from the identity matrix; hence, the use of the GK-FKM algorithm appears to be fruitful.

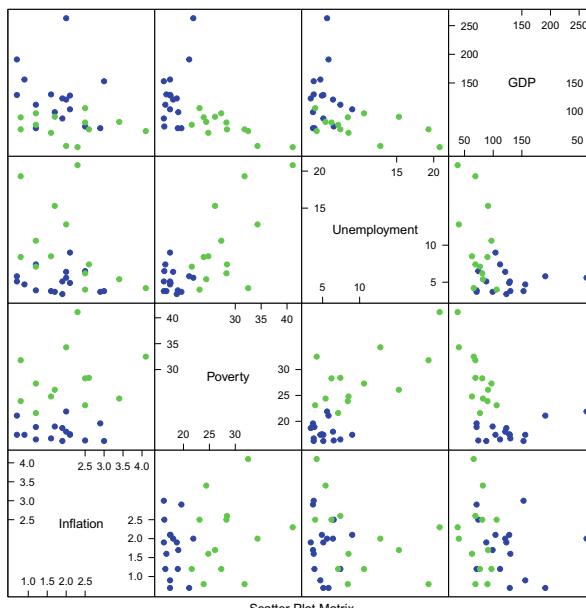
```
> corr1 <- matrix(0, nrow = 4, ncol = 4)
> for (i in 1:4)
+ {
+   for (j in 1:4)
+   {
+     corr1[i,j] <- EURO.FKM.gk$F[i, j, 1]/
+                   (sqrt(EURO.FKM.gk$F[i, i, 1])*
+                    sqrt(EURO.FKM.gk$F[j, j, 1]))
+   }
+ }
> corr2 <- matrix(0, nrow = 4, ncol = 4)
> for (i in 1:4)
+ {
+   for (j in 1:4)
+   {
+     corr2[i,j] <- EURO.FKM.gk$F[i, j, 2]/
+                   (sqrt(EURO.FKM.gk$F[i, i, 2])*
+                    sqrt(EURO.FKM.gk$F[j, j, 2]))
+   }
+ }
> rownames(corr1) <- rownames(cor(Eurostat))
> colnames(corr1) <- rownames(corr1)
> rownames(corr2) <- rownames(cor(Eurostat))
> colnames(corr2) <- rownames(corr2)
> corr1
      Inflation    Poverty Unemployment
Inflation 1.0000000 -0.1903391 -0.108247693
Poverty   -0.1903391 1.0000000  0.127781531
Unemployment -0.1082477 0.1277815  1.000000000
GDP       -0.1926163 0.4685627 -0.001202916
                                         GDP
Inflation -0.192616257
Poverty   0.468562675
Unemployment -0.001202916
GDP       1.000000000
> corr2
      Inflation    Poverty Unemployment
Inflation 1.0000000  0.1896178 -0.4787360
Poverty   0.1896178 1.0000000  0.6441123
Unemployment -0.4787360 0.6441123  1.0000000
GDP       -0.1777475 -0.8206797 -0.4833651
                                         GDP
Inflation -0.1777475
Poverty   -0.8206797
Unemployment -0.4833651
GDP       1.0000000
> cor(Eurostat)
      Inflation    Poverty Unemployment
```

Inflation	1.0000000	0.2096722	-0.2085215
Poverty	0.2096722	1.0000000	0.7176703
Unemployment	-0.2085215	0.7176703	1.0000000
GDP	-0.2248368	-0.4938166	-0.4048039
	GDP		
Inflation	-0.2248368		
Poverty	-0.4938166		
Unemployment	-0.4048039		
GDP	1.0000000		

The partition obtained by means of the FKM.gk algorithm is shown in Fig. 5.6 by using the function `spalom` of the package **lattice** [47].

```
> library(lattice)
> spalom(Eurostat, groups = EURO.FKM.gk$clus[, 1],
+         pch = 19, col = c("blue", "green"))
```

The cluster sizes are 16 and 13, respectively. Cluster 1 contains Denmark, Germany, Ireland, France, Luxembourg, Hungary, Malta, Netherlands, Austria, Poland, Slovenia, Slovakia, Finland, Sweden, Norway, and Switzerland, while European countries in Cluster 2 are Estonia, Greece, Spain, Croatia, Italy, Cyprus, Latvia, Lithuania, Portugal, Romania, UK, North Macedonia, and Serbia.



**Fig. 5.6** Eurostat data: matrix of scatterplots of the GK-FkM solution with  $k = 2$  clusters (blue points: cluster 1, green points: cluster 2)

```

> cl.size(EURO.FKM.gk$U)
Clus 1 Clus 2
  16      13
> EURO.FKM.gk$clus
           Cluster Membership degree
Denmark            1       0.8045272
Germany           1       0.7924806
Estonia            2       0.7830440
Ireland            1       0.9611438
Greece             2       0.9709922
Spain              2       0.9075574
France             1       0.6596737
Croatia            2       0.8035894
Italy               2       0.8667271
Cyprus              2       0.7564900
Latvia              2       0.9812209
Lithuania          2       0.9508444
Luxembourg         1       0.9709924
Hungary             1       0.6999522
Malta               1       0.6364979
Netherlands        1       0.8412127
Austria             1       0.9621626
Poland              1       0.7677517
Portugal            2       0.5681399
Romania             2       0.9274432
Slovenia            1       0.9227918
Slovakia            1       0.9066941
Finland             1       0.6464076
Sweden              1       0.9046242
UK                  2       0.7334378
Norway              1       0.8411200
Switzerland         1       0.9169116
NorthMacedonia     2       0.9740918
Serbia              2       0.9617765
> row.names(Eurostat[EURO.FKM.gk$clus[, 1] == 1, ])
[1] "Denmark"        "Germany"        "Ireland"
[4] "France"          "Luxembourg"     "Hungary"
[7] "Malta"           "Netherlands"   "Austria"
[10] "Poland"          "Slovenia"       "Slovakia"
[13] "Finland"         "Sweden"        "Norway"
[16] "Switzerland"
> row.names(Eurostat[EURO.FKM.gk$clus[, 1] == 2, ])
[1] "Estonia"         "Greece"        "Spain"
[4] "Croatia"         "Italy"          "Cyprus"
[7] "Latvia"          "Lithuania"     "Portugal"
[10] "Romania"         "UK"             "NorthMacedonia"
[13] "Serbia"

```

Finally, we can further characterize the partition by means of the centroid features.

```
> EURO.FKM.gk$H
      Inflation Poverty Unemployment      GDP
Clus 1  1.761105 18.31658      5.22551 130.71526
Clus 2  2.088690 28.58108     10.33598  74.08919
```

Countries in Cluster 1 are characterized by the lowest values of Inflation, Poverty, and Unemployment and the highest value of GDP, while, on the contrary, those in Cluster 2 by the highest values of Inflation, Poverty, and Unemployment and the lowest value of GDP.

The highest and the lowest membership degrees to Cluster 1 correspond to Luxembourg and Malta, respectively. Comparing their features with the corresponding centroid, we can notice that Luxembourg is closer to it than Malta. On the other hand, Latvia and Portugal have the highest and the lowest membership degree to Cluster 2. In particular, taking a look at features for Portugal, we can see that it shares the features of the two clusters.

```
> which.max(EURO.FKM.gk$clus[
+   EURO.FKM.gk$clus[, 1] == 1, 2])
Luxembourg
      5
> EURO.FKM.gk$clus["Luxembourg", 2]
[1] 0.9709924
> Eurostat["Luxembourg", ]
      Inflation Poverty Unemployment      GDP
Luxembourg          2     21.9      5.6 263
> which.min(EURO.FKM.gk$clus[
+   EURO.FKM.gk$clus[, 1] == 1, 2])
Malta
      7
> EURO.FKM.gk$clus["Malta", 2]
[1] 0.6364979
> Eurostat["Malta", ]
      Inflation Poverty Unemployment      GDP
Malta            1.7       19      3.7 99
> which.max(EURO.FKM.gk$clus[
+   EURO.FKM.gk$clus[, 1] == 2, 2])
Latvia
      7
> EURO.FKM.gk$clus["Latvia", 2]
[1] 0.9812209
> Eurostat["Latvia", ]
      Inflation Poverty Unemployment      GDP
Latvia           2.6     28.4      7.4 69
> which.min(EURO.FKM.gk$clus[
+   EURO.FKM.gk$clus[, 1] == 2, 2])
Portugal
      9
```

```
> EURO.FKM.gk$clus ["Portugal", 2]
[1] 0.5681399
> Eurostat ["Portugal", ]
      Inflation Poverty Unemployment GDP
Portugal          1.2       21.6           7.1    77
```

An alternative implementation of the GK-FkM algorithm is provided by the function **GK** of the package **ppclust**.

### 5.3.2 **FKM.gkb** and **fuzzy.GK**

This section describes two implementations of the GKB-FkM clustering algorithm: the functions **FKM.gkb** and **fuzzy.GK** of the packages **fclust** and **advclust**, respectively. To better clarify the benefit of the GKB-FkM algorithm in comparison with the GK-FkM one, we use a synthetic dataset, called **synt.data2**, contained in the package **fclust**.

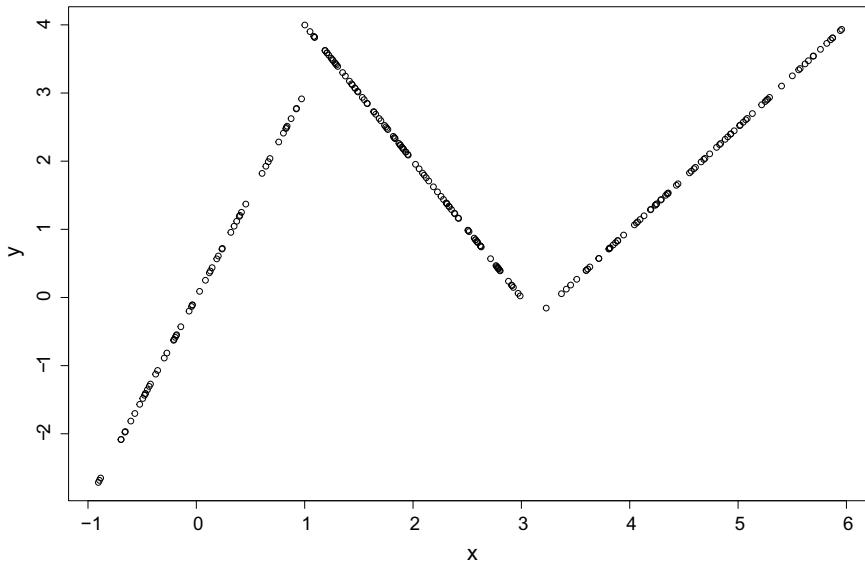
```
> library(fclust)
> data("synt.data2")
> plot(synt.data2)
```

It entails a data matrix with  $n = 240$  rows and  $p = 2$  columns. As we can see in Fig. 5.7, it consists of three non-spherical clusters. Although three clusters are clearly visible, **FKM.gk** fails due to the clear singularity of the cluster covariance matrices.

```
> synt.FKM.gk <- FKM.gk(synt.data2, k = 3, RS = 1,
+                         seed = 123)
Warning message:
In FKM.gk(synt.data2, k = 3, RS = 1, seed = 123) :
  When k=3, at least one cluster covariance matrix
  seems to be singular. Increase the number of
  starting points RS or use FKM.gkb
```

The output displays a warning message inviting to run the function **FKM.gkb** instead, to avoid singularity problems.

```
> synt.FKM.gkb <- FKM.gkb(synt.data2, k = 3, RS = 1,
+                           seed = 123)
```



**Fig. 5.7** `synt.data2` data: scatterplot

The partition obtained by means of the `FKM.gkb` algorithm is shown in Fig. 5.8 by using the function `plot.fclust`.

```
> plot.fclust(synt.FKM.gkb)
```

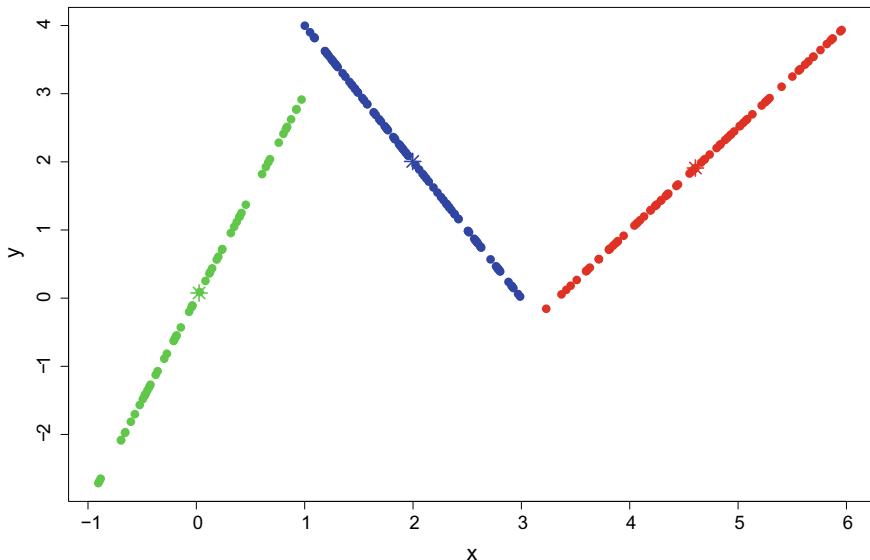
As we can notice, the clusters are properly recognized. See, for more details, [21].

The same results are obtained by using the function `fuzzy.GK` of the package **advclust**.

```
> library(advclust)
> synt.fuzzy.GK = fuzzy.GK(synt.data2, K = 3, m = 2,
+                           max.iteration = 1e+6,
+                           threshold = 1e-9,
+                           RandomNumber = 123)
Default Gamma (0) will be used
Default rho will be used

Membership initialized randomly

iteration:      16
Finish :)
```



**Fig. 5.8** `synt.data2`: scatterplot of the GKB-FkM solution (the different colors identify the clusters)

The membership degrees are randomly initialized, and the convergence is reached after 16 iterations. By comparing the two partitions, it results that there is a one-to-one correspondence between them.

```
> table(synt.FKM.gkb$clus[, 1],
+        synt.fuzzy.GK@hard.label)

      1     2     3
1  80    0    0
2    0   60    0
3    0    0  100
```

## 5.4 Entropic Fuzzy $k$ -Means

This section describes the entropic variant of the FkM (Entropic Fuzzy  $k$ -Means, EFkM) [37, 38]. This method arises to avoid the use of the fuzziness parameter  $m$  that has not a realistic meaning. In detail, the main difference with the FkM is the introduction of a regularization term in place of  $m$ . This term is the Shannon entropy, and it can be interpreted as a measure of entropy for fuzzy sets:

$$\sum_{i=1}^n \sum_{g=1}^k u_{ig} \log u_{ig}. \quad (5.16)$$

The optimization problem is now modified as

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{H}} J_{\text{EFKM}} &= \sum_{i=1}^n \sum_{g=1}^k u_{ig} d^2(\mathbf{x}_i, \mathbf{h}_g) + \tau \sum_{i=1}^n \sum_{g=1}^k u_{ig} \log u_{ig}, \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n, \end{aligned} \quad (5.17)$$

where  $\tau$  is a non-negative weighting parameter measuring the degree of fuzzy entropy. The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible membership degree matrix  $\mathbf{U}$ .
2. Given  $\mathbf{U}$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n u_{ig} \mathbf{x}_i}{\sum_{i=1}^n u_{ig}}, \quad g = 1, \dots, k. \quad (5.18)$$

3. Given  $\mathbf{H}$ , update the membership degree matrix  $\mathbf{U}$ :

$$u_{ig} = \frac{\exp\left(-\frac{d^2(\mathbf{x}_i, \mathbf{h}_g)}{\tau}\right)}{\sum_{g'=1}^k \exp\left(-\frac{d^2(\mathbf{x}_i, \mathbf{h}_{g'})}{\tau}\right)}, \quad i = 1, \dots, n, \quad g = 1, \dots, k. \quad (5.19)$$

4. Repeat steps 2 and 3 until convergence is reached.

As usual, the updates are obtained by means of the Lagrange multiplier method. In this case, the centroids in (5.18) are obtained as weighted means of the variables with weights equal to the membership degrees (natural choice) instead of the membership degrees at the power of  $m$ , as in (5.2).

There exists a connection between EF $k$ M and the model-based approach. In fact, the EM algorithm can be viewed as a particular coordinate descent algorithm minimizing a loss function equal to the negative sum of the entropies of the posterior probabilities plus a weighted sum of probabilistic distances between units and mixture components (see, for more details, [27]). Note that model-based clustering and the EM algorithm will be discussed in Part IV.

### 5.4.1 **FKM.ent**

The EF $k$ M clustering algorithm is implemented in the function **FKM.ent** of the package **felust**. The main characteristic of this function, compared with the others of

the package, is the introduction of the degree of fuzzy entropy ( $\tau$ ), `ent`. It replaces the fuzziness parameter  $m$ . As for the function `FKM`, we use the well-known dataset `wine` contained in the package **datasetsICR**. The function is run using standardized data (`stand = 1`) and 10 random starts (`RS = 10`) to limit the risk of local optima (the default value for  $\tau$ , `ent = 1`, is selected).

```
> library(fclust)
> library(datasetsICR)
> data("wine")
> Class <- wine[, 1]
> wine <- wine[, -1]
> wine.FKM.ent <- FKM.ent(wine, stand = 1, RS = 10,
+                               seed = 264)
The default value k=2:6 has been set
The default index SIL.F has been set
The default value alpha=1 has been set for computing
SIL.F
```

The optimal number of clusters, corresponding to the maximum value of  $FS$  (0.462517), is  $k = 3$ .

```
> wine.FKM.ent$k
Number of clusters
3
> wine.FKM.ent$criterion
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.4280005 0.4625170 0.4345109 0.4024889 0.3523851
```

The cluster sizes are 62, 51, and 65, respectively.

```
> cl.size(wine.FKM.ent$U)
Clus 1 Clus 2 Clus 3
62      51      65
```

The obtained partition is exactly the same obtained by the `FkM` clustering algorithm. Note that, in general, the partitions of `FkM` and `EFkM` differ, and the current result shows that, for this particular dataset, the partition remains stable.

```
> table(wine.FKM.ent$clus[, 1], wine.FKM.3$clus[, 1])
    1   2   3
1   0 62  0
2 51  0  0
3   0  0 65
```

Since the centroids obtained by means of the EF $k$ M algorithm are weighted means with weights equal to the membership degrees, the centroids are slightly different from those obtained by F $k$ M.

```
> Hraw(wine.FKM.ent$X, wine.FKM.ent$H)
      Alcohol Malic acid      Ash
Clus 1 13.67345    1.974026 2.464530
Clus 2 13.13606    3.311503 2.421631
Clus 3 12.25810    1.922536 2.230883
      Alkalinity of ash Magnesium Total phenols
Clus 1           17.47954 108.01412   2.852122
Clus 2           21.28342  98.74465   1.682447
Clus 3           20.01568  92.67898   2.243089
      Flavanoids Nonflavanoid phenols
Clus 1     3.0006449          0.2908490
Clus 2     0.8156277          0.4516395
Clus 3     2.0512739          0.3594136
      Proanthocyanins Color intensity      Hue
Clus 1     1.929994          5.446505 1.0681476
Clus 2     1.150407          7.234660 0.6915436
Clus 3     1.611642          3.000419 1.0590154
      OD280/OD315 of diluted wines   Proline
Clus 1           3.161490 1097.3804
Clus 2           1.698021 622.1416
Clus 3           2.800160 511.7270
> Hraw(wine.FKM.3$X, wine.FKM.3$H)
      Alcohol Malic acid      Ash Alkalinity of ash
Clus 1 13.04447    3.198086 2.403085   21.01413
Clus 2 13.58018    1.950055 2.423150   17.55505
Clus 3 12.45312    1.977425 2.268191   20.00721
      Magnesium Total phenols Flavanoids
Clus 1 98.32535    1.761977  1.014878
Clus 2 105.62368    2.790073  2.866781
Clus 3 93.73464    2.253092  2.058545
      Nonflavanoid phenols Proanthocyanins
Clus 1           0.4441319  1.209568
Clus 2           0.2943256  1.903188
Clus 3           0.3574444  1.584297
      Color intensity      Hue
Clus 1     6.577139 0.7523029
Clus 2     5.456684 1.0525165
Clus 3     3.506681 1.0299486
      OD280/OD315 of diluted wines   Proline
Clus 1           1.890480 632.1152
Clus 2           3.077742 1044.0895
Clus 3           2.794070 568.7919
```

In particular, we can assess the recovery performance of EFkM and FkM by using the Euclidean distance to compare the standardized estimated centroids with the standardized true ones ( $H.t$ ), obtained as sample means of the three groups of cultivars. We can notice that those got by the EFkM algorithm are closer to the true centroids. Note that the function `aggregate` needs as input argument a data frame, while the function `scale` returns a matrix. Note also that the rows of `wine.FKM.3$H` and `wine.FKM.ent$H` are reordered to fix the cluster label switching.

```
> wine.zd <- as.data.frame(scale(wine))
> H.t <- aggregate(. ~ Class, data = wine.zd,
+                     FUN = mean) [, -1]
> sqrt(apply(abs(H.t - wine.FKM.3$H[c(2, 3, 1), ]),
+             1, sum))
[1] 1.109220 1.045644 1.420407
> sqrt(apply(abs(H.t - wine.FKM.ent$H[c(1, 3, 2), ]),
+             1, sum))
[1] 0.7913076 0.7063597 0.6402451
```

Before going on, we may note that the entropic FkM with the Gustafson-Kessel approach is also implemented in the package **fclust** via the function `FKM.gk.ent`.

## 5.5 Fuzzy $k$ -Means with Polynomial Fuzzifier

A generalization of the FkM algorithm has been proposed in [31, 53] by considering an alternative fuzzifier function, to overcome a drawback of FkM-type algorithms. The fuzzifier  $m$ , used to control the overlapping between clusters by giving low membership degrees to units with unclear assignments, usually leads to assign the units to all clusters with non-zero membership degrees. This prevents a hard classification even when the units are very close to a single prototype. In general, a fuzzifier function is a continuous, strictly increasing function  $f : [0, 1] \rightarrow [0, 1]$  with  $f(0) = 0$  and  $f(1) = 1$ . In the FkM case,  $f(u_{ig}) = u_{ig}^m$ . The suggested polynomial fuzzifier function is instead defined as

$$f(u_{ig}) = \left( \frac{1-\beta}{1+\beta} u_{ig}^2 + \frac{2\beta}{1+\beta} u_{ig} \right), \quad (5.20)$$

where  $\beta \in [0, 1]$ .

The general optimization problem of the Fuzzy  $k$ -Means with Polynomial Fuzzifier (FkMPF) becomes

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{H}} J_{\text{FkMPF}} &= \sum_{i=1}^n \sum_{g=1}^k f(u_{ig}) d^2(\mathbf{x}_i, \mathbf{h}_g), \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1 \quad i = 1, \dots, n. \end{aligned} \quad (5.21)$$

A common choice is  $\beta = 0.5$ ; for  $\beta = 0$ , we obtain the FkM with  $m = 2$  and for  $\beta = 1$  the standard  $k$ -Means.

Unlike the FkM algorithm, here the constraint of non-negative membership degrees is not automatically satisfied. Thus, in the iterative process, based on the Lagrange multiplier method, we have to take into account explicitly the constraint  $u_{ig} > 0$ , and only the prototypes fulfilling those conditions are included in the updates of the membership degrees. The membership degrees for all other prototypes are set to zero. In detail, for unit  $i$ , the distances between the unit and the prototypes are sorted and only those prototypes for which the membership degrees are larger than 0 are kept in the membership degree updates (selected prototypes for  $i$ ). The number of clusters for the units corresponding to the selected subset of prototypes is denoted by  $\hat{k}$  and  $g_{sel}$  refers to a selected prototype for  $i$ . The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible membership degree matrix  $\mathbf{U}$ .
2. Given  $\mathbf{U}$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n f(u_{ig}) \mathbf{x}_i}{\sum_{i=1}^n f(u_{ig})}, \quad g = 1, \dots, k. \quad (5.22)$$

3. Given  $\mathbf{H}$ , update the membership degree matrix  $\mathbf{U}$ :

$$u_{ig} = \begin{cases} \frac{1}{1-\beta} \frac{1 + \beta(\hat{k} - 1)}{\sum_{g'=1}^{\hat{k}} \frac{d^2(\mathbf{x}_i, \mathbf{h}_{g'})}{d^2(\mathbf{x}_i, \mathbf{h}_g)}} - \frac{\beta}{1-\beta}, & g = g_{sel}, \\ 0, & \text{otherwise,} \end{cases} \quad (5.23)$$

for  $i = 1, \dots, n$  and  $g = 1, \dots, k$ .

4. Repeat steps 2 and 3 until convergence is reached.

### 5.5.1 ***FKM.pf***

This section details the implementation of the FkMPF clustering algorithm by the function **FKM.pf** of the package **fclust**. The input arguments are the same as for the other functions contained in the package except for **b**, the parameter of the polynomial fuzzifier, that is set to 0.5 by default. We run the function for the dataset **wine** of the package **datasetsICR**.

```
> library(fclust)
> library(datasetsICR)
> data("wine")
> Class <- wine[, 1]
> wine <- wine[, -1]
> wine.FKM.pf <- FKM.pf(wine, stand = 1, RS = 10,
+                           seed = 264)
The default value k=2:6 has been set
The default index SIL.F has been set
The default value alpha=1 has been set for computing
SIL.F
```

The optimal number of clusters according to the default criterion (the FS index) is  $k = 3$ .

```
> wine.FKM.pf$k
Number of clusters
3
> wine.FKM.pf$criterion
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.4699136 0.5025005 0.4653834 0.3963534 0.3868160
```

The same optimal number of clusters is also obtained by specifying **index = "XB"**.

```
> wine.FKM.pf <- FKM.pf(wine, stand = 1, RS = 10,
+                           index = "XB", seed = 264)
The default value k=2:6 has been set
> wine.FKM.pf$k
Number of clusters
3
> wine.FKM.pf$criterion
XB k=2     XB k=3     XB k=4     XB k=5     XB k=6
0.5400992 0.4638960 1.1367147 0.9500389 0.9090030
```

Now we better inspect the partition with  $k = 3$ .

```
> wine.FKM.pf.3 <- FKM.pf(wine, k = 3, stand = 1,
+                               RS = 10, seed = 264)
```

As we can easily notice, it is the same resulting by applying the FkM clustering algorithm.

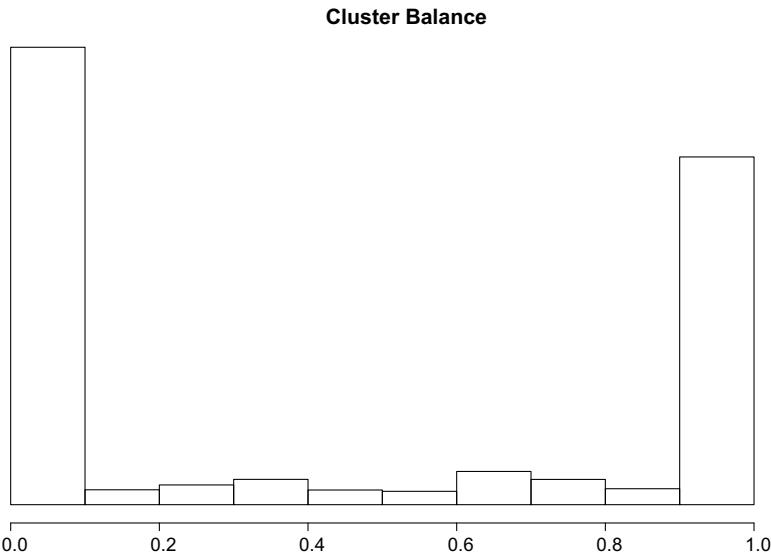
```
> cl.size(wine.FKM.pf.3$U)
Clus 1 Clus 2 Clus 3
      51       62       65
```

It can also be shown by comparing the partition with the classification given by the variable Class.

```
> table(Class, wine.FKM.pf.3$clus[, 1])
Class   1   2   3
  1    0  59   0
  2    3   3  65
  3   48   0   0
```

Also in this case just 6 out 178 wines are misclassified. However, some differences with respect to the FkM solution are visible. First of all, the prototypes in the original units of measurement are slightly different.

```
> Hraw(wine.FKM.pf.3$X, wine.FKM.pf.3$H)
          Alcohol Malic acid           Ash Alkalinity of ash
Clus 1 13.10151     3.312307  2.424787             21.30820
Clus 2 13.66572     1.954635  2.442591             17.30557
Clus 3 12.25996     1.897970  2.239419             20.15494
          Magnesium Total phenols Flavanoids
Clus 1    98.7934      1.686941  0.8409379
Clus 2   107.3730      2.851736  2.9743374
Clus 3    92.2656      2.237041  2.0686859
          Nonflavanoid phenols Proanthocyanins
Clus 1              0.4522984      1.149179
Clus 2              0.2876687      1.946575
Clus 3              0.3610080      1.593996
          Color intensity        Hue
Clus 1            7.070349  0.7032355
Clus 2            5.453559  1.0660201
Clus 3            3.009443  1.0621313
          OD280/OD315 of diluted wines   Proline
Clus 1                      1.725794  621.5158
Clus 2                      3.144992 1089.3366
Clus 3                      2.825121  508.9150
```



**Fig. 5.9** wine data: chart diagram of the scaled frequency related to the membership degrees of the FkMPF solution with  $k = 3$  clusters

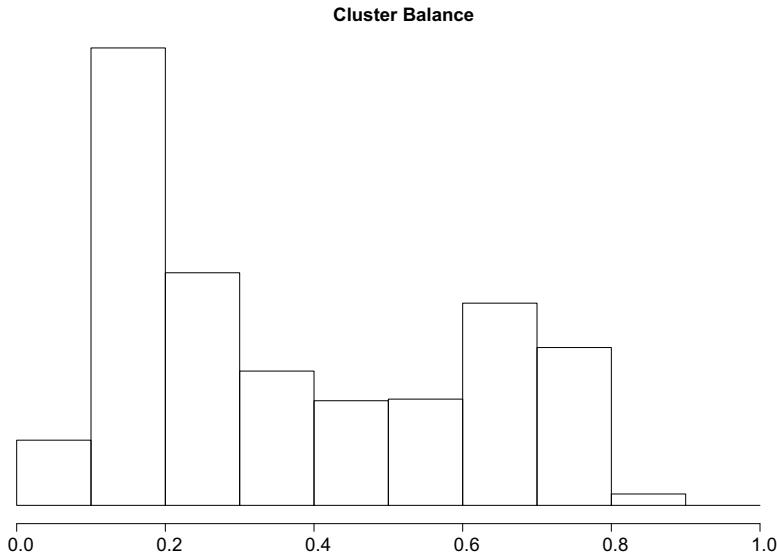
The main difference with the FkM algorithm consists in the membership degrees. If we take a look at the output argument `clus` (omitted for the sake of brevity), the units that are very close to a prototype are assigned to that specific cluster with a membership degree equal to 1. This can be easily seen graphically. To obtain a visual inspection of fuzzy clustering results [30], the function `VIFCR`, included in the package `fclust`, can be used. Three plots (selected by `which`) are available. Plot 1 (`which = 1`) is a chart diagram of the scaled frequency of the membership degrees

$$\frac{1}{n} \sum_{(i,g): a \leq u_{ig} < b} \left( \frac{k(k-2)}{k-1} u_{ig} + \frac{k}{k-1} \right),$$

with  $a, b \in [0, 1]$  and  $a < b$ . The values are scaled in such a way that the weights for membership degrees equal to 0 and 1 are  $k/(k-1)$  and  $k$ , respectively. The weights of the membership degrees in  $[0, 1]$  are linearly increasing from  $k/(k-1)$  to  $k$ . Thus, in case of crisp assignments, the chart diagram would show a value of 1 on both the left and the right sides. The plot is displayed in Fig. 5.9.

```
> VIFCR(wine.FKM.pf.3, which = 1)
```

The highest bars correspond to extreme values 0 and 1, and just a few units have unclear assignments. If we inspect the same graphical representation for the FkM



**Fig. 5.10** `wine` data: chart diagram of the scaled frequency related to the membership degrees of the FKM solution with  $k = 3$  clusters

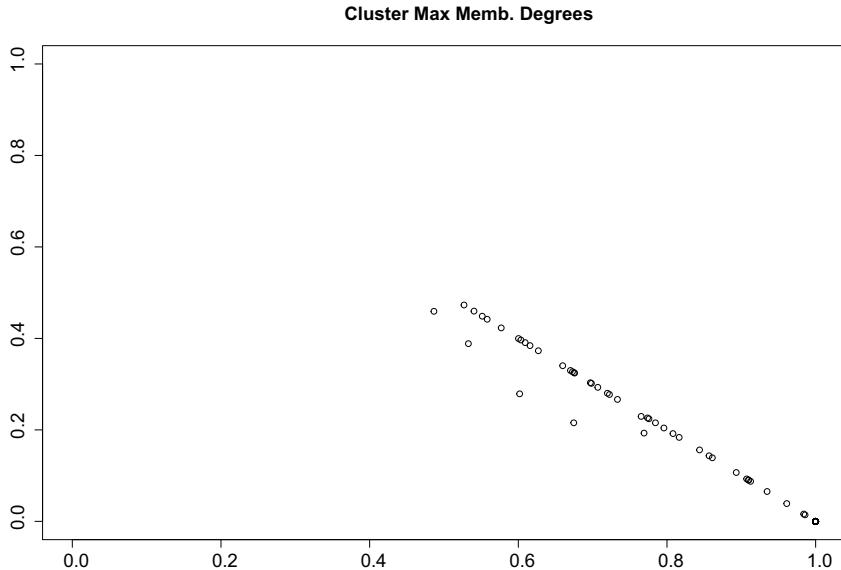
partition, reported in Fig. 5.10, we notice that the fuzziness is more evident, in fact, the highest bars do not correspond to the extreme values and the scaled frequencies of membership degrees between 0.4 and 0.6 are higher. In detail, units that are very close to a single prototype are not assigned with a membership equal to 1 (hard classification).

```
> VIFCR(wine.FKM.3, which = 1)
```

Plot 2 (`which = 2`) is a scatterplot with coordinates, for each unit (point)  $i$ , given by the highest ( $u_{ig_1}$ ) and the second highest membership degree ( $u_{ig_2}$ ). All the points are included in the triangle of vertices  $(0, 0)$  (noisy data),  $(0.5, 0.5)$  (ambiguous assignments), and  $(1, 0)$  (crisp assignments). The above scatterplot is displayed in Fig. 5.11.

```
> VIFCR(wine.FKM.pf.3, which = 2)
```

Most of the points (units) in Fig. 5.11 are close to vertex  $(1, 0)$ , and hence are characterized by a crisp/hard assignment.



**Fig. 5.11** wine data: scatterplot of the highest and the second highest membership degrees of the FkMPP solution with  $k = 3$  clusters

Plot 3 (which = 3), for each cluster, is a scatterplot such that the coordinates of each unit (point)  $i$  are  $(d_{ig}, u_{ig})$ . The ideal situation is to obtain high membership degrees for small distances and low membership degrees for large distances. If so the points are located in the upper left area or in the lower right area.

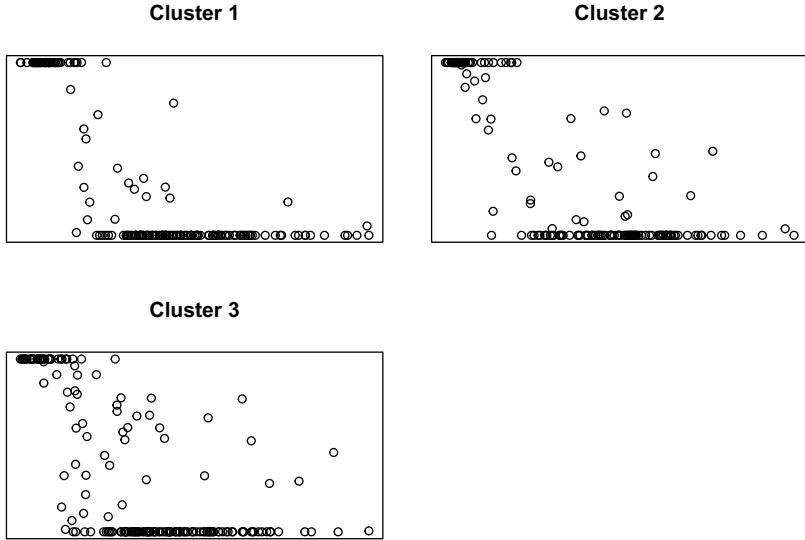
```
> VIFCR(wine.FKM.pf.3, which = 3)
```

The three scatterplots are reported in Fig. 5.12.

## 5.6 Fuzzy $k$ -Medoids

This section details the generalization of the standard  $k$ -Medoids algorithm, introduced in Chap. 3. Unlike the fuzzy clustering methods we have previously described, clusters are not characterized by centroids, artificial units, but in terms of medoids, a subset of the observed units. The partitioning around medoids algorithm [29] has been extended to the fuzzy approach in [33].

The Fuzzy  $k$ -Medoids (FkMed) optimization problem can be formalized as



**Fig. 5.12** wine data: scatterplots of the coordinates  $(d_{ig}, u_{ig})$  of the FkMPF solution with  $k = 3$  clusters

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{H}} J_{\text{FkMed}} &= \sum_{i=1}^n \sum_{g=1}^k u_{ig}^m d^2(\mathbf{x}_i, \mathbf{h}_g), \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n, \\ \{\mathbf{h}_1, \dots, \mathbf{h}_g, \dots, \mathbf{h}_k\} &\subseteq \{\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_n\}, \end{aligned} \quad (5.24)$$

where  $\mathbf{h}_g$  is now the medoid of the  $g$ -th cluster.

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose  $k$  initial medoids, i.e., the medoid matrix  $\mathbf{H}$ .
2. Given  $\mathbf{H}$ , update the membership degree matrix  $\mathbf{U}$ :

$$u_{ig} = \frac{1}{\sum_{g'=1}^k \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_{g'})}{d^2(\mathbf{x}_i, \mathbf{h}_g)} \right)^{\frac{1}{m-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k. \quad (5.25)$$

3. Given  $\mathbf{U}$ , update the medoid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \operatorname{argmin}_{i=1, \dots, n} \sum_{i'=1}^n u_{ig}^m d^2(\mathbf{x}_i, \mathbf{x}_{i'}), \quad g = 1, \dots, k. \quad (5.26)$$

4. Repeat steps 2 and 3 until convergence is reached.

Since the medoid has always a membership degree equal to 1 to the cluster, raising its membership to the power of  $m$  has no effect. Nonetheless, it affects the assignment of non-medoid units. When  $m$  is high, the mobility of the medoids from iteration to iteration may be lost. Thus, a value between 1 and 1.5 is usually recommended for  $m$ .

As its crisp counterpart, the FkMed algorithm is computationally more expensive than the FkM-type algorithms. This is due to the medoids update that needs the computation of distances between each pair of units.

### 5.6.1 **FKM.med**

This section describes the implementation of the FkMed clustering algorithm available in the function `FKM.med` of the package **fclust**. The input arguments are the same as for the function `FKM`. The default values are all the same but the parameter of fuzziness, which is usually lower than the one used in `FKM` (default: 1.5).

We use the dataset `USArrests` of the package **datasets** [44]. It contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973, and the percent of the population living in urban areas.

```
> library(datasets)
> data("USArrests")
> str(USArrests)
'data.frame': 50 obs. of 4 variables:
 $ Murder : num 13.2 10 8.1 8.8 9 7.9 3.3 ...
 $ Assault : int 236 263 294 190 276 204 110 ...
 $ UrbanPop: int 58 48 80 50 91 78 77 ...
 $ Rape   : num 21.2 44.5 31 19.5 40.6 38.7 11.1 ...
> row.names(USArrests)
[1] "Alabama"          "Alaska"           "Arizona"          "
[4] "Arkansas"         "California"       "Colorado"         "
[7] "Connecticut"       "Delaware"         "Florida"          "
[10] "Georgia"          "Hawaii"           "Idaho"           "
[13] "Illinois"         "Indiana"          "Iowa"            "
[16] "Kansas"           "Kentucky"         "Louisiana"        "
[19] "Maine"             "Maryland"          "Massachusetts"   "
[22] "Michigan"          "Minnesota"        "Mississippi"     "
[25] "Missouri"          "Montana"          "Nebraska"         "
[28] "Nevada"            "New Hampshire"    "New Jersey"      "
[31] "New Mexico"        "New York"          "North Carolina"  "
[34] "North Dakota"      "Ohio"              "Oklahoma"        "
[37] "Oregon"            "Pennsylvania"     "Rhode Island"    "
[40] "South Carolina"    "South Dakota"     "Tennessee"       "
[43] "Texas"             "Utah"              "Vermont"         "
[46] "Virginia"          "Washington"       "West Virginia"   "
[49] "Wisconsin"         "Wyoming"          "
```

We run the function `FKM.med` by selecting the standardization option and 10 random starts, while the default values of  $k$ , `index`, and `alpha` have been set.

```
> library(fclust)
> USA.FKM.med <- FKM.med(USArrests, stand = 1,
+                               RS = 10, seed = 264)
The default value k=2:6 has been set
The default index SIL.F has been set
The default value alpha=1 has been set for computing
SIL.F
```

The optimal number of clusters, corresponding to the maximum value of  $FS$  ( $0.5183611$ ), is  $k = 3$ .

```
> USA.FKM.med$k
Number of clusters
[1] 3
> USA.FKM.med$criterion
SIL.F k=2  SIL.F k=3  SIL.F k=4  SIL.F k=5  SIL.F k=6
0.4058561 0.5183611 0.4318490 0.3929743 0.4960089
```

Unlike the other functions contained in the package, there is a further output argument, `medoid`, that is, a vector containing the indices of the medoid units.

```
> USA.FKM.med$medoid
[1] 36 31 26
```

The above indices, in the present context, correspond to Oklahoma, New Mexico, and Montana.

```
> row.names(USArrests)[USA.FKM.med$med]
[1] "Oklahoma"    "New Mexico"   "Montana"
```

The medoid features are easily obtained.

```
> USArrests[USA.FKM.med$medoid, ]
      Murder Assault UrbanPop Rape
Oklahoma     6.6     151       68 20.0
New Mexico   11.4     285       70 32.1
Montana      6.0     109       53 16.4
```

Note that they are the same values contained in the output argument `H` transformed by using the original units of measurement.

```
> Hraw(USA.FKM.med$X, USA.FKM.med$H)
      Murder Assault UrbanPop Rape
Clus 1     6.6      151       68 20.0
Clus 2    11.4      285       70 32.1
Clus 3     6.0      109       53 16.4
```

Further details on the partition of any `fclust` function can be observed by inspecting the summary output by the command `summary.fclust`. It displays the number of units, the number of clusters, the cluster sizes, the closest hard clustering partition, the cluster memberships (using the closest hard clustering partition), the number of units with unclear assignment, the units with unclear assignment and the cluster sizes without unclear assignments (only if units with unclear assignment are present), the cluster summary (for every cluster: size, minimal membership degree, maximum membership degree, average membership degree, number of units with unclear assignment), and the Euclidean distance matrix for the cluster medoids.

```
> summary.fclust(USA.FKM.med)

Fuzzy clustering object of class 'fclust'

Number of objects:
50

Number of clusters:
3

Cluster sizes:
Clus 1 Clus 2 Clus 3
    17      19      14

Clustering index values:
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.4058561 0.5183611 0.4318490 0.3929743 0.4960089

Closest hard clustering partition:
          Alabama        Alaska        Arizona
          2                  2                  2
          Arkansas       California       Colorado
          3                  2                  2
Connecticut       Delaware        Florida
          1                  1                  2
          Georgia         Hawaii         Idaho
          2                  1                  3
```

	Illinois	Indiana	Iowa
	2	1	3
Kansas		Kentucky	Louisiana
	1	3	2
Maine		Maryland	Massachusetts
	3	2	1
Michigan		Minnesota	Mississippi
	2	3	2
Missouri		Montana	Nebraska
	1	3	3
Nevada	New Hampshire		New Jersey
	2	3	1
New Mexico		New York	North Carolina
	2	2	2
North Dakota		Ohio	Oklahoma
	3	1	1
Oregon	Pennsylvania		Rhode Island
	1	1	1
South Carolina	South Dakota		Tennessee
	2	3	2
Texas		Utah	Vermont
	2	1	3
Virginia	Washington		West Virginia
	1	1	3
Wisconsin		Wyoming	
	3	1	

Cluster memberships :

Clus 1

```
[1] "Connecticut"    "Delaware"      "Hawaii"
[4] "Indiana"        "Kansas"        "Massachusetts"
[7] "Missouri"       "New Jersey"    "Ohio"
[10] "Oklahoma"      "Oregon"       "Pennsylvania"
[13] "Rhode Island"   "Utah"         "Virginia"
[16] "Washington"    "Wyoming"
```

Clus 2

```
[1] "Alabama"       "Alaska"        "Arizona"
[4] "California"    "Colorado"      "Florida"
[7] "Georgia"        "Illinois"      "Louisiana"
[10] "Maryland"       "Michigan"     "Mississippi"
[13] "Nevada"         "New Mexico"   "New York"
[16] "North Carolina" "South Carolina" "Tennessee"
[19] "Texas"
```

Clus 3

```
[1] "Arkansas"      "Idaho"        "Iowa"
[4] "Kentucky"       "Maine"        "Minnesota"
[7] "Montana"        "Nebraska"    "New Hampshire"
[10] "North Dakota"   "South Dakota" "Vermont"
[13] "West Virginia"  "Wisconsin"
```

Number of objects with unclear assignment (maximal membership degree <0.5) :

1

```

Objects with unclear assignment:
"Mississippi"

Cluster sizes (without unclear assignments):
Clus 1   Clus 2   Clus 3   No clus
      17       18       14        1

Membership degree matrix (rounded):
          Clus 1   Clus 2   Clus 3
Alabama      0.27     0.60     0.13
Alaska       0.12     0.80     0.08
Arizona      0.04     0.95     0.01
Arkansas     0.34     0.05     0.61
California   0.09     0.88     0.03
Colorado     0.18     0.78     0.04
Connecticut   0.72     0.02     0.27
Delaware     0.84     0.07     0.10
Florida      0.02     0.97     0.01
Georgia      0.17     0.72     0.10
Hawaii       0.74     0.04     0.22
Idaho        0.08     0.00     0.92
Illinois     0.21     0.76     0.03
Indiana      0.93     0.00     0.07
Iowa         0.14     0.01     0.85
Kansas        0.91     0.00     0.09
Kentucky      0.10     0.01     0.89
Louisiana    0.13     0.82     0.05
Maine         0.12     0.01     0.88
Maryland      0.00     1.00     0.00
Massachusetts 0.87     0.03     0.10
Michigan      0.00     1.00     0.00
Minnesota    0.38     0.01     0.61
Mississippi  0.26     0.46     0.28
Missouri      0.71     0.23     0.06
Montana       0.00     0.00     1.00
Nebraska      0.25     0.00     0.74
Nevada        0.06     0.92     0.02
New Hampshire 0.15     0.01     0.85
New Jersey    0.83     0.08     0.09
New Mexico    0.00     1.00     0.00
New York      0.13     0.85     0.02
North Carolina 0.24     0.55     0.21
North Dakota  0.15     0.02     0.84
Ohio          0.97     0.00     0.02
Oklahoma      1.00     0.00     0.00
Oregon        0.84     0.05     0.11
Pennsylvania  0.87     0.00     0.13
Rhode Island   0.72     0.07     0.21
South Carolina 0.18     0.68     0.14
South Dakota   0.03     0.00     0.96
Tennessee     0.29     0.58     0.13
Texas         0.26     0.69     0.05

```

```

Utah           0.87   0.03   0.11
Vermont        0.13   0.02   0.85
Virginia       0.94   0.00   0.05
Washington     0.91   0.02   0.07
West Virginia  0.07   0.01   0.93
Wisconsin      0.32   0.01   0.67
Wyoming         0.60   0.00   0.40

Cluster summary:
  Cl.size Min.memb.deg. Max.memb.deg. Av.memb.deg.
Clus 1      17          0.60            1          0.84
Clus 2      19          0.46            1          0.79
Clus 3      14          0.61            1          0.83
  N.uncl.assignm.
Clus 1          0
Clus 2          1
Clus 3          0

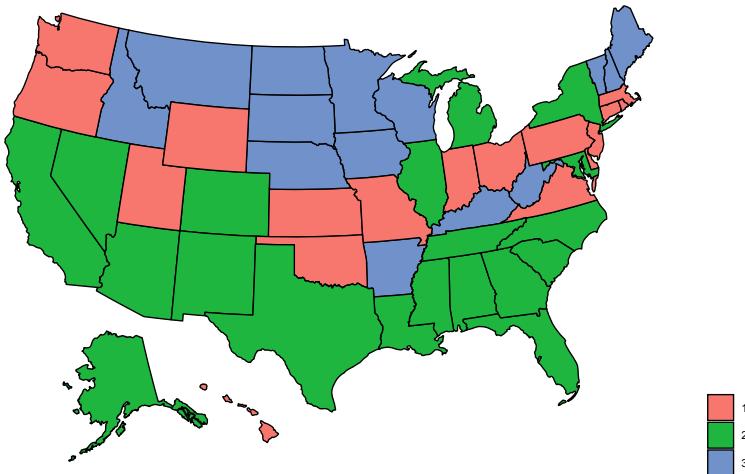
Euclidean distance matrix for the prototypes (rounded):
    Clus 1 Clus 2
Clus 2  2.34
Clus 3  1.22    3.19

Available components:
[1] "U"          "H"          "F"          "clus"
[5] "medoid"     "value"      "criterion"   "iter"
[9] "k"          "m"          "ent"         "b"
[13] "vp"         "delta"      "stand"       "Xca"
[17] "X"          "D"          "call"

```

We discover that Cluster 1 is composed of 17 US states, Connecticut, Delaware, Hawaii, Indiana, Kansas, Massachusetts, Missouri, New Jersey, Ohio, Oklahoma, Oregon, Pennsylvania, Rhode Island, Utah, Virginia, Washington, and Wyoming. The 19 states in Cluster 2 are Alabama, Alaska, Arizona, California, Colorado, Florida, Georgia, Illinois, Louisiana, Maryland, Michigan, Mississippi, Nevada, New Mexico, New York, North Carolina, South Carolina, Tennessee, and Texas. Cluster 3 contains Arkansas, Idaho, Iowa, Kentucky, Maine, Minnesota, Montana, Nebraska, New Hampshire, North Dakota, South Dakota, Vermont, West Virginia, and Wisconsin. The membership degrees to Cluster 1 range from 0.60 (Wyoming) to 1 (the medoid), and the average value is 0.84. For those related to Cluster 2, the minimum value is 0.46 (Mississippi), unclear assignment, and the average one is 0.79. Finally, they range from 0.61 (Arkansas and Minnesota) to 1, with average of 0.83, for Cluster 3.

To represent the obtained clusters in a map, we use the function `plot_usmap` of the package `usmap` [18]. Note that `ggplot2` [52] must be installed to use



**Fig. 5.13** USArrests data: map of the US states by cluster (FkMed solution with  $k = 3$  clusters)

`plot_usmap`. The dataset `USstate` of the package **datasetsICR** refers to FIPS codes for the US States and the district of Columbia. The variables are `fips` (FIPS State Numeric Code), `usps` (Official USPS Code), and `name` (Name). Note that, in `plot_usmap`, we use the option `exclude = "11"` to specify that the District of Columbia (FIPS code 11) is not displayed.

```
> library(datasetsICR)
> data("USstate")
> names(USstate)
[1] "fips" "usps" "name"
> USstate$clus <- as.factor(USA.FKM.med$clus[, 1])
> library(usmap)
> library(ggplot2)
> plot_usmap(data = USstate, values = "clus",
+             exclude = "11") +
+   theme(legend.position = "right") +
+   theme(legend.title = element_blank())
```

Figure 5.13 reports the clusters in different colors. It can be noticed that the clusters can be interpreted geographically. In particular, Cluster 2 is mainly composed of states from the South, while states in Cluster 3 are mainly in the Northern area.

To produce a graphical representation of the distributions of the four variables in the clusters, we can use boxplots. The medoids are denoted by the blue points.

```

> par(mfrow = c(1, 2))
> boxplot(USArrests$Murder ~ USA.FKM.med$clus[, 1],
+           names = c("Cluster 1", "Cluster 2",
+                     "Cluster 3"),
+           sub = "Murder")
> points(USArrests[USA.FKM.med$medoid, 1], pch = 16,
+           col = "blue")
> boxplot(USArrests$Assault ~ USA.FKM.med$clus[, 1],
+           names = c("Cluster 1", "Cluster 2",
+                     "Cluster 3"),
+           sub = "Assault")
> points(USArrests[USA.FKM.med$medoid, 2], pch = 16,
+           col = "blue")
> par(mfrow = c(1, 2))
> boxplot(USArrests$UrbanPop ~ USA.FKM.med$clus[, 1],
+           names = c("Cluster 1", "Cluster 2",
+                     "Cluster 3"),
+           sub = "UrbanPop")
> points(USArrests[USA.FKM.med$medoid, 3], pch = 16,
+           col = "blue")
> boxplot(USArrests$Rape ~ USA.FKM.med$clus[, 1],
+           names = c("Cluster 1", "Cluster 2",
+                     "Cluster 3"),
+           sub = "Rape")
> points(USArrests[USA.FKM.med$medoid, 4], pch = 16,
+           col = "blue")

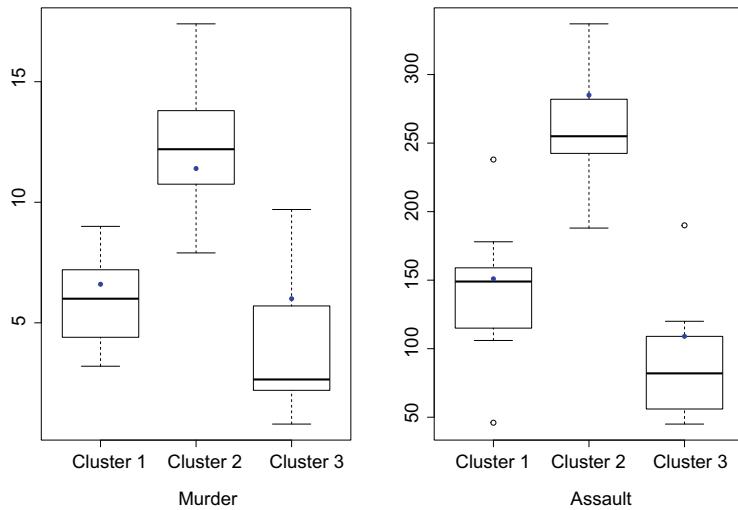
```

As we can notice in Fig. 5.14, Cluster 2 is mainly characterized by the highest values of assaults, murders, and rapes, while, on the contrary, low values of the three variables are reported for states assigned to Cluster 3. In addition, the percent of the population living in urban areas is more or less the same for the median values of Cluster 1 and Cluster 2, but the distribution is more dispersed for Cluster 2 (Fig. 5.15).

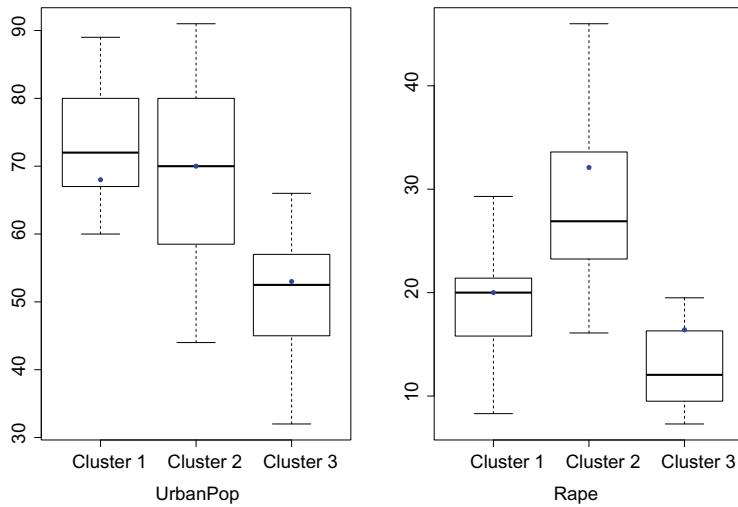
Alternative functions implementing the FkMed clustering algorithm are contained in the package **ClusterR**: `Cluster_Medoids` and `Clara_Medoids`, already mentioned in Sect. 3.5.3. An interesting peculiarity of `Cluster_Medoids` and `Clara_Medoids` is the input option `fuzzy` that can be either TRUE or FALSE. If `fuzzy = TRUE`, then the membership degrees for each cluster will be returned based on the distance between units and medoids (`fuzzy_probs`).

## 5.7 Fuzzy Clustering for Relational Data

In several practical applications, the information is available in terms of relational data. We recall that relational data consist in pair-wise relations (similarity or dissimilarity/distance) between units, stored in a matrix  $\mathbf{D}$  with generic element  $d(\mathbf{x}_i, \mathbf{x}_j)$ , not necessarily based on a set of observed variables. There exist different proposals of fuzzy clustering algorithms for such kind of data. A first proposal has been introduced in [29]: the FANNY algorithm. The optimization problem is formalized as



**Fig. 5.14** USArrests data: boxplots of Murder and Assault, by cluster (FkMed solution with  $k = 3$  clusters). The blue points denote the values corresponding to the medoids



**Fig. 5.15** USArrests data: boxplots of UrbanPop and Rape, by cluster (FkMed solution with  $k = 3$  clusters). The blue points denote the values corresponding to the medoids

$$\begin{aligned} \min_{\mathbf{U}} J_{\text{FANNY}} &= \sum_{g=1}^k \frac{\sum_{i=1}^n \sum_{i'=1}^n u_{ig}^2 u_{i'g}^2 d(\mathbf{x}_i, \mathbf{x}_{i'})}{2 \sum_{i=1}^n u_{ig}^2}, \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n. \end{aligned} \quad (5.27)$$

A limit of FANNY is that it might fail when the matrix  $\mathbf{D}$  does not contain Euclidean distances. In fact, the non-negativity of the membership degrees is not guaranteed unless Euclidean distances are used. For this reason, Davé and Sen [16] propose a relational fuzzy clustering algorithm for any kind of distance matrix  $\mathbf{D}$  with generic element  $d_{NE}(\mathbf{x}_i, \mathbf{x}_l)$ . The Non-Euclidean Fuzzy Relational data Clustering algorithm (NEFRC) consists in solving the following minimization problem:

$$\begin{aligned} \min_{\mathbf{U}} J_{\text{NEFRC}} &= \sum_{g=1}^k \frac{\sum_{i=1}^n \sum_{i'=1}^n u_{ig}^m u_{i'g}^m d_{NE}(\mathbf{x}_i, \mathbf{x}_{i'})}{2 \sum_{i=1}^n u_{ig}^m}, \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n, \end{aligned} \quad (5.28)$$

where  $d_{NE}(\cdot, \cdot)$  is a generic distance measure, not necessarily the Euclidean one. The NEFRC algorithm also differs from FANNY since a general fuzzifier  $m$  is used.

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible membership degree matrix  $\mathbf{U}$ .
2. Given  $\mathbf{U}$ , compute  $b_{ig}$  ( $i = 1, \dots, n, g = 1, \dots, k$ ) according to

$$b_{ig} = a_{ig} u_{ig}^{m-2}, \quad (5.29)$$

where

$$a_{ig} = \frac{m \sum_{i'=1}^n u_{i'g}^m d_{NE}(\mathbf{x}_i, \mathbf{x}_{i'})}{\sum_{i'=1}^n u_{i'g}^m} - \frac{m \sum_{i'=1}^n \sum_{i''=1}^n u_{i'g}^m u_{i''g}^m d_{NE}(\mathbf{x}_{i'}, \mathbf{x}_{i''})}{2 \left( \sum_{i'=1}^n u_{i'g}^m \right)^2}, \quad (5.30)$$

considering  $u_{i'g}^{(r+1)}$  if  $i' < i$  or  $u_{i'g}^{(r)}$  if  $i' \geq i$ .

3. Update the membership degree matrix  $\mathbf{U}$ ,

$$u_{ig} = \begin{cases} \frac{\frac{1}{b_{ig}}}{\sum_{g' \in I_i^+} \left( \frac{1}{b_{i'g'}} \right)}, & \text{if } g \in I_i^+, \\ 0, & \text{if } g \in I_i^-, \end{cases} \quad (5.31)$$

with

$$I_i^- = \left\{ g : \frac{\frac{1}{b_{ig}}}{\sum_{g'=1}^k \left( \frac{1}{b_{ig'}} \right)} \leq 0 \right\}, \quad i = 1, \dots, n, \quad (5.32)$$

$$I_i^+ = \left\{ g : \frac{\frac{1}{b_{ig}}}{\sum_{g'=1}^k \left( \frac{1}{b_{ig'}} \right)} > 0 \right\}, \quad i = 1, \dots, n, \quad (5.33)$$

for  $i = 1, \dots, n$  and  $g = 1, \dots, k$ .

4. Repeat steps 2 and 3 until convergence is reached.

### 5.7.1 **fanny**

This section describes the implementation of the FANNY clustering algorithm by the function **fanny** of the package **cluster**. We consider the Facial Expression dataset (**FaceExp**) included in the package **smacof** [17]. It consists of a distance matrix of  $n = 13$  facial expressions [1]. The ratings are based on a 9-point Likert scale. The dataset also contains external scales taken from [19]. Such external scales reflect the following three perceptual dimensions: pleasant-unpleasant (PU), attention-rejection (AR), and tension-sleep (TS).

```
> library(smacof)
> data("FaceExp")
> face <- FaceExp
> str(face)
'dist' num [1:78] 4.05 8.25 5.57 1.15 2.97 4.34 ...
- attr(*, "Size")= int 13
- attr(*, "call")= language as.dist.default(m = delta)
- attr(*, "Diag")= logi FALSE
- attr(*, "Upper")= logi FALSE
- attr(*, "Labels")= chr [1:13] "Grief at death of
  mother" "Savoring a Coke" "Very pleasant surprise"
  "Maternal love-baby in arms" ...
```

The main input argument of the function **fanny** is **x**, the data matrix or data frame, or the distance matrix, depending on the value of the **diss** argument, a logical flag such that **diss = TRUE** (default for **dist** or **distance** objects) implies that **x** is a distance matrix. If **diss = FALSE**, then **x** is treated as a matrix of units by variables. The fuzziness parameter is **memb.exp** (default: 2). The standardization option is provided by **stand** (default: FALSE).

To choose the optimal number of clusters, we compute the S values varying  $k$  from 2 to 5, by means of the output argument `silinfo$avg.width` of the function `fanny`.

```
> library(cluster)
> sil <- NULL
> for (g in 2:5)
+ {
+   set.seed(264)
+   sil[g-1] <- fanny(face, k = g, diss = TRUE,
+                      stand = TRUE)$silinfo$avg.width
+ }
> names(sil) <- paste("k =", 1:length(sil) + 1)
> round(sil, 2)
k = 2 k = 3 k = 4 k = 5
0.46 0.54 0.51 0.49
```

The maximum value is obtained for  $k = 3$ ; hence, we run the function `fanny` by setting this specific value.

```
> face.fanny.3 <- fanny(face, k = 3, diss = TRUE)
```

The output of `fanny` contains the following arguments.

```
> str(face.fanny.3)
List of 10
 $ membership : num [1:13, 1:3] 0.7933 0.2479 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:13] "Grief at death of mother" "
    "Savoring a Coke" "Very pleasant surprise" "
    "Maternal love-baby in arms" ...
  .. ..$ : NULL
 $ coeff       : Named num [1:2] 0.599 0.399
  ..- attr(*, "names")= chr [1:2] "dunn_coeff" "
    "normalized"
 $ memb.exp   : num 2
 $ clustering  : Named int [1:13] 1 2 2 2 1 3 3 2 ...
  ..- attr(*, "names")= chr [1:13] "Grief at death of
    mother" "Savoring a Coke" "Very pleasant surprise"
    "Maternal love-baby in arms" ...
 $ k.crisp    : num 3
 $ objective   : Named num [1:2] 8.28 1.00e-15
  ..- attr(*, "names")= chr [1:2] "objective" "
    "tolerance"
 $ convergence: Named int [1:3] 31 1 500
  ..- attr(*, "names")= chr [1:3] "iterations" "
    "converged" "maxit"
 $ diss        : NULL
```

```
$ call      : language fanny(x = face, k = 3, diss =
  TRUE)
$ silinfo   :List of 3
..$ widths    : num [1:13, 1:3] 1 1 1 1 1 2 2
...
... --- attr(*, "dimnames")=List of 2
... ..$ : chr [1:13] "Physical exhaustion" "Grief
  at death of mother" "Revulsion" "Light sleep" ...
... ..$ : chr [1:3] "cluster" "neighbor" "sil_
  width"
..$ clus.avg.widths: num [1:3] 0.481 0.607 0.517
..$ avg.width   : num 0.538
- attr(*, "class")= chr [1:2] "fanny" "partition"
```

To interpret the partition, we focus on the membership degree matrix (`membership`).

```
> face.fanny$.membership
          [,1]      [,2]
Grief at death of mother 0.79326769 0.08563603
Savoring a Coke          0.24791089 0.63719807
Very pleasant surprise   0.08380700 0.84832177
Maternal love-baby in arms 0.16818793 0.75430529
Physical exhaustion      0.81053863 0.10367431
Something wrong with plane 0.32893492 0.14642937
Anger at seeing dog beaten 0.04735385 0.02555723
Pulling hard on seat of chair 0.24636971 0.58796454
Unexpectedly meets old boyfriend 0.03584315 0.94274016
Revulsion                0.68785647 0.12484015
Extreme pain              0.55618164 0.17062186
Knows plane will crash    0.11900189 0.09624096
Light sleep                0.63981809 0.22159692
          [,3]
Grief at death of mother 0.12109628
Savoring a Coke          0.11489104
Very pleasant surprise   0.06787123
Maternal love-baby in arms 0.07750678
Physical exhaustion      0.08578706
Something wrong with plane 0.52463571
Anger at seeing dog beaten 0.92708892
Pulling hard on seat of chair 0.16566575
Unexpectedly meets old boyfriend 0.02141670
Revulsion                0.18730339
Extreme pain              0.27319649
Knows plane will crash    0.78475715
Light sleep                0.13858499
```

We can see that, although FaceExp does not contain Euclidean distances, all the membership degrees are non-negative. The facial expressions assigned to Cluster 1 are Grief at death of mother, Physical exhaustion, Revulsion, Extreme pain, and Light sleep.

```
> round(face.fanny.3$membership
+       [(face.fanny.3$clustering == 1), 1], 2)
Grief at death of mother      Physical exhaustion
          0.79                      0.81
      Revulsion      Extreme pain
          0.69                      0.56
    Light sleep
          0.64
```

Cluster 2 contains Savoring a Coke, Very pleasant surprise, Maternal love-baby in arms, Pulling hard on seat of chair, and Unexpectedly meets old boyfriend.

```
> round(face.fanny.3$membership
+       [(face.fanny.3$clustering == 2), 2])
      Savoring a Coke
          0.64
      Very pleasant surprise
          0.85
  Maternal love-baby in arms
          0.75
  Pulling hard on seat of chair
          0.59
Unexpectedly meets old boyfriend
          0.94
```

Finally, three facial expressions are assigned to Cluster 3: Something wrong with plane, Anger at seeing dog beaten, and Knows plane will crash.

```
> round(face.fanny.3$membership
+       [(face.fanny.3$clustering == 3), 3], 2)
Something wrong with plane Anger at seeing dog beaten
          0.52                      0.93
  Knows plane will crash
          0.78
```

In order to better characterize the obtained clusters, we use the dataset FaceScale.

```
> data("FaceScale")
> scale <- FaceScale
> str(scale)
'data.frame':   13 obs. of  3 variables:
 $ PU: num  3.8 5.9 8.8 7 3.3 3.5 2.1 6.7 7.4 2.9 ...
 $ AR: num  4.2 5.4 7.8 5.9 2.5 6.1 8 4.2 6.8 3 ...
 $ TS: num  4.1 4.8 7.1 4 3.1 6.8 8.2 6.6 5.9 5.1 ...
```

We run the FKM algorithm on such data.

```
> library(fclust)
> scale.FKM <- FKM(scale, seed = 264)
The default value k=2:6 has been set
The default index SIL.F has been set
The default value alpha=1 has been set for computing
SIL.F
```

The function `print.fclust` is used to display the obtained clustering results.

```
> print.fclust(scale.FKM)

Fuzzy clustering object of class 'fclust'

Number of units:
13

Number of clusters:
3

Clustering index values:
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.5448655 0.7201412 0.6156551 0.6564286 0.6309071

Closest hard clustering partition:
          Grief at death of mother
                                1
          Savoring a Coke
                                2
          Very pleasant surprise
                                2
          Maternal love-baby in arms
                                2
          Physical exhaustion
                                1
          Something wrong with plane
                                3
          Anger at seeing dog beaten
                                3
          Pulling hard on seat of chair
                                2
Unexpectedly meets old boyfriend
                                2
          Revulsion
                                1
          Extreme pain
                                1
          Knows plane will crash
                                3
```

```

Light sleep
      1

Membership degree matrix (rounded):
                                         Clus 1  Clus 2  Clus 3
Grief at death of mother             0.83   0.11   0.05
Savoring a Coke                     0.13   0.82   0.05
Very pleasant surprise               0.10   0.76   0.14
Maternal love-baby in arms          0.09   0.86   0.05
Physical exhaustion                 0.96   0.03   0.01
Something wrong with plane         0.21   0.29   0.50
Anger at seeing dog beaten          0.00   0.00   1.00
Pulling hard on seat of chair       0.16   0.75   0.08
Unexpectedly meets old boyfriend    0.02   0.95   0.02
Revulsion                           0.91   0.05   0.04
Extreme pain                         0.71   0.14   0.15
Knows plane will crash              0.02   0.03   0.95
Light sleep                          0.74   0.17   0.09

Available components:
[1] "U"           "H"           "F"           "clus"
[5] "medoid"      "value"       "criterion"   "iter"
[9] "k"            "m"           "ent"          "b"
[13] "vp"          "delta"       "stand"        "Xca"
[17] "X"           "D"           "call"

```

The optimal number of clusters is  $k = 3$ , corresponding to the maximum value of the FS index (0.7201412). There is a one-to-one correspondence with the partition obtained by fanny.

```

> table(face.fanny.3$clustering, scale.FKM$clus[, 1])

  1 2 3
1 5 0 0
2 0 5 0
3 0 0 3

```

To better characterize the three clusters, we take a look at the prototypes.

```

> scale.FKM$H
      PU          AR          TS
Clus 1 3.337131 2.811558 3.989618
Clus 2 6.990007 6.010149 5.583598
Clus 3 1.948255 7.918375 8.250620

```

We can notice that Cluster 1 is characterized by low values of the three perceptual dimensions (lowest values for AR and TS). Cluster 2 contains facial expressions with the highest value of PU and all the perceptual dimensions higher than those of Cluster

1. Finally, Cluster 3 is identified with the lowest value of PU and the highest values of AR and TS.

### 5.7.2 NEFRC

In this section, the implementation of the NEFRC clustering algorithm is presented by considering the function **NEFRC** contained in the package **fclust**. The input arguments are the same as for the other functions available in **fclust** except for **D**, the matrix, or data frame containing distances, which is needed in place of the unit-by-variable matrix or data frame **X**. In the specific case of relational data, the input option **index** can take all the cluster validity indices, except for **XB**.

### 5.7.2.1 Case Study with a Distance Matrix from Likert Scale Data

We start with the dataset FaceExp, already analyzed in the previous section.

```
> face.NEFLC <- NEFLC(D = face, seed = 264)
The default value k=2:6 has been set
The default index SIL.F has been set
The default value alpha=1 has been set for computing
    SIL.F
```

The resulting partition is exactly the same obtained by means of fanny.

```
+      [(face.NEFCR$clus[, 1] == 3), 2], 2)
Something wrong with plane Anger at seeing dog beaten
                                0.52                      0.93
    Knows plane will crash
                                0.78
```

### 5.7.2.2 Case Study with Categorical Data

A peculiarity of the NEFRC algorithm, unlike the other fuzzy clustering methods introduced in this chapter, is that it can also handle categorical data. We therefore illustrate now the use of NEFRC by its application to the dataset `houseVotes` contained in `fclust`, already analyzed in Sect. 2.6.2. Again we remove the level `yn` and the missing values.

```
> data("houseVotes")
> level.drop <- droplevels(houseVotes, exclude = "yn")
> houseVotesComplete <- level.drop[
+   complete.cases(level.drop), ]
> str(houseVotesComplete)
'data.frame': 232 obs. of 17 variables:
 $ class                               : Factor w/ 2
   levels "democrat","republican": 1 2 1 1 1 1 1 2 1
   2 ...
 $ handicapped-infants                 : Factor w/ 2
   levels "n","y": 1 1 2 2 2 2 2 2 1 ...
 $ water-project-cost-sharing          : Factor w/ 2
   levels "n","y": 2 2 2 2 1 1 2 1 2 ...
 $ adoption-of-the-budget-resolution  : Factor w/ 2
   levels "n","y": 2 1 2 2 2 2 2 1 2 1 ...
 $ physician-fee-freeze                : Factor w/ 2
   levels "n","y": 1 2 1 1 1 1 1 2 1 2 ...
 $ el-salvador-aid                    : Factor w/ 2
   levels "n","y": 2 2 1 1 1 1 1 2 1 2 ...
 $ religious-groups-in-schools        : Factor w/ 2
   levels "n","y": 2 2 1 1 1 1 1 1 2 ...
 $ anti-satellite-test-ban           : Factor w/ 2
   levels "n","y": 1 1 2 2 2 2 2 2 1 ...
 $ aid-to-nicaraguan-contras         : Factor w/ 2
   levels "n","y": 1 1 2 2 2 2 2 2 1 ...
 $ mx-missile                          : Factor w/ 2
   levels "n","y": 1 1 2 2 2 2 2 2 1 ...
 $ immigration                          : Factor w/ 2
   levels "n","y": 1 1 1 1 2 1 1 1 1 1 ...
 $ synfuels-corporation-cutback       : Factor w/ 2
   levels "n","y": 1 1 2 1 1 2 2 1 2 1 ...
 $ education-spending                  : Factor w/ 2
   levels "n","y": 1 2 1 1 1 1 1 2 1 2 ...
 $ superfund-right-to-sue             : Factor w/ 2
   levels "n","y": 2 2 1 1 1 1 1 2 1 2 ...
```

```
$ crime : Factor w/ 2
  levels "n","y": 2 2 1 1 1 1 1 2 1 2 ...
$ duty-free-exports : Factor w/ 2
  levels "n","y": 2 1 2 2 2 2 2 1 2 1 ...
$ export-administration-act-south-africa: Factor w/ 2
  levels "n","y": 2 2 2 2 2 2 2 2 2 1 ...
```

The first variable identifies the known classification, and it is not considered in the clustering algorithm. It can be saved in an object (`class`) for further analysis.

```
> votes <- houseVotesComplete[, -1]
> class <- houseVotesComplete[, 1]
```

Since categorical variables are observed, we computed the Gower distance, by means of the option `metric = "gower"` in the function `daisy` and we used it as input argument in the function `NEFRC`.

```
> library(cluster)
> D.votes <- daisy(x = votes, metric = "gower")
> votes.NEFRC <- NEFRC(D = D.votes, RS = 5,
+ seed = 264)
```

To obtain an outline of the `NEFRC` output, as for the other `fclust` clustering algorithms, we can use the `summary.fclust` function (the output is not displayed for the sake of brevity). The optimal number of clusters, corresponding to the maximum value of FS (0.64), is  $k = 2$ . Cluster 1 is composed of 119 Congressional Voting Records while Cluster 2 by 113. The average membership degrees of the units assigned to Clusters 1 and 2 are 0.80 and 0.78, respectively. In addition, the membership degrees of Cluster 1 range from 0.51 to 0.91, while those of Cluster 2 from 0.50 to 0.90.

To compare the `NEFRC` partition with the classification contained in the variable `class`, a cross-tabulation is built.

```
> table(votes.NEFRC$clus[, 1], class)
   class
   democrat republican
1      19       100
2     105        8
```

As we can notice, Cluster 1 is mainly composed of republicans, while Cluster 2 mainly contains democrats. The similarity between the true and the observed partition is measured by means of the indices included in the function `Fclust.compare`.

```
> Fclust.compare(class, votes.NEFRC$U)
   ARI.F      RI.F JACCARD.F
0.3378474  0.6689236  0.5024775
```

These values confirm a quite good level of agreement and a better recovery of the two parties if compared with the results reported in Sect. 2.6.2.

### 5.7.2.3 Case Study with Mixed Data

The function NEFRC can be also used for mixed-type data. This is an additional value of this fuzzy clustering algorithm. To show such a feature, we analyze a dataset contained in the package **PCAmixdata** [12]. The data `gironde` are a list of four datasets characterizing conditions of life in 542 cities, Gironde (France). In particular, we analyze the dataset `gironde$housing`. It comes from the 2009 population census realized in Gironde by INSEE (Institut National de la Statistique et des Études Économiques). The dataset contains the description of the cities by  $p = 5$  variables (two categorical and three quantitative variables).

```
> library(PCAmixdata)
> data("gironde")
> housing <- gironde$housing
> str(housing)
'data.frame': 542 obs. of 5 variables:
 $ density    : num 131.7 21.2 532 101.2 551.9 ...
 $ primaryres: num 88.8 87.5 94.9 93.8 62.1 ...
 $ houses     : Factor w/ 2 levels "inf" 90%,"sup" 90%
   ": 1 2 1 2 1 2 2 2 1 2 ...
 $ owners     : num 64.2 77.1 65.7 66.5 71.5 ...
 $ council    : Factor w/ 2 levels "inf" 5%,"sup" 5%:
   2 1 2 2 1 1 1 1 2 1 ...
```

The variables are the population density (`density`), the percentage of primary residences (`primaryres`), the percentage of houses (`houses`) with two levels, the percentage of homeowners living in their primary residence (`owners`), and the percentage of council housing within the cities (`council`) with two levels.

To obtain a fuzzy partition of the cities, we compute the distance matrix by using the option `metric = "gower"` in `daisy`, and we use it as an input argument for the function `NEFRC`. Note that `NEFRC` is computationally intensive.

```
> D.housing <- daisy(housing, metric = "gower")
> housing.NEFRC <- NEFRC(D = D.housing, RS = 5,
+                           seed = 264)
```

The optimal number of clusters is  $k = 2$ . The corresponding FS value, 0.82, is much higher than the other ones.

```
> housing.NEFRC$k
Number of clusters
2
> housing.NEFRC$criterion
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.8216079 0.5890470 0.5042780 0.6092894 0.6394809
```

The obtained partition includes two clusters of size 389 and 153, respectively.

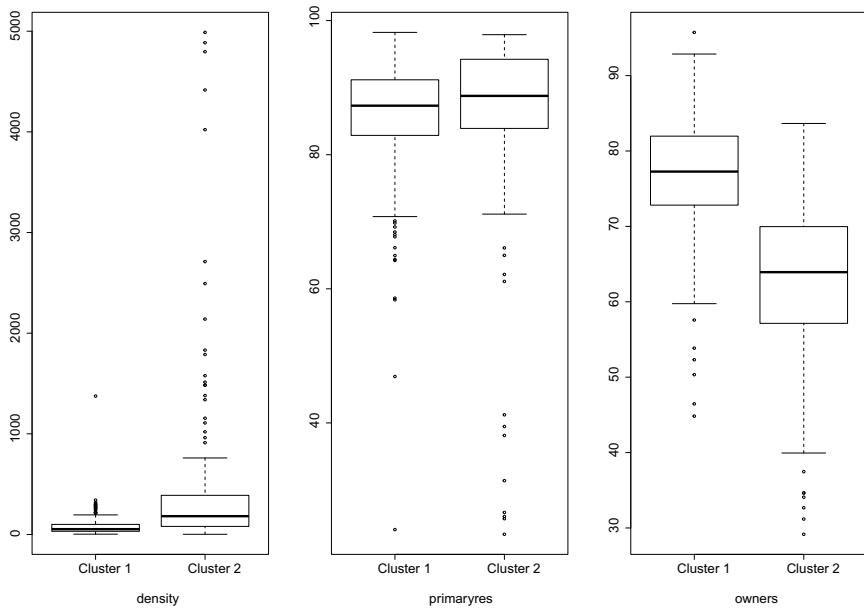
```
> cl.size(housing.NEFRC$U)
Clus 1 Clus 2
389     153
```

To characterize the obtained clusters, we use a graphical representation for the distributions of the variables. In detail, for the quantitative ones, the boxplots are provided in Fig. 5.16.

```
> par(mfrow = c(1, 3))
> boxplot(housing$density~housing.NEFRC$clus[, 1],
+           names = c("Cluster 1", "Cluster 2"),
+           sub = "density")
> boxplot(housing$primaryres~housing.NEFRC$clus[, 1],
+           names = c("Cluster 1", "Cluster 2"),
+           sub = "primaryres")
> boxplot(housing$owners~housing.NEFRC$clus[, 1],
+           names = c("Cluster 1", "Cluster 2"),
+           sub = "owners")
```

For the categorical variables, we analyze the distributions stratified by cluster.

```
> perc.houses <- table(housing$houses,
+                        housing.NEFRC$clus[, 1]) /
+  rbind(colSums(table(housing$houses,
+                      housing.NEFRC$clus[, 1])),
+        colSums(table(housing$houses,
+                      housing.NEFRC$clus[, 1]))*100
> perc.council <- table(housing$council,
+                        housing.NEFRC$clus[, 1]) /
+  rbind(colSums(table(housing$council,
+                      housing.NEFRC$clus[, 1])),
+        colSums(table(housing$council,
+                      housing.NEFRC$clus[, 1]))*100
```



**Fig. 5.16** housing data: boxplots of population density, primary residences, homeowners living in their primary residence, by cluster (NEFRC solution with  $k = 2$  clusters)

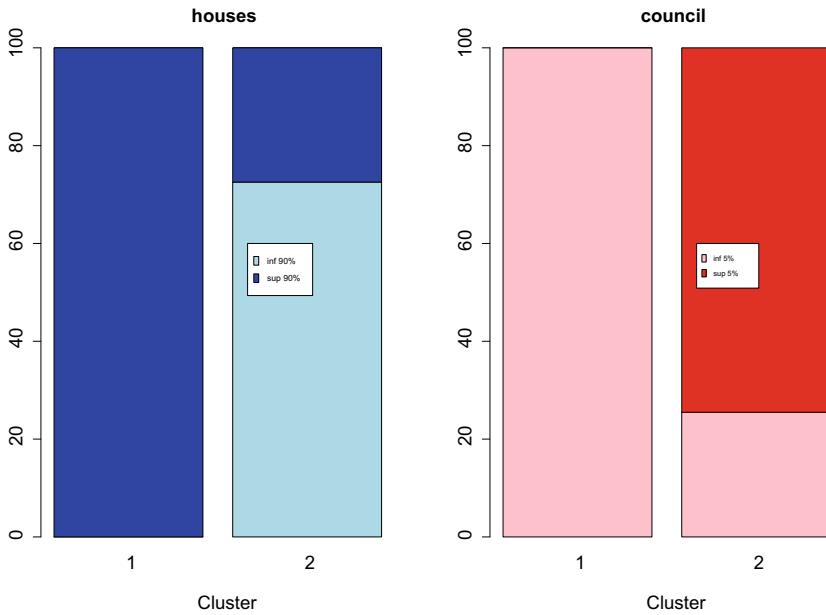
**Table 5.2** housing data: distributions of houses by cluster (NEFRC solution with  $k = 2$  clusters)

Houses	Cluster 1	Cluster 2
Less than 90%	0.00	72.55
More than 90%	100.00	27.45

**Table 5.3** housing data: distributions of council housing by cluster (NEFRC solution with  $k = 2$  clusters)

Council	Cluster 1	Cluster 2
Less than 5%	100.00	25.49
More than 5%	0.00	74.51

We get Tables 5.2 and 5.3. All the cities in Cluster 1 have a percentage of houses greater than 90% and a percentage of council housing lower than 5%. 72.55% of cities in Cluster 2 have a percentage of houses lower than 90%, while the remaining 27.45% greater than 90%. 74.51% of cities assigned to Cluster 2 have a percentage of council housing greater than 5%, while 25.49% lower than 5%.



**Fig. 5.17** housing data: barplots of houses and council housing by cluster (NEFRC solution with  $k = 2$  clusters)

The stratified distributions are graphically represented by barplots (see Fig. 5.17).

```
> par(mfrow = c(1, 2))
> barplot(perc.houses,
+           beside = FALSE,
+           col = c("lightblue", "blue"),
+           cex.names = 0.7,
+           sub = "Cluster",
+           main = "houses")
> legend(1.5, 60, levels(housing$houses),
+         fill = c("lightblue", "blue"), bg = "white",
+         cex = 0.7, text.width = 0.3)
> barplot(perc.council,
+           beside = FALSE,
+           col = c("pink", "red"),
+           cex.names = 0.7,
+           sub = "Cluster",
+           main = "council")
> legend(1.5, 60, levels(housing$council),
+         fill = c("pink", "red"), bg = "white",
+         cex = 0.6, text.width = 0.3)
```

## 5.8 Fuzzy $k$ -Means with Noise Cluster

In this section, some robust versions of fuzzy clustering algorithms are provided. The performances of FkM-type algorithms may be affected by the presence of outliers, due to the unit-sum constraints of the membership degrees. To fulfill the above constraints, even the units that are far from all the clusters are forced to belong to the closest one. This leads to centroids that are constructed taking into account also units having characteristics that are quite different from those of the remaining ones. For this reason, the results may be not realistic. Several kinds of robustification have been proposed in the literature. Among them, a powerful approach is based on the concept of noise cluster [14], containing all the units with a high membership degree recognized as outliers. A penalization term is added for this purpose in the optimization problem of the Fuzzy  $k$ -Means algorithm with Noise cluster (FkMN):

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{H}} J_{\text{FkMN}} &= \sum_{i=1}^n \sum_{g=1}^k u_{ig}^m d^2(\mathbf{x}_i, \mathbf{h}_g) + \sum_{i=1}^n \delta^2 \left( 1 - \sum_{g=1}^k u_{ig} \right)^m, \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^{k+1} u_{ig} &= 1, \quad i = 1, \dots, n, \end{aligned} \quad (5.34)$$

where  $\delta^2$  is a non-negative parameter that needs to be chosen in advance, denoting the square distance of each unit to the noise cluster, and  $u_{i(k+1)} = 1 - \sum_{g=1}^k u_{ig}$  represents the membership degree of unit  $i$  to the noise cluster.

Solving (5.34) leads to a partition with  $k + 1$  clusters. The first  $k$  standard clusters are homogeneous, whereas the noise cluster contains all the outliers and it is usually formed by units with heterogeneous features.

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible membership degree matrix  $\mathbf{U}$ .
2. Given  $\mathbf{U}$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n u_{ig}^m \mathbf{x}_i}{\sum_{i=1}^n u_{ig}^m}, \quad g = 1, \dots, k. \quad (5.35)$$

3. Given  $\mathbf{H}$ , update the membership degree matrix  $\mathbf{U}$ :

$$u_{ig} = \frac{1}{\sum_{g'=1}^k \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_{g'})}{d^2(\mathbf{x}_i, \mathbf{h}_{g'})} \right)^{\frac{1}{m-1}} + \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_g)}{\delta^2} \right)^{\frac{1}{m-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k, \quad (5.36)$$

and

$$u_{i(k+1)} = 1 - \sum_{g=1}^k u_{ig}, \quad i = 1, \dots, n. \quad (5.37)$$

4. Repeat steps 2 and 3 until convergence is reached.

All the fuzzy clustering approaches, analyzed in the previous sections, can be extended by introducing a noise cluster.

In the package **fclust**, the following functions correspond to robust versions of fuzzy clustering algorithms: **FKM.noise** (FkMN, Fuzzy  $k$ -Means with Noise cluster), **FKM.ent.noise** (Fuzzy  $k$ -Means with Entropy regularization and Noise cluster), **FKM.gk.ent.noise** (Gustafson and Kessel-like Fuzzy  $k$ -Means with Entropy regularization and Noise cluster), **FKM.gk.noise** (Gustafson and Kessel-like Fuzzy  $k$ -Means with Noise cluster), **FKM.gkb.ent.noise** (Gustafson, Kessel, and Babuska-like Fuzzy  $k$ -Means with Entropy regularization and Noise cluster), **FKM.gkb.noise** (Gustafson, Kessel, and Babuska-like Fuzzy  $k$ -Means with Noise cluster), **FKM.med.noise** (Fuzzy  $k$ -Medoids with Noise cluster), **FKM.pf.noise** (Fuzzy  $k$ -Means with Polynomial Fuzzifier and Noise cluster), and **NEFRC.noise** (Non-Euclidean Fuzzy Relational Clustering with Noise cluster).

### 5.8.1 **FKM.noise**

This section describes the noise variant of the most known algorithm, FkM, implemented in the function **FKM.noise**. Unlike FkM, the robust version includes an additional input argument, **delta**, denoting the noise distance (default: average Euclidean distance between units and prototypes from FkM using the same values of **k** and **m**).

We use the dataset **mtcars** [28] included in the package **datasets**. The data are extracted from the 1974 Motor Trend US magazine and cover fuel consumption and 10 aspects of automobile design and performance for 32 automobiles. It is a data frame with  $n = 32$  units on  $p = 11$  variables: miles/(US) gallon, number of cylinders, displacement (cu.in.), gross horsepower, rear axle ratio, weight (1000 lbs), 1/4 mile time, V/S (engine cylinder configuration), transmission, number of forward gears, and number of carburetors. The variables are all quantitative but **vs** and **am**, even if in the data frame they are reported as numeric.

```
> library(datasets)
> data("mtcars")
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
```

```
$ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 ...
$ wt : num 2.62 2.88 2.32 3.21 3.44 ...
$ qsec: num 16.5 17 18.6 19.4 17 ...
$ vs : num 0 0 1 1 0 1 0 1 1 ...
$ am : num 1 1 1 0 0 0 0 0 0 ...
$ gear: num 4 4 4 3 3 3 3 4 4 ...
$ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

We remove categorical variables for the following cluster analysis.

```
> mtcars.r <- mtcars[, -(8:9)]
```

Since the variables have different units of measurement, the standardization option is needed.

```
> library(fclust)
> mtcars.FKM.noise <- FKM.noise(mtcars.r,
+                                     stand = 1, RS = 10,
+                                     seed = 264)
The default value k=2:6 has been set
The default index SIL.F has been set
The default value alpha=1 has been set for computing
SIL.F
```

The optimal number of cluster is  $k = 2$ , corresponding to the highest value of FS with  $k$  varying from 2 to 6.

```
> mtcars.FKM.noise$k
Number of clusters
2
> mtcars.FKM.noise$criterion
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.6870745 0.6111064 0.5687289 0.4705014 0.5780071
```

We use different random starts, and we see that the values of the objective function are always the same for all the initializations.

```
> mtcars.FKM.noise$value
Start 1 Start 2 Start 3 Start 4 Start 5 Start 6
75.88538 75.88538 75.88538 75.88538 75.88538 75.88538
Start 7 Start 8 Start 9 Start 10
75.88538 75.88538 75.88538 75.88538
```

The assignments and the corresponding membership degrees of the cars are reported in the matrix `clus`.

```
> mtcars.FKM.noise$clus
      Cluster Membership degree
Mazda RX4                 1        0.5548905
Mazda RX4 Wag              1        0.5758461
Datsun 710                 1        0.9210961
Hornet 4 Drive              2        0.4291432
Hornet Sportabout           2        0.8925012
Valiant                     2        0.4347773
Duster 360                 2        0.8027059
Merc 240D                  1        0.8200124
Merc 230                   1        0.5802317
Merc 280                   1        0.5209460
Merc 280C                  1        0.5040038
Merc 450SE                 2        0.9621231
Merc 450SL                 2        0.9543703
Merc 450SLC                2        0.9430261
Cadillac Fleetwood          2        0.6513206
Lincoln Continental          2        0.6406502
Chrysler Imperial            2        0.6946155
Fiat 128                   1        0.7811004
Honda Civic                 1        0.6444503
Toyota Corolla              1        0.6999164
Toyota Corona               1        0.6735422
Dodge Challenger             2        0.8486327
AMC Javelin                 2        0.8882985
Camaro Z28                 2        0.7002444
Pontiac Firebird             2        0.8798698
Fiat X1-9                   1        0.8780788
Porsche 914-2               1        0.6953395
Lotus Europa                 1        0.6457875
Ford Pantera L              2        0.3614103
Ferrari Dino                 1        0.3308944
Maserati Bora               2        0.2946065
Volvo 142E                  1        0.9218643
```

The cars that have an unclear assignment to the proper clusters are five: Hornet 4 Drive, Valiant, Ford Pantera L, Ferrari Dino, and Maserati Bora.

```
> ind_fuzzy <- which(mtcars.FKM.noise$clus[, 2] < 0.5)
> mtcars.r[ind_fuzzy, ]
      mpg cyl disp hp drat    wt  qsec gear
Hornet 4 Drive  21.4   6  258 110 3.08 3.215 19.44     3
Valiant         18.1   6  225 105 2.76 3.460 20.22     3
Ford Pantera L 15.8   8  351 264 4.22 3.170 14.50     5
Ferrari Dino    19.7   6  145 175 3.62 2.770 15.50     5
Maserati Bora   15.0   8  301 335 3.54 3.570 14.60     5
      carb
Hornet 4 Drive   1
```

Valiant	1
Ford Pantera L	4
Ferrari Dino	6
Maserati Bora	8

Only Maserati Bora is indeed recognized as an outlier. In fact, its membership degrees to the noise cluster is 0.54.

```
> ind_noise <- which(rowSums(mtcars.FKM.noise$U) < 0.5)
> length(ind_noise)
[1] 1
> row.names(mtcars.r[ind_noise, ])
[1] "Maserati Bora"
> 1 - sum(mtcars.FKM.noise$U[ind_noise, ])
[1] 0.5452703
```

In order to better explain this outlier, we compare its values with the centroids.

```
> mtcars.r[ind_noise, ]
      mpg cyl disp hp drat wt qsec
Maserati Bora 15     8 301 335 3.54 3.57 14.6
      gear carb
Maserati Bora 5      8
> Hraw(mtcars.FKM.noise$X, mtcars.FKM.noise$H)
      mpg      cyl      disp      hp      drat
Clus 1 24.88878 4.434880 121.9081 92.19589 3.987704
Clus 2 16.03562 7.797646 327.7147 184.23053 3.159514
      wt      qsec      gear      carb
Clus 1 2.483594 18.73966 4.046483 1.964255
Clus 2 3.868353 17.24989 3.107105 2.958960
```

Cluster 1 identifies cars characterized by high fuel efficiency, four gears in the transmission on average, higher than that of cars in Cluster 2 and better performance in terms of acceleration (qsec), while, on the other hand, cars in Cluster 2 have higher values of number of cylinders (cyl), total amount of power the engine can generate (disp), gross horsepower (hp), weight (wt), and number of carburetors (carb). Maserati Bora is far from the centroids especially with regard to some variables. In particular, it is mainly characterized by low fuel efficiency and very high values of gross horsepower and number of carburetors.

Further comments can be made by considering the categorical variables (vs and am). The  $p$ -value of the  $\chi^2$  test suggests to reject the null hypothesis of independence between the obtained partition and vs and between it and am. In particular, by looking at the cross-tabulations, we can discover that most units assigned to Cluster 1 present a straight line configuration of the cylinders in the engine (vs = 1) and manual transmission (am = 1). On the contrary, cars in Cluster 2 are mainly characterized by v-shape configuration of the cylinders (vs = 0) and automatic

transmission ( $\text{am} = 0$ ).

```
> table(mtcars.FKM.noise$clus[, 1], mtcars$vs)
      0   1
  1   4 12
  2 14  2
> chisq.test(mtcars.FKM.noise$clus[, 1],
+             mtcars$vs)$p.value
[1] 0.001340641
> table(mtcars.FKM.noise$clus[, 1], mtcars$am)
      0   1
  1   5 11
  2 14  2
> chisq.test(mtcars.FKM.noise$clus[, 1],
+             mtcars$am)$p.value
[1] 0.003983112
```

## 5.9 Possibilistic $k$ -Means

As we have already discussed in Sect. 5.8, the fuzzy approach leads to unrealistic results when outliers are contained in the dataset. This is due to the constraints on the membership degrees. A further alternative for dealing with this issue is based on the possibilistic approach [34]. It consists in relaxing the unit-sum constraints of the membership degrees and in adding a regularization term in the optimization problem.

The so-called Possibilistic  $k$ -Means (PkM) clustering method [34, 35] can be formalized as

$$\begin{aligned} \min_{\mathbf{T}, \mathbf{H}} J_{\text{PkM}} &= \sum_{i=1}^n \sum_{g=1}^k t_{ig}^\eta d^2(\mathbf{x}_i, \mathbf{h}_g) + \sum_{g=1}^k \gamma_g \sum_{i=1}^n (1 - t_{ig})^\eta, \\ \text{s.t. } t_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \end{aligned} \quad (5.38)$$

where the matrix  $\mathbf{T}$  of order  $(n \times k)$  stores the (possibilistic) membership degrees. The second term of the loss function in (5.38) avoids the trivial solution with  $\mathbf{T} = \mathbf{0}$ . This approach is named possibilistic due to the membership degrees recognized as possibility values. These are the so-called typicality degrees. The cluster-specific parameter  $\gamma_g$  determines the distance at which the membership degree of a unit to the  $g$ -th cluster becomes 0.5 and  $\eta (> 1)$  is the fuzzifier (usually,  $\eta = 2$ ). The parameter  $\gamma_g$  can be defined as

$$\gamma_g = \gamma \frac{\sum_{i=1}^n u_{ig}^m d^2(\mathbf{x}_i, \mathbf{h}_g)}{\sum_{i=1}^n u_{ig}^m}, \quad g = 1, \dots, k, \quad (5.39)$$

where, usually,  $\gamma = 1$  and  $\mathbf{h}_g$  and  $t_{ig}$  are obtained from FkM.

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible typicality degree matrix  $\mathbf{T}$ .
2. Given  $\mathbf{T}$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n t_{ig}^\eta \mathbf{x}_i}{\sum_{i=1}^n t_{ig}^\eta}, \quad g = 1, \dots, k. \quad (5.40)$$

3. Given  $\mathbf{H}$ , update the typicality degree matrix  $\mathbf{T}$ :

$$t_{ig} = \frac{1}{1 + \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_g)}{\gamma_g} \right)^{\frac{1}{\eta-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k. \quad (5.41)$$

4. Repeat steps 2 and 3 until convergence is reached.

Unlike the fuzzy approach, the typicality degrees are computed by using only the distance between the unit and the closest centroid, regardless of the centroids of the remaining clusters. This explains the reason why they are referred to as degrees of typicality (of units to clusters).

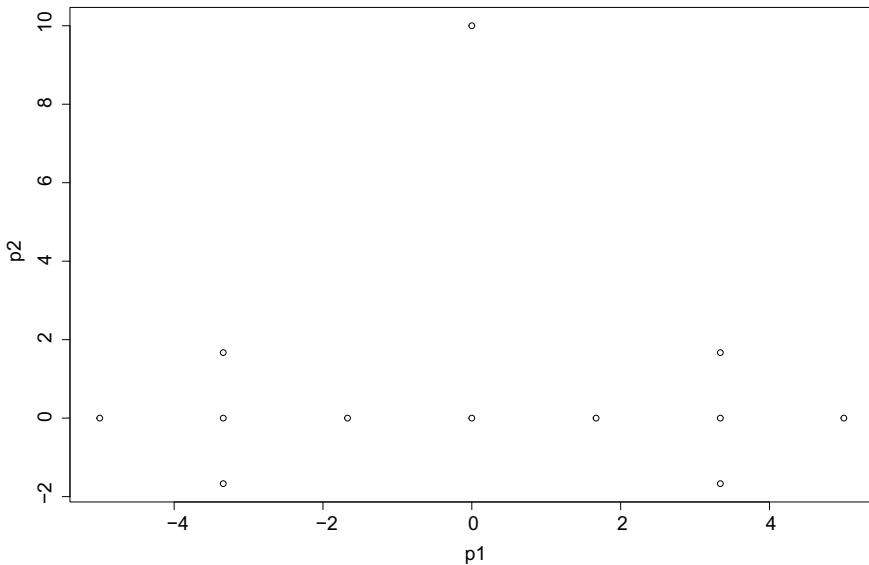
### 5.9.1 **pcm**

This section describes the implementation of the PkM clustering algorithm via the function **pcm**. The function is contained in the package **ppclust** [10]. Note that the function implements two versions of the PkM algorithm. The one reported in (5.41) is considered by setting the default option `oftype = 1`.

We use a synthetic dataset described in [43], contained in the package **ppclust**: `x12`. It consists of two well-separated clusters and two noisy units (see Fig. 5.18).

```
> library(ppclust)
> data("x12")
> str(x12)
num [1:12, 1:2] -5 -3.34 -3.34 -3.34 -1.67 1.67 ...
- attr(*, "dimnames")=List of 2
..$ : NULL
..$ : chr [1:2] "p1" "p2"
```

The compulsory input arguments of the function **pcm** are a quantitative vector, data frame or matrix (`x`) and an integer specifying the number of clusters or a matrix containing the initial cluster centers (`centers`). For the remaining arguments, default values are used. In particular, the typicality exponent `eta` is equal to 2, and the data are not standardized (`stand = FALSE`). A matrix containing the



**Fig. 5.18** x12 data: scatterplot

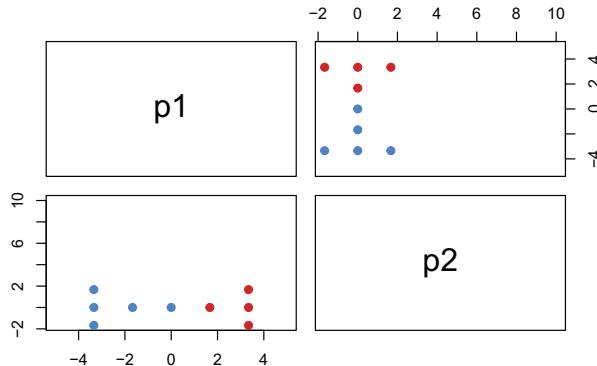
initial membership degrees can be specified via the option `memberships` (note that a good starting point is very useful in PkM).

```
> pcm.x12 <- pcm(x = x12, centers = 2, numseed = 264)
```

The estimated typicality degrees are contained in the output argument `t`.

```
> pcm.x12$t
   Cluster 1   Cluster 2
1  0.49182466  0.13369024
2  0.65502885  0.19360831
3  0.84687495  0.20750206
4  0.64821836  0.19300893
5  0.97196717  0.35111743
6  0.35111743  0.97196717
7  0.19360831  0.65502885
8  0.20750206  0.84687495
9  0.19300893  0.64821836
10 0.13369024  0.49182466
11 0.63107911  0.63107911
12 0.07030045  0.07030045
```

The first five units are assigned to Cluster 1, while the following five to Cluster 2. Unit 11 is exactly in the middle of the two clusters as its typicality degree is constant



**Fig. 5.19**  $x_{12}$  data: scatterplot of the units with typicality degrees greater than 0.5 by cluster (Pkm solution with  $k = 2$  clusters)

across clusters ( $t_{11}$  and  $t_{12}$  equal to 0.63). Unit 12 is equally far from both clusters, the typicalities are again constant over clusters, but close to 0 because it is located very far from the bulk of data.

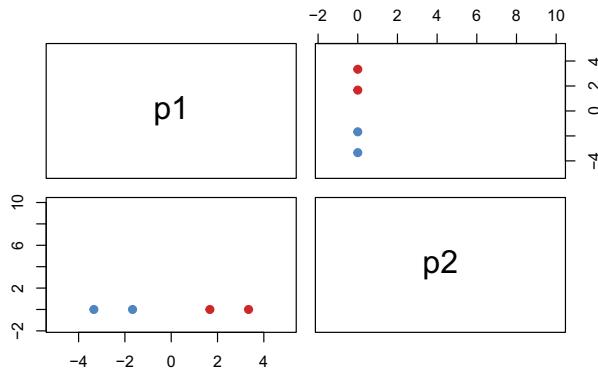
The cluster centroids are provided by the output argument v.

```
> pcm.x12$v
          p1           p2
Cluster 1 -2.146395 0.01892445
Cluster 2  2.146395 0.01892445
```

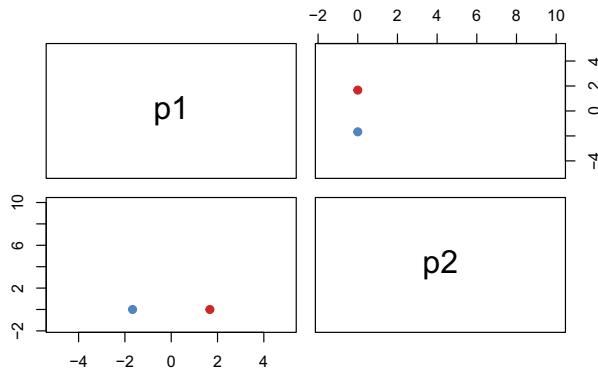
The function `plotcluster` of the package **ppclust** can be used to plot the clustering results. It needs as input argument an object of class **ppclust**, such as `pcm.x12`. In addition, we can select the membership degree type, `mt` (the default option is "u" for fuzzy membership degrees when the object of class **ppclust** is the output of `fcm`; "t" for typicality degrees) and a character to specify the cluster membership, `cm` (the default option is "max", i.e., maximum membership degree for each unit; "threshold" for units assigned to a cluster when the maximum membership degree is higher than a threshold specified by `tv`).

We plot units with typicality degrees greater than 0.5 (`tv = 0.5`) (see Fig. 5.19), greater than 0.7 (`tv = 0.7`) (see Fig. 5.20), and greater than 0.9 (`tv = 0.9`) (see Fig. 5.21).

```
> plotcluster(pcm.x12, mt = "t", cm = "threshold",
+              tv = 0.5)
> plotcluster(pcm.x12, mt = "t", cm = "threshold",
+              tv = 0.7)
> plotcluster(pcm.x12, mt = "t", cm = "threshold",
+              tv = 0.9)
```



**Fig. 5.20**  $\times 12$  data: scatterplot of the units with typicality degrees greater than 0.7 by cluster (P<sub>k</sub>M solution with  $k = 2$  clusters)



**Fig. 5.21**  $\times 12$  data: scatterplot of the units with typicality degrees greater than 0.9 by cluster (P<sub>k</sub>M solution with  $k = 2$  clusters)

Note that a different possibilistic algorithm, called the possibilistic clustering algorithm [55], is also implemented in the package **ppclust** via the function **pca**.

## 5.10 Hybrid (Fuzzy/Possibilistic) Clustering Methods

In this section, some hybrid fuzzy/possibilistic methods are described. They arise to exploit the benefits (and overcome the drawbacks) of both approaches. In this perspective, the first proposal [42] is the Fuzzy Possibilistic  $k$ -Means (FP<sub>k</sub>M). The idea is to combine the membership degrees of the F<sub>k</sub>M and the typicality degrees of the P<sub>k</sub>M algorithm in the objective function. This leads to avoid the limit of the F<sub>k</sub>M in the presence of outliers and the well-known coincident cluster problem of the P<sub>k</sub>M algorithm [5]. Specifically, the P<sub>k</sub>M may lead to a trivial solution with coincident

clusters, that is, clusters with the same centroids, because its objective function can usually be split into  $k$  terms, one for each cluster, to be minimized independently of each other.

The FPkM optimization problem can be formalized as

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{T}, \mathbf{H}} J_{\text{FPkM}} &= \sum_{i=1}^n \sum_{g=1}^k \left( u_{ig}^m + t_{ig}^\eta \right) d^2(\mathbf{x}_i, \mathbf{h}_g), \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n, \\ t_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{i=1}^n t_{ig} &= 1, \quad g = 1, \dots, k. \end{aligned} \quad (5.42)$$

As we can notice, the squared distances are weighted using the terms  $(u_{ig}^m + t_{ig}^\eta)$  as weights. In addition to the usual constraints on the fuzzy membership degrees in  $\mathbf{U}$ , it is also requested that the column-wise sums of  $\mathbf{T}$  are equal to 1. In other words, for each cluster, the typicality values are constrained so that their sum across units is equal to one.

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible fuzzy membership degree matrix  $\mathbf{U}$  and a typicality degree matrix  $\mathbf{T}$ .
2. Given  $\mathbf{U}$  and  $\mathbf{T}$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n \left( u_{ig}^m + t_{ig}^\eta \right) \mathbf{x}_i}{\sum_{i=1}^n \left( u_{ig}^m + t_{ig}^\eta \right)}, \quad g = 1, \dots, k. \quad (5.43)$$

3. Given  $\mathbf{H}$ , update the fuzzy membership degree  $\mathbf{U}$  and the typicality degree matrix  $\mathbf{T}$ :

$$u_{ig} = \frac{1}{\sum_{g'=1}^k \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_{g'})}{d^2(\mathbf{x}_i, \mathbf{h}_g)} \right)^{\frac{1}{m-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k, \quad (5.44)$$

$$t_{ig} = \frac{1}{\sum_{i'=1}^n \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_g)}{d^2(\mathbf{x}_{i'}, \mathbf{h}_g)} \right)^{\frac{1}{\eta-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k. \quad (5.45)$$

4. Repeat steps 2 and 3 until convergence is reached.

A Modified Fuzzy Possibilistic  $k$ -Means (MFPkM) has been proposed in [46]. It consists in incorporating a weighting parameter in the objective function of the FPkM method to tune the importance of the two terms.

The constraints on the column-wise sums of  $\mathbf{T}$  in (5.42) lead to unrealistic values for large datasets. For this reason, a different kind of hybrid approach is suggested in [43]. This is known as the Possibilistic Fuzzy  $k$ -Means (PFkM) algorithm. The PFkM exploits the benefits of both approaches and overcomes the issues due to the unit column-wise sums of  $\mathbf{T}$  in the FPkM.

The PFkM clustering method can be formalized as

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{T}, \mathbf{H}} J_{\text{PFkM}} &= \sum_{i=1}^n \sum_{g=1}^k \left( au_{ig}^m + bt_{ig}^\eta \right) d^2(\mathbf{x}_i, \mathbf{h}_g) + \sum_{g=1}^k \gamma_g \sum_{i=1}^n (1 - t_{ig})^\eta, \\ \text{s.t. } u_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \\ \sum_{g=1}^k u_{ig} &= 1, \quad i = 1, \dots, n, \\ t_{ig} &\in [0, 1], \quad i = 1, \dots, n, \quad g = 1, \dots, k, \end{aligned} \quad (5.46)$$

where  $a$  and  $b$  are non-negative parameters tuning the importance of the two terms. The higher the  $a$  ( $b$ ) is, the more relevant is the emphasis of the fuzzy (possibilistic) approach. The parameter of fuzziness  $m > 1$  has the same meaning as in the FkM algorithm, while  $\eta$  and  $\gamma_g$  have the same meanings as in the PkM algorithm.

The optimal solution can be found by means of the following iterative algorithm.

1. Rationally or randomly choose a feasible fuzzy membership degree matrix  $\mathbf{U}$  and a typicality degree matrix  $\mathbf{T}$ .
2. Given  $\mathbf{U}$  and  $\mathbf{T}$ , update the centroid matrix  $\mathbf{H}$ :

$$\mathbf{h}_g = \frac{\sum_{i=1}^n \left( au_{ig}^m + bt_{ig}^\eta \right) \mathbf{x}_i}{\sum_{i=1}^n \left( au_{ig}^m + bt_{ig}^\eta \right)}, \quad g = 1, \dots, k. \quad (5.47)$$

3. Given  $\mathbf{H}$ , update the fuzzy membership degree matrix  $\mathbf{U}$  and the typicality degree matrix  $\mathbf{T}$ :

$$u_{ig} = \frac{1}{\sum_{g'=1}^k \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_{g'})}{d^2(\mathbf{x}_i, \mathbf{h}_g)} \right)^{\frac{1}{m-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k, \quad (5.48)$$

$$t_{ig} = \frac{1}{1 + \left( \frac{d^2(\mathbf{x}_i, \mathbf{h}_g)}{\gamma_g} \right)^{\frac{1}{\eta-1}}}, \quad i = 1, \dots, n, \quad g = 1, \dots, k. \quad (5.49)$$

4. Repeat steps 2 and 3 until convergence is reached.

In all the above hybrid proposals, the cluster structure can be evaluated by means of the centroid matrix  $\mathbf{H}$ . The fuzzy membership degree matrix  $\mathbf{U}$  helps to recognize the partition of the units. On the other hand, the typicality degree matrix  $\mathbf{T}$  is useful to detect the outliers.

### 5.10.1 **fpcm**, **mfpcm** and **pfcm**

This section details the implementation of the FP $k$ M, the MFP $k$ M, and the PF $k$ M clustering algorithm, using the functions **fpcm**, the **mfpcm**, and the **pfcm** provided in the package **ppclust**. We use the well-known dataset **iris** [2, 22], available in the package **datasets**. It contains the measurements, in centimeters, of the sepal length and width and petal length and width for 50 flowers from each of three species of iris. The species are setosa, versicolor, and virginica. One of the three clusters (setosa) is well separated from the other two, while versicolor and virginica have some overlap.

```
> library(datasets)
> data("iris")
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 ...
 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 ...
 $ Species      : Factor w/ 3 levels "setosa",
                  "versicolor", ...: 1 1 1 1 1 1 1 ...
```

We run the function **pcm** by setting  $k = 3$  clusters (**centers**) and fixing the seed number (**numseed**).

```
> library(ppclust)
> iris.pcm <- pcm(iris[, 1:4], centers = 3,
+                   numseed = 321)
```

Unfortunately, even if we have selected a number of clusters equal to  $k = 3$ , the function **pcm** leads to two coincident clusters, as we can notice taking a look at the centroids contained in the output argument **v**.

```
> iris.pcm$v
      Sepal.Length Sepal.Width Petal.Length
Cluster 1       6.172303    2.877981     4.763067
Cluster 2       6.172882    2.879010     4.763516
Cluster 3       5.002622    3.398096     1.484792
      Petal.Width
Cluster 1       1.6077434
Cluster 2       1.6065564
Cluster 3       0.2472753
```

In fact, Clusters 1 and 3 contain 100 and 50 flowers, respectively, while Cluster 2 is empty, as it substantially overlaps with Cluster 1.

```
> iris.pcm$csizes
  1   2   3
100   0   50
```

Virtually, the same results are obtained by using the function `pca`.

```
> iris.pca <- pca(iris[, 1:4], centers = 3,
+                     numseed = 321)
> iris.pca$v
      Sepal.Length Sepal.Width Petal.Length
Cluster 1       6.173748    2.874387     4.770252
Cluster 2       6.173748    2.874387     4.770252
Cluster 3       4.993286    3.406194     1.473917
      Petal.Width
Cluster 1       1.6100255
Cluster 2       1.6100255
Cluster 3       0.2443817
```

Hence, both the possibilistic clustering algorithms incur the coincident cluster problem. To avoid it, a further proposal is based on the introduction of a repulsion term among centroids, see [49]. It is implemented in the function `pcmr` of the package **ppclust**. We run it for the dataset `iris`.

```
> iris.pcmr <- pcmr(iris[, 1:4], centers = 3,
+                     numseed = 321)
```

The obtained centroids are not coincident, as it can be observed from below.

```
> iris.pcmr$v
      Sepal.Length Sepal.Width Petal.Length
Cluster 1       5.915065    3.059402     3.468176
Cluster 2       5.887080    2.759138     4.363142
Cluster 3       5.937133    2.940217     4.147449
      Petal.Width
Cluster 1       0.8021035
Cluster 2       1.3964572
Cluster 3       1.3526800
```

In this case, the repulsion constraint avoids the issue but if we take a look at the partition, the results are not satisfactory.

```
> iris.pcmr$cluster
  1   2   3   4   5   6   7   8   9   10  11  12  13
  1   1   1   1   1   1   1   1   1   1   1   1   1   1
 14  15  16  17  18  19  20  21  22  23  24  25  26
  1   1   1   1   1   1   1   1   1   1   1   1   1
 27  28  29  30  31  32  33  34  35  36  37  38  39
```

1	1	1	1	1	1	1	1	1	1	1	1	1	1
40	41	42	43	44	45	46	47	48	49	50	51	52	
1	1	1	1	1	1	1	1	1	1	1	2	2	
53	54	55	56	57	58	59	60	61	62	63	64	65	
2	2	2	2	2	1	2	2	1	3	2	2	1	
66	67	68	69	70	71	72	73	74	75	76	77	78	
2	2	2	2	2	2	3	2	2	2	2	2	2	
79	80	81	82	83	84	85	86	87	88	89	90	91	
2	1	1	1	2	2	2	2	2	2	3	2	2	
92	93	94	95	96	97	98	99	100	101	102	103	104	
2	2	1	2	3	2	2	1	2	2	2	2	2	
105	106	107	108	109	110	111	112	113	114	115	116	117	
2	2	2	2	2	2	2	2	2	2	2	2	2	
118	119	120	121	122	123	124	125	126	127	128	129	130	
2	2	2	2	2	2	2	2	2	2	2	2	2	
131	132	133	134	135	136	137	138	139	140	141	142	143	
2	2	2	2	2	2	2	2	2	2	2	2	2	
144	145	146	147	148	149	150							
2	2	2	2	2	2	2							

Comparing the obtained partition with the known classification, contained in the last column of the dataset `iris` (`Species`), we get the following.

```
> class <- iris[, 5]
> table(class, iris.pcmr$cluster)

class      1 2 3
  setosa    50 0 0
  versicolor 8 38 4
  virginica   0 50 0
```

We can notice that the setosa type units are assigned to Cluster 1. Similarly, all the virginica type units are assigned to Cluster 2. However, most of the remaining flowers (versicolor type) are assigned to Cluster 2.

To obtain better results and avoid the coincident cluster problem, we run the functions implementing hybrid methods. We start by using the function `fpcm` that implements the FPkM clustering algorithm, setting the same input arguments used for `pcmr`.

```
> iris.fpcm <- fpcm(iris[, 1:4], centers = 3,
+                      numseed = 321)
> str(iris.fpcm)
List of 19
 $ u           : num [1:150, 1:3] 0.00107 0.0075 0.00641
   0.01011 0.00177 ...
 ..- attr(*, "dimnames")=List of 2
 ... ..$ : chr [1:150] "1" "2" "3" "4" ...
 ... ..$ : chr [1:3] "Cluster 1" "Cluster 2" "Cluster 3"
 $ t           : num [1:150, 1:3] 0.000205 0.0002 0.000188
   0.000197 0.000201 ...
 $ v           : num [1:3, 1:4] 6.77 5.89 5 3.05 2.76 ...
```

```

... - attr(*, "dimnames")=List of 2
... ..$ : chr [1:3] "Cluster 1" "Cluster 2" "Cluster 3"
... ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
$ v0           : num [1:3, 1:4] 6.7 6.1 5.5 3.3 2.8 4.2
      5.7 4.7 1.4 2.5 ...
... - attr(*, "dimnames")=List of 2
... ..$ : chr [1:3] "Cluster 1" "Cluster 2" "Cluster 3"
... ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
$ d           : num [1:150, 1:3] 24.5 25 26.7 25.4 24.9
      ...
... - attr(*, "dimnames")=List of 2
... ..$ : chr [1:150] "1" "2" "3" "4" ...
... ..$ : chr [1:3] "Cluster 1" "Cluster 2" "Cluster 3"
$ x           : num [1:150, 1:4] 5.1 4.9 4.7 4.6 5 5.4
      4.6 5 4.4 4.9 ...
... - attr(*, "dimnames")=List of 2
... ..$ : chr [1:150] "1" "2" "3" "4" ...
... ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
$ cluster     : Named int [1:150] 3 3 3 3 3 3 3 3 3 3 ...
... - attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
$ csizes      : Named num [1:3] 41 59 50
... - attr(*, "names")= chr [1:3] "1" "2" "3"
$ sumssqrs   : List of 4
... $ between.ss : num 586
... $ within.ss  : Named num [1:3] 28.1 36 15.2
... ... - attr(*, "names")= chr [1:3] "1" "2" "3"
... $ tot.within.ss: num 79.3
... $ tot.ss       : num 665
$ k           : num 3
$ m           : num 2
$ eta         : num 2
$ iter        : num [1:10] 40
$ best.start  : int 1
$ func.val    : num [1:10] 60.5
$ comp.time   : num [1:10] 0.37
$ inpargs     : List of 8
... $ iter.max: int 1000
... $ con.val  : num 1e-09
... $ dmetric  : chr "squeuclidean"
... $ alginitv: chr "kmpp"
... $ alginitu: chr "imembrand"
... $ fixcent: logi FALSE
... $ fixmemb: logi FALSE
... $ stand   : logi FALSE
$ algorithm   : chr "FPCM"
$ call         : language fpcm(x = iris[, 1:4], centers =
      3, numseed = 321)
- attr(*, "class")= chr "ppclust"

```

We get the centroids contained in v.

```

> iris.fpcm$v
      Sepal.Length Sepal.Width Petal.Length
Cluster 1       6.774919   3.052355   5.646483
Cluster 2       5.888873   2.761129   4.363902
Cluster 3       5.003983   3.414050   1.482851
      Petal.Width
Cluster 1       2.0536317

```

```
Cluster 2    1.3972651
Cluster 3    0.2533794
```

The resulting partition consists of three clusters of size 41, 59, and 50, respectively.

```
> iris.fpcm$csize
 1 2 3
41 59 50
```

By comparing the partition with the information provided by `class`, we obtain the following cross-tabulation. The output argument `cluster` is an integer vector containing the cluster labels according to the closest hard partition obtained by considering the fuzzy membership degree matrix ( $u$ ).

```
> table(iris.fpcm$cluster, class)
   class
   setosa versicolor virginica
1      0         3       38
2      0        47       12
3     50         0       0
```

We can note that all the setosa flowers are allocated to Cluster 3. Cluster 1 is mainly composed of virginica flowers, while Cluster 2 of versicolor flowers.

Therefore, in this particular case, the FPkM results are satisfactory. However, as we marked above, the applicability of the FPkM algorithm is often prevented from the constraint on the column-wise sums that must be equal to 1.

```
> all(round(rowSums(iris.fpcm$u)) == 1)
[1] TRUE
> all(colSums(iris.fpcm$t) == 1)
[1] TRUE
```

We then run the function `mfpcm` implementing the MFPkM clustering algorithm.

```
> iris.mfpcm <- mfpcm(iris[, 1:4], centers = 3,
+                         numseed = 321)
```

Results are very similar.

```
> iris.mfpcm$v
          Sepal.Length Sepal.Width Petal.Length
Cluster 1      6.787250   3.056275    5.663409
Cluster 2      5.898733   2.764134    4.381504
```

```

Cluster 3      5.003115    3.415077    1.479617
          Petal.Width
Cluster 1      2.0610002
Cluster 2      1.4051719
Cluster 3      0.2520234
> iris.mfpcm$csize
 1 2 3
40 60 50
> table(iris.mfpcm$cluster, class)
  class
  setosa versicolor virginica
1     0         3        37
2     0         47       13
3    50         0        0

```

Finally, we run the PF $k$ M clustering algorithm. Unlike the other functions of the package **ppclust**, pfcm has two additional input argument: *a* and *b*, the numeric values for the relative importance of the fuzzy term (default: 1) and the relative importance of the possibilistic term of the objective function (default: 1). We use the default values in the current application.

```

> iris.pfcm <- pfcm(iris[, 1:4], centers = 3,
+                      numseed = 321)

```

The following centroids are returned.

```

> iris.pfcm$csize
          Sepal.Length Sepal.Width Petal.Length
Cluster 1      6.623684    3.014813    5.462464
Cluster 2      5.921892    2.788865    4.396931
Cluster 3      5.004632    3.410189    1.484259
          Petal.Width
Cluster 1      1.9919289
Cluster 2      1.4071926
Cluster 3      0.2520773

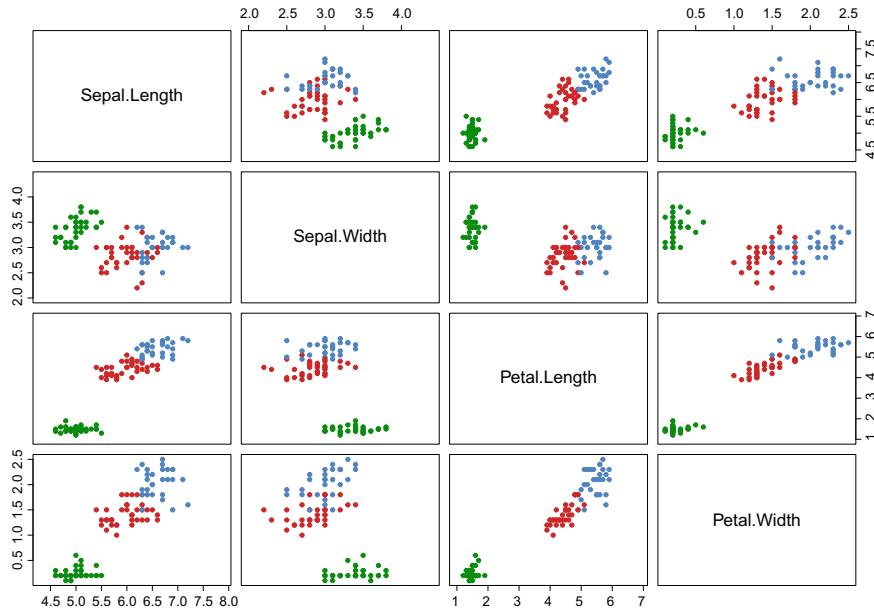
```

The resulting partition is composed of three clusters of size 46, 54, and 50, respectively. Only 14 out of 150 flowers are misclassified.

```

> iris.pfcm$csize
 1 2 3
46 54 50
> table(iris.pfcm$cluster, class)
  class
  setosa versicolor virginica
1     0         5        41
2     0         45        9
3    50         0        0

```



**Fig. 5.22** `iris` data: matrix of scatterplots of the PF $k$ M solution with  $k = 3$  clusters (blue points: cluster 1, red points: cluster 2, green points: cluster 3)

By means of the function `plotcluster` of the package `ppclust`, a matrix of scatterplots for the resulting partition is produced (see Fig. 5.22). Note that the plot is based on the fuzzy membership degrees (default option `mt = "u"`), but, as we already observed, it could also be based on the typicality degrees (`mt = "t"`).

```
> plotcluster(iris.pfcm)
```

Cluster 1 mainly contains virginica type and only five versicolor-type flowers. All the setosa flowers are assigned, also in this case, to Cluster 3, highlighting a perfect recovery. Cluster 2 is mainly composed of versicolor plus nine virginica flowers. By analyzing only the misclassified flowers, we notice that most of them have characteristics that are intermediate between Clusters 1 and 2.

```
> ind1 <- which(iris.pfcm$cluster == 1 &
+                  class == "versicolor")
> ind2 <- which(iris.pfcm$cluster == 2 &
+                  class == "virginica")
> round(iris.pfcm$u[ind1, ], 2)
      Cluster 1 Cluster 2 Cluster 3
51       0.54     0.42     0.04
```

```

53      0.66      0.32      0.02
77      0.50      0.47      0.03
78      0.77      0.21      0.01
87      0.48      0.50      0.03
> round(iris.pfcm$t[ind1, ], 2)
  Cluster 1 Cluster 2 Cluster 3
51      0.38      0.29      0.02
53      0.52      0.31      0.02
77      0.44      0.38      0.02
78      0.69      0.35      0.02
87      0.45      0.42      0.02
> round(iris.pfcm$u[ind2, ], 2)
  Cluster 1 Cluster 2 Cluster 3
102     0.44      0.53      0.02
107     0.21      0.72      0.07
114     0.38      0.59      0.03
120     0.32      0.66      0.03
122     0.32      0.65      0.03
127     0.35      0.63      0.02
128     0.43      0.55      0.02
139     0.29      0.69      0.02
143     0.44      0.53      0.02
> round(iris.pfcm$t[ind2, ], 2)
  Cluster 1 Cluster 2 Cluster 3
102     0.43      0.43      0.02
107     0.14      0.32      0.03
114     0.34      0.41      0.02
120     0.31      0.45      0.02
122     0.33      0.45      0.02
127     0.50      0.60      0.02
128     0.52      0.55      0.02
139     0.44      0.61      0.02
143     0.43      0.43      0.02

```

The above results confirm that the versicolor and the virginica groups have some overlap, but the hybrid strategy performs well in partitioning the flowers.

## References

1. Abelson, R.P., Sermat, V.: Multidimensional scaling of facial expressions. *J. Exp. Psychol. Gen.* **63**, 546–554 (1962)
2. Anderson, E.: The irises of the Gaspe Peninsula. *Bull. Am. Iris Soc.* **59**, 2–5 (1936)
3. Babuska, R., Van der Veen, P.J., Kaymak, U.: Improved covariance estimation for Gustafson-Kessel clustering. In: Proceedings of the 2002 IEEE International Conference on Fuzzy Systems, pp. 1081–1085 (2002)
4. Bagus, F.A.F., Pramana, S.: *advclust*: Object Oriented Advanced Clustering. R package version 0.4 (2016). <https://CRAN.R-project.org/package=advclust>

5. Barni, M., Cappellini, V., Mecocci, A.: Comments on ‘A possibilistic approach to clustering’. *IEEE T. Fuzzy Syst.* **4**, 393–396 (1996)
6. Bezdek, J.C.: Cluster validity with fuzzy sets. *J. Cybern.* **3**, 58–73 (1974)
7. Bezdek, J.C.: *Pattern Recognition with Fuzzy Objective Function Algorithm*. Plenum Press, New York (1981)
8. Campello, R.J.G.B.: A fuzzy extension of the Rand index and other related indexes for clustering and classification assessment. *Pattern Recognit. Lett.* **28**, 833–841 (2007)
9. Campello, R.J.G.B., Hruschka, E.R.: A fuzzy extension of the silhouette width criterion for cluster analysis. *Fuzzy Sets Syst.* **157**, 2858–2875 (2006)
10. Cebezi, Z.: Comparison of internal validity indices for fuzzy clustering. *J. Agr. Inform.* **10**, 1–14 (2019)
11. Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P.A., Lukasik, S., Zak, S.: A complete gradient clustering algorithm for features analysis of X-ray images. In: Pietka, E., Kawa, J. (eds.) *Information Technologies in Biomedicine*, pp. 15–24. Springer-Verlag, Berlin (2010)
12. Chavent, M., Kuentz, V., Labenne, A., Liquet, B., Saracco, J.: PCAmixdata: Multivariate Analysis of Mixed Data. R package version 3.1 (2017). <https://CRAN.R-project.org/package=PCAmixdata>
13. Chou, C.-H., Su, M.-C.: A modified version of the  $K$ -means algorithm with a distance based on cluster symmetry. *IEEE T. Pattern Anal. Mach. Intell.* **23**, 674–680 (2001)
14. Davé, R.N.: Characterization and detection of noise in clustering. *Pattern Recognit. Lett.* **12**, 657–664 (1991)
15. Davé, R.N.: Validating fuzzy partitions obtained through c-shells clustering. *Pattern Recognit. Lett.* **17**, 613–623 (1996)
16. Davé, R.N., Sen, S.: Robust fuzzy clustering of relational data. *IEEE T. Fuzzy Syst.* **10**, 713–727 (2002)
17. de Leeuw, J., Mair, P.: Multidimensional scaling using majorization: SMACOF in R. *J. Stat. Softw.* **31**, 1–30 (2009)
18. Di Lorenzo, P.: usmap: US Maps including Alaska and Hawaii. R package version 0.5.0 (2019). <https://CRAN.R-project.org/package=usmap>
19. Engen, B., Levy, N., Schlossberg, H.: The dimensional analysis of a new series of facial expressions. *J. Exp. Psychol. Gen.* **55**, 454–458 (1958)
20. Ferraro, M.B., Giordani, P.: A toolbox for fuzzy clustering using the R programming language. *Fuzzy Sets Syst.* **279**, 1–16 (2015)
21. Ferraro, M.B., Giordani, P., Serafini, A.: fclust: an R package for fuzzy clustering. *R J.* **11**(1), 205–233 (2019)
22. Fisher, R.A.: The use of multiple measurements in taxonomic problems. *Ann. Eugen.* **7**, 179–188 (1935)
23. Fukuyama, Y., Sugeno, M.: A new method of choosing the number of clusters for the fuzzy  $c$ -means method. In: *Proceedings of the 5th Fuzzy System Symposium*, pp. 247–250 (1989) (in Japanese)
24. Gath, I., Geva, A.B.: Unsupervised optimal fuzzy clustering. *IEEE T. Pattern Anal. Mach. Intell.* **11**, 773–781 (1989)
25. Giordani, P., Ferraro, M.B., Martella, F.: datasetsICR: Datasets from the Book “An Introduction to Clustering with R”, R package version 1.0 (2020). <https://CRAN.R-project.org/package=datasetsICR>
26. Gustafson, D.E., Kessel, W.C.: Fuzzy clustering with a fuzzy covariance matrix. In: *Proceedings of the 1978 IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes*, pp. 761–766 (1979)
27. Hathaway, R.J.: Another interpretation of the EM algorithm for mixture distributions. *Stat. Probab. Lett.* **4**, 53–56 (1986)
28. Henderson, H.V., Velleman, P.F.: Building multiple regression models interactively. *Biometrics* **37**, 391–411 (1981)

29. Kaufman, L., Rousseeuw, P.J.: *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York (1990)
30. Klawonn, F., Chekhtman, V., Janz, E.: Visual inspection of fuzzy clustering results. In: Benitez, J.M., Cordon, O., Hoffmann, F., Roy, R. (eds.) *Advances in Soft Computing - Engineering Design and Manufacturing*, pp. 65–76. Springer, London (2003)
31. Klawonn, F., Höppner, F.: An alternative approach to the fuzzifier in fuzzy clustering to obtain better clustering. In: *Proceedings of Eusflat Conference*, pp. 730–734 (2003)
32. Klawonn, F., Kruse, R., Winkler, R.: Fuzzy clustering: more than just fuzzification. *Fuzzy Sets Syst.* **281**, 272–279 (2015)
33. Krishnapuram, R., Joshi, A., Nasraoui, O., Yi, L.: Low-complexity fuzzy relational clustering algorithms for web mining. *IEEE T. Fuzzy Syst.* **9**, 595–607 (2001)
34. Krishnapuram, R., Keller, J.M.: A possibilistic approach to clustering. *IEEE T. Fuzzy Syst.* **1**, 98–110 (1993)
35. Krishnapuram, R., Keller, J.M.: The possibilistic  $c$ -means algorithm: insights and recommendations. *IEEE T. Fuzzy Syst.* **4**, 385–393 (1996)
36. Kwon, S.H.: Cluster validity index for fuzzy clustering. *Electron. Lett.* **34**, 2176–2177 (1998)
37. Li, R.-P., Mukaidono, M.: A maximum-entropy approach to fuzzy clustering. In: *Proceedings of 1995 IEEE International Conference on Fuzzy Systems*, pp. 2227–2232 (1995)
38. Li, R.-P., Mukaidono, M.: Gaussian clustering method based on maximum-fuzzy-entropy interpretation. *Fuzzy Sets Syst.* **102**, 253–258 (1999)
39. MacQueen, J.: Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, vol. 1, pp. 281–298 (1967)
40. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.: *cluster: Cluster Analysis Basics and Extensions*. R package version 2.1.0 (2019)
41. Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., Leisch, F., Chang, C.-C., Lin, C.-C.: *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071)*, TU Wien. R package version 1.7-3 (2019). <https://CRAN.R-project.org/package=e1071>
42. Pal, N.R., Pal, K., Bezdek, J.C.: A mixed  $c$ -means clustering model. In: *Proceedings of FUZZ-IEEE'97*, pp. 11–21 (1997)
43. Pal, N.R., Pal, K., Keller, J.M., Bezdek, J.C.: A possibilistic fuzzy  $c$ -means clustering algorithm. *IEEE T. Fuzzy Syst.* **13**, 517–530 (2005)
44. R Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna (2020). <https://www.R-project.org>
45. Ruspini, E.H.: Numerical methods for fuzzy clustering. *Inf. Sci.* **2**, 319–350 (1970)
46. Saad, M.F., Alimi, A.M.: Modified fuzzy possibilistic  $c$ -means. In: *Proceedings of the International Multiconference of Engineers and Computer Scientists*, pp. 18–20 (2009)
47. Sarkar, D., Andrews, F., Wright, K., Klepeis, N., Murrell, P.: *lattice: Trellis graphics for R*, R package version 0.20-41 (2020). <https://CRAN.R-project.org/package=lattice>
48. Tang, Y.G., Sun, F.C., Sun, Z.Q.: Improved validation index for fuzzy clustering. In: *American Control Conference*, June 8–10, Portland, OR, USA (2005)
49. Timm, H., Borgelt, C., Döring, C., Kruse, R.: An extension to possibilistic fuzzy cluster analysis. *Fuzzy Sets Syst.* **147**, 3–16 (2004)
50. Wachs, J., Shapira, O., Stern, H.: A method to enhance the ‘possibilistic  $C$ -means with repulsion’ algorithm based on cluster validity index. In: Abraham, A., de Baets, B., Köppen, M., Nickolay, B. (eds.) *Applied Soft Computing Technologies: The Challenge of Complexity. Advances in Soft Computing*, vol. 34. Springer, Berlin (2006)
51. Wang, W., Zhang, Y.: On fuzzy cluster validity indices. *Fuzzy Sets Syst.* **158**, 2095–2117 (2007)
52. Wickham, H.: *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York (2016)

53. Winkler, R., Klawonn, F., Kruse, R.: Fuzzy clustering with polynomial fuzzifier function in connection with  $m$ -estimators. *Appl. Comput. Math.* **10**, 146–163 (2011)
54. Xie, X.L., Beni, G.: A validity measure for fuzzy clustering. *IEEE T. Pattern Anal. Mach. Intell.* **13**, 841–847 (1991)
55. Yang, M.-S., Wu, K.-L.: Unsupervised possibilistic clustering. *Pattern Recognit.* **39**, 5–21 (2006)

## **Part IV**

# **Model-Based Clustering**

# Chapter 6

## Model-Based Clustering



### 6.1 Introduction

As we observed in previous chapters, a wide range of clustering methods does exist based on measures of similarity (dissimilarity/distance) between units and groups of units. These approaches do not assume an underlying statistical model and, for this reason, are often referred to as heuristic methods. In this chapter, we introduce an alternative approach, the so-called model-based clustering, that is explicitly based on a probability model. This approach describes data as being generated by some probabilistic process and the goal of clustering is to recover the parameters of such a process. The clustering algorithm fits a probability model to the observed data (usually by maximum likelihood, with an overfitting penalty), and the quality of the fitted model is evaluated by some likelihood-based criteria. The basic idea of model-based clustering is to approximate the data density by a mixture model. The number of distinct clusters in the data is often assumed to be the number of mixture components and the units are partitioned into clusters according to a pre-specified rule (Bayes rule). Note that, while the goal of finite mixture modeling is typically limited to inference on model parameters, model-based clustering is mostly focused on providing a good partition of data into groups of homogeneous units.

The model-based clustering framework has led to improvements in the application of clustering by making definition and purposes clearer than in the context of heuristic clustering methods. Further, some of the most popular heuristic clustering methods can be viewed as estimation methods for certain known probability models. For example, the  $k$ -Means clustering algorithm has been shown to be closely related to the model-based Gaussian clustering using the equal volume spherical constraint. In other words,  $k$ -Means is an approximate estimation method for a parsimonious model based on simple independent Gaussian distributions. As already observed, model-based clustering has also interesting links with fuzzy clustering methods. Even though in the latter no probabilistic assumptions are made, from a practical point of view, it is quite obvious that such two clustering categories share similar

features. Both of them produce a soft partition of units and the posterior probability may play a role similar to the membership degree [42].

Model-based clustering techniques provide a statistical approach to solve important practical questions that arise in applying clustering methods. An advantage of using a statistical model is that choosing the cluster notion and the number of clusters is less arbitrary with respect to the heuristic methods since the approach includes rigorous formal criteria based on the log-likelihood function penalized by some function of the number of model parameters. Moreover, a probabilistic framework is intuitively appealing to allow for comparisons with other methods and inference on model parameters. Another advantage of model-based approach is that no standardization of the observed variables is needed. For instance, when working with Gaussian distributions with unknown variances, the results will be the same irrespective of whether the variables are standardized or not [62]. This is not so for heuristic cluster methods like  $k$ -Means, where, as we saw, standardization is always an issue.

FMMs date back to the pioneering works in [79, 81] and, although the idea of defining a cluster in terms of a mixture component goes back to Tiedeman [96], the first example of Gaussian mixture model used for clustering is attributed to Wolfe [102]. For a comprehensive survey on the history, theory, and applications of mixture models, the interested reader is referred to [33, 38, 69, 72, 97]. A crucial point for the dissemination of FMMs has surely been the work in [28] on the Expectation-Maximization (EM) algorithm, which makes model parameter estimation possible by means of a maximum likelihood approach. The EM algorithm is investigated in [68].

## 6.2 Finite Mixture Models

FMMs represent a family of very interesting and flexible models, often used to describe heterogeneous populations. For their own nature, they have been widely used for density estimation and, only later, as a formal framework for clustering and classification. Let us start supposing we have data consisting of  $n$  multivariate units  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$  assumed to be independent realizations of a random variable vector  $\mathbf{X} \in \mathbb{R}^p$  coming from a population  $\mathcal{P}$  formed by  $k$  subpopulations  $\mathcal{P}_g$ ,  $g = 1, \dots, k$ , characterized by internal cohesion and external separation. Let  $\pi_g = \Pr(i \in \mathcal{P}_g)$ ,  $0 < \pi_g \leq 1$ ,  $g = 1, \dots, k$ , be the prior probability (also called mixing proportion or weight) that a generic unit comes from the subpopulation  $\mathcal{P}_g$ , where  $\sum_{g=1}^k \pi_g = 1$  clearly holds. The marginal probability density function of  $\mathbf{x}_i$  is described by the following FMM

$$f(\mathbf{x}_i | \Psi) = \sum_{g=1}^k \pi_g f_g(\mathbf{x}_i | \boldsymbol{\theta}_g), \quad (6.1)$$

where  $\Psi = (\pi_1, \dots, \pi_{k-1}, \theta_1, \dots, \theta_k)$  represents the overall parameter vector. Note that the parameter  $\pi_k$  is not included in  $\Psi$  since it can be computed by  $\pi_k = 1 - \sum_{g=1}^{k-1} \pi_g$ . In this context, the internal cohesion (homogeneity) and the external separation are usually summarized through a unique choice of the parametric density form, i.e.,  $f_g(\cdot) = f(\cdot)$ , where each subpopulation is characterized by a parameter vector  $\theta_g$ ,  $g = 1, \dots, k$ , which can vary among different subpopulations (entirely or partially). We will refer to the subpopulation  $\mathcal{P}_g$  or to the density  $f_g(\cdot)$  with the term component (also known as kernel density in the literature). The interpretation of a component as a cluster is straightforward: each cluster has its density  $f(\mathbf{x}_i | \theta_g)$ . Then, the assignment of units to clusters is made through the Maximum A Posteriori (MAP) rule, which allocates each unit to its most likely cluster (i.e., to the one with the highest posterior probability that the unit comes from that cluster). If more than one posterior probability is maximum, randomization can be used [69]. Note that extensions of model-based clustering where each unit potentially belongs to more than one cluster have been proposed in the literature (for more details, see [43, 85]).

### 6.3 Parameter Estimation

A commonly used approach for estimating the parameters of a FMM is Maximum Likelihood (ML). As it is clear, maximizing the likelihood or the log-likelihood function is equivalent but it is often convenient to work with the latter. The general log-likelihood function for a sample from (6.1) has the form

$$\ell(\Psi) = \sum_{i=1}^n \log \left[ \sum_{g=1}^k \pi_g f(\mathbf{x}_i | \theta_g) \right]. \quad (6.2)$$

The ML Estimate (MLE) of  $\Psi$  can be derived by looking for the zeros of the first derivative of the (log-)likelihood function, under the constraint  $\sum_{g=1}^k \pi_g = 1$ . However, even in simple mixture models, this log-likelihood rarely leads to closed-form solutions for the ML estimates. In general, we get a set of non-linear equations that must be solved in an iterative manner. The EM algorithm, introduced in [28], is one of the most used tools for inferring mixture models. A brief history of the EM algorithm can be found in [67]. Specifically, the EM algorithm is a fixed-point iterative method that locally maximizes the likelihood (or log-likelihood) function in an efficient way in presence of “incomplete” data. Thus, the mixture model must be conveniently formulated by assuming the existence of missing data and posing the mixture model into an incomplete data problem. In general, the EM algorithm is easily implemented and numerically stable and, under fairly general conditions, it has reliable global convergence. However, like Newton-Raphson algorithms, there is no guarantee that the convergence will be to a global maximum when the likelihood (or log-likelihood) function has multiple local maxima [68]. As a consequence, convergence will strongly depend on starting values. In Sect. 6.3.3, we will describe

how this problem can be addressed. The description of the EM algorithm for FMMs is given in the next section, while two well-known examples of such algorithm will be described in Sect. 6.3.2.

### 6.3.1 EM Algorithm for Finite Mixture Models

To formulate the FMM in an incomplete data framework, the assumption is that each unit is associated to a label, denoting the mixture component the unit belongs to. Specifically, the indicator variable vector  $\mathbf{z}_i = (z_{i1}, \dots, z_{ik})$ ,  $z_{ig} \in \{0, 1\}$ ,  $\sum_{g=1}^k z_{ig} = 1$ , is defined as

$$z_{ig} = \begin{cases} 1 & \text{if } i \in \mathcal{P}_g, \\ 0 & \text{otherwise,} \end{cases} \quad (6.3)$$

$g = 1, \dots, k$ . In other words, the EM algorithm assumes that the observed data, say  $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ , are incomplete since we have no information on the indicator vectors  $\mathbf{z} = (\mathbf{z}_1, \dots, \mathbf{z}_n)$ . Thus, the “complete” data  $\mathbf{y} = (\mathbf{x}, \mathbf{z})$  are thought of as split into two parts: the observed  $\mathbf{x}$  and the unobserved part  $\mathbf{z}$ . The corresponding complete log-likelihood function is defined as

$$\ell_c(\Psi) = \sum_{i=1}^n \log [f(\mathbf{y}_i \mid \Psi)] = \sum_{i=1}^n \{\log [f(\mathbf{x}_i \mid \Psi)] + \log [f(\mathbf{z}_i \mid \Psi)]\}, \quad (6.4)$$

where the indicator variables  $(\mathbf{z}_1, \dots, \mathbf{z}_n)$  are assumed to be independent realizations of a multinomial random variable  $\mathbf{Z}$  with probabilities  $\pi_1, \dots, \pi_k$ . Consequently, the complete data log-likelihood function becomes

$$\ell_c(\Psi) = \sum_{i=1}^n \sum_{g=1}^k \log [(f(\mathbf{x}_i \mid \theta_g) \pi_g)^{z_{ig}}] = \sum_{i=1}^n \sum_{g=1}^k z_{ig} \{\log [f(\mathbf{x}_i \mid \theta_g)] + \log (\pi_g)\}. \quad (6.5)$$

As previously specified, the EM algorithm is an iterative algorithm: it starts from some initial estimate of the parameter vector and then proceeds to iteratively update parameter estimates until convergence is obtained. Each iteration consists of the E-step and the M-step. In the  $(r+1)$ -th iteration of the E-step, the expected value of the complete data log-likelihood function,  $\ell_c(\Psi)$ , conditional on the observed data  $\mathbf{x}$  and the current parameter estimate  $\hat{\Psi}^{(r)} = (\hat{\pi}_1^{(r)}, \dots, \hat{\pi}_{k-1}^{(r)}, \hat{\theta}_1^{(r)}, \dots, \hat{\theta}_k^{(r)})$  is computed:

$$Q(\Psi \mid \hat{\Psi}^{(r)}) = E_{\hat{\Psi}^{(r)}} \{\ell_c(\Psi) \mid \mathbf{x}\}. \quad (6.6)$$

Since it is linear in the missing variables, the E-step reduces to the computation of

$$w_{ig}^{(r+1)} = \text{E}_{\hat{\Psi}^{(r)}} \{ Z_{ig} \mid \mathbf{x}_i \} = \Pr \left\{ Z_{ig} = 1 \mid \mathbf{x}_i, \hat{\Psi}^{(r)} \right\}, \quad (6.7)$$

which represents the posterior probability that the  $i$ -th unit comes from the subpopulation  $\mathcal{P}_g$ ,  $i = 1, \dots, n$ ,  $g = 1, \dots, k$ , computed by using the parameter estimates obtained at the previous step. By directly applying the Bayes rule, it follows that

$$w_{ig}^{(r+1)} = \frac{\hat{\pi}_g^{(r)} f(\mathbf{x}_i \mid \hat{\theta}_g^{(r)})}{\sum_{g'=1}^k \hat{\pi}_{g'}^{(r)} f(\mathbf{x}_i \mid \hat{\theta}_{g'}^{(r)})}. \quad (6.8)$$

On the other side, the  $(r + 1)$ -th iteration of the M-step updates the parameter estimate  $\hat{\Psi}^{(r+1)}$  by maximizing the (conditional) expected value of the complete log-likelihood function  $Q(\Psi \mid \hat{\Psi}^{(r)})$  with respect to  $\Psi$ , that is,

$$\hat{\Psi}^{(r+1)} = \arg \max_{\Psi} Q(\Psi \mid \hat{\Psi}^{(r)}). \quad (6.9)$$

In practice, we look for roots of the following equation:

$$\frac{\partial Q(\Psi \mid \hat{\Psi}^{(r)})}{\partial \Psi} = 0. \quad (6.10)$$

Note that  $\hat{\Psi}^{(r+1)}$  may not necessarily have a closed-form; in such a case,  $Q(\Psi \mid \hat{\Psi}^{(r)})$  should be maximized numerically. A peculiar property of the EM algorithm is that the observed data (log-)likelihood function does not decrease between successive iterations. In particular, in [103], it has been shown that the sequence of likelihood values  $\{L(\hat{\Psi}^{(r)})\}_{r=1,2,\dots}$  converges monotonically to some stationary value  $L^*$  of the likelihood under quite general conditions. Thus, from an initial solution  $\hat{\Psi}^{(0)}$ , the E- and M-steps are repeatedly alternated until convergence is achieved, which may be determined by using a suitable stopping criterion which will be introduced in Sect. 6.3.4.

### 6.3.2 EM-Type Algorithms

In this section, we describe in detail two of the most used extensions of the EM algorithm, namely, the Stochastic EM algorithm (SEM, [17]) and the Classification EM algorithm (CEM, [19]). Other modifications of the EM algorithm have been proposed in the literature such as the Generalized EM algorithm (GEM, [28]) and its subclasses: the Expectation Conditional Maximization (ECM, [77]), the Expectation Conditional Maximization Either (ECME, [61]), the Alternating ECM (AECM,

[78]), and the Monte Carlo EM algorithm (MCEM, [99]). The interested reader can refer to [68] for an overall overview.

### 6.3.2.1 SEM Algorithm

The SEM algorithm is a stochastic version of the EM incorporating a Stochastic (S-)step between the E- and M-steps. Specifically, the S-step generates a complete sample  $\mathbf{y}^{pse} = (\mathbf{x}, \mathbf{z}^{pse})$  by drawing  $\mathbf{z}^{pse}$  from the conditional density of  $\mathbf{z}$  given  $\mathbf{x}$  and the current estimates  $\hat{\Psi}^{(r)}$ , namely,  $f(\mathbf{z}|\mathbf{x}, \hat{\Psi}^{(r)})$ . Then, it updates  $\hat{\Psi}^{(r)}$  on the basis of the pseudo-complete sample  $\mathbf{y}^{pse} = (\mathbf{x}, \mathbf{z}^{pse})$ .

By using random draws at each iteration, the SEM algorithm gives an answer to the well-known limitations of the EM: the strong dependence on initial values and the slow convergence which can occur when the mixture components are not well separated.

Starting from an initial parameter  $\Psi^{(0)}$ , an iteration of the SEM algorithm consists of the following three steps:

- E-step: The posterior probabilities  $w_{ig}$ ,  $i = 1, \dots, n$ ,  $g = 1, \dots, k$ , are calculated as in the standard E-step.
- S-step: Pseudo-values  $\mathbf{z}_i^{pse}$  are simulated from a multinomial distribution with probabilities  $w_{ig}$ ,  $g = 1, \dots, k$ .
- M-step:  $\hat{\Psi}$  estimates are updated as in a standard M-step replacing the  $w_{ig}$  by the simulated  $\mathbf{z}_i^{pse}$  in the likelihood equations.

Notice that the SEM algorithm does not converge point-wise: the process generated by the SEM is a Markov chain whose stationary distribution is concentrated around the ML parameter estimator. A natural estimate  $\hat{\Psi}^{(r)}$  from a SEM sequence,  $r = 1, \dots$ , is the mean of the iterated values, typically obtained after a burn-in period [17]. An alternative estimate is to consider the parameter value leading to the highest likelihood in a SEM sequence. In practice, both point-wise estimators perform similarly.

The theoretical convergence properties of SEM algorithms are difficult to assess since they involve the study of the ergodicity of the Markov chain and the existence of the corresponding stationary distribution. Under standard regularity conditions, weak convergence to a local maximum can be proved, for details see [29]. However, computational studies show that the SEM algorithm is even better than the EM algorithm in several cases, such as censored data [23] and mixture models [16]. A drawback of this procedure is that it requires thousands of simulations and the computational burden could be, again, very high.

### 6.3.2.2 CEM Algorithm

The CEM algorithm incorporates a classification step between the E- and the M-steps of the EM algorithm. Starting from an initial parameter vector  $\Psi^{(0)}$ , an iteration of the CEM consists of the following three steps:

- E-step: The conditional probabilities  $w_{ig}$ ,  $i = 1, \dots, n$ ,  $g = 1, \dots, k$ , are calculated as in the standard E-step.
- C-step: A partition  $P = (P_1, \dots, P_k)$  of  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$  is obtained by assigning each unit to the component maximizing the posterior probability  $w_{ig}$ ,  $g = 1, \dots, k$ , i.e., adopting a MAP approach.
- M-step:  $\hat{\Psi}$  estimates are updated using the cluster  $P_g$  as a sub-sample for the parameters specific to the  $g$ -th mixture component ( $g = 1, \dots, k$ ).

In other words, CEM maximizes a complete data log-likelihood where the missing component indicator vector for each sample point is estimated and included in the dataset. As a consequence, the CEM algorithm is not expected to converge to the ML estimate of  $\Psi$  and may yield inconsistent estimates of the parameters, especially when the mixture components are overlapping or are in disparate proportions [69].

Notice that the SEM algorithm can be thought of as a stochastic version of the CEM as well as of the EM; in fact, it appears to be a natural stochastic version of both algorithms though they are designed to optimize different criteria: the log-likelihood function for the EM algorithm and the classification log-likelihood function for CEM. Moreover, the two algorithms may lead to different partitions. The convex hulls of the clusters generated by the CEM are disjoint whereas the clusters generated by the SEM are generally overlapping. From this point of view, it turns out that the sequence of mixture estimates obtained via the SEM is closer to the EM estimates when compared to the CEM ones.

### 6.3.3 Initializing the EM Algorithm

Choosing initial values for the parameters of an FMM is crucial for finding MLEs through the EM algorithm since the (log-)likelihood surface has often multiple local maxima. The cost per iteration is generally low, but the number of iterations needed for convergence may be large making the EM algorithm relatively slow. Thus, it is clear that a good initialization strategy may lead to attain a potential global maximum in fewer iterations. The literature is rich in proposals for finding reasonable starting values; however, no strategy uniformly outperforms the others. Here, we list only the most common and better performing (at least to our knowledge) strategies.

An easy way to initialize the EM algorithm is based on the use of several random starts, as we already saw in the previous chapters. However, this strategy has a high chance of providing bad starting points. The use of several random starts may help to reach the global maximum even though it can be time-consuming, and thus impracticable for high-dimensional data, with no guarantee that the global maximum

is found [35]. In fact, the successful search for the global maximum depends not only on the number of random starts but also on both the complexity of the function being optimized and the procedure for generating the random starting points.

A different way to proceed for initializing the EM algorithm consists of starting from the solution obtained by other clustering algorithms belonging to the standard approach (Part II). The most commonly used method is the Model-Based Hierarchical Agglomerative Clustering (MBHAC) introduced in [6] in the context of Gaussian mixtures. Specifically, “hierarchical” clusters are obtained by recursively merging the two clusters that provide the smallest decrease in the classification likelihood. It can be shown that the MBHAC approach works well when the components are well separated and the datasets have moderate size [64]. In [50], several methods for choosing initial values for the EM algorithm in the case of finite mixtures with Gaussian and Poisson kernels are compared and some extensions of existing methods are proposed.

Other interesting initialization strategies are proposed in [11]. They are based on the solutions given by the CEM and the SEM algorithms and from solutions obtained after short runs of the EM algorithm (smallEM or emEM). In particular, the latter strategy is often preferred, for details, see [11]. An extension of the smallEM is the RndEM procedure [63], where the smallEM stage is replaced by choosing multiple starting points and evaluating the log-likelihood at these values without running any EM iterations. A second proposal introduced in [63] consists of a staged approach based on finding a large number of local modes of the dataset and then choosing representatives from the most widely separated ones. It can be shown that the latter is very time-consuming for high-dimensional data, and it has worse performance than both the smallEM and RndEM algorithms. Recently, a strategy for initializing the model parameters in Gaussian mixture models has been proposed in [76] and simple transformation-based methods to refine the EM initialization step derived from MBHAC have been proposed in [91].

Since none of the aforementioned strategies can be regarded as the best one, a usual good strategy consists in exploring several strategies and then choosing the solution with the highest log-likelihood value.

### 6.3.4 Stopping Criteria for the EM Algorithm

The issue of choosing a suitable stopping rule has been dealt with by several authors. They are modifications of those for fuzzy clustering (see Sect. 5.2) specifically tuned for the model-based framework. The most used stopping criteria consist in declaring convergence when the relative increase in the observed (log-)likelihood function or the difference between two successive estimates of  $\hat{\Psi}$  is lower than a pre-specified threshold  $\varepsilon > 0$  [59], that is,

$$\frac{l(\hat{\Psi}^{(r+1)}) - l(\hat{\Psi}^{(r)})}{|l(\hat{\Psi}^{(r)})|} < \varepsilon \quad (6.11)$$

or

$$\|\hat{\Psi}^{(r+1)} - \hat{\Psi}^{(r)}\| < \varepsilon. \quad (6.12)$$

Clearly, the smaller is  $\varepsilon$ , the more severe the criterion. In general, criteria based on the relative change of the (log-)likelihood indicate lack of progress rather than actual convergence [59]. See [71, 92] for a dedicated study on the effect of stopping too early, [12, 82] for stopping rules based on Aitken's acceleration scheme and [57, 58, 83] for stopping rules based on the gradient function. Clearly, when the (log-)likelihood is trapped in a flat area, any aforementioned criteria is likely to work appropriately.

## 6.4 On Identifiability of Finite Mixture Models

Even though we do not want to deeply deal with parameter identifiability, it is worth spending few words on this topic and refer to [38, 69] for a general discussion on identifiability. Starting with the pioneering work in [94], several authors have looked into identifiability for finite mixture distributions [4, 22, 26, 86, 95, 104].

In the finite mixture framework, identifiability can be defined as follows. Let

$$f(\mathbf{x}_i | \Psi) = \sum_{g=1}^k \pi_g f_g(\mathbf{x}_i | \boldsymbol{\theta}_g) \quad \text{and} \quad f(\mathbf{x}_i | \Psi^*) = \sum_{g=1}^{k^*} \pi_g^* f_g(\mathbf{x}_i | \boldsymbol{\theta}_g^*) \quad (6.13)$$

be two FMMs from the same parametric family. A FMM is said identifiable with respect to  $\Psi$  when

$$f(\mathbf{x}_i | \Psi) \equiv f(\mathbf{x}_i | \Psi^*) \quad (6.14)$$

holds if and only if  $k = k^*$  and there exists a permutation of the mixture component indices such that

$$\pi_g = \pi_g^* \quad \text{and} \quad f_g(\mathbf{x}_i | \boldsymbol{\theta}_g) = f_g(\mathbf{x}_i | \boldsymbol{\theta}_g^*), \quad g = 1, \dots, k. \quad (6.15)$$

In general, three different meanings of nonidentifiability may be distinguished for FMMs [38]. Firstly, nonidentifiability due to invariance to relabeling the mixture components [86]. This is essentially what we already saw on cluster label switching in the previous chapters. Such an identifiability problem is related to the fact that, for the general finite mixture distribution with  $k$  components defined in (6.1), there exist  $k!$  equivalent ways of arranging the components. From a practical point of view, this is not a severe identifiability issue and it is usually treated in a formal manner by

imposing constraints on the parameter space so that no different parameters generate the same distribution and conditions in (6.14) and (6.15) are fulfilled. A very common constraint consists in ordering the mixture components according to the prior probabilities as

$$\pi_1 \leq \pi_2 \leq \cdots \leq \pi_k, \quad (6.16)$$

after model parameters have been estimated.

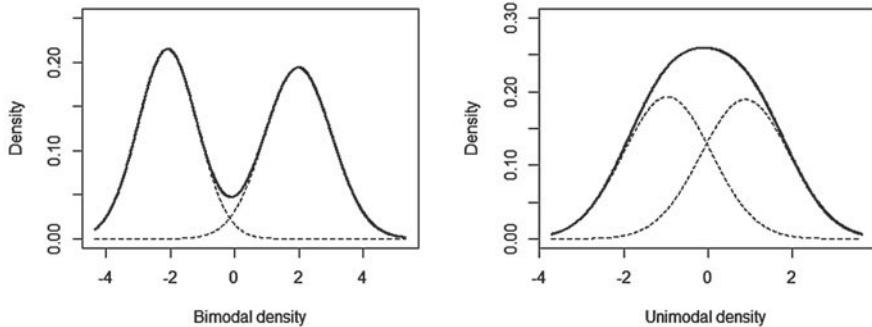
A further identifiability problem is nonidentifiability due to potential overfitting [26]. It can be shown that any mixture with  $k - 1$  components defines a nonidentifiability subset in the larger parameter space, say  $\mathbf{H}(k)$  corresponding to mixtures with  $k$  components, where either one component is empty or two components are associated to the same parameter value. This identifiability is achieved, for example, by constraining the parameter space  $\mathbf{H}(k)$  in such a way that the density in (6.1) is a mixture of  $k$  distinct, nonempty components (i.e.,  $\pi_g > 0$ ,  $g = 1, \dots, k$ ).

Note that finite mixture distributions may remain unidentifiable, even if a formal identifiability constraint excludes nonidentifiability. Examples of nonidentifiable families are mixtures of uniform or binomial distributions. The last type of nonidentifiability has been defined in [104] as a generic property of a certain class of mixture distributions. Mixtures of negative binomial distributions and several multivariate families, such as the multivariate Gaussian mixtures, are generically identifiable and discrete mixtures need not be identifiable [104]. See [38] for specific details on the topic.

## 6.5 Model Selection

When discussing the process of parameter estimation, we assumed that the number  $k$  of mixture components is fixed; however, in practice, it is usually unknown and, thus, it must be estimated as well. It should be noted that the choice of the number of components may not be a fundamental issue when mixture models are used to provide a semi-parametric estimate of an unknown distribution function; in this case, overestimation of the number of components is not an issue. On the other hand, if the goal is to use the mixture model to provide a partition for the observed data, the choice of the number of components must be carefully and critically addressed. In the framework of model-based clustering, the problem of determining the number of clusters and choosing an appropriate clustering model becomes a problem of model selection. Clearly, the two aspects are linked to each other: if a simpler model is selected, more clusters may be needed to provide a good representation of the data; vice versa, if a more complex model is selected, fewer clusters may be enough.

The problem of overfitting arises from intrinsic nonidentifiability of a mixture model so that a distribution which can be well approximated by a mixture of  $k$  component densities can also be well approximated by a mixture of  $k_0 \geq k$  component densities, in the sense that the two mixture distributions can be empirically indistinguishable. Thus, we can only get a lower bound for the number of components;



**Fig. 6.1** Examples of mixtures of two univariate Gaussian densities. On the left:  $\pi_1 = \pi_2 = 0.5$ ,  $\mu_1 = -2$  and  $\mu_2 = 2$ ; on the right:  $\pi_1 = \pi_2 = 0.5$ ,  $\mu_1 = -1$  and  $\mu_2 = 1$

in this respect, the right question to ask is not “how many components are there in the mixture model?” but “what is the smallest number of components in the mixture needed to make the model appropriate for the observed data?”.

A wide literature is devoted to the issue of choosing  $k$ . One of the oldest methods, mainly based on intuition, consists in estimating the number of components by analyzing the number of modes; in [97], some inferential procedures for assessing the number of modes are described. See also [5]. However, the obvious drawback of this method is that if the component densities are not sufficiently far apart, the mixture distribution would still be unimodal and estimating the number of components by the number of modes would fail, especially in a clustering context. Note, however, that the practical interest could lie in finding components that correspond to separate modes, so that true separation occurs.

We illustrate the distinction between modes and components through a simple example considering a mixture of two univariate Gaussian densities. Figure 6.1 shows mixtures of two Gaussians with equal weights. The means are four standard deviations apart on the left and two standard deviations apart on the right, being the standard deviations equal to one. In both figures, the dotted lines represent the component densities and the solid line is the mixture density. As it can be seen, in the left figure, we have a bimodal distribution, while on the right the mixture distribution results unimodal. Thus, in the latter case, we cannot infer about the number of components using the number of modes. Moreover, visually inspecting the modes in a multivariate context is overly difficult. Alternative and more popular approaches are based on the so-called information criteria:  $k$  is chosen by minimizing two times the negative log-likelihood function penalized by some penalty term that is intended to reflect its complexity. Examples are the Akaike Information Criterion (AIC, [2]) which takes the form

$$\text{AIC} = -2\ell(\hat{\Psi}) + 2\nu, \quad (6.17)$$

where  $\nu = \dim(\Psi)$  is the number of free parameters to be estimated and  $\hat{\Psi}$  is the MLE for  $\Psi$ ; the Bayesian Information Criterion (BIC, [87]) given by

$$\text{BIC} = -2\ell(\hat{\Psi}) + v \log(n); \quad (6.18)$$

the Integrated Completed Likelihood (ICL, [10]) is expressed as

$$\text{ICL} = -2\ell_{ic}(\hat{\Psi}) + v \log(n), \quad (6.19)$$

where  $\ell_{ic}(\cdot) = \log(f(\mathbf{x}, \hat{\mathbf{z}}|\hat{\Psi}))$  with  $f(\mathbf{x}, \hat{\mathbf{z}}|\hat{\Psi})$  being the complete likelihood obtained by substituting  $\mathbf{z}$  for  $\hat{\mathbf{z}}$  and  $\Psi$  for  $\hat{\Psi}$ . Note that ICL can also be calculated via the following approximation:

$$\text{ICL} = \text{BIC} - \sum_{g=1}^k \sum_{i=1}^n w_{ig} \log(w_{ig}). \quad (6.20)$$

Finally, the Normalized Entropy Criterion (NEC, [21]) is defined by

$$\text{NEC} = \begin{cases} \frac{-\sum_{g=1}^k \sum_{i=1}^n w_{ig} \log(w_{ig})}{\ell(\hat{\Psi}_k) - \ell(\hat{\Psi}_1)} & \text{if } k > 1, \\ 1 & \text{otherwise,} \end{cases} \quad (6.21)$$

where  $\hat{\Psi}_k$  denotes the estimated parameter vector setting  $k$  clusters. From (6.20), it is clear that the number of clusters selected by the ICL tends to be generally smaller than the number of clusters selected by the BIC due to the additional entropy term that reflects the uncertainty in the posterior classification. As a result, ICL tends to prefer models that produce clearly separated clusters and to choose a number of clusters equal or smaller than the one selected by BIC. In general, ICL and NEC provide more parsimonious solutions when compared to the others and the same comment holds for BIC with respect to AIC. Note that the performance of such information criteria has been studied and compared looking at different aspects. For instance, it can be proved that AIC tends to asymptotically not underestimate the “correct” number of clusters [21, 93]. Moreover, several studies suggest the use of BIC (see, for example, [69]). A method for combining the point of view underlying BIC and ICL to achieve the best of both is proposed in [7]. Although information criteria are easy to be implemented, it may be difficult to compare different model fits. Usually, differences in BIC greater than 10 are considered strong evidence to support a specific model [51]. However, it is unclear how this value should be calibrated as a function of  $n$  and  $p$ .

Other alternative information criteria are available in the literature but they are out of the scope of this book. In addition to information criteria, model selection can be formulated within a hypothesis testing problem. An obvious way of approaching the problem of testing the smallest value of the number of components in a mixture model is to use a Likelihood Ratio Test (LRT). We consider testing the following null and alternative hypotheses:

$$\begin{cases} H_0 : k = k_0, \\ H_1 : k = k_1, \end{cases} \quad (6.22)$$

for some  $k_1 > k_0$ , usually  $k_1 = k_0 + 1$ . Let  $\hat{\Psi}_{k_i}$  be the MLE of  $\Psi$  under  $H_i$  ( $i = 0, 1$ ), and let  $\lambda = \frac{L(\hat{\Psi}_{k_0})}{L(\hat{\Psi}_{k_1})}$  be the corresponding likelihood ratio. In the context of mixture models, it is well known that regularity conditions ( $\Theta_0 \cap \Theta_1 = \emptyset$  and  $\Theta = \Theta_0 \cup \Theta_1$ , where  $\Theta$  is the parameter space and  $\Theta_0$  and  $\Theta_1$  are the parameter spaces under  $H_0$  and  $H_1$ , respectively) do not hold, and therefore it is no longer guaranteed that  $-2 \log \lambda$  has under  $H_0$  an asymptotic  $\chi^2$  distribution with degrees of freedom equal to the difference between the number of parameters under the alternative and the null hypotheses, that is, the Wald theorem does not hold anymore [34]. In fact, in this context, the null hypothesis  $H_0$  can be achieved from the alternative hypothesis  $H_1$  or more formally the null hypothesis lies on the boundary of the alternative by simply considering an empty component. To explain this, let us consider the null hypothesis of one Gaussian component (with the usual notation,  $\mu$  and  $\sigma$  denote the mean and standard deviation, respectively)

$$H_0 : f(x) = \varphi(x; \mu, \sigma^2) \quad (6.23)$$

versus the alternative hypothesis of two Gaussian components

$$H_1 : f(x) = \pi \varphi(x; \mu_1, \sigma_1^2) + (1 - \pi) \varphi(x; \mu_2, \sigma_2^2). \quad (6.24)$$

The model under  $H_0$  can be obtained by setting  $\pi = 0$  or  $\pi = 1$ , or any  $\pi$  with  $\mu_1 = \mu_2$  and  $\sigma_1 = \sigma_2$ . Thus, the null hypothesis is nested within the alternative, i.e.,  $\Theta_0 \cap \Theta_1 \neq \emptyset$ . In other words,  $\Theta_0$  is on the boundary of  $\Theta_1$ .

A bootstrap approach, where the null distribution of  $-2 \log \lambda$  is estimated by fitting the mixture model to  $B$  samples drawn under the null hypothesis is proposed in [66]. The procedure can be summarized as follows:

1. Using the observed data, estimate a mixture model with a number of components equal to  $k_0$  (i.e., the number of components assumed under  $H_0$ ) and obtain the MLE  $\hat{\Psi}_{k_0}$ .
2. Generate  $B$  bootstrap samples from such a mixture model.
3. For each bootstrap sample, estimate mixture models for  $k = k_0$  and  $k_0 + 1$  components and compute the  $B$  test statistics  $-2 \log(\lambda_b)$ ,  $b = 1, \dots, B$ . These replicated values of  $-2 \log \lambda$  provide an assessment of the corresponding bootstrap distribution under  $H_0$ .

This approach can be used to compare the LRT value on the original sample with a specific quantile of the bootstrap distribution, corresponding to a given level  $\alpha$ . The test rejects  $H_0$  if the observed value of the statistic  $-2 \log \lambda$  is greater than the  $j$ -th ordered value in the  $B$  bootstrap replications, which can be used as an estimate of the quantile of order  $j/(B+1)$ . A bootstrap-based approximation to the  $p$ -value may be computed as

$$p\text{-value} \approx \frac{1 + \sum_{b=1}^B I(-2 \log \lambda_b \geq -2 \log \lambda_{obs})}{(B+1)}, \quad (6.25)$$

where  $-2 \log \lambda_{obs}$  is the test statistic computed on the observed sample and  $I(\cdot)$  denotes the indicator function, equal to 1 if the argument is true and 0 otherwise [66]. The number of bootstrap samples,  $B$ , must be very large to reduce the sensibility of the EM algorithm to the starting points:  $B = 1000$  can be considered a good number of bootstrap samples. Results in [66, 70] discuss the performance of the bootstrap method in the case of limited dimension (few data and two or three clusters). Simulation studies in [70] illustrate that there is a tendency of the bootstrap approach to underestimate the upper percentiles of the null distribution of  $-2 \log \lambda$ , and hence a tendency toward the null hypothesis (conservative test).

## 6.6 Gaussian Mixture Models for Continuous Data

The Gaussian Mixture Model (GMM) is by far the most popular mixture model in this setting since estimation methods are well established and component distributions are deeply analyzed; thus, interpretation of results is simplified. In such a case, the assumption is that a Gaussian (or Normal) distribution describes each mixture component, that is,

$$\mathbf{x}_i | z_{ig} = 1 \sim \text{MVN}_p(\cdot | \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g), \quad i = 1, \dots, n, g = 1, \dots, k, \quad (6.26)$$

where MVN stands for MultiVariate Normal. In other words, given the mixture component label,  $\mathbf{X}_i$  is a multivariate Gaussian random variable with mean vector  $\boldsymbol{\mu}_g$  and covariance matrix  $\boldsymbol{\Sigma}_g$ . The generic  $g$ -th cluster is ellipsoidal, centered at  $\boldsymbol{\mu}_g$  and with geometric features determined by  $\boldsymbol{\Sigma}_g$ . The marginal density function in (6.1) becomes

$$f(\mathbf{x}_i | \Psi) = \sum_{g=1}^k \pi_g \phi(\mathbf{x}_i | \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g), \quad (6.27)$$

where  $\phi(\cdot)$  represents the density function of a  $p$ -variate Gaussian random variable and the overall parameter vector is

$$\Psi = \{\pi_1, \dots, \pi_{k-1}, \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_k\} \in \mathbb{R}^{k[1+p+(p^2+p)/2]-1}, \quad (6.28)$$

subject to  $\sum_{g=1}^k \pi_g = 1$ ,  $0 < \pi_g \leq 1$ ,  $\|\boldsymbol{\mu}_g\| < \infty$  and  $0 < |\boldsymbol{\Sigma}_g| < \infty$ ,  $g = 1, \dots, k$ .

In the  $(r+1)$ -th iteration of the E-step, (6.8) becomes

$$w_{ig}^{(r+1)} = \frac{\hat{\pi}_g^{(r)} \phi(\mathbf{x}_i | \hat{\mu}_g^{(r)}, \hat{\Sigma}_g^{(r)})}{\sum_{g'=1}^k \hat{\pi}_{g'}^{(r)} \phi(\mathbf{x}_i | \hat{\mu}_{g'}^{(r)}, \hat{\Sigma}_{g'}^{(r)})}, \quad (6.29)$$

while, in the M-step, the maximization of the expected value of the complete log-likelihood function  $\mathcal{Q}(\Psi | \hat{\Psi}^{(r)})$  leads to the following closed-form updates (conditional on  $w_{ig}^{(r+1)}$ ) for model parameters  $\pi_g$ ,  $\mu_g$ , and  $\Sigma_g$ :

$$\hat{\pi}_g^{(r+1)} = \frac{\sum_{i=1}^n w_{ig}^{(r+1)}}{n}, \quad (6.30)$$

$$\hat{\mu}_g^{(r+1)} = \frac{\sum_{i=1}^n w_{ig}^{(r+1)} \mathbf{x}_i}{\sum_{i=1}^n w_{ig}^{(r+1)}} \quad (6.31)$$

and

$$\hat{\Sigma}_g^{(r+1)} = \frac{\sum_{i=1}^n w_{ig}^{(r+1)} (\mathbf{x}_i - \hat{\mu}_g^{(r+1)}) (\mathbf{x}_i - \hat{\mu}_g^{(r+1)})'}{\sum_{i=1}^n w_{ig}^{(r+1)}}, \quad (6.32)$$

$g = 1, \dots, k$ . The posterior probabilities of component membership can be used, at convergence, to cluster unit by the MAP criterion

$$\hat{z}_{ig} = \begin{cases} 1 & \text{if } g = \arg \max_{g'=1,\dots,k} w_{ig'}, \\ 0 & \text{otherwise,} \end{cases} \quad (6.33)$$

$i = 1, \dots, n$ . The update in (6.32) refers to an unstructured component-specific covariance matrix  $\Sigma_g$  with a number of free parameters equal to  $(p^2 + p)/2$ ; since this can be quite a large number, to avoid convergence difficulties or identifiability issues, parsimonious GMMs can be considered. Seminal parsimonious GMMs based on the spectral decomposition of the covariance matrices have been introduced [6, 20, 36]. Specifically,  $\Sigma_g$  is reparametrized by the following eigendecomposition:

$$\Sigma_g = \lambda_g \mathbf{D}_g \mathbf{A}_g \mathbf{D}'_g, \quad (6.34)$$

where

- $\lambda_g = |\Sigma_g|^{1/p}$  denotes the volume of the  $g$ -th cluster;
- $\mathbf{D}_g$  is the eigenvector matrix of  $\Sigma_g$  which determines the orientation of the  $g$ -th cluster; and
- $\mathbf{A}_g$  is a diagonal matrix,  $|\mathbf{A}_g| = 1$ , with diagonal entries corresponding to the normalized eigenvalues of  $\Sigma_g$  in decreasing order.  $\mathbf{A}_g$  determines the shape of the  $g$ -th cluster.

**Table 6.1** Set of 14 parsimonious Gaussian mixture models with  $k$  clusters and  $p$  variables.  $\mathbf{I}_p$  denotes the identity matrix of order  $p$

Model	$\Sigma_g$	Distribution	Volume	Shape	Orientation
EII	$\lambda \mathbf{I}_p$	Spherical	Equal	Equal	
VII	$\lambda_g \mathbf{I}_p$	Spherical	Variable	Equal	
EEI	$\lambda \mathbf{A}$	Diagonal	Equal	Equal	Coordinate axes
VEI	$\lambda_g \mathbf{A}$	Diagonal	Variable	Equal	Coordinate axes
EVI	$\lambda \mathbf{A}_g$	Diagonal	Equal	Variable	Coordinate axes
VVI	$\lambda_g \mathbf{A}_g$	Diagonal	Variable	Variable	Coordinate axes
EEE	$\lambda \mathbf{DAD}'$	Ellipsoidal	Equal	Equal	Equal
VEE	$\lambda_g \mathbf{DAD}'$	Ellipsoidal	Variable	Equal	Equal
EVE	$\lambda \mathbf{DA}_g \mathbf{D}'$	Ellipsoidal	Equal	Variable	Equal
VVE	$\lambda_g \mathbf{DA}_g \mathbf{D}'$	Ellipsoidal	Variable	Variable	Equal
EEV	$\lambda \mathbf{D}_g \mathbf{AD}'_g$	Ellipsoidal	Equal	Equal	Variable
VEV	$\lambda_g \mathbf{D}_g \mathbf{AD}'_g$	Ellipsoidal	Variable	Equal	Variable
EVV	$\lambda \mathbf{D}_g \mathbf{A}_g \mathbf{D}'_g$	Ellipsoidal	Equal	Variable	Variable
VVV	$\lambda_g \mathbf{D}_g \mathbf{A}_g \mathbf{D}'_g$	Ellipsoidal	Variable	Variable	Variable

This parametrization of the GMM yields a family of parsimonious models by constraining volume, shape, or orientation to vary within and across clusters. Also, the overall component-specific covariance matrix can be restricted to be spherical or diagonal. A set of 14 specific models with different geometric features can be specified, see Table 6.1, which reports all such models with the corresponding volume, shape, and orientation and Table 6.2, which contains the corresponding number of free parameters. In both tables, each model is denoted by three letters, where “E” stands for equal, “I” stands for identity (spherical clusters), and “V” stands for varying. Specifically, the first letter refers to the volumes of the clusters, the second letter refers to the shapes of the clusters, and finally the third letter refers to the orientations of the clusters. Thus, just as an example, the model “EVI” considers clusters with equal volumes, different shapes, and spherical. Note that the first six models assume that the variables are independent while the following eight do not.

One attractive feature of GMMs is that the obtained clustering is invariant under affine transformations of the data (i.e., under changes in location, scale, and rotation of the data). Note that this does not hold, in general, for the reparameterization in (6.34), i.e., it is not true for all the models in Table 6.1. See [31] and Table 1 therein. On the other hand, a drawback of GMMs is that the likelihood function is unbounded for arbitrary covariance matrices  $\Sigma_g$  [27, 52] and, in such cases, the EM algorithm may either break down or give inaccurate results (spurious solutions, lying close to the boundary of the parameter space). The reason of an unbounded

**Table 6.2** Number of parameters for the set of 14 parsimonious Gaussian mixture models with  $k$  clusters and  $p$  variables

Model	Number of model parameters
EII	$(k - 1) + kp + 1$
VII	$(k - 1) + kp + k$
EEI	$(k - 1) + kp + p$
VEI	$(k - 1) + kp + k + (p - 1)$
EVI	$(k - 1) + kp + 1 + k(p - 1)$
VVI	$(k - 1) + kp + kp$
EEE	$(k - 1) + kp + p(p + 1)/2$
VEE	$(k - 1) + kp + k + p(p - 1)/2 + (p - 1)$
EVE	$(k - 1) + kp + 1 + p(p - 1)/2 + k(p - 1)$
VVE	$(k - 1) + kp + k + p(p - 1)/2 + k(p - 1)$
EEV	$(k - 1) + kp + 1 + kp(p - 1)/2 + (p - 1)$
VEV	$(k - 1) + kp + k + kp(p - 1)/2 + (p - 1)$
EVV	$(k - 1) + kp + 1 + kp(p - 1)/2 + k(p - 1)$
VVV	$(k - 1) + kp + kp(p + 1)/2$

likelihood function, which however occurs very rarely, can be that the covariance matrix associated with one or more components is singular or nearly singular as a consequence of one or more clusters containing only few units (it may happen when there are too many components in the mixture), or when the units in the clusters are concentrated close to a linear subspace of reduced dimension. However, when homogeneous components are considered,  $\pi_1 = \dots = \pi_k$ , covariance matrices are restricted in the parameter space and it is impossible to obtain components with only one unit [64]. There are several methods proposed in the literature to address this problem often based on adding constraints on the component covariance matrices. See [69] for a detailed discussion and [25, 30, 47, 56] for recent proposals.

## 6.7 Mixture Models for Categorical and Mixed Data

As mentioned in the previous sections, GMM is a benchmark model-based clustering approach for continuous data. When facing non-continuous data, different component densities have to be specified. For instance, for count data, Poisson component densities may be used leading to the Poisson mixture model. On the other hand, when facing nominal variables, multinomial component densities may be used leading to one of the most commonly used forms of model-based clustering for categorical data, namely, the Latent Class Analysis (LCA, [54]) model. If the values are ordinal, a standard way to proceed is to treat them as nominal, ignoring the order. However, this procedure often leads to lose information and poor clustering results. When continu-

ous and categorical variables are available in the same dataset, local independence is often assumed, i.e., variables are considered independent conditionally on the cluster membership and a mixture model for mixed data is used, where, within each component, a Gaussian distribution is used (with diagonal covariance) for the continuous and a multinomial distribution is adopted for the categorical data. In this case, model parameters can be estimated quite easily. However, if the dataset contains a large number of categorical variables when compared to the number of the continuous, a possible strategy may be to transform the few continuous variables into categorical data and use a LCA model on the resulting dataset. Clearly, a drawback of this strategy is that it may lead to an important loss of information. Recently, to overcome such problems, an appealing approach to deal with any combination of continuous, binary, ordinal, or nominal variables, named the clustMD model, has been proposed in [73]. See also the model proposed in [48] and implemented in [65], where each component distribution is allowed to be the product of different distributions chosen according to the type of variables (possible distributions are Gamma, Lognormal, Poisson, Multinomial, Binomial, and Gaussian).

A few examples of model-based clustering specific to ordinal data are available in the literature [84, 98]. In particular, a probit-like link with an Underlying Response Variable (URV) approach is proposed in [84] which avoids the requirement of the local independence assumption in model specification, thus preventing a fitted model with too many components. For a comprehensive overview of model-based clustering for categorical and mixed data, see [13, 72]. In the following sections, we describe the LCA and clustMD models.

### 6.7.1 Latent Class Analysis Model

Let us suppose we have  $n$  units described by  $p$  categorical variables, with  $m_1, \dots, m_p$  levels. The data can be represented by  $n$  binary vectors  $\mathbf{x}_i = \{x_{ijh}; j = 1, \dots, p, h = 1, \dots, m_j - 1\}, i = 1, \dots, n$ , defined as

$$x_{ijh} = \begin{cases} 1 & \text{if } x_{ij} = h, \\ 0 & \text{otherwise,} \end{cases} \quad (6.35)$$

where  $x_{ij}$  represents the response of variable  $j$  measured for unit  $i$ . Binary data can be seen as a particular case of such categorical data with  $m_j = 2, j = 1, \dots, p$ .

The LCA model assumes that data arise from a mixture of  $k$  multinomial distributions

$$f(\mathbf{x}_i | \Psi) = \sum_{g=1}^k \pi_g \text{Multi}(\mathbf{x}_i | \tau_g) = \sum_{g=1}^k \pi_g \prod_{j=1}^p \prod_{h=1}^{m_j} (\tau_{jhg})^{x_{ijh}}, \quad (6.36)$$

where  $\tau_{jhg}$  denotes the probability that variable  $j$  has level  $h$  for units in the  $g$ -th component, i.e.,  $\tau_{jhg} = \Pr(x_{ij} = h | z_{ig} = 1)$ ,  $\boldsymbol{\tau}_g = (\tau_{1hg}, \dots, \tau_{phg}; h = 1, \dots, m_j)$ ,  $\pi_g (g = 1, \dots, k)$  denote the mixing proportions, and  $\Psi$  is the parameter vector to be estimated. The LCA model assumes that the  $p$  categorical variables are independent given the component membership (local independence). This model may present problems of identifiability but most situations of interest are identifiable [3].

In order to propose more parsimonious models, it is possible to denote [18]

$$h_{jg} = \arg \max_h \tau_{jhg} \quad (6.37)$$

and

$$\varepsilon_g^{jh} = \begin{cases} 1 - \tau_{jhg} & \text{if } h = h_{jg}, \\ \tau_{jhg} & \text{otherwise,} \end{cases} \quad (6.38)$$

where  $\mathbf{h}_g = (h_{1g}, \dots, h_{pg})$  represents the modal level vector in the  $g$ -th component while the elements of  $\boldsymbol{\varepsilon}_g = (\varepsilon_g^{11}, \dots, \varepsilon_g^{pm_p})$  can be thought as scattering values. For instance, for two variables with  $m_1 = 4$  and  $m_2 = 3$  levels, if  $\boldsymbol{\tau}_g = (0.1, 0.3, 0.2, 0.4; 0.2, 0.7, 0.1)$ , then  $\mathbf{h}_g = (4, 2)$  and  $\boldsymbol{\varepsilon}_g = (0.1, 0.3, 0.2, 0.6; 0.2, 0.3, 0.1)$ . Note that an interpretation similar to the cluster center and the covariance matrix used for continuous data in the GMM context may be given: low values of the elements in  $\boldsymbol{\varepsilon}_g$  mean low dispersion around the center  $\mathbf{h}_g$ .

Imposing various constraints on the scattering parameters  $\varepsilon_g^{jh}$ , five parsimonious models can be derived:

- $\varepsilon_g^{jh}$ : the scattering depends on clusters, variables, and levels;
- $\varepsilon_g^j$ : the scattering depends on clusters and variables but not on levels;
- $\varepsilon_g$ : the scattering depends on clusters, but not on variables;
- $\varepsilon^j$ : the scattering depends on variables, but not on clusters and levels; and
- $\varepsilon$ : the scattering is constant over variables and clusters.

Further, five models can be obtained by allowing free proportions. All these models are available in the package **Rmixmod** [53] implementing the LCA models and are summarized in Table 6.3.

### 6.7.2 ClustMD Model

The clustMD model [73] is a GMM where ordinal variables arise from latent univariate Gaussian variables and nominal variables are categorizations of latent multivariate variables. Specifically, the observed mixed-type variables in  $\mathbf{x}_i$  are assumed to be the manifestation of an underlying continuous vector,  $\mathbf{u}_i$ , which follows a GMM, i.e.,

$$f(\mathbf{u}_i | \Psi) = \sum_{g=1}^k \pi_g \phi(\mathbf{u}_g, \boldsymbol{\Sigma}_g), \quad (6.39)$$

**Table 6.3** Set of ten parsimonious multinomial mixture models available in the package **Rmixmod** [53]

Model	Rmixmod name	Prop.	Var.	Comp.
$[\varepsilon]$	Binary_pk_E	Equal	True	True
$[\varepsilon^j]$	Binary_pk_Ej	Equal	False	True
$[\varepsilon_g]$	Binary_pk_Ek	Equal	True	False
$[\varepsilon_g^j]$	Binary_pk_Ekj	Equal	False	False
$[\varepsilon_g^{jh}]$	Binary_pk_Ekjh	Equal	False	False
	Binary_pk_E	Free	True	True
	Binary_pk_Ej	Free	False	True
	Binary_pk_Ek	Free	True	False
	Binary_pk_Ekj	Free	False	False
	Binary_pk_Ekjh	Free	False	False

where, as usual,  $\Psi$  denotes the overall parameter vector.

For ordinal variable  $j$  with  $m_j$  levels, let  $\gamma_j = (\gamma_{j0}, \dots, \gamma_{jm_j})$  be a  $m_{j+1}$ -dimensional vector of thresholds that partitions the real line. If the latent variable  $u_{ij}$  is such that

$$\gamma_{jh-1} < u_{ij} < \gamma_{jh}, \quad (6.40)$$

then the observed ordinal response is

$$x_{ij} = h, \quad (6.41)$$

$h = 1, \dots, m_j$ . Thus, since the latent variable  $u_{ij}$  follows a Gaussian distribution  $N(\mu_j, \sigma_j^2)$ , the probability of observing level  $h$  can be expressed as

$$P(x_{ij} = h) = \phi\left(\frac{\gamma_{jh} - \mu_j}{\sigma_j}\right) - \phi\left(\frac{\gamma_{jh-1} - \mu_j}{\sigma_j}\right), \quad (6.42)$$

where  $\phi(\cdot)$  is the Gaussian cumulative distribution function. Usually,  $\gamma_{jh}$  is taken equal to  $\phi^{-1}(\delta_h)$ , where  $\delta_h$  is the proportion of the observed values which are less than or equal to level  $h$ .

For nominal variable  $j$  with  $m_j$  levels, the latent variable is multivariate. In particular, the underlying continuous vector  $\mathbf{u}_{ij} = (u_{ij}^1, \dots, u_{ij}^{m_j-1})$  is assumed

$$\mathbf{u}_{ij} \sim MVN_{m_j-1}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j). \quad (6.43)$$

The nominal response  $x_{ij}$  is a realization of the elements in  $\mathbf{u}_{ij}$  compared to each other and to a threshold assumed to be 0, that is,

**Table 6.4** Set of six parsimonious clustMD models. Parameters are “Unconstrained”, “Constrained” to be equal across clusters or equal to the identity matrix

Model	$\lambda_g$	$\mathbf{A}_g$
EII	Constrained	Identity
VII	Unconstrained	Identity
EEI	Constrained	Constrained
VEI	Unconstrained	Constrained
EVI	Constrained	Unconstrained
VVI	Unconstrained	Unconstrained

$$x_{ij} = \begin{cases} 1 & \text{if } \max_h u_{ij}^h < 0, \\ h & \text{if } u_{ij}^{h-1} = \max_h u_{ij}^h \text{ and } u_{ij}^{h-1} > 0, \quad h = 2, \dots, m_j. \end{cases} \quad (6.44)$$

Six parsimonious diagonal clusMD models (namely, EII, VII, EEI, VEI, EVI, VVI) can be derived by means of the eigendecomposition of the cluster-specific covariance matrix  $\Sigma_g = \lambda_g \mathbf{A}_g$ , with  $|\mathbf{A}_g| = 1$ . As usual, these parameters control the volume ( $\lambda_g$ ) and the shape ( $\mathbf{A}_g$ ) of the clusters. In other words, the covariance matrix for the clustMD models is assumed to be diagonal implying that all variables are conditionally independent given the cluster membership. The six clustMD models are detailed in Table 6.4.

To estimate the clustMD model parameters an EM algorithm is used. In the presence of nominal variables, the E-step of the EM algorithm requires approximating the conditional probabilities of the latent data and the cluster labels by Monte Carlo simulations of the multivariate Gaussian distributions.

An approximation of the BIC is used for selecting the best clustMD model (the best fitting covariance structure and number of clusters) since the observed likelihood, needed for the calculation of the standard BIC, relies on the calculation of intractable integrals. Further details are present in [73].

## 6.8 mclust, Rmixmod, and clustMD

A large number of R packages are available for model-based clustering and each covers a specific set of mixture models. Among others, we may refer to **bgmm** [9], **EMCluster** [24], **flexmix** [40], **flexCWM** [65], **mclust** [90], **MixAll** [49], **mixtools** [105], **mixture** [15], and **Rmixmod** just to mention a few. See [90] for a comparative study of the functionalities of the abovementioned packages.

Here, we illustrate the use of two popular libraries, namely, **mclust** and **Rmixmod**. While the former is specific for estimating GMMs via the EM algorithm and it can be applied to data involving only continuous variables, the latter also allows for estimating multinomial FMMs via the EM algorithm and it can be used for cat-

egorical variables. By assuming local independence between continuous and categorical variables according to the procedure described in Sect. 6.7, **Rmixmod** allows for clustering of mixed data. Finally, we give an illustration of the **clustMD** model implemented by the package **clustMD** [74] in the case of mixed data for comparison with the **Rmixmod** solution.

### 6.8.1 Case Study with Continuous Data

To discuss clustering for continuous data, we consider the dataset **customers** [1] included in the package **datasetsICR** [39] regarding clients of a wholesale distributor already presented in Sect. 2.6.1. Also, in this case, we limit our attention to the  $p = 6$  continuous variables giving the annual spending of different types of goods (i.e., **customers[, 3:8]**). The aim is to cluster customers so that customers in a given cluster present similar annual spending shares. In Sect. 6.8.1.1, we illustrate the use of **mclust** and in Sect. 6.8.1.2 the use of **Rmixmod**.

#### 6.8.1.1 **customers** data: **mclust**

Typically, a model-based clustering strategy consists in comparing models for different numbers of clusters and different sets of constraints on the component-specific covariance matrices and then choosing the best combination. In the package **mclust**, the function **mclustBIC** computes the BIC values for all the models corresponding to the 14 covariance structures described in Table 6.1 for a vector, matrix, or data frame of quantitative variables specified in the argument **data**. The available models can also be visualized by means of the command **help(mclustModelNames)**. The number of clusters for which the BIC is calculated is  $G = 1:9$  unless otherwise specified, while the default option for initializing the EM algorithm is the solution obtained from the MBHAC approach we have shortly described in Sect. 6.3.3. Unusually, the best model corresponds to the model having the maximum BIC value over all specifications, as the function actually provides the  $-BIC$  value. As a side note, we advise the readers to always check, within the packages involving the computation of BIC, how it is implemented.

```
> library(datasetsICR)
> data("customers")
> library(mclust)
> BIC <- mclustBIC(data = customers[, 3:8])
> BIC
Bayesian Information Criterion (BIC):
      EII        VII       EEI       VEI       EVI
1 -54803.18 -54803.18 -53682.50 -53682.50 -53682.50
2 -54316.13 -52961.91 -53006.53 -51441.37 -52527.12
3 -53647.73 -52457.97 -52663.14 -50569.68 -51549.29
```

```

4 -53324.13 -51729.15 -52681.34 -50214.79 -51156.24
5 -53038.81 -51457.71 -52004.36 -50034.81 -51178.62
6 -52777.98 -51268.11 -52066.76 -50085.59 -50828.52
7 -52743.00 -51045.07 -52267.31 -49905.49 -50632.43
8 -52784.91 -50940.48 -52308.76 -49818.64 -50680.49
9 -52319.44 -50865.96 -52252.09 -49724.50 -50812.60

          VVI         EEE         EVE         VEE
1 -53682.50 -52283.37 -52283.37 -52283.37
2 -51327.70 -52036.09 -51668.35 -50477.43
3 -50027.81 -52078.78 -51049.97 -50011.99
4 -49650.69 -52123.25        NA -49930.19
5 -49480.53 -51798.47        NA -49840.32
6 -49379.84 -51840.84        NA -49840.18
7 -49359.41 -51715.99        NA -49717.08
8 -49296.33 -51758.54        NA -49674.40
9 -49297.93 -51766.77        NA -49679.22

          VVE         EEV         VEV         EVV         VVV
1 -52283.37 -52283.37 -52283.37 -52283.37 -52283.37
2 -50510.98 -51233.31 -50501.13 -51199.88 -50474.18
3 -50043.88 -51166.60 -49863.02 -51070.11 -49682.83
4 -49574.91 -50547.94 -49581.46 -50331.92 -49517.86
5 -49535.25 -50433.33 -49576.04 -50331.69 -49520.79
6 -49522.48 -50396.81 -49598.76 -50283.92 -49573.43
7 -49480.75 -50378.86 -49559.33        NA -49542.27
8 -49518.14 -50390.40 -49571.01        NA -49606.77
9 -49541.36 -50322.63 -49636.17        NA -49711.31

Top 3 models based on the BIC criterion:
      VVI , 8      VVI , 9      VVI , 7
-49296.33 -49297.93 -49359.41

```

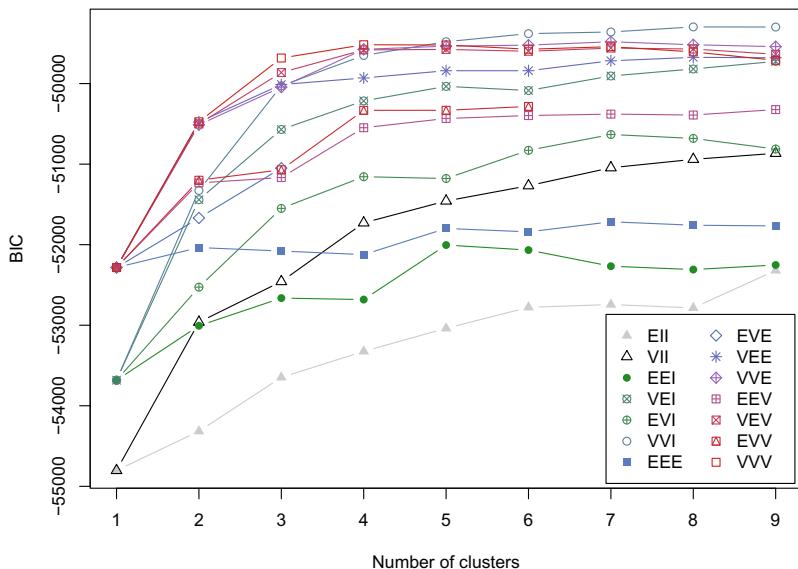
To visualize only the three best BIC values and the classification table for the best model, we can use the following code.

```

> summary(BIC, customers[, 3:8])
Best BIC values:
              VVI , 8           VVI , 9           VVI , 7
BIC      -49296.33 -49297.928459 -49359.40977
BIC diff     0.00       -1.594514      -63.07582

Classification table for model (VVI , 8):
  1   2   3   4   5   6   7   8
 85  62  25  55  33 109  11  60

```



**Fig. 6.2** `customers` data: BIC plots for the 14 GMMs fitted by `mclustBIC` for  $k = 1, \dots, 9$

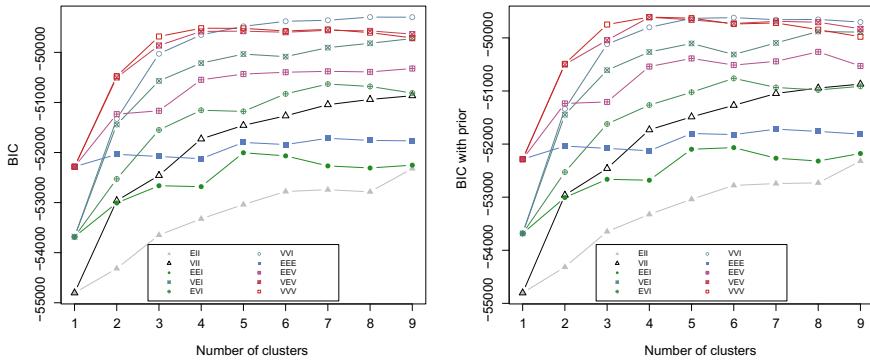
Figure 6.2 graphically shows the BIC values for each estimated model. Graphic parameters such as legend options and axes limits can be defined by the user, see `help(plot.mclustBIC)` for more details. The code to obtain such a visualization is given below.

```
> plot(BIC, xlab = "Number of clusters")
```

Optional arguments in `mclustBIC` allow for specifying the number of clusters,  $G$ , and the model covariance structures, `modelName`. Some examples are listed below.

```
> # BIC values only for k = 1, 3, 5
> k <- seq(from = 1, to = 5, by = 2)
> mclustBIC(customers[, 3:8], G = k)
> # BIC values only for models "EEE" and "VVV"
> mclustBIC(customers[, 3:8],
+             modelName = c("EEE", "VVV"))
> # BIC values for k = 1, 3, 5 and models "EEE" and
> # "VVV"
> mclustBIC(customers[, 3:8], k,
+             modelName = c("EEE", "VVV"))
```

Notice that, in the output results of `mclustBIC`, we have NA values for the combination (EEE,  $G = 5:9$ ) and (EVV,  $G = 7:9$ ). It means that these models cannot



**Fig. 6.3** customers data: BIC plots (on the left) and with Bayesian regularization (on the right) for the 10 GMMs fitted by `mclustBIC` for  $k = 1, \dots, 9$

be estimated; this usually happens when the likelihood function becomes infinite due to singularity of some cluster-specific covariance matrix. A way to solve this issue is based on Bayesian regularization as proposed in [37]. By specifying proper prior distributions, the sequence of BIC values tends to decrease smoothly, rather than jumping spuriously to infinity. This is because the prior distribution tends to smooth or regularize these features. Such a regularization is implemented in `mclust` by specifying the `prior` argument which allows to adopt a conjugate prior for the component-specific means and covariance matrices. This is described below.

```
> BICprior <- mclustBIC(customers[, 3:8],
+                         prior = priorControl())
```

To appreciate the differences, we run the following code to get Fig. 6.3.

```
> par(mfrow = c(1, 2))
> plot(BIC, modelNames = c("EII", "VII", "EEI", "VEI",
+                           "EVI", "VVI", "EEE", "EEV", "VEV", "VVV"),
+       xlab = "Number of clusters", ylab = "BIC",
+       legendArgs = list(x = "bottom", ncol = 2,
+                          cex = .55))
> plot(BICprior, xlab = "Number of clusters",
+       ylab = "BIC with prior",
+       legendArgs = list(x = "bottom", ncol = 2,
+                          cex = .55))
```

The figure shows the BIC values without (on the left) and with the prior (on the right). To have a direct comparison, we specify `modelNames` to obtain plots without prior for the same models. However, in this case, even though the Bayesian regularization is specified, models EVE and EVV cannot be estimated.

As already mentioned, the EM algorithm in **mclust** is initialized using the MBHAC solution. In most cases, it provides reasonable starting points, even though it does not guarantee convergence to a global maximum as explained in Sect. 6.3.3. The argument `initialization` allows for choosing different starting points for the EM algorithm through the following three components: `hcPairs`, `subset`, and `noise`. Specifically, `hcPairs` is a matrix of merge pairs for agglomerative hierarchical clustering similar to the one produced by the function `hc`, a function of **mclust** implementing MBHAC. The default option already shown above is equivalent to the following use of `hc`.

```
> hc1 <- hc(customers[, 3:8], modelName = "VVV",
+             use = "SVD")
> BIC <- mclustBIC(customers[, 3:8],
+                     initialization =
+                     list(hcPairs = hc1))
```

Examples of two different initializations are shown below.

```
> # Standardized variables and model "EEE"
> hc2 <- hc(customers[, 3:8], modelName = "EEE",
+             use = "SVD")
> BIC2 <- mclustBIC(customers[, 3:8],
+                     initialization =
+                     list(hcPairs = hc2))
> # Random initialization by creating random
> # agglomerations, see help(randomPairs)
> BIC3 <- mclustBIC(customers[, 3:8], verbose = FALSE,
+                     initialization = list(hcPairs =
+                     randomPairs(customers[, 3:8])))
> summary(BIC)
Best BIC values:
          VVI , 8           VVI , 9           VVI , 7
BIC      -49296.33       -49297.92       -49359.41
BIC diff    0.00          -1.59          -63.08
> summary(BIC2)
Best BIC values:
          EEV , 8           EEV , 5           EEV , 6
BIC      -50351.41       -50380.88       -50382.12
BIC diff    0.00          -29.47          -30.70
> summary(BIC3)
Best BIC values:
          VVI , 7           VVI , 8           VVE , 6
BIC      -49375.25       -49395.34       -49408.86
BIC diff    0.00          -20.09          -33.62
```

As it can be observed, two out of three initializations produce a solution with  $k = 8$  clusters and different covariance structures. However, looking at the BIC values, the

best solution corresponds to the one having diagonal covariance with varying volume and shape (i.e., VVI). On the other hand, the random initialization produces a different solution at each try (result not shown here). The latter problem is clearly related to local maxima. To obtain results that are reproducible, we recommend to set the seed of the random number generated using the function `set.seed`. A way to overcome the problem of local maxima consists in considering several random starts as described in Sect. 6.3.3. For this purpose, the function `mclustBICupdate` may be useful to merge objects of class `mclustBIC` and then keep the best models. An example with 100 random starts is specified by the following code.

```
> BICt <- NULL
> for(j in 1:100) {
+   rBIC <- mclustBIC(customers[, 3:8],
+                       verbose = FALSE,
+                       initialization =
+                       list(hcPairs =
+                             randomPairs(customers[, 3:8])))
+   BICt <- mclustBICupdate(BICt, rBIC)
+ }
> summary(BICt)
Best BIC values:
          VVI , 9           VVI , 8           VVI , 7
BIC      -49222        -49242.10       -49303.70
BIC diff     0          -20.10        -81.70
```

By using a random initialization with 100 starting points, the best solution corresponds to  $k = 9$  clusters with the same covariance structure of the abovementioned best solution (i.e., VVI). Note that, when the default option of `initialization` is used, such a solution with  $k = 9$  and covariance structure VVI is the second best solution and it has a BIC value that is very close to the best one (`BIC diff = -1.59`). However, for the sake of parsimony, we prefer the one with  $k = 8$  clusters.

Note that the argument `noise` may be used for indicating an initial guess as to which units in the data are noisy. An illustration of the latter will be given in Sect. 7.5.

In order to estimate model parameters, the function `Mclust` can be used. Similarly to `mclustBIC`, it considers by default 14 different covariance structures, up to  $k = 9$  clusters and the MBHAC initialization for the EM algorithm. These options can be modified as for `mclustBIC`. The output is a list of components corresponding to the best model selected according to the BIC as shown below.

```
> mod <- Mclust(customers[, 3:8])
> names(mod)
[1] "call"      "data"       "modelName"  "n"
[5] "d"         "G"          "BIC"        "bic"
[9] "loglik"    "df"         "hypvol"    "parameters"
[13] "z"         "classification" "uncertainty"
```

Thus, for visualizing only the estimated means and the corresponding best partition we can use the following script.

```
> round(mod$parameters$mean, 2)
      [,1]      [,2]      [,3]
Fresh     7812.31   3902.20   6209.16
Milk      4830.34   8867.54   16838.35
Grocery   6635.72   14404.02   26620.46
Frozen    852.53    1119.39    1887.55
Detergents_Paper 2173.27   6541.25   12718.08
Delicassen 1073.24   1162.56   2182.03
      [,4]      [,5]      [,6]
Fresh     23191.94  17273.02  13661.78
Milk      3884.24   9035.88   1805.28
Grocery   4461.43   9207.82   2529.03
Frozen    7709.25   3841.60   3296.78
Detergents_Paper 575.00    2635.45   423.72
Delicassen 1918.05   2681.12   902.85
      [,7]      [,8]
Fresh     35962.34  6984.09
Milk      31746.52  1080.09
Grocery   37244.80  1579.71
Frozen    13053.55  1451.94
Detergents_Paper 15112.75  211.35
Delicassen 10934.65  449.74
> table(mod$classification)
  1   2   3   4   5   6   7   8
 85   62   25   55   33  109   11   60
```

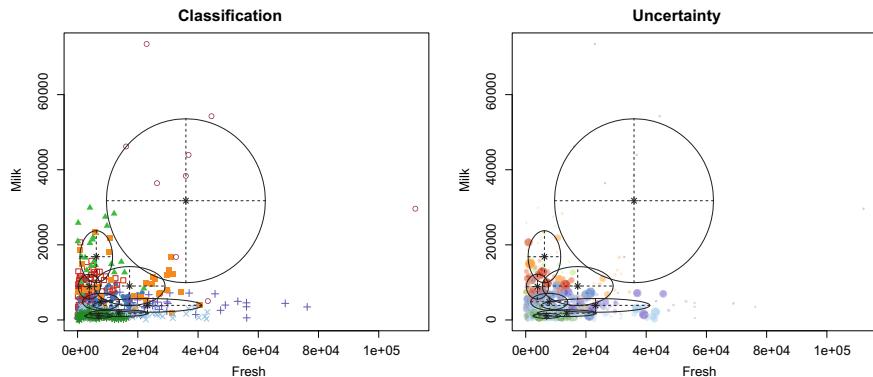
In practice, especially for large datasets, a good strategy for saving both time and memory consists in exploring the potential best models and different initialization strategies with the function `mclustBIC` and then apply `Mclust` for getting the results only of the (few) best model(s). For this purpose, we may specify the option `x = BICt`.

```
> modt <- Mclust(customers[, 3:8], x = BICt)
```

In this way, the BIC values for models that have already been computed, and are available in `x`, are not recomputed.

Graphical representation of classification and uncertainty associated with the obtained clustering can be done through the following code.

```
> plot(mod, what = "classification")
> plot(mod, what = "uncertainty")
```



**Fig. 6.4** Customers data: classification (on the left) and uncertainty (on the right) on the plane (Fresh, Milk) corresponding to the best GMM ( $k = 8$ , model=VV<sub>I</sub>, BIC= -49296.33) fitted by Mclust with default options. Different colors correspond to different clusters

Two matrices of scatterplots are produced for each pair of variables with points marked according to the corresponding cluster and component-specific ellipses superimposed. To have a better view of these plots, especially if we have a large number of variables, we may select only a subset of variables to be displayed with the argument `dimens`. For instance, Fig. 6.4 shows the classification (on the left) and uncertainty (on the right) with respect to the variables `Fresh` and `Milk`. The code used for obtaining such a plot is listed below.

```
> par(mfrow = c(1, 2))
> plot(mod, what = "classification", dimens = 1:2)
> title(main = "Classification")
> plot(mod, what = "uncertainty", dimens = 1:2)
> title(main = "Uncertainty")
```

To visualize a summary of the best model, we can use `summary(mod)` which provides the log-likelihood value, the number of units  $n$ , the degrees of freedom, the BIC, the ICL, and the clustering table. Note that, specifying the argument `parameters = TRUE`, also parameter estimates corresponding to the best model are provided.

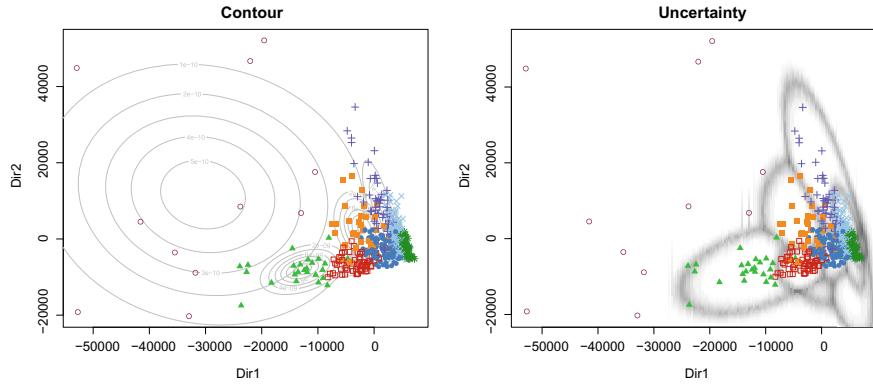
Looking at the chosen partition, we observe that Cluster 7 has the smallest size and it is characterized by customers with extremely high annual spending for all types of goods except for `Fresh` and `Frozen`, while Cluster 8 is characterized by customers with low annual spending for all types of goods (followed by Cluster 1). Cluster 6 is similar to Clusters 1 and 8 except for `Fresh` and `Frozen`. Customers assigned to Cluster 3 have high annual spending for a subset of types of goods, namely, `Milk`, `Grocery`, and `Detergents_Paper`, and low for the others, while the opposite is true for Cluster 4. Clusters 2 and 5 do not have a clear interpretation. Furthermore, to check whether the obtained partition has link with the `Channel` and/or `Region`

variables, providing information on the customer channel (two levels: Horeca, i.e., Hotel/Restaurant/Café, Retail) and region (three levels: Lisbon, Oporto, Other), respectively, we report the following cross-tabulations and the corresponding Adjusted Rand Index (ARI, [46]).

```
> attach(customers)
> table(Channel, mod$classification)
Channel   1   2   3   4   5   6   7   8
  Horeca 52   5   0  51  18 107   5  60
  Retail 33  57  25   4  15   2   6   0
> table(Region, mod$classification)
Region   1   2   3   4   5   6   7   8
  Lisbon 14  11   5   9   7  19   0  12
  Oporto  8   7   4   6   2  14   2   4
  Other  63  44  16  40  24  76   9  44
> ARI1 <- adjustedRandIndex(Channel,
+                               mod$classification)
> ARI2 <- adjustedRandIndex(Region, mod$classification)
> round(ARI1, 2)
[1] 0.16
> round(ARI2, 2)
[1] -0.00
```

As it can be noticed, all the units belonging to Cluster 8 and most of those in Cluster 6 (107 out to 109) present the level Horeca, while all the units belonging to Cluster 3 and a large portion of those in Cluster 2 present the level Retail. On the other hand, no relationship between the obtained partition and Region can be recorded. The computed ARI values are low even though ARI1 is slightly higher than ARI2, probably for the correspondence we observe among some clusters and Channel that, in any case, is masked by the presence of clusters without such a strong connection. All in all, by comparing this solution with those obtained by using DIANA (Sect. 2.6.1), we can conclude that the current one offers a more elaborated partition characterized by a higher number of clusters. However, some of these clusters appear to be related to the ones discovered by DIANA.

Once the clusters have been defined, we may use the function `MclustDR` to visualize the clustering structure into a low-dimensional subspace according to the proposal in [89]. The directions which span the reduced subspace are defined as the set of linear combinations of the original variables, ordered by importance through the corresponding eigenvalues. `MclustDR` contains a tuning parameter `lambda`, taking values in [0, 1], which enables the recovery of most of the separating directions, i.e., those that lead to maximal separation among groups. The default (0.5) gives equal importance to differences in means and covariances among clusters. By setting `lambda = 1`, only the information on cluster means is used for estimating the directions and, as a result, the projected data show the maximal separation among cluster means. In our case, the dimension of the subspace is  $d = \min(p = 6, k - 1 = 7) = 6$ . For more details, see `help(MclustDR)` and [88, 89]. Figure 6.5 shows the bivariate contour plot of estimated mixture densities



**Fig. 6.5** customers data: contour plot of the estimated densities (on the left) and uncertainty boundaries (on the right) on the projection subspace, estimated by `MclustDR`, corresponding to the best GMM ( $k = 8$ , `model=VVI`,  $\text{BIC} = -49296.33$ ) fitted by `Mclust` with default options. Different colors correspond to different clusters

(on the left) and uncertainty boundaries (on the right) when the projection subspace is considered. The uncertainty is shown using a gray scale with darker regions indicating higher uncertainty. The corresponding code is reported below.

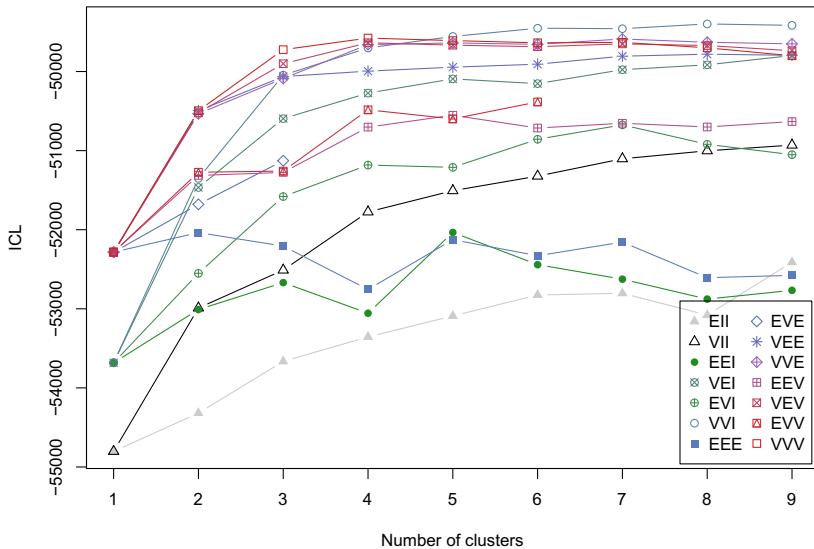
```
> drmod <- MclustDR(mod, lambda = 1)
> par(mfrow = c(1, 2))
> plot(drmod, what = "contour")
> title(main = "Contour")
> plot(drmod, what = "boundaries", ngrid = 200)
> title(main = "Uncertainty")
```

Many other plots can be obtained by specifying the argument `what`. For a comprehensive list and examples, see `help(plot.MclustDR)`. Other dimension reduction techniques for finding the directions of maximum separation can be found in [44] and are implemented in the package **fpc** [45].

Until now, we have used the default option of **mclust** for model selection, that is, the BIC. However, as already mentioned in Sect. 6.5, BIC is a good choice in a density estimation framework rather than in a clustering one. Alternatively, **mclust** allows to use the ICL by means of the function `mclustICL`.

```
> ICL <- mclustICL(customers[, 3:8])
> plot(ICL, xlab = "Number of clusters")
```

Figure 6.6 shows the ICL values. It is well known that ICL privileges solutions with well-separated clusters; however, in this example, BIC and ICL recommend the same optimal solution.

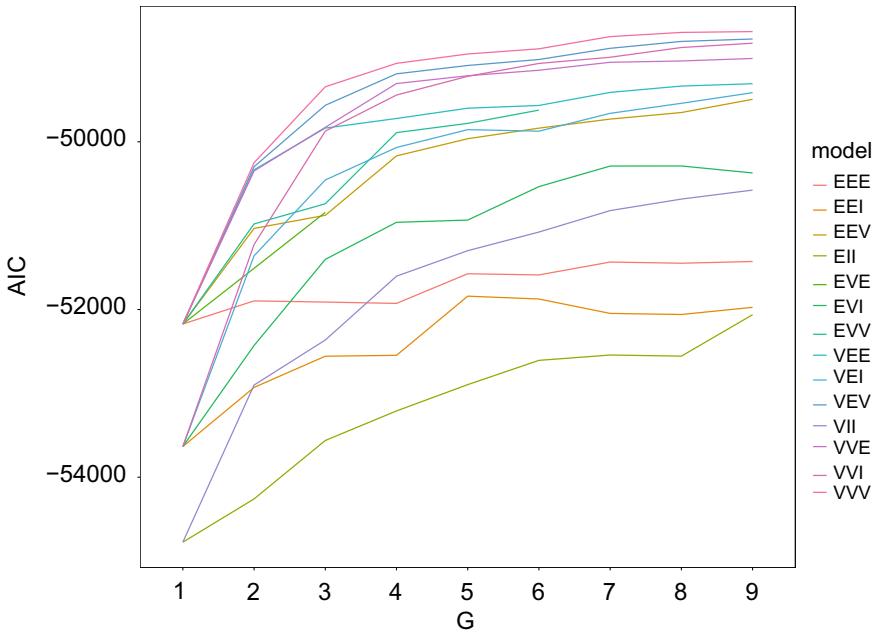


**Fig. 6.6** customers data: ICL plots for the 14 models fitted by `mclustICL` for  $k = 1, \dots, 9$

Other information criteria may be computed for verifying whether the same model is selected. To investigate this further, we compute the values of  $-\text{AIC}$ , see (6.17), for all the models and numbers of clusters considered above. By using the function `ggplot` of the package `ggplot2` [101], we graphically visualize these values in Fig. 6.7.

```
> library(ggplot2)
> G <- attr(BIC, "G")
> modelN <- attr(BIC, "modelName")
> d <- attr(BIC, "d")
> criteria <- data.frame(G = rep(G, length(modelN)),
+                           model = rep(modelN,
+                           each = length(G)), AIC = NA,
+                           stringsAsFactors = FALSE)
> for (i in 1:nrow(criteria)){
+   m = Mclust(customers[, 3:8],
+   G = criteria$G[i],
+   modelN = criteria$model[i])
+   if (!is.null(m)) {
+     p = nMclustParams(criteria$model[i],
+     d, criteria$G[i])
+     criteria[i, "AIC"] =
+     2 * m$loglik - 2 * p}
+ }
> head(criteria[order(criteria$AIC,
+ decreasing = TRUE), ], 3)
```

G	model	AIC
1	EII	-54500
1	EVE	-52500
1	VII	-54800



**Fig. 6.7** customers data: AIC plots for the 14 models fitted by Mclust for  $k = 1, \dots, 9$

```

126 9      VVV      -48685.53
125 8      VVV      -48695.42
124 7      VVV      -48745.35
> ggplot(criteria, aes(G, AIC, color = model)) +
+   geom_line() +
+   scale_x_discrete(limits = G) +
+   theme_bw() +
+   theme(panel.grid.major.x = element_blank()) +
+   theme(panel.grid.major.y = element_blank()) +
+   theme(panel.grid.minor.y = element_blank())

```

As expected, AIC tends to select more clusters than BIC and ICL. **mclust** also allows for selecting the number of clusters by LRT through the bootstrap procedure described in Sect. 6.5, implemented in the function `mclustBootstrapLRT`. Besides the data, it is mandatory to specify the model we want to test, other optional arguments are `nboot`, `level`, and `maxG` which allow for setting the number of bootstrap samples (default 999), the significance level (default 0.05), and the maximum number of clusters to test, respectively. In our example, the bootstrap *p*-values indicate the presence of  $k = 9$  clusters for model VVI, as indicated by the following output.

```

> LRT <- mclustBootstrapLRT(customers[, 3:8], maxG = 9,
+                               modelName = "VVI")
> LRT
-----
Bootstrap sequential LRT for the number of mixture c...
-----
Model      = VVI
Replications = 999
    LRTS bootstrap p-value
1 vs 2     2433.93      0.001
2 vs 3     1379.02      0.001
3 vs 4     456.25       0.001
4 vs 5     249.29       0.001
5 vs 6     179.82       0.001
6 vs 7     99.56        0.001
7 vs 8     142.20       0.001
8 vs 9     77.53        0.001
9 vs 10    -1.34        1.000

```

Figure 6.8 displays the histograms of the LRT bootstrap distributions (the dotted vertical lines refer to the sample values of LRTs) which can be obtained by the following code.

```

> par(mfrow = c(3, 3))
> plot(LRT, G = 1)
> plot(LRT, G = 2)
> plot(LRT, G = 3)
> plot(LRT, G = 4)
> plot(LRT, G = 5)
> plot(LRT, G = 6)
> plot(LRT, G = 7)
> plot(LRT, G = 8)
> plot(LRT, G = 9)

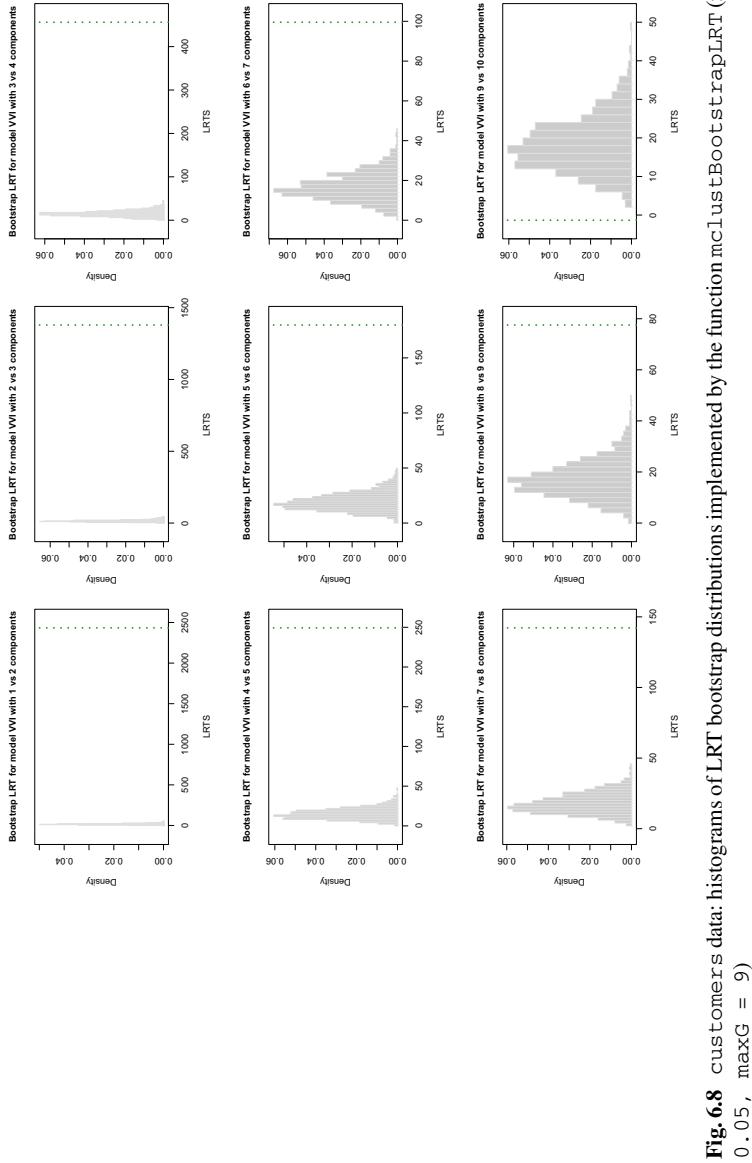
```

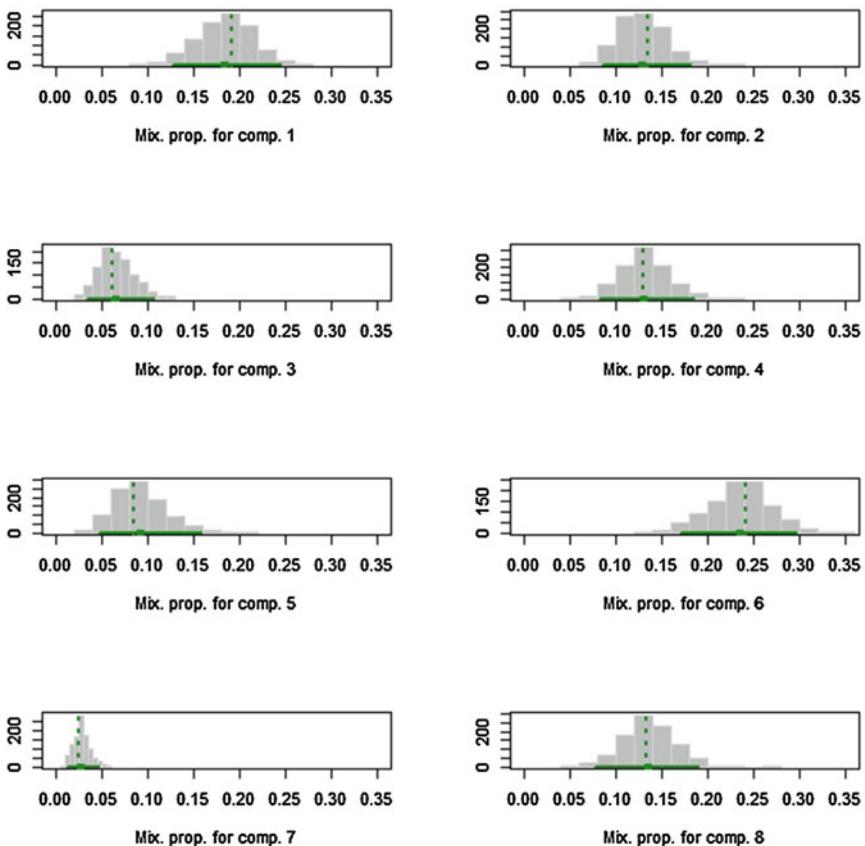
The package **mclust** allows for computing the bootstrap (standard or weighted likelihood) and jackknife estimates of standard errors and percentile bootstrap confidence intervals for model parameters through the function `MclustBootstrap`. It needs as input argument an object returned by a call to `Mclust`. The arguments `nboot` and `type` are optional for specifying the number of bootstrap samples and the type of bootstrap to perform, respectively. By default, `nboot = 999` and `type = "bs"` for parametric bootstrap.

```

> boot <- MclustBootstrap(mod, nboot = 999,
+                           type = "bs")

```





**Fig. 6.9** customers data: bootstrap distribution for the mixing proportions `pro`, corresponding to the best GMM ( $k = 8$ , model=VV1,  $\text{BIC} = -49296.33$ , fitted by `Mclust` with default options), implemented by the function `MclustBootstrap` (`nboot = 999`, `type = "bs"`). The vertical dotted lines refer to the MLEs

The command `summary(boot)` allows for visualizing the standard errors of parameter estimates while, by specifying the argument `what = "ci"`, the bootstrap percentile confidence intervals with a confidence level equal to 0.95 (it can be modified by setting the `conf.level` argument) are given. Other types of resampling can be visualized calling `help(MclustBootstrap)`. Details are available in [80]. Finally, plots of the bootstrap distribution of parameters `pro`, `mean`, and `var` can be drawn. For instance, Fig. 6.9 shows the bootstrap distribution for the mixing proportions (`pro`) obtained by using the following code.

```
> par(mfrow = c(4, 2))
> plot(boot, what = "pro")
```

### 6.8.1.2 **customers** data: Rmixmod

Let us now consider the analysis of the *customers* data by using the package **Rmixmod** to highlight potential differences with the solution provided by **mclust**. The function to estimate model parameters is `mixmodCluster`. By default, for continuous data, it applies the EM algorithm initialized by the `smallEM`, described in Sect. 6.3.3. This latter estimates the model `Gaussian_pk_Lk_C` which corresponds to the model `VEE` in **mclust**. In contrast to **mclust**, the vector containing the possible numbers of clusters is mandatory and has to be specified by the argument `nbCluster`. The default criterion to select the best model is `BIC`, even though it can be differently specified by means of the argument `criterion`. Possible alternatives are `"ICL"`, `"NEC"`, `c("BIC", "ICL", "NEC")`. Contrarily to **mclust** and according to the standard information criteria definition, the best model is the one with the lowest value of the chosen criterion. Considering all the available information criteria, we estimate the default model up to  $k = 12$  clusters by the following code.

```
> library(Rmixmod)
> ALL_Rmixmod <- mixmodCluster(customers[, 3:8],
+                                 nbCluster = 1:12,
+                                 criterion = c("BIC", "ICL", "NEC"))
> table(ALL_Rmixmod@bestResult@partition)
  1   2   3   4   5   6   7   8   9 
 24  20  40  38  31  72  19  74 122
> # Repeat again
> ALL_Rmixmod <- mixmodCluster(customers[, 3:8],
+                                 nbCluster = 1:12,
+                                 criterion = c("BIC", "ICL", "NEC"))
> table(ALL_Rmixmod@bestResult@partition)
  1   2   3   4   5   6   7   8 
123  55 104  38  44  20  43  13
```

As it can be observed, the solution obtained in `ALL_Rmixmod` substantially changes at each entry due to the presence of local maxima of the log-likelihood function. In such a situation, **Rmixmod** is able to avoid these traps by tuning its input parameters as we will show later on.

Behind the default model, the same family of 14 models implemented in **mclust** is provided. A further set of 14 models allowing for identical prior probabilities ( $\pi_g = 1/k$ ,  $g = 1, \dots, k$ ) can be set. The complete list of 28 models can be displayed with the command `mixmodGaussianModel()`. To run all the available models, we have to specify `mixmodGaussianModel(family = "all")` in the argument `models` of the main function `mixmodCluster`. Other family options are `"general"` (eight models where the volumes, the shapes, and the orientations of the clusters are allowed to vary among clusters), `"diagonal"` (four models with diagonal covariance matrices), and `"spherical"` (two models assuming spherical

**Table 6.5** The family of common parsimonious models as implemented in **mclust** and **Rmixmod**. For models with equal proportions, “\_pk\_” is replaced by “\_p\_”.  $\mathbf{B}_g$  is a diagonal matrix with  $|\mathbf{B}_g| = 1$

<b>mclust</b>	<b>Rmixmod</b>	<b>Rmixmod</b> name	<b>Rmixmod</b> family
EII	$[\lambda \mathbf{I}_p]$	Gaussian_pk_L_I	spherical
VII	$[\lambda_g \mathbf{I}_p]$	Gaussian_pk_Lk_I	spherical
EEI	$[\lambda \mathbf{B}]$	Gaussian_pk_L_B	diagonal
VEI	$[\lambda_g \mathbf{B}]$	Gaussian_pk_Lk_B	diagonal
EVI	$[\lambda \mathbf{B}_g]$	Gaussian_pk_L_Bk	diagonal
VVI	$[\lambda_g \mathbf{B}_g]$	Gaussian_pk_Lk_Bk	diagonal
EEE	$[\lambda \mathbf{D}\mathbf{A}\mathbf{D}']$	Gaussian_pk_L_C	general
VEE	$[\lambda_g \mathbf{D}\mathbf{A}_g\mathbf{D}']$	Gaussian_pk_Lk_C	general
EVE	$[\lambda \mathbf{D}\mathbf{A}_g\mathbf{D}']$	Gaussian_pk_L_D_Ak_D	general
VVE	$[\lambda_g \mathbf{D}\mathbf{A}_g\mathbf{D}']$	Gaussian_pk_Lk_D_Ak_D	general
EEV	$[\lambda \mathbf{D}_g\mathbf{A}\mathbf{D}_g']$	Gaussian_pk_L_Dk_A_Dk	general
VEV	$[\lambda_g \mathbf{D}_g\mathbf{A}_g\mathbf{D}_g']$	Gaussian_pk_Lk_Dk_A_Dk	general
EVV	$[\lambda \mathbf{D}_g\mathbf{A}_g\mathbf{D}_g']$	Gaussian_pk_L_Ck	general
VVV	$[\lambda_g \mathbf{D}_g\mathbf{A}_g\mathbf{D}_g']$	Gaussian_pk_Lk_Ck	general

shapes). Default is “general”. In Table 6.5, the 14 models common to **mclust** and **Rmixmod** are shown.

Some model specifications are shown in the following code.

```
> # All the available models
> ALL_Rmixmod2 <- mixmodCluster(customers[, 3:8],
+ nbCluster = 1:12,
+ models = mixmodGaussianModel(family = "all"))
> # General family
> ALL_Rmixmod3 <- mixmodCluster(customers[, 3:8],
+ nbCluster = 1:12,
+ models = mixmodGaussianModel
+ (family = "general"))
> # Spherical and Diagonal families
> ALL_Rmixmod4 <- mixmodCluster(customers[, 3:8],
+ nbCluster = 1:12,
+ models = mixmodGaussianModel(
+ family = c("spherical",
+ "diagonal")))
> table(ALL_Rmixmod4@bestResult@partition)
 1  2  3  4  5  6  7  8
 9 51 75 20 74 67 81 63
```

In this example, `family = "all"` and `family = "general"` do not give possible solutions (details not reported) while the solution obtained in `ALL_Rmixmod4` needs to be adjusted since, as before, multiple re-runs of `mixmodCluster` provide local optima (details not reported). Therefore, we now check whether results improve by modifying the estimation strategy. For this purpose, the argument `strategy` in `mixmodCluster` allows for specifying the estimation strategy by means of the function `mixmodStrategy`. In the latter, a list of arguments can be tuned; in particular, increasing the value of `nbTry`, which defines the number of tries (default 1), can help to find the global optimum solution.

Other important arguments are: `algo` which allows to specify alternative estimation algorithms (options are "EM", "SEM", "CEM", `c("EM", "SEM")`) and `initMethod` which specifies the method of initialization of the algorithm in the `algo` argument. For `initMethod`, options are "random", "smallEM", "CEM", "SEMMax", "parameter", "label". For a deeper insight, see `help(mixmodStrategy)` and [53].

Without loss of generality, two different ways to set a strategy are shown below. In the first example, we decide to increase the number of tries by setting `nbTry = 100`; in the second example, we also change the initialization by choosing a random-starting point strategy (`initMethod = "random"`).

```
> strategy1 <- mixmodStrategy(nbTry = 100)
> ALL_Rmixmod1 <- mixmodCluster(customers[, 3:8],
+                                   nbCluster = 1:12,
+                                   models = mixmodGaussianModel(
+                                       family = c("spherical", "diagonal")),
+                                   strategy = strategy1,
+                                   criterion = c("BIC", "ICL", "NEC"))
> strategy2 <- mixmodStrategy(initMethod = "random",
+                               nbTry = 100)
> ALL_Rmixmod2 <- mixmodCluster(customers[, 3:8],
+                                   nbCluster = 1:12,
+                                   models = mixmodGaussianModel(
+                                       family = c("spherical", "diagonal")),
+                                   strategy = strategy2,
+                                   criterion = c("BIC", "ICL", "NEC"))
> table(ALL_Rmixmod1@bestResult@partition)
 1  2  3  4  5  6  7  8  9
41 70 87 10 54 14 63 55 46
> table(ALL_Rmixmod2@bestResult@partition)
 1  2  3  4  5  6  7  8  9
63 54 55 10 14 70 87 46 41
> table(ALL_Rmixmod1@bestResult@partition,
+        ALL_Rmixmod2@bestResult@partition)
    1  2  3  4  5  6  7  8  9
 1  0  0  0  0  0  0  0  0 41
 2  0  0  0  0  0 70  0  0  0
 3  0  0  0  0  0  0 87  0  0
 4  0  0  0 10  0  0  0  0  0
 5  0 54  0  0  0  0  0  0  0
```

```

6   0   0   0   0  14   0   0   0   0
7  63   0   0   0   0   0   0   0   0
8   0   0  55   0   0   0   0   0   0
9   0   0   0   0   0   0   0  46   0

```

As it can be observed, both strategies lead to the same best solution corresponding to the model Gaussian\_pk\_Lk\_Bk (diagonal, varying volume and shape) with  $k = 9$  clusters. A list of all the estimated models, sorted by the BIC values, is given by the command ALL\_Rmixmod1@results or ALL\_Rmixmod1 [ "results" ]. Alternatively, we can choose to sort this list according to a different information criterion by the function sortByCriterion. For instance, the following command shows the best model according to ICL. In our example, BIC and ICL give the same solution.

```

> icl.out <- sortByCriterion(ALL_Rmixmod1, "ICL")
> # Or
> icl.out [ "bestResult" ]

```

For convenience and for sake of space, we report the most important results as follows.

```

> round(ALL_Rmixmod1@bestResult@parameters@mean, 2)
      [,1]      [,2]      [,3]      [,4]
[1,] 53054.7 13597.05 21424.54 1580.43
[2,] 12977.94 6375.45 9082.19 1746.19
[3,] 15443.81 1692.41 2463.45 3810.00
[4,] 17182.07 32303.31 48241.69 2781.65
[5,] 6185.11 3676.55 3520.91 899.17
[6,] 28628.42 18052.67 12149.83 15819.15
[7,] 7022.26 1084.83 1577.66 1503.59
[8,] 23500.81 3898.30 4583.81 7217.92
[9,] 1930.00 6811.47 11712.52 464.85
      [,5]      [,6]
[1,] 9555.40 1567.28
[2,] 3386.61 1756.47
[3,] 368.08 965.23
[4,] 23944.84 3191.66
[5,] 827.48 824.84
[6,] 1501.99 9610.83
[7,] 210.36 452.21
[8,] 663.97 1933.72
[9,] 5173.66 946.75
> ALL_Rmixmod1@bestResult@criterion
[1] "BIC" "ICL" "NEC"
> ALL_Rmixmod1@bestResult@criterionValue
[1] 4.920908e+04 4.941137e+04 3.961451e-02
> ALL_Rmixmod1@bestResult@nbCluster
[1] 9
> ALL_Rmixmod1@bestResult@model
[1] "Gaussian_pk_Lk_Bk"

```

To investigate about the comparison of **mclust** and **Rmixmod**, let us give a look to the following cross-tabulation and ARI.

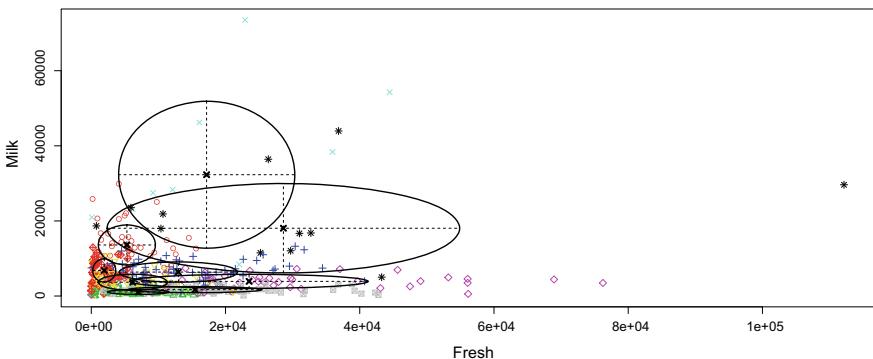
```
> table(ALL_Rmixmod1@bestResult@partition,
+        mod$classification)
   1 2 3 4 5 6 7 8
1 0 20 20 0 1 0 0 0
2 35 11 0 0 24 0 0 0
3 0 0 0 4 0 83 0 0
4 0 0 5 0 0 0 5 0
5 34 0 0 0 0 20 0 0
6 0 0 0 1 7 0 6 0
7 0 0 0 0 0 3 0 60
8 1 0 0 50 1 3 0 0
9 15 31 0 0 0 0 0 0
> round(adjustedRandIndex(ALL_Rmixmod1@bestResult
+                           @partition, mod$classification), 2)
[1] 0.58
```

We can observe that most of the units belonging to Clusters 4, 6, and 8 of the **mclust** solution belong to Clusters 8, 3, and 7 of the **Rmixmod** solution, respectively, while no further clear one-to-one relationships are visible. For instance, units belonging to Cluster 1 of the solution by **mclust** are splitted into Clusters 2, 5, and 9 of the solution by **Rmixmod**, as well as units belonging to Cluster 2 of the **mclust** solution are split into Clusters 1, 2, and 9 of the **Rmixmod** solution. The difference between the two solutions is also confirmed by the value of ARI (0.58). Note that the different solutions obtained by the packages **mclust** and **Rmixmod** highlight the impact of different initialization strategies on the final parameter estimates and the dependence on the starting values.

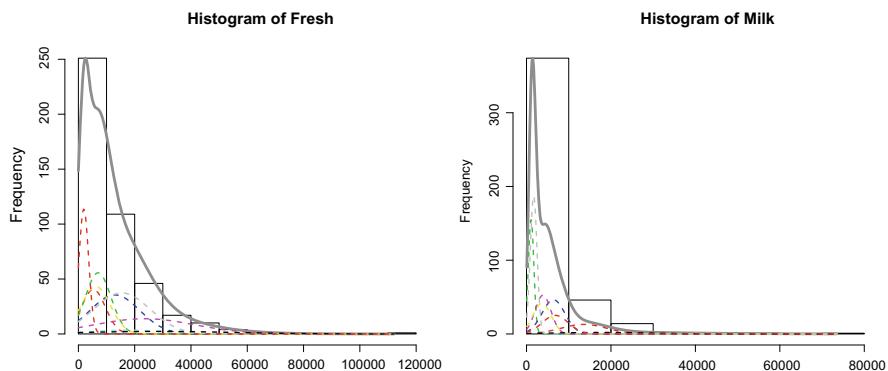
Finally, we may plot the obtained results by using `plot(ALL_Rmixmod1)` or `plotCluster(ALL_Rmixmod1["bestResult"])` which produce the same output but, for the input object, the former uses an object of class `Mixmod` while the latter an object of class `MixmodResults`. If we have a large number of variables, we can also show only part of them by specifying the following code to produce Fig. 6.10.

```
> plotCluster(ALL_Rmixmod1["bestResult"],
+              customers[, 3:8], variable1 = 1,
+              variable2 = 2)
```

Notice that different colors represent different clusters and ellipsoids are centered at the mean using the estimated parameters. Similarly, we can make histograms where, for each variable, component-specific and marginal densities are drawn (see Fig. 6.11).



**Fig. 6.10** customers data: classification on the plane (Milk, Fresh) corresponding to the best GMM ( $k = 9$ , model=Gaussian\_pk\_Lk\_Bk, BIC= 49209.08) fitted by mixmodCluster with strategy1. Different colors represent different clusters



**Fig. 6.11** customers data: histograms of the variables Fresh (on the left) and Milk (on the right) corresponding to the best GMM ( $k = 9$ , model=Gaussian\_pk\_Lk\_Bk, BIC= 49209.08) fitted by mixmodCluster with strategy1. Different colors represent different component-specific and marginal densities

```
> hist(ALL_Rmixmod1)
> # Or
> histCluster(ALL_Rmixmod1[ "bestResult" ] ,
+               customers[, 3:8], 1:2)
```

### 6.8.2 Case Studies with Categorical and Mixed Data

In the following two sections, we illustrate examples of model-based clustering for categorical and mixed data implemented via **Rmixmod**. It is worth to note that other packages for fitting LCA models are available, including **polLCA** [60]

and **BayesLCA** [100], among others. Finally, we also show the use of the package **clustMD** in the case of mixed data.

### 6.8.2.1 Case Study with Categorical Data

The dataset **birds** [14] is available in the package **Rmixmod** and it contains details on the morphology of  $n = 69$  birds (puffins). Specifically, each bird is described by  $p = 5$  categorical features:

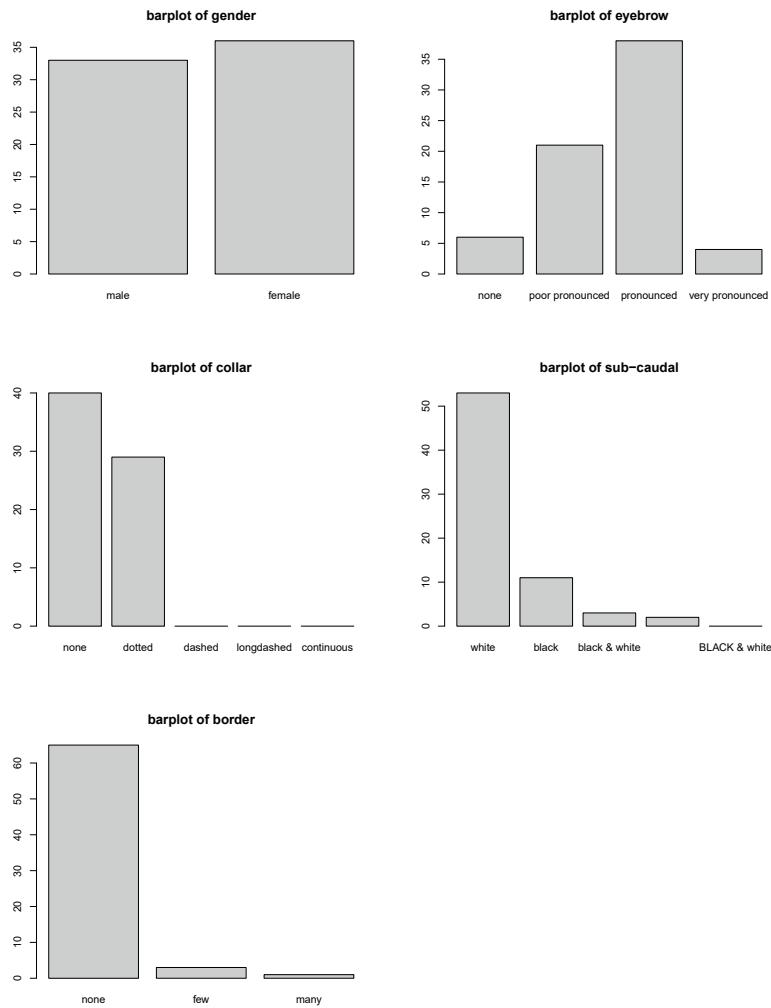
- Gender (male, female);
- Eyebrow (none, poorly pronounced, pronounced, very pronounced);
- Collar (none, dotted, dashed, longdashed, continuous);
- Sub-caudal (white, black, black and white, black and WHITE, BLACK and white);
- Border (none, few, many).

```
> data("birds")
> str(birds)
'data.frame': 69 obs. of 5 variables:
 $ gender     : Factor w/ 2 levels "male", "female":...
 $ eyebrow    : Factor w/ 4 levels "none", "poor...
 $ collar     : Factor w/ 5 levels "none", "dotted", ...
 $ sub-caudal: Factor w/ 5 levels "white", "black", ...
 $ border     : Factor w/ 3 levels "none", "few", ...
> colnames(birds)[4] = "sub.caudal"
```

Moreover, information on bird species is provided: 34 are *Iherminieri* and 35 are *subalaris*. Figure 6.12 shows the barplot of the morphological features obtained by the following code.

```
> attach(birds)
> par(mfrow = c(3, 2))
> barplot(table(gender), main = "barplot of gender")
> barplot(table(eyebrow), main = "barplot of eyebrow")
> barplot(table(collar), main = "barplot of collar")
> barplot(table(sub.caudal),
+           main = "barplot of sub-caudal")
> barplot(table(border), main = "barplot of border")
```

We start inspecting the data and looking at the potential importance that each feature has in assessing whether the puffin is *Iherminieri* or *subalaris*. For this purpose, we analyze the following cross-tabulations and  $\chi^2$  tests.



**Fig. 6.12** birds data: barplots of morphological features

```
> birds.class <- as.integer(c(rep(1, 34), rep(2, 35)))
> table(gender, birds.class)
      birds.class
gender 1 2
male   18 15
female 16 20
> round(chisq.test(gender, birds.class)$p.value, 4)
[1] 0.5502
> table(eyebrow, birds.class)
      birds.class
eyebrow 1 2
```

```

none          6   0
poor pronounced 21   0
pronounced      7 31
very pronounced  0   4
> chisq.test(eyebrow, birds.class)$p.value
[1] 5.261924e-10
> table(collar, birds.class)
      birds.class
collar      1   2
none        5 35
dotted      29  0
dashed       0  0
longdashed   0  0
continuous   0  0
> chisq.test(collar, birds.class)$p.value
[1] 4.144118e-12
> table(sub.caudal, birds.class)
      birds.class
sub.caudal    1   2
white        19 34
black        11  0
black & white  3   0
black & WHITE  1   1
BLACK & white  0   0
> round(chisq.test(sub.caudal, birds.class)$p.value, 4)
[1] 0.0004
> table(border, birds.class)
      birds.class
border     1   2
none      32 33
few       2   1
many      0   1
> round(chisq.test(border, birds.class)$p.value, 4)
[1] 0.5131

```

The *p*-values suggest that `eyebrow`, `collar`, and `sub.caudal` could be important factors for assessing the group of each bird.

Assuming that the bird species is unknown, our task is to allocate birds to meaningful clusters based on their morphological features by means of the LCA model described in Sect. 6.7.1. Once obtained the partition, we will verify its link with the species. For categorical data, the argument `models` in the function `mixmodCluster` of **Rmixmod** allows for specifying 10 multinomial models (summarized in Table 6.3) which can be viewed by the command `mixmodMultinomialModel()`. By default `Binary_pk_Ekjh` is considered. As in the case of `mixmodGaussianModel`, the function `mixmodMultinomialModel` allows to specify a different model or a list of models to run by using the arguments `listModels`, `variable.independency`, `free.proportions`, `equal.proportions`, and `component.independency`. The last two arguments allow to include models which are independent of a certain variable or component.

We fit all the available models for  $G = 1:10$  considering all the available information criteria (BIC, ICL, and NEC), `strategy1` and `strategy2` defined in Sect. 6.8.1.2.

```
> strategy1 <- mixmodStrategy(nbTry = 100)
> birds.out1 <- mixmodCluster(birds,
+                               nbCluster = 1:10,
+                               dataType = "qualitative",
+                               model = mixmodMultinomialModel(),
+                               strategy = strategy1,
+                               criterion = c("BIC", "ICL", "NEC"))
> strategy2 <- mixmodStrategy(initMethod = "random",
+                               nbTry = 100)
> birds.out2 <- mixmodCluster(birds,
+                               nbCluster = 1:10,
+                               dataType = "qualitative",
+                               model = mixmodMultinomialModel(),
+                               strategy = strategy2,
+                               criterion = c("BIC", "ICL", "NEC"))
```

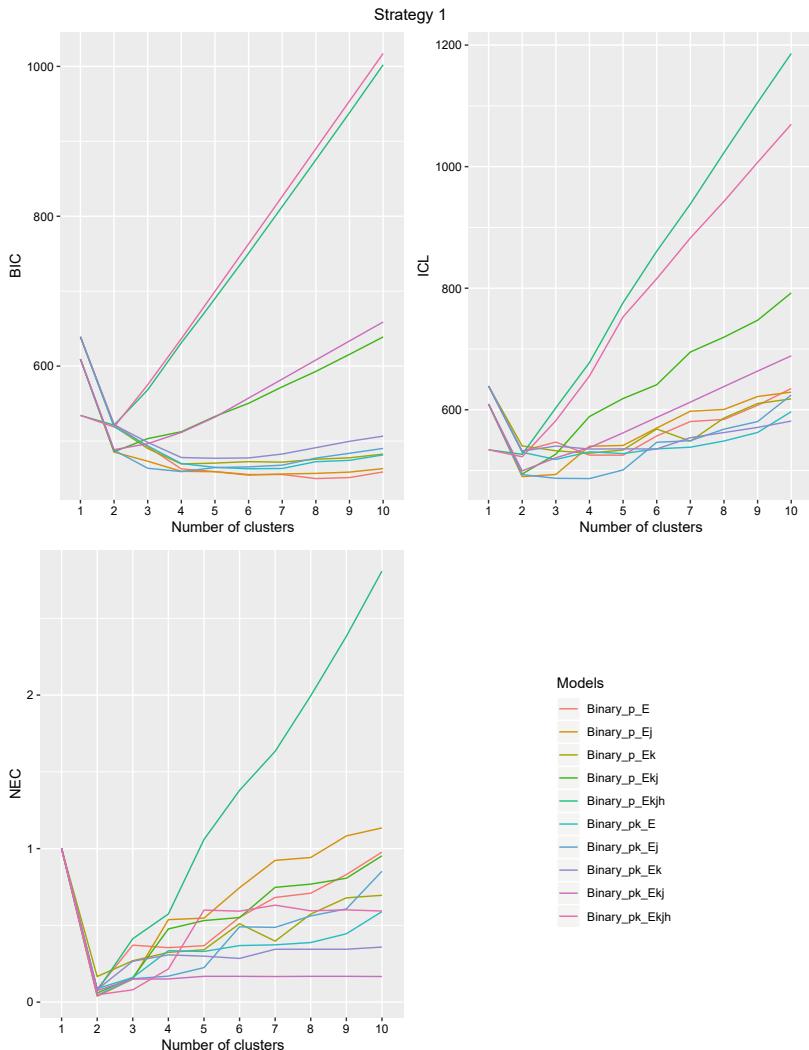
Let us give a look at the results obtained from both strategies by the following commands.

```
> birds.out1@bestResult@nbCluster
[1] 8
> birds.out1@bestResult@model
[1] "Binary_p_E"
> ICL1 <- sortByCriterion(birds.out1, "ICL")
> ICL1@bestResult@nbCluster
[1] 4
> ICL1@bestResult@model
[1] "Binary_pk_Ej"
> NEC1 <- sortByCriterion(birds.out1, "NEC")
> NEC1@bestResult@nbCluster
[1] 2
> NEC1@bestResult@model
[1] "Binary_p_Ej"
> birds.out2@bestResult@nbCluster
[1] 8
> birds.out2@bestResult@model
[1] "Binary_p_E"
> ICL2 <- sortByCriterion(birds.out2, "ICL")
> ICL2@bestResult@nbCluster
[1] 4
> ICL2@bestResult@model
[1] "Binary_pk_Ej"
> NEC2 <- sortByCriterion(birds.out2, "NEC")
> NEC2@bestResult@nbCluster
[1] 2
> NEC2@bestResult@model
```

```
[1] "Binary_p_Ej"
> round(birds.out1@bestResult@criterionValue, 2)
[1] 449.92 584.14 0.71
> round(birds.out2@bestResult@criterionValue, 2)
[1] 449.92 584.14 0.71
> round(ICL1@bestResult@criterionValue, 2)
[1] 459.22 486.81 0.17
> round(ICL2@bestResult@criterionValue, 2)
[1] 459.22 486.80 0.11
> round(NEC1@bestResult@criterionValue, 2)
[1] 485.27 490.05 0.04
> round(NEC2@bestResult@criterionValue, 2)
[1] 485.27 490.05 0.04
```

The two initialization strategies lead to similar results; however, different information criteria suggest different solutions. Specifically, BIC suggests the model `Binary_p_E` with  $k = 8$  clusters, NEC suggests the model `Binary_p_Ej` with  $k = 2$  clusters, while ICL selects the model `Binary_p_k_E_j` with  $k = 4$  clusters. Moreover, the best solution obtained by NEC (i.e.,  $k = 2$  clusters) has an ICL value of 490.05 which is very close to the value 486.80 for the best solution obtained by ICL (i.e.,  $k = 4$  clusters). For reasons of parsimony, we prefer the one with  $k = 2$  clusters corresponding to the model `Binary_p_Ej`. On the other hand, the best BIC values (449.92) corresponding to the solution with  $k = 8$  clusters are very close to the BIC value corresponding to the best solution obtained by ICL (459.22) with  $k = 4$  clusters. Therefore, for reasons of parsimony, we prefer the one with  $k = 4$  clusters corresponding to the model `Binary_p_k_E_j`. To confirm our conclusion, we can graphically visualize all the information criteria (BIC, ICL, NEC) values for all the models by using `strategy1` (Fig. 6.13). The corresponding code follows.

```
> G <- attr(birds.out1, "nbCluster")
> modelN <- attr(birds.out1, "models") [1]
> Criteria <- data.frame(G = rep(G, 3*length(modelN)),
+                           model = rep(modelN,
+                                       each = length(G)),
+                           BIC = NA, ICL = NA, NEC = NA,
+                           stringsAsFactors = FALSE)
> for (i in 1:length(birds.out1@results)){
+   Criteria[i, 1] = birds.out1@results[[i]]@nbCluster
+   Criteria[i, 2] = birds.out1@results[[i]]@model
+   Criteria[i, 3:5] =
+     birds.out1@results[[i]]@criterionValue
+ }
> library(gridExtra)
> lx <- xlab("Number of clusters")
> g_legend <- function(a.gplot){
+   tmp <- ggplot_gtable(ggplot_build(a.gplot))
+   leg <- which(sapply(tmp$grobs,
+                      function(x) x$name) == "guide-box")
+   legend <- tmp$grobs[[leg]]}
```



**Fig. 6.13** birds data: BIC, ICL, and NEC plots for the LCA models fitted by `mixmodCluster` with `strategy1` for  $k = 1, \dots, 10$

```

+           return(legend)      }
> xscale <- scale_x_discrete(limits = c("1", "2", "3",
+                                         "4", "5", "6", "7", "8", "9", "10"))
> A <- ggplot(Criteria, aes(Criteria[, 1],
+                           Criteria[, 3], color = Criteria[, 2])) +
+     stat_summary(aes(group = Criteria[, 2]), +
+                  fun = mean, geom = "line") +
+     ylab("BIC") + lx +
+ 
```

```

+           labs(color = 'Models') + xscale
> B <- ggplot(Criteria, aes(Criteria[, 1],
+                             Criteria[, 4], color = Criteria[, 2])) +
+     stat_summary(aes(group = Criteria[, 2]), fun =
+     mean, geom = "line") +
+     ylab("ICL") + lx +
+     labs(color = 'Models') + xscale
> C <- ggplot(Criteria, aes(Criteria[, 1],
+                             Criteria[, 5], color = Criteria[, 2])) +
+     stat_summary(aes(group = Criteria[, 2]), fun =
+     mean, geom = "line") +
+     ylab("NEC") + lx +
+     labs(color = 'Models') + xscale
> mylegend <- g_legend(A)
> grid.arrange(A + theme(legend.position = "none"), B +
+                 theme(legend.position = "none"), C +
+                 theme(legend.position = "none"),
+                 nrow = 2, top = "Strategy 1", mylegend)

```

Thus, we are going to analyze the solutions with  $k = 2$  and  $k = 4$  clusters for models `Binary_p_Ej` and `Binary_pk_Ej`, respectively, in order to reach a final decision on the best solution. The corresponding two-dimensional representations are shown in Figs. 6.14 and 6.15 which are obtained through the following code.

```

> plot(NEC1)
> legend("bottomleft", c("Cluster1", "Cluster2"),
+         col = c(2, 3), pch = c(1, 2))
> plot(ICL1)
> legend("bottomleft", c("Cluster1", "Cluster2",
+                         "Cluster3", "Cluster4"), col = c(2, 3, 4, 5),
+         pch = c(1, 2, 3, 4))

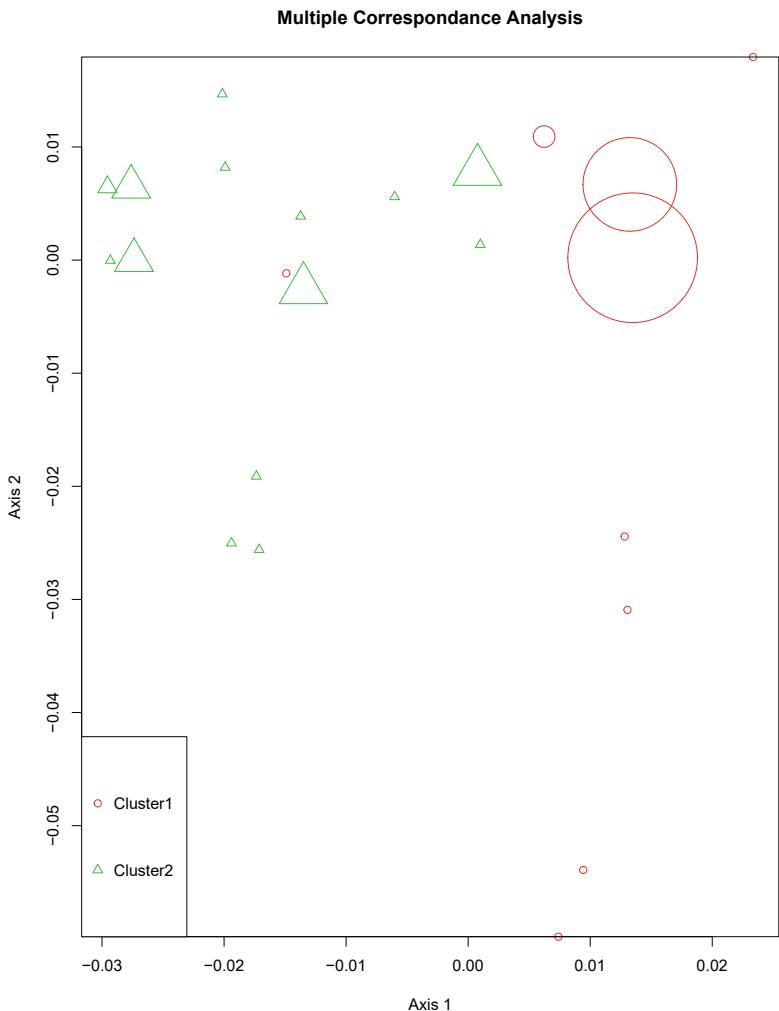
```

Notice that these representations are provided by performing Multiple Correspondence Analysis (MCA): bigger symbols mean that units are more similar. To further investigate, we look at the cluster-specific center vectors and scatter matrices as follows.

```

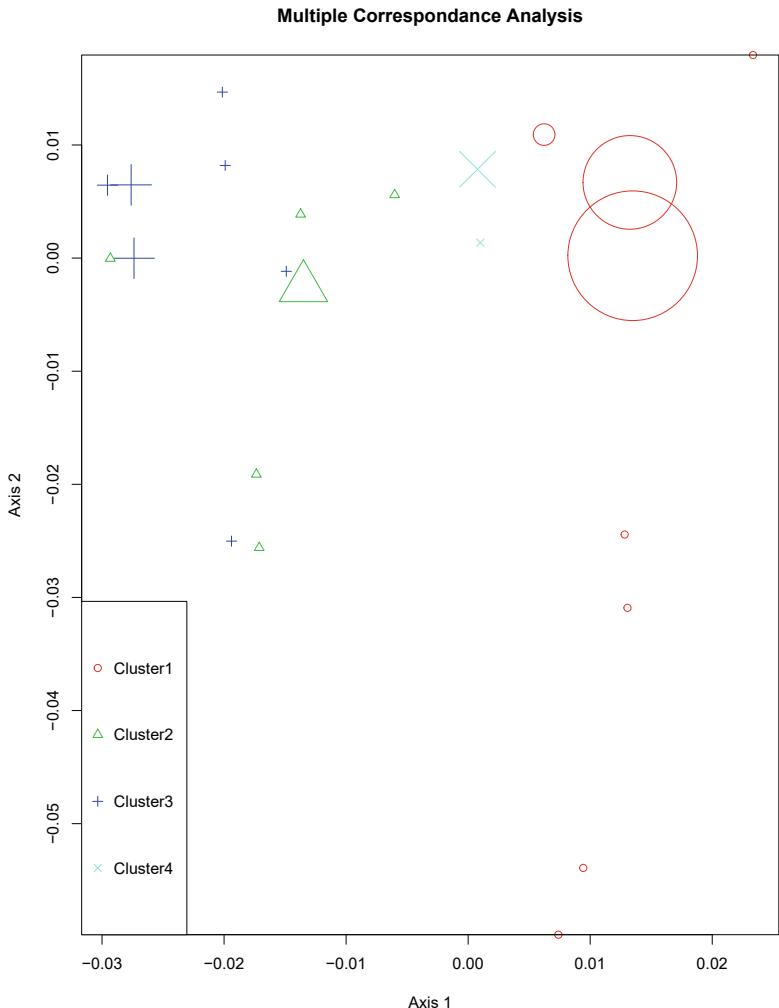
> NEC1@bestResult@parameters@center
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,]      2      3      1      1      1
[2,]      1      2      2      1      1
> NEC1@bestResult@parameters@scatter
[[1]]
      [,1]  [,2]  [,3]
[1,]  0.468  0.468  0.009
[2,]  0.085  0.085  0.255
[3,]  0.030  0.008  0.008
[4,]  0.248  0.062  0.062

```



**Fig. 6.14** *birds* data: classification on the plane of the first two components resulting from MCA corresponding to the NEC-best LCA model ( $k = 2$ , model=Binary\_p\_Ej, NEC=449.92) fitted by mixmodCluster with strategy1

```
[5,]  0.075  0.038  0.038
      [,4]  [,5]
[1,]  0.000  0.000
[2,]  0.085  0.000
[3,]  0.008  0.008
[4,]  0.062  0.062
[5,]  0.000  0.000
[[2]]
      [,1]  [,2]  [,3]
```



**Fig. 6.15** *birds* data: classification on the space of the first two components resulting from MCA corresponding to the ICL-best LCA model ( $k = 4$ , `model=Binary_p_k_E_j`,  $ICL=459.22$ ) fitted by `mixmodCluster` with strategy

```
[1,] 0.468 0.468 0.000
[2,] 0.085 0.255 0.085
[3,] 0.008 0.030 0.007
[4,] 0.248 0.062 0.062
[5,] 0.075 0.038 0.038
[ ,4] [ ,5]
[1,] 0.000 0.000
[2,] 0.085 0.000
[3,] 0.008 0.008
```

```
[4,] 0.062 0.062
[5,] 0.000 0.000
> table(NEC1@bestResult@partition)
 1 2
40 29
> ICL1@bestResult@parameters@center
 [,1] [,2] [,3] [,4] [,5]
[1,] 2     3     1     1     1
[2,] 2     2     2     1     1
[3,] 1     2     2     2     1
[4,] 1     3     2     1     1
> ICL1@bestResult@parameters@scatter
[[1]]
 [,1]   [,2]   [,3]
[1,] 0.425 0.425 0.000
[2,] 0.066 0.066 0.199
[3,] 0.061 0.015 0.015
[4,] 0.128 0.032 0.032
[5,] 0.091 0.046 0.046
 [,4]   [,5]
[1,] 0.000 0.000
[2,] 0.066 0.000
[3,] 0.015 0.015
[4,] 0.032 0.032
[5,] 0.000 0.000
[[2]]
 [,1]   [,2]   [,3]
[1,] 0.425 0.425 0.000
[2,] 0.066 0.199 0.066
[3,] 0.015 0.061 0.015
[4,] 0.128 0.032 0.032
[5,] 0.091 0.046 0.046
 [,4]   [,5]
[1,] 0.000 0.000
[2,] 0.066 0.000
[3,] 0.015 0.015
[4,] 0.032 0.032
[5,] 0.000 0.000
[[3]]
 [,1]   [,2]   [,3]
[1,] 0.425 0.425 0.000
[2,] 0.066 0.199 0.066
[3,] 0.015 0.061 0.015
[4,] 0.032 0.128 0.032
[5,] 0.091 0.046 0.046
 [,4]   [,5]
[1,] 0.000 0.000
[2,] 0.066 0.000
[3,] 0.015 0.015
[4,] 0.032 0.032
[5,] 0.000 0.000
[[4]]
 [,1]   [,2]   [,3]
```

```
[1,] 0.425 0.425 0.000
[2,] 0.066 0.066 0.199
[3,] 0.015 0.061 0.015
[4,] 0.128 0.032 0.032
[5,] 0.091 0.046 0.046
[,4] [,5]
[1,] 0.000 0.000
[2,] 0.066 0.000
[3,] 0.015 0.015
[4,] 0.032 0.032
[5,] 0.000 0.000
> table(ICL1@bestResult@partition)
 1 2 3 4
39 10 14 6
```

These results are summarized in Fig. 6.16 (solution corresponding to  $k = 2$  clusters) and Fig. 6.17 (solution corresponding to  $k = 4$  clusters) representing, for each variable, the cluster-specific probabilities to have a given modality, obtained by the following code.

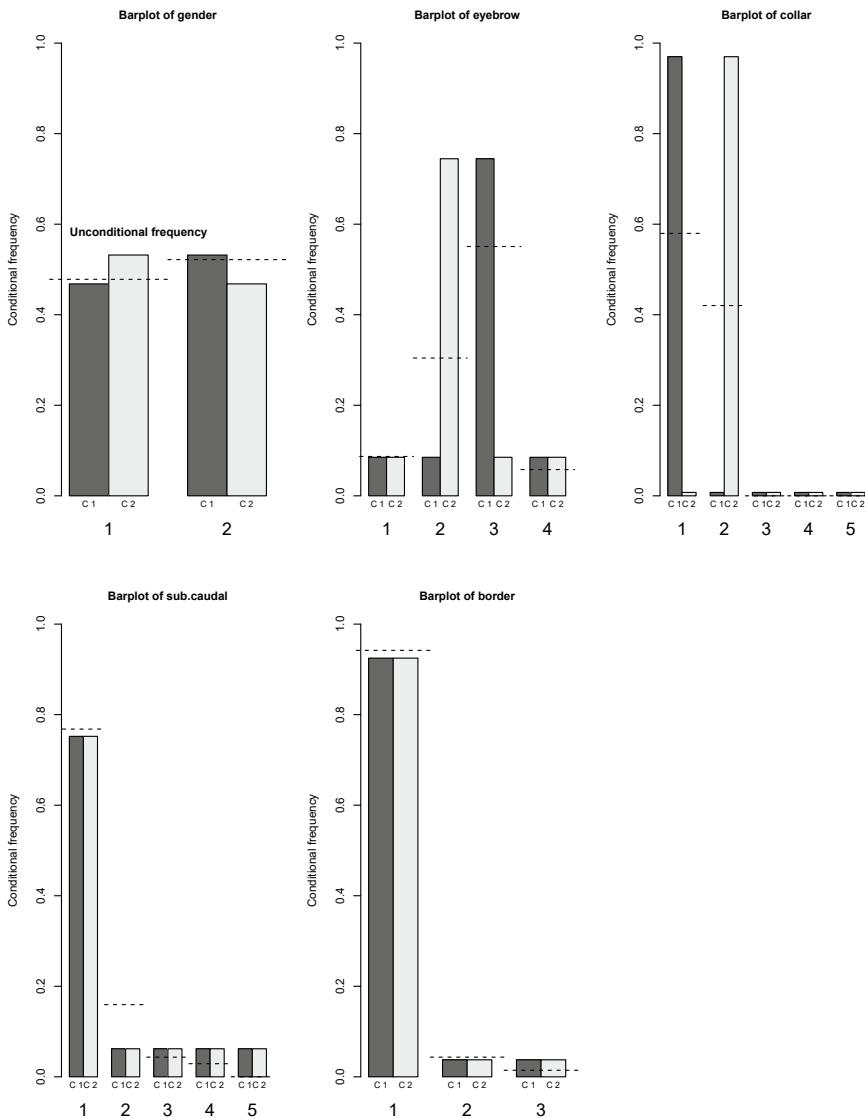
```
> barplot(NEC1)
> barplot(ICL1)
```

As far as the solution with  $k = 2$  clusters is concerned,  $n_1 = 40$  and  $n_2 = 29$  birds, variables `sub-caudal` and `border` do not seem to be able to discriminate between the two clusters: in both clusters birds have high probabilities to be white ( $1 - 0.25 = 0.75$ ) and not to have a border ( $1 - 0.08 = 0.92$ ). Cluster 1 has a slightly higher probability than Cluster 2 to include females ( $1 - 0.47 = 0.53$ ), a high probability to contain birds with pronounced eyebrow ( $1 - 0.26 = 0.74$ ) and, especially, no collar ( $1 - 0.03 = 0.97$ ). On the other hand, Cluster 2 has a slightly higher probability than Cluster 1 to include males ( $1 - 0.47 = 0.53$ ), a high probability to contains birds with dotted collar ( $1 - 0.03 = 0.97$ ) and poorly pronounced eyebrow ( $1 - 0.26 = 0.74$ ).

To check whether the obtained partition has links with bird species, we report the following cross-tabulations and the corresponding ARI.

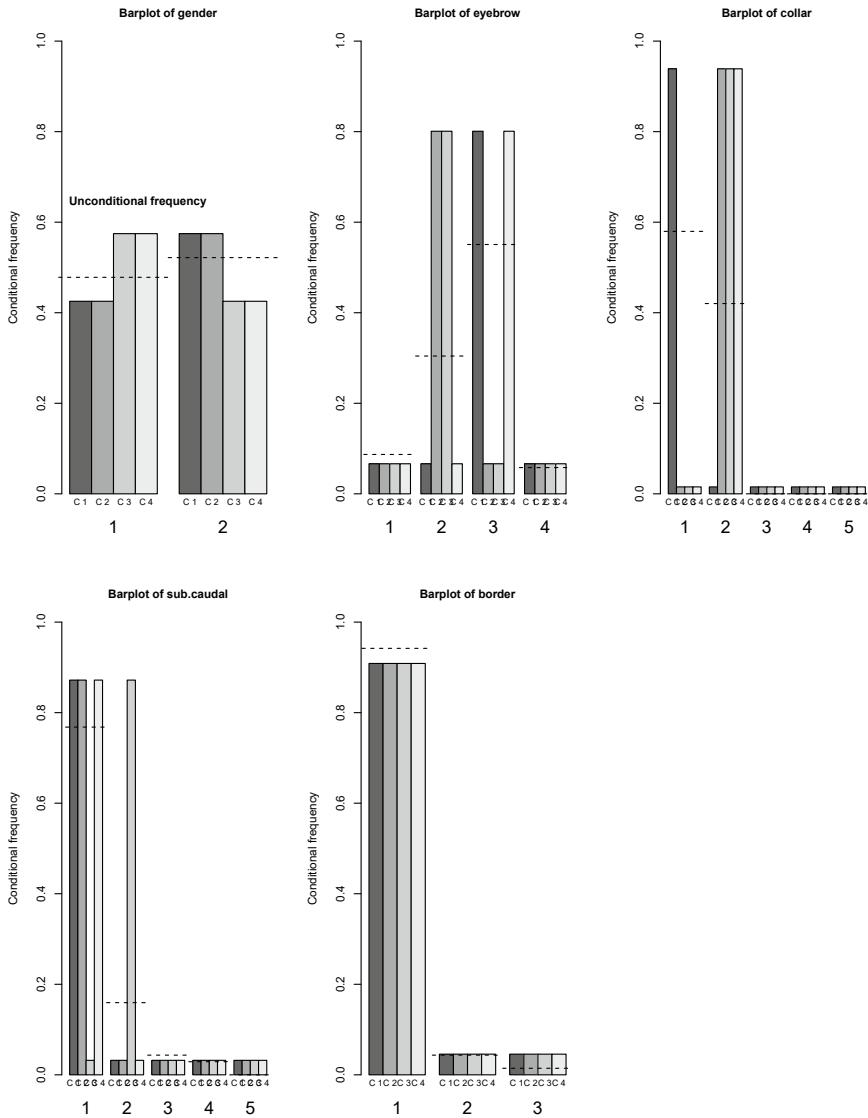
```
> table(NEC1@bestResult@partition, birds.class)
   birds.class
     1   2
  1 35  5
  2  0 29
> round(adjustedRandIndex(NEC1@bestResult@partition,
+                           birds.class), 2)
[1] 0.73
```

As it can be observed, Cluster 1 is mostly formed by *subalatis* (35 out of 40) and all birds belonging to Cluster 2 are *lherminieri*. We can conclude that the two clusters' solution recovers pretty well the bird species using the  $p = 5$  morphological features.



**Fig. 6.16** birds data: barplots for the NEC-best LCA model ( $k = 2$ , `model=Binary_p_Ej`,  $\text{NEC}=449.92$ ) fitted by `mixmodCluster` with `strategy1`

Looking at the solution with  $k = 4$  clusters, containing  $n_1 = 39$ ,  $n_2 = 10$ ,  $n_3 = 14$ , and  $n_4 = 6$  birds, we confirm that variable `border` does not help to discriminate between the  $k = 4$  clusters. Moreover, Cluster 1 is mostly formed by birds belonging to Cluster 1 in the previous solution (39 out of 40 birds). Cluster 2 has the same probability to be formed by females as Cluster 1 ( $1 - 0.43 = 0.57$ ) but it has a higher



**Fig. 6.17** birds data: barplots for the ICL-best LCA model ( $k = 4$ , `model=Binary_p_k_E_j`, `ICL=459.22`) fitted by `mixmodCluster` with `strategy1`

probability to include birds with poorly pronounced eyebrow ( $1 - 0.20 = 0.80$ ) and dotted collar ( $1 - 0.06 = 0.94$ ). Clusters 3 and 4 have the same probability to include males ( $1 - 0.43 = 0.57$ ), but Cluster 3 has a higher probability to include birds with poorly pronounced eyebrow when compared to Cluster 2 ( $1 - 0.20 = 0.80$ ) and it is more characterized by birds with black sub-caudal ( $1 - 0.12 = 0.88$ ). In contrast,

Cluster 4 has a higher probability to include birds with pronounced eyebrow than Cluster 1 ( $1 - 0.20 = 0.80$ ) and it is characterized by birds with white sub-caudal ( $1 - 0.12 = 0.88$ ). We report below the following cross-tabulations and ARI values.

```
> table(ICL1@bestResult@partition,
+       NEC1@bestResult@partition)
   1   2
1 39  0
2  0 10
3  1 13
4  0  6
> round(adjustedRandIndex(ICL1@bestResult@partition,
+                           NEC1@bestResult@partition), 2)
[1] 0.73
> table(ICL1@bestResult@partition, birds.class)
  birds.class
   1   2
1  4 35
2 10  0
3 14  0
4  6  0
> round(adjustedRandIndex(ICL1@bestResult@partition,
+                           birds.class), 2)
[1] 0.54
```

As it can be observed, the solution with  $k = 4$  clusters splits Cluster 2 in the previous solution into three clusters. We may conclude that the solution with  $k = 4$  clusters is able to further characterize the lherminieri species by assuming a potential presence of lherminieri sub-species. To conclude, if the main interest is to recover the two puffin species, the solution with  $k = 2$  clusters represents a good choice. On the other hand, if the biological interest consists in discovering potential puffin sub-species, the solution with  $k = 4$  clusters may be preferred.

### 6.8.2.2 Case Study with Mixed Data

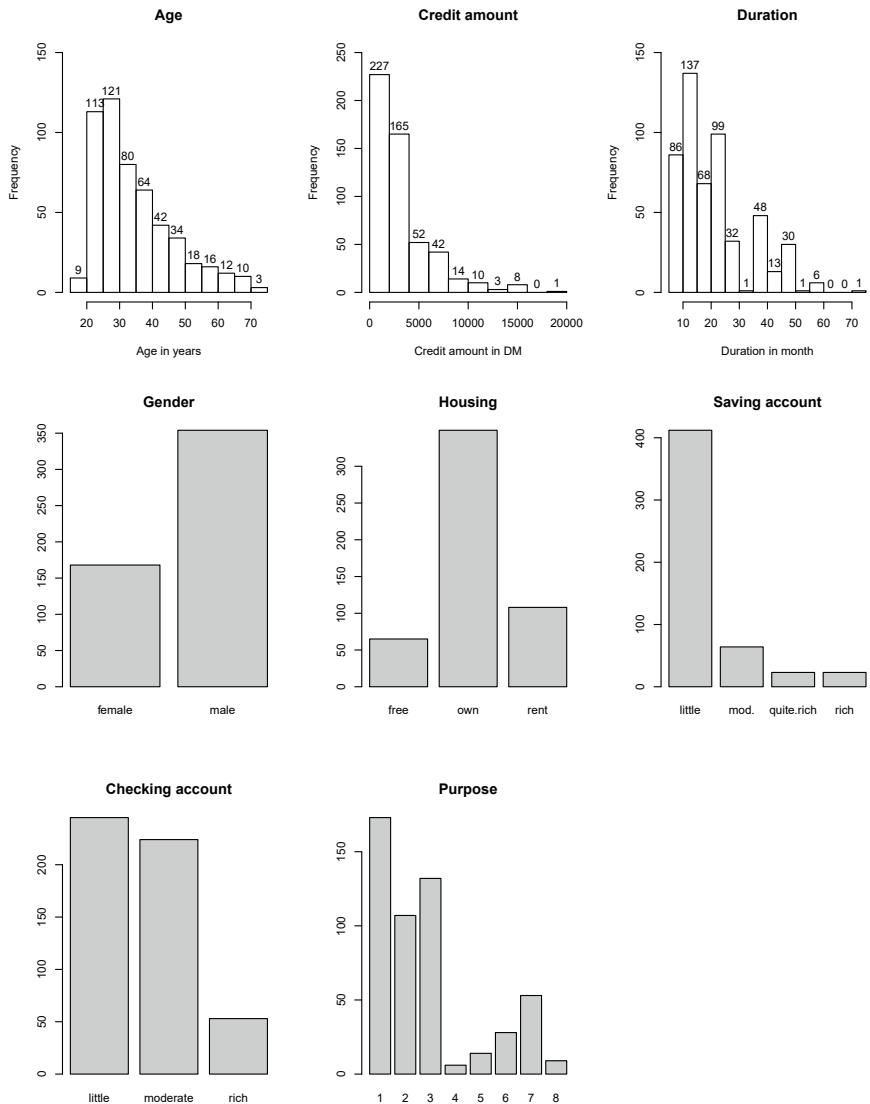
The original German credit risk data contains 1000 units with several mixed measurements [32]. In this dataset, each entry represents a person who takes a bank credit. Each person is either classified as a good or a bad customer according to her/his ability to repay. This information is described by a variable, class risk (1 stands for Good, 2 stands for Bad). A subset of this dataset, containing  $p = 8$  mixed measurements (three continuous, two ordinal, and three nominal) is considered here. The selected attributes are as follows:

- Age (in years);
- Credit amount (in Deutsch Mark, DM);
- Credit duration (in month);

- Gender: male, female;
- Housing: own, rent, free;
- Saving accounts: little < 100 DM, moderate ( $100 \leq \dots < 500$  DM), quite rich ( $500 \leq \dots < 1000$  DM), rich ( $\geq 1000$  DM);
- Checking account: little (< 0 DM), moderate ( $0 \leq \dots < 200$  DM), rich ( $\geq 200$  DM). It represents the status of the existing checking account;
- Purpose: car (1), furniture/equipment (2), radio/TV (3), domestic appliances (4), repairs (5), education (6), business (7), vacation/others (8).

The data frame `german` containing the eight mixed measurements and the class risk variable is available in the package **datasetsICR**. The aim of this study is to find clusters of customers with similar profiles and assess whether such groups are related to the levels good and bad. We remove all the records containing at least one NA value, obtaining a dataset with 522 customers. Moreover, we order the variables according to the type so that the continuous variables come first, the ordinal variables come second, and the nominal variables at the end, as it is needed for using the package **clustMD**. Figure 6.18 shows the histogram and the barplot for each variable. The corresponding code follows.

```
> data("german")
> colnames(german) [4] <- "Saving.acc"
> colnames(german) [5] <- "Checking.acc"
> colnames(german) [6] <- "Credit.amou"
> colnames(german) [9] <- "Class.risk"
> level.drop <- droplevels(german, exclude = "NA")
> German <- level.drop[complete.cases
+                         (level.drop), ]
> dim(German)
[1] 522    9
> German <- German[, c(1, 6, 7, 4:5, 2:3, 8, 9)]
> German$Saving.acc[German$Saving.acc ==
+                     'quite rich'] <- 'quite.rich'
> German$Saving.acc[German$Saving.acc ==
+                     'moderate'] <- 'mod.'
> German$Purpose[German$Purpose == 'car'] <- '1'
> German$Purpose[German$Purpose ==
+                  'furniture/equipment'] <- '2'
> German$Purpose[German$Purpose == 'radio/TV'] <- '3'
> German$Purpose[German$Purpose ==
+                  'domestic appliances'] <- '4'
> German$Purpose[German$Purpose == 'repairs'] <- '5'
> German$Purpose[German$Purpose == 'education'] <- '6'
> German$Purpose[German$Purpose == 'business'] <- '7'
> German$Purpose[German$Purpose ==
+                  'vacation/others'] <- '8'
> attach(German)
> par(mfrow = c(3, 3))
> hist(Age, main = "Age", label = TRUE,
+       ylim = c(0, 150), xlab = "Age in years")
```



**Fig. 6.18** german data: histograms for the continuous variables and barplots for the ordinal and nominal variables

```
> hist(Credit.amou, main = "Credit amount",
+       label = TRUE, ylim = c(0, 250),
+       xlab = "Credit amount in DM")
> hist(Duration, main = "Duration", label = TRUE,
+       ylim = c(0, 150), xlab = "Duration in month")
> barplot(table(Gender), main = "Gender")
> barplot(table(Housing), main = "Housing")
> barplot(table(Saving.acc), main = "Saving account")
```

```
> barplot(table(Checking.acc),
+          main = "Checking account")
> barplot(table(Purpose), main = "Purpose")
```

We standardize the continuous variables and encode the ordinal and the nominal variables as factors.

```
> German <- as.data.frame(German)
> German[, 1:3] <- scale(German[, 1:3])
> German[, 4] <- factor(German[, 4])
> German[, 5] <- factor(German[, 5])
> German[, 6] <- factor(German[, 6])
> German[, 7] <- factor(German[, 7])
> German[, 8] <- factor(German[, 8])
> German[, 9] <- factor(German[, 9])
```

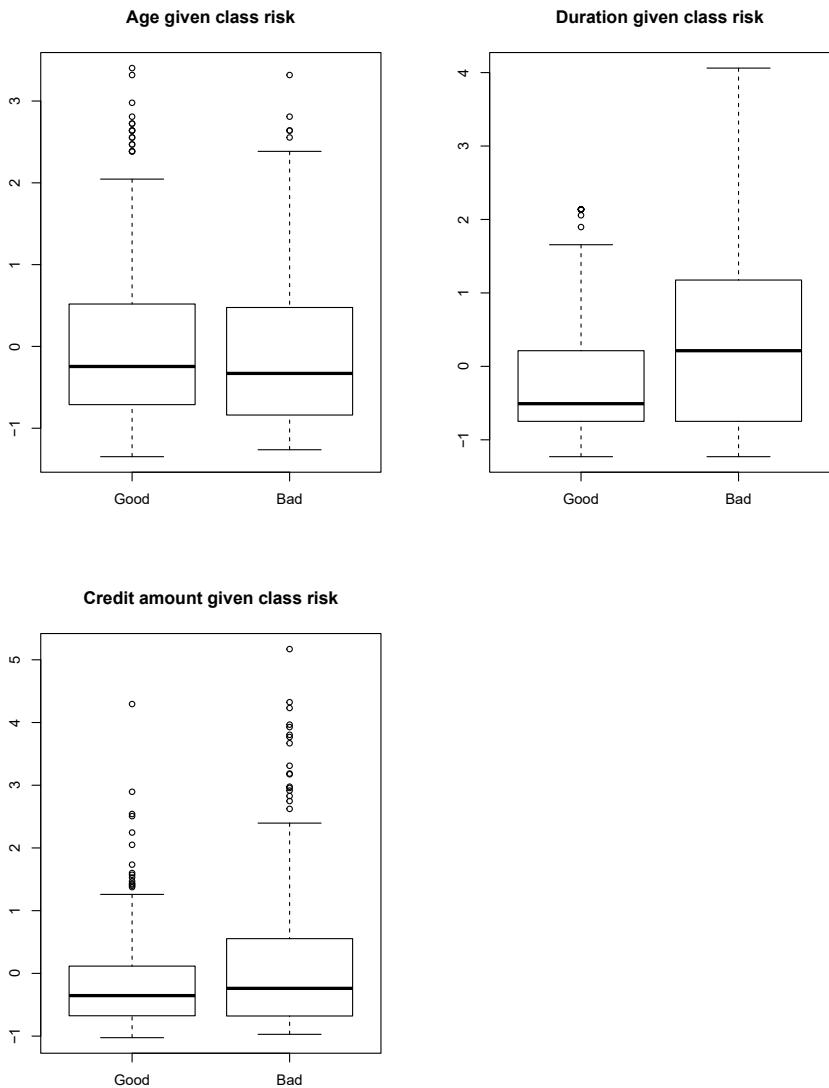
We get the boxplots in Fig. 6.19 for the continuous variables conditional on the variable `Class.risk` through the following code.

```
> attach(German)
> par(mfrow = c(2, 2))
> boxplot(Age ~ Class.risk,
+           main = "Age given class risk")
> boxplot(Duration ~ Class.risk,
+           main = "Duration given class risk")
> boxplot(Credit.amou ~ Class.risk,
+           main = "Credit amount given class risk")
> boxplot(Age ~ Class.risk, names = c("Good", "Bad"),
+           main = "Age given class risk")
```

As we can observe, the median value of Age for Bad customers is slightly lower than for Good; it might be premature to say young customers tend to have bad credit, but we can guess they tend to be riskier. The median value of Duration appears to be lower for the Good customers as compared to Bad customers, and the range appears to be stricter for Good customers when compared to Bad ones. Finally, we observe that Credit.amou for Bad customers is larger than for Good.

We report the following cross-tabulations and  $\chi^2$  tests for categorical variables.

```
> table(Gender, Class.risk)
      Class.risk
Gender    1     2
  female   86   82
  male    205  149
> round(chisq.test(Gender, Class.risk)$p.value, 4)
[1] 0.1771
```



**Fig. 6.19** german data: boxplots for the continuous variables given the variable `Class.risk`

```
> table(Housing, Class.risk)
      Class.risk
Housing 1   2
  free  29  36
  own 210 139
  rent  52  56
> round(chisq.test(Housing, Class.risk)$p.value, 4)
[1] 0.0138
```

```

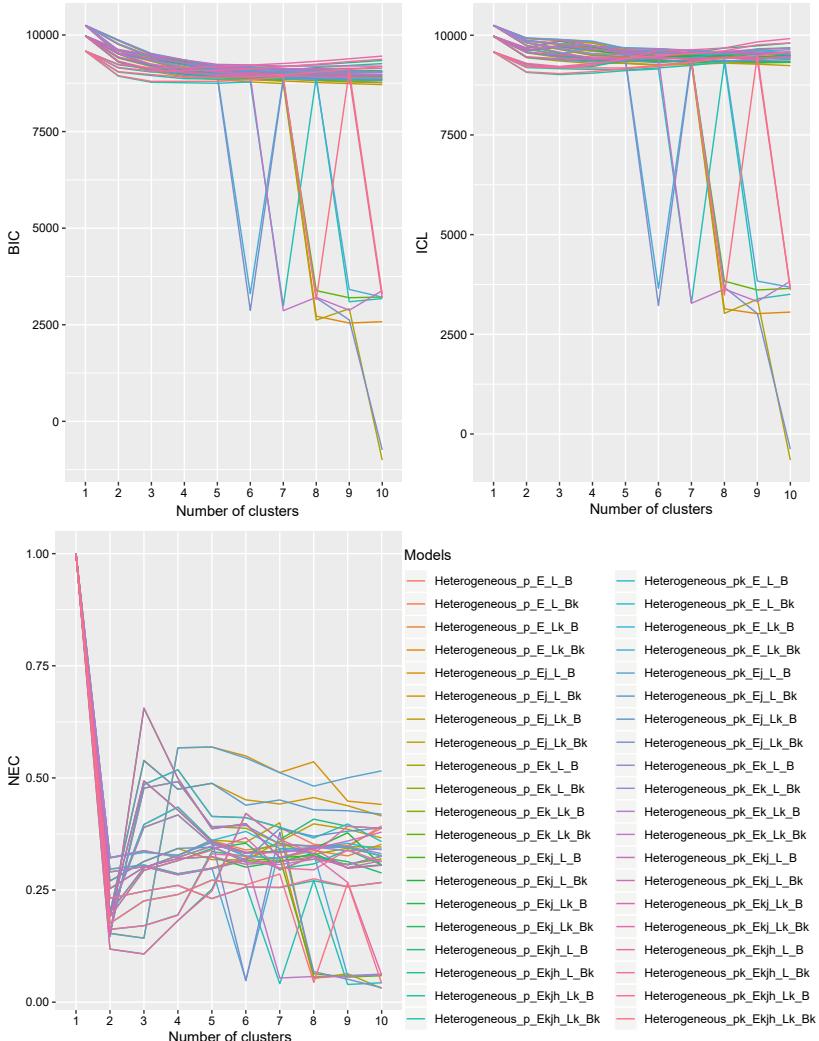
> table(Saving.acc, Class.risk)
      Class.risk
Saving.acc   1   2
  little    221 191
  mod.     34  30
  quite.rich 17   6
  rich     19   4
> round(chisq.test(Saving.acc, Class.risk)$p.value, 4)
[1] 0.0133
> table(Checking.acc, Class.risk)
      Class.risk
Checking.acc 1   2
  little    124 121
  moderate 126  98
  rich     41  12
> round(chisq.test(Checking.acc, Class.risk)$p.value,
+         4)
[1] 0.0018
> table(Purpose, Class.risk)
      Class.risk
Purpose 1   2
  1  93 80
  2  62 45
  3  81 51
  4  3   3
  5  8   6
  6 10  18
  7 29  24
  8  5   4
> round(chisq.test(Purpose, Class.risk)$p.value, 4)
[1] 0.4453

```

The observed  $p$ -values suggest that `Checking.acc`, `Saving.acc`, and `Housing` could be important to discriminate whether a customer is Good. For clustering purposes, we now ignore the information on the class risk and proceed to characterize different banking behavior profiles by means of the following two approaches.

- Heterogeneous mixture models applied to ordinal variables considered nominal and assuming local independence, according to the LCA model described in Sect. 6.7. For this purpose, the functions of the package **Rmixmod** are used. The argument `model` in `mixmodCluster` allows to define a list of 40 heterogeneous mixture models, where the covariance matrix for the Gaussian model can only be diagonal while the multinomial models are defined as in the categorical case. The complete list of such models can be viewed by means of the command `mixmodCompositeModel()`.
- `clustMD` models, described in Sect. 6.7.2, where ordinal variables are categorical manifestations of latent, univariate, Gaussian variables and nominal variables



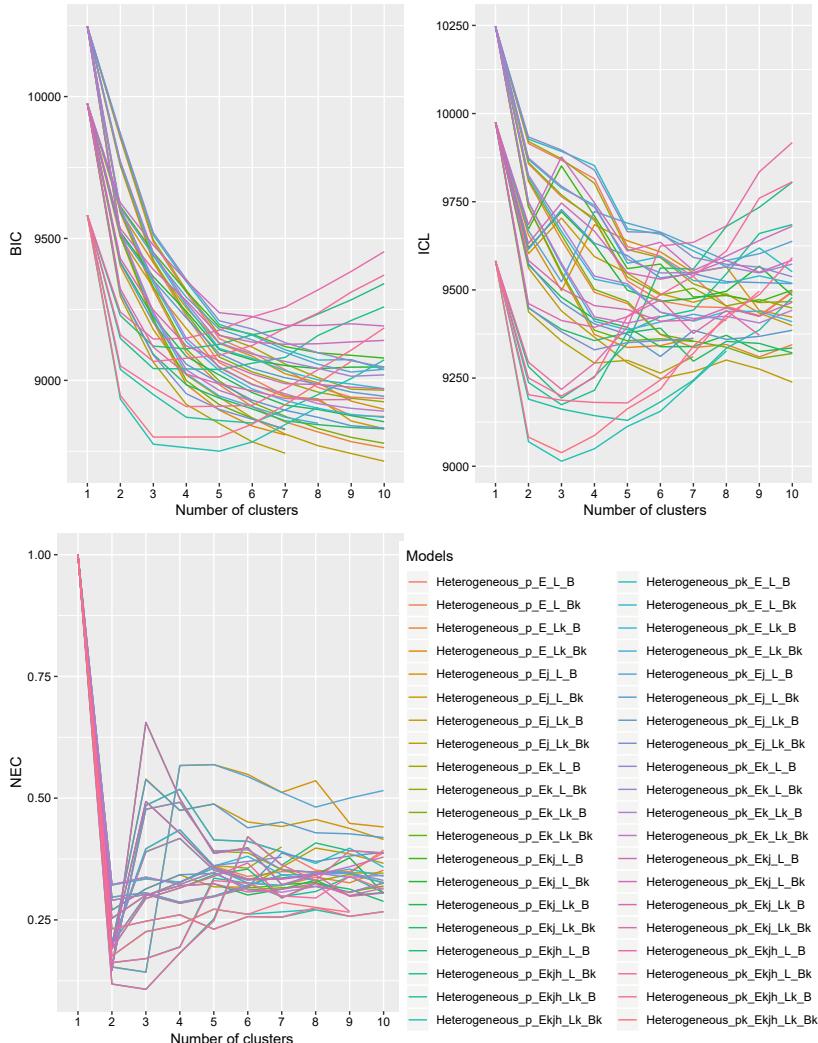


**Fig. 6.20** german data: BIC, ICL, and NEC plots for the 40 heterogeneous mixture models fitted by mixmodCluster with strategy1 for  $k = 1, \dots, 10$

```

+                               ("Heterogeneous_p_Ekjh_Lk_Bk"),
+                               strategy = strategy1,
+                               criterion = c("BIC", "ICL", "NEC"))
> out2@bestResult@criterionValue
[1] 8775.16 9014.75      0.23
> table(out2@bestResult@partition)
  1    2    3
134 238 150

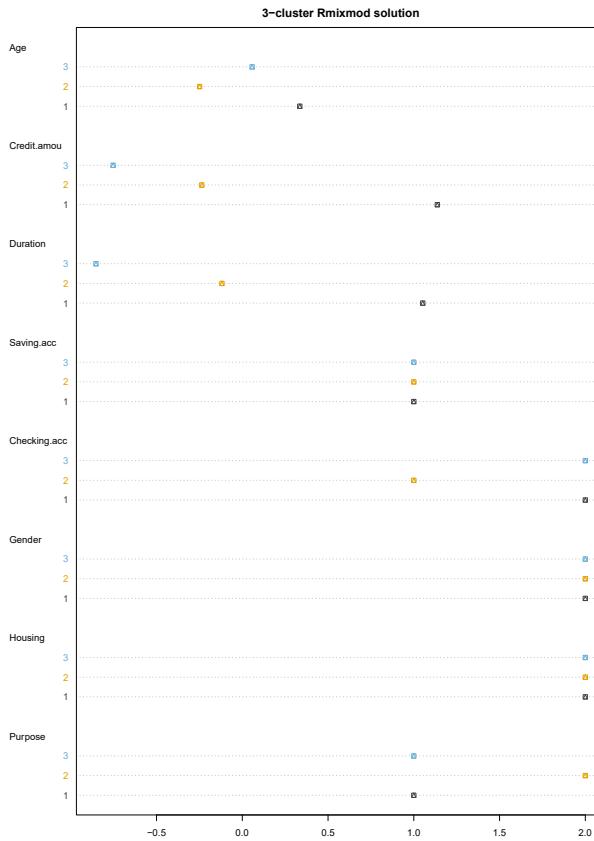
```



**Fig. 6.21** german data: corrected BIC, ICL, and NEC plots for the 40 heterogeneous mixture models fitted by mixmodCluster with strategy1 for  $k = 1, \dots, 10$

```
> out2@bestResult@parameters@m_parameter@center
  [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    2    2    1
[2,]    1    1    2    2    2
[3,]
> out2@bestResult@parameters@m_parameter@scatter
[[1]]
  [,1]   [,2]   [,3]   [,4]
```

```
[1,] 0.192 0.166 0.009 0.018
[2,] 0.426 0.466 0.040 0.000
[3,] 0.221 0.221 0.000 0.000
[4,] 0.282 0.439 0.157 0.000
[5,] 0.587 0.109 0.144 0.008
[,5] [,6] [,7] [,8]
[1,] 0.000 0.000 0.000 0.000
[2,] 0.000 0.000 0.000 0.000
[3,] 0.000 0.000 0.000 0.000
[4,] 0.000 0.000 0.000 0.000
[5,] 0.034 0.070 0.173 0.049
[[2]]
[,1] [,2] [,3] [,4]
[1,] 0.207 0.101 0.061 0.045
[2,] 0.461 0.345 0.116 0.000
[3,] 0.375 0.375 0.000 0.000
[4,] 0.063 0.347 0.284 0.000
[5,] 0.272 0.675 0.283 0.001
[,5] [,6] [,7] [,8]
[1,] 0.000 0.000 0.000 0.000
[2,] 0.000 0.000 0.000 0.000
[3,] 0.000 0.000 0.000 0.000
[4,] 0.000 0.000 0.000 0.000
[5,] 0.007 0.018 0.090 0.005
[[3]]
[,1] [,2] [,3] [,4]
[1,] 0.246 0.116 0.056 0.073
[2,] 0.401 0.545 0.144 0.000
[3,] 0.342 0.342 0.000 0.000
[4,] 0.069 0.208 0.139 0.000
[5,] 0.661 0.112 0.311 0.034
[,5] [,6] [,7] [,8]
[1,] 0.000 0.000 0.000 0.000
[2,] 0.000 0.000 0.000 0.000
[3,] 0.000 0.000 0.000 0.000
[4,] 0.000 0.000 0.000 0.000
[5,] 0.053 0.095 0.049 0.007
> out2@bestResult@parameters@g_parameter@mean
[,1] [,2] [,3]
[1,] 0.335 1.137 1.052
[2,] -0.249 -0.236 -0.119
[3,] 0.057 -0.753 -0.853
> out2@bestResult@parameters@g_parameter@variance
[[1]]
[,1] [,2] [,3]
[1,] 1.326 0.000 0.000
[2,] 0.000 1.434 0.000
[3,] 0.000 0.000 1.090
[[2]]
[,1] [,2] [,3]
[1,] 0.532 0.000 0.000
[2,] 0.000 0.114 0.000
[3,] 0.000 0.000 0.357
```



**Fig. 6.22** german data: the mean and center values for the Heterogeneous\_p\_Ekjh\_Lk\_Bk model with  $k = 3$  ( $\text{BIC}=8775.16$ ) fitted by mixmodCluster with strategy1

```
[ [ 3 ] ]
      [,1]     [,2]     [,3]
[1, ] 1.190    0.000    0.000
[2, ] 0.000    0.018    0.000
[3, ] 0.000    0.000    0.065
> means3 <- matrix(NA, ncol(German[, 1:8]), 3)
> means3[1:3, ] <- t(out2@bestResult@
+                           parameters@g_parameter@mean)
> means3[4:8, ] <- t(out2@bestResult@
+                           parameters@m_parameter@center)
> rownames(means3) <- colnames(German[, 1:8])
> dotchart(t(means3), col = 1:3, pch = 19)
```

These results are summarized in Fig. 6.22 where the mean and center values of the continuous and categorical variables are reported. It can be seen that the center

values for the  $k = 3$  clusters (with size  $n_1 = 238, n_2 = 134, n_3 = 150$ ) are approximately the same for variables `Saving.acc`, `Gender`, and `Housing`, meaning that these variables do not seem to discriminate between the three clusters. All clusters mainly contain customers with little saving accounts (in Clusters 1 and 2,  $1 - 0.19 = 0.80$ ), males (Cluster 1:  $1 - 0.22 = 0.78$ , Cluster 2:  $1 - 0.37 = 0.63$ ; Cluster 3:  $1 - 0.34 = 0.66$ ) and with owned house (Cluster 1:  $1 - 0.44 = 0.56$ ; Cluster 2:  $1 - 0.37 = 0.63$ ; Cluster 3:  $1 - 0.34 = 0.66$ ). The center values for `Checking.acc` and `Purpose` are the same for Clusters 1 and 3 and differ only for Cluster 2: Clusters 1 and 3 are most likely characterized by moderate checking account (Cluster 1:  $1 - 0.47 = 0.53$ ; Cluster 3:  $1 - 0.55 = 0.45$ ) and mainly asking for credit to buy furniture/equipment (Cluster 1:  $1 - 0.59 = 0.41$  Cluster 3:  $1 - 0.66 = 0.34$ ), while Cluster 2 is most likely characterized by customers with little checking account ( $1 - 0.47 = 0.53$ ), asking for credit to buy car ( $1 - 0.47 = 0.53$ ). On the other hand, the continuous variables (`Age`, `Credit.amou`, and `Duration`) have mean values that vary across clusters suggesting they are rather important to discriminate behavior banking profiles. In particular, Cluster 1 is characterized by old customers with high credit amount and duration of credit, while Cluster 2 by young customers with average credit amount and duration of credit. Finally, Cluster 3 contains middle-age customers with low amount and duration of credit.

We now move to analyze the data by means of the package **clustMD**. For this purpose, we use the function `clustMDparallel` of the package **clustMD** which allows to run multiple `clustMD` models in parallel. The most important arguments are the following.

- X: data matrix where the continuous variables come first, the binary (coded 1 and 2) and ordinal variables (coded 1, 2, ...) come second, and the nominal variables (coded 1, 2, ...) are in the last position.
- G: the number of clusters to be fitted.
- CnsIndx: the number of continuous variables in the dataset.
- OrdIndx: the sum of the number of continuous, binary, and ordinal variables in the dataset.
- models: a string indicating which `clustMD` model is to be fitted. Options are "EII", "VII", "EEI", "VEI", "EVI", "VVI" or "BD".
- startCL: a string indicating which clustering method should be used to initialize the (MC)EM algorithm. Different options are "kmeans", "hclust", "mclust", "hc\_mclust" (default), and "random".

The code for running all the available models for  $G = 1:10$  clusters and plotting the estimated approximated BIC values for each of the fitted models is detailed in the following.

```
> library(clustMD)
> X.MD <- German[, 1:8]
> # Transformation of the variables as required
> X.MD[, 4] <- as.integer(X.MD[, 4])
> X.MD[, 5] <- as.integer(X.MD[, 5])
```

```

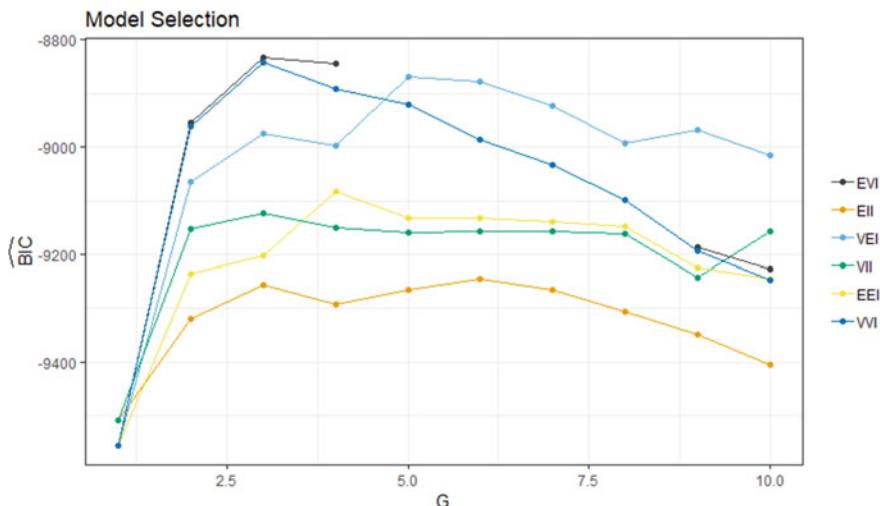
> X.MD[, 6] <- as.integer(X.MD[, 6])
> X.MD[, 7] <- as.integer(X.MD[, 7])
> X.MD[, 8] <- as.integer(X.MD[, 8])
> out.MD <- clustMDparallel(X = X.MD, G = 1:10,
+                               CnsIndx = 3, OrdIndx = 8,
+                               Nnorms = 20000,
+                               MaxIter = 500,
+                               models = c("EVI", "EII",
+                                         "VEI", "VII", "EEI", "VVI"),
+                               store.params = FALSE,
+                               scale = TRUE,
+                               startCL = "kmeans",
+                               autoStop = TRUE,
+                               ma.band = 30,
+                               stop.tol = 0.0001)
> out.MD$BICarray
      EVI       EII       VEI       VII       EEI
G=1 -9555.29 -9507.11 -9555.29 -9507.11 -9555.29
G=2 -8953.92 -9320.13 -9065.01 -9152.51 -9236.21
G=3 -8833.38 -9255.68 -8974.58 -9122.82 -9201.80
G=4 -8845.35 -9291.80 -8998.45 -9150.73 -9081.78
G=5      NA -9266.04 -8868.63 -9160.21 -9131.78
G=6      NA -9244.24 -8878.68 -9156.47 -9133.11
G=7      NA -9264.27 -8922.06 -9157.38 -9138.92
G=8      NA -9305.24 -8993.47 -9161.48 -9148.85
G=9 -9186.97 -9349.16 -8967.02 -9243.25 -9224.47
G=10 -9226.67 -9404.35 -9015.59 -9157.79 -9245.75
      VVI
G=1 -9555.29
G=2 -8960.21
G=3 -8842.45
G=4 -8890.71
G=5 -8920.30
G=6 -8985.05
G=7 -9032.66
G=8 -9098.11
G=9 -9192.95
G=10 -9247.85
> out.MD$ICLarray
      EVI       EII       VEI       VII
G=1 -11516.54 -11142.08 -11516.54 -11142.08
G=2 -10979.04 -10498.05 -11251.07 -10112.14
G=3 -10990.25 -10142.45 -11204.46 -9791.59
G=4 -10806.15 -10354.22 -11195.16 -9724.44
G=5      NA -10195.86 -11187.45 -9743.19
G=6      NA -10081.61 -11189.00 -9690.36
G=7      NA -9938.94 -11035.43 -9457.00
G=8      NA -10063.89 -11199.19 -9201.88
G=9 -11281.68 -10019.66 -11143.07 -9495.98
G=10 -11015.61 -9976.85 -11052.28 -8906.30
      EEI       VVI
G=1 -11516.54 -11516.54
G=2 -11211.45 -10977.98

```

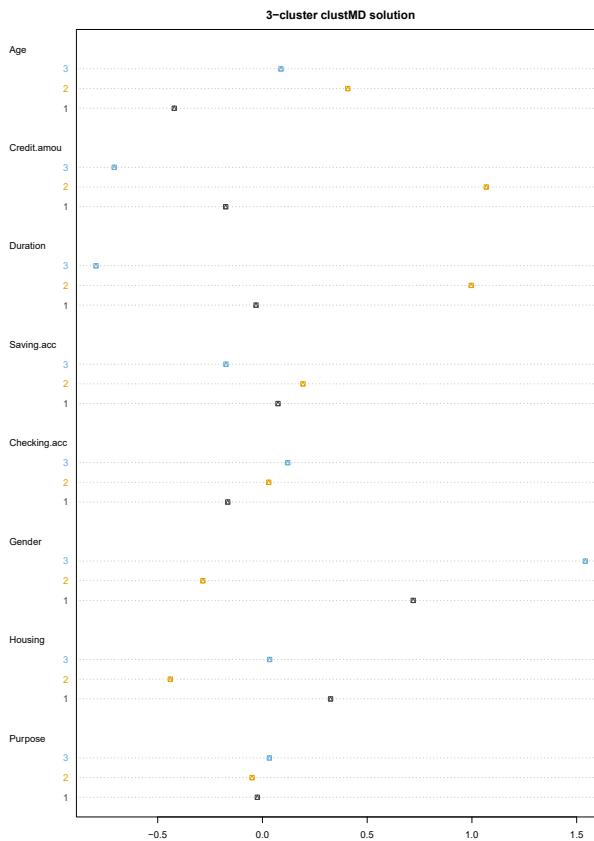
```
G=3   -11206.26   -11006.73
G=4   -11081.98   -10989.37
G=5   -11330.96   -11071.35
G=6   -11318.72   -11027.15
G=7   -11309.32   -11108.14
G=8   -11284.50   -11097.73
G=9   -11427.85   -11152.62
G=10  -11419.70   -11118.28
> plot(out.MD)
```

Figure 6.23 suggests that the best model is EVI (very similar to model VVI) with  $k = 3$  clusters. Thus, to estimate the model parameters, we fit such a model by the function `clustMD` where the inputs are similar to `clustMDparallel` but for  $G$  that is now fixed and `model` that is a string vector with the `clustMD` model to be fitted.

```
out.MD.EVI3 <- clustMD(X.MD, G = 3, CnsIndx = 3,
+                           OrdIndx = 8, Nnorms = 20000,
+                           MaxIter = 500, model = "EVI",
+                           store.params = FALSE,
+                           scale = TRUE,
+                           startCL = "kmeans")
> table(out.MD.EVI3$cl)
  1   2   3
144 183 195
> dotchart(t(out.MD.EVI3$means), col = 1:3, pch = 19,
+           main = "3-cluster clustMD solution")
```



**Fig. 6.23** german data: approximated BIC plots for the six `clustMD` models fitted by `clustMDparallel` (`startCL = "kmeans"`) for  $k = 1, \dots, 10$



**Fig. 6.24** german data: the mean and center values corresponding to the best clustMD model ( $k = 3$ , model = EVI, approximated BIC =  $-8833.38$ ) fitted by `clustMDparallel` (startCL = "kmeans")

Figure 6.24 displays the mean and center values for the continuous and the categorical variables, from top to bottom, according to the three-cluster solution. As it can be observed, as far as the continuous variables are considered, the mean values are similar to the ones obtained by **Rmixmod** while the center values of Gender, Housing, and Saving are more far apart: the clustMD model seems to produce a more pronounced cluster separation than **Rmixmod**.

Finally, to compare the two solutions obtained via the two packages with the known partition `Class.risk`, we report the cross-tabulations and the corresponding ARI below.

```
> table(out2@bestResult@partition, Class.risk)
  Class.risk
    1   2
  1  54  80
```

```

2 143 95
3 94 56
> round(adjustedRandIndex(out2@bestResult@partition,
+                         Class.risk), 2)
[1] 0.02
> table(out.MD.EVI3$c1, Class.risk)
  Class.risk
    1   2
1 101 82
2 60 84
3 130 65
> round(adjustedRandIndex(out.MD.EVI3$c1,
+                         Class.risk), 2)
[1] 0.03
> table(out2@bestResult@partition,
+        out.MD.EVI3$c1)
  1   2   3
1 3 130 1
2 180 14 44
3 0 0 150
> round(adjustedRandIndex(out2@bestResult@partition,
+                         out.MD.EVI3$c1), 2)
[1] 0.66

```

As it can be observed, the two methods produce quite similar clusters: most of customers in Clusters 1, 2, and 3 of the **clustMD** solution correspond to customers in Clusters 2, 1, and 3 of the **Rmixmod** solution, respectively. This is confirmed by a quite high ARI value (0.66). On the other hand, we may conclude that if the aim is to recover the Good and Bad partition, the two classes of models failed, while providing a well-separated partition.

## References

1. Abreu, N.: Analise do perfil do cliente Recheio e desenvolvimento de um sistema promocional. Mestrado em Marketing, ISCTE-IUL, Lisbon (2011)
2. Akaike, H.: Information theory and an extension of the maximum likelihood principle. In: Petrov, B.N., Csaki, F. (eds.) 2nd International Symposium on Information Theory, pp. 267–281. Akademiai Kiado, Budapest (1973)
3. Allman, E.S., Matias, C., Rhodes, J.A.: Identifiability of parameters in latent structure models with many observed variables. Ann. Stat. **37**, 3099–3132 (2009)
4. Atienza, N., Garcia-Heras, J., Muñoz-Pichardo, J.M.: A new condition for identifiability of finite mixture distributions. Metrika **63**, 215–221 (2006)
5. Bagnato, L., Punzo, A.: Finite mixtures of unimodal beta and gamma densities and the k-bumps algorithm. Comput. Stat. **28**, 1571–1597 (2013)
6. Banfield, J.D., Raftery, A.E.: Model-based Gaussian and non-Gaussian clustering. Biometrics **49**, 803–821 (1993)
7. Baudry, J.P., Raftery, A.E., Celeux, G., Lo, K., Gottardo, R.: Combining mixture components for clustering. J. Comput. Graph. Stat. **19**, 332–353 (2010)

8. Bensmail, H., Celeux, G.J.: Regularized Gaussian discriminant analysis through eigenvalue decomposition. *Am. Stat. Assoc.* **91**, 1743–1748 (1996)
9. Biecek, P., Szczurek, E.: bgmm: Gaussian Mixture Modeling Algorithms and the Belief-Based Mixture Modeling. R package version 1.8.4 (2020). <https://CRAN.R-project.org/package=bgmm>
10. Biernacki, C., Celeux, G., Govaert, G.: Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Trans. Pattern Anal. Mach. Intell.* **22**, 719–725 (2000)
11. Biernacki, C., Celeux, G., Govaert, G.: Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models. *Comput. Stat. Data Anal.* **41**, 561–575 (2003)
12. Bohnning, D., Dietz, E., Schaub, R., Schlattmann, P., Lindsay, B.G.: The distribution of the likelihood ratio for mixtures of densities from the one-parameter exponential family. *Ann. Inst. Statist. Math.* **46**, 373–388 (1994)
13. Bouveyron, C., Celeux, G., Murphy, T.B., Raftery, A.E.: Model-Based Clustering and Classification for Data Science: With Applications in R. Cambridge University Press, Singapore (2019)
14. Bretagnolle, V.: Personal communication, source: Museum (2007)
15. Browne, R.P., ElSherbiny, A., McNicholas, P.D.: mixture: Mixture Models for Clustering and Classification. R package version 1.5 (2018). <https://CRAN.R-project.org/package=mixture>
16. Celeux, G., Chauveau, D., Diebolt, J.: Stochastic versions of the EM algorithm: an experimental study in the mixture case. *J. Stat. Comput. Simul.* **55**, 287–314 (1996)
17. Celeux, G., Diebolt, J.: The SEM algorithm: a probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. *Comput. Stat. Q.* **2**, 73–82 (1985)
18. Celeux, G., Govaert, G.: Clustering criteria for discrete data and latent class models. *J. Classif.* **8**, 157–176 (1991)
19. Celeux, G., Govaert, G.: A classification EM algorithm for clustering and two stochastic versions. *Comput. Stat. Data An.* **14**, 315–332 (1992)
20. Celeux, G., Govaert, G.: Gaussian parsimonious clustering models. *Pattern Recognit.* **28**, 781–793 (1995)
21. Celeux, G., Soromenho, G.: An entropy criterion for assessing the number of clusters in a mixture model. *J. Classif.* **13**, 195–212 (1996)
22. Chandra, S.: On the mixtures of probability distributions. *Scand. J. Statist.* **4**, 105–112 (1977)
23. Chauveau, D.: A stochastic EM algorithm for mixtures with censored data. *J. Stat. Plann. Infer.* **46**, 1–25 (1995)
24. Chen, W.C., Maitra, R.: EMCluster: EM Algorithm for Model-Based Clustering of Finite Mixture Gaussian Distribution. R package version 0.2-12 (2019). <https://CRAN.R-project.org/package=EMCluster>
25. Chen, J., Tan, X., Zhang, R.: Inference for normal mixtures in mean and variance. *Stat. Sinica* **18**, 443–465 (2008)
26. Crawford, S.L.: An application of the Laplace method to finite mixture distributions. *J. Am. Stat. Assoc.* **89**, 259–267 (1994)
27. Day, N.E.: Estimating the components of a mixture of normal distributions. *Biometrika* **56**, 463–474 (1969)
28. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM-algorithm. *J. R. Stat. Soc. B* **39**, 1–38 (1977)
29. Diebolt, J., Celeux, G.: Asymptotic properties of a stochastic EM algorithm for estimating mixing proportions. *Stoch. Models* **9**, 599–613 (1993)
30. Di Mari, R., Rocci, R., Gattone, S.A.: Constrained maximum likelihood estimation of cluster-wise linear regression models with unknown number of components. ArXiv e-prints (2018). <https://arxiv.org/abs/1804.05185>
31. Dotto, F., Farcomeni, A.: Robust inference for parsimonious model-based clustering. *J. Stat. Comput. Simul.* **89**, 414–442 (2019)
32. Dua, D., Graff, C.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine (2019). <http://archive.ics.uci.edu/ml>

33. Everitt, B.S., Hand, D.J.: Finite Mixture Distributions. Chapman and Hall, London (1981)
34. Feng, Z.D., McCulloch, C.E.: Using bootstrap likelihood ratios in finite mixture models. *J. R. Stat. Soc. B* **58**, 609–617 (1996)
35. Finch, S., Mendell, N., Thode, H.: Probabilistic measures of adequacy of a numerical search for a global maximum. *J. Am. Stat. Assoc.* **84**, 1020–1023 (1989)
36. Fraley, C., Raftery, A.E.: Model-based clustering, discriminant analysis, and density estimation. *J. Am. Stat. Assoc.* **97**, 611–631 (2002)
37. Fraley, C., Raftery, A.E.: Bayesian regularization for normal mixture estimation and model-based clustering. *J. Classif.* **24**, 155–181 (2007)
38. Frühwirth-Schnatter, S.: Finite Mixture and Markov Switching Models. Springer-Verlag, New York (2006)
39. Giordani, P., Ferraro, M.B., Martella, F.: datasetsICR: Datasets from the Book “An Introduction to Clustering with R. R package version 1.0 (2020). <https://CRAN.R-project.org/package=datasetsICR>
40. Gruen, B., Leisch, F.: flexmix: Flexible Mixture Modeling. R package version 2.3-15 (2019). <https://CRAN.R-project.org/package=flexmix>
41. Hastie, T., Tibshirani, R.: Discriminant analysis by Gaussian mixtures. *J. R. Stat. Soc. B* **58**, 155–176 (1996)
42. Hathaway, R.J.: Another interpretation of the EM algorithm for mixture distributions. *Stat. Prob. Lett.* **4**, 53–56 (1986)
43. Heller, A.K., Williamson, S., Ghahramani, Z.: Statistical models for partial membership. In: Proceedings of the 25th International Conference on Machine Learning, pp. 392–399. ACM, New York, NY, USA (2008)
44. Hennig, C.: Breakdown points for maximum likelihood-estimators of location-scale mixtures. *Ann. Stat.* **32**, 1313–1340 (2004)
45. Hennig, C.: fpc: Flexible Procedures for Clustering. R package version 2.2-5 (2020). <https://CRAN.R-project.org/package=fpc>
46. Hubert, L., Arabie, P.A.: Comparing partitions. *J. Classif.* **2**, 193–218 (1985)
47. Ingrassia, S.: A likelihood-based constrained algorithm for multivariate normal mixture models. *Stat. Met. Appl.-Ger.* **13**, 151–166 (2004)
48. Ingrassia, S., Punzo, A., Vittadini, G., Minotti, S.C.: The generalized linear mixed cluster-weighted model. *J. Classif.* **32**, 85–113 (2015)
49. Iovleff, S., Bathia, P.: MixAll: Clustering and Classification Using Model-Based Mixture Models. R package version 1.5.1 (2018). <https://CRAN.R-project.org/package=MixAll>
50. Karlis, D., Xekalaki, E.: Choosing initial values for the EM algorithm for finite mixtures. *Comput. Stat. Data Anal.* **41**, 577–590 (2003)
51. Kass, R.E., Raftery, A.E.: Bayes factors. *J. Am. Stat. Assoc.* **90**, 773–795 (1995)
52. Kiefer, J., Wolfowitz, J.: Consistency of the maximum likelihood estimator in the presence of infinitely many incidental parameters. *Ann. Math. Stat.* **27**, 887–906 (1956)
53. Langrognet, F., Lebret R., Poli, C., Iovleff, S., Auder, B., Iovleff, S.: Rmixmod: Classification with Mixture Modelling. R package version 2.1.2.2 (2019). <https://CRAN.R-project.org/package=Rmixmod>
54. Lazarsfeld, P.F., Henry, N.W.: Latent Structure Analysis. Houghton Mifflin, Boston (1968)
55. Lebret, R., Iovleff, S., Langrognet, F., Biernacki, C., Celeux, G., Govaert, G.: Rmixmod: The R package of the model-based unsupervised, supervised, and semi-supervised classification Mixmod library. *J. Stat. Softw.* **67** (2015)
56. Li, P., Chen, J., Marriott, P.: Non-finite Fisher information and homogeneity: the EM approach. *Biometrika* **96**, 411–426 (2009)
57. Lindsay, B.G.: The geometry of mixture likelihood: a general theory. *Ann. Stat.* **11**, 86–94 (1983)
58. Lindsay, B.G.: Mixture models: theory, geometry and applications. In: NSF-CBMS Regional Conference Series in Probability and Statistics. Institute of Mathematical Statistics, California, vol. 5 (1995)

59. Lindstrom, M.J., Bates, D.M.: Newton-Raphson and EM algorithms for linear mixed-effects models for repeated-measures data. *J. Am. Stat. Assoc.* **83**, 1014–1022 (1988)
60. Linzer, D.A., Lewis, J.B.: poLCA: An R Package for polytomous variable latent class analysis. *J. Stat. Softw.* **42**, 1–29 (2011)
61. Liu, C., Rubin, D.B.: The ECME algorithm: a simple extension of EM and ECM with faster monotone convergence. *Biometrika* **81**, 633–648 (1994)
62. Magidson, J., Vermunt, J.K.: Latent class models for clustering: a comparison with  $K$ -means. *Can. J. Mark. Res.* **20**, 36–43 (2002)
63. Maitra, R.: Initializing partition-optimization algorithms. *IEEE/ACM T. Comput. Biol. Bioinform.* **6**, 144–157 (2009)
64. Maitra, R., Melnykov, V.: Simulating data to study performance of finite mixture modeling and clustering algorithms. *J. Comput. Graph. Stat.* **19**, 354–376 (2010)
65. Mazza, A., Punzo, A., Ingrassia, S.: flexCWM: A flexible framework for cluster-weighted models. *J. Stat. Softw.* **86**, 1–30 (2018)
66. McLachlan, G.J.: On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture. *J. R. Stat. Soc. C* **36**, 318–324 (1987)
67. McLachlan, G.J., Krishnan, T.: The EM Algorithm and Extensions. Wiley, New York (1997)
68. McLachlan, G.J., Krishnan, T.: The EM Algorithm and Extensions, 2n edn. Wiley, New York (2008)
69. McLachlan, G.J., Peel, D.: Finite Mixture Models. Wiley, New York (2000)
70. McLachlan, G.J., Peel, D.: On a resampling approach to choosing the number of components in normal mixture models. In: Billard, L., Fisher, N.I. (eds.) Computing Science and Statistics, vol. 28, pp. 260–266. Interface Foundation of North America, Fairfax Station, Virginia (1997)
71. McNicholas, P.D., Murphy, T.B., McDaid, A.F., Frost, D.: Serial and parallel implementations of model-based clustering via parsimonious Gaussian mixture models. *Comput. Stat. Data Anal.* **54**, 711–723 (2010)
72. McNicholas, P.D.: Mixture Model-Based Classification. Chapman & Hall/CRC Press, Boca Raton (2016)
73. McParland, D., Gormley, I.C.: Model-based clustering for mixed data: clustMD. *Adv. Data Anal. Classif.* **10**, 155–169 (2016)
74. McParland, D., Gormley, I.C.: clustMD: Model-based clustering for mixed data. R package version 1.2.1 (2017). <https://CRAN.R-project.org/package=clustMD>
75. Melnykov, V., Maitra, R.: Finite mixture models and model-based clustering. *Stat. Surv.* **4**, 80–116 (2010)
76. Melnykov, V., Melnykov, I.: Initializing the EM algorithm in Gaussian mixture models with an unknown number of components. *Comput. Stat. Data Anal.* **56**, 1381–1395 (2012)
77. Meng, X.L., Rubin, D.B.: Maximum likelihood estimation via the ECM algorithm: a general framework. *Biometrika* **80**, 267–278 (1993)
78. Meng, X.L., Van Dyk, D.A.: The EM algorithm - an old folk song sung to a fast new tune. *J. R. Stat. Soc. B* **59**, 511–567 (1997)
79. Newcomb, S.: A generalized theory of the combination of observations so as to obtain the best result. *Am. J. Math.* **8**, 343–366 (1886)
80. O'Hagan, A., Murphy, T.B., Gormley, I.C.: On estimation of parameter uncertainty in model-based clustering. ArXiv e-prints (2015). <http://arxiv.org/abs/1510.00551>
81. Pearson, K.: Contributions to the mathematical theory of evolution. *Philos. T. R. Soc. A.* **185**, 71–110 (1894)
82. Pilla, R.S., Kamarthi, S.V., Lindsay, B.G.: Aitken-based acceleration methods for assessing convergence of multilayer neural networks. *IEEE T. Neural Netw.* **12**, 998–1012 (2001)
83. Pilla, R.S., Lindsay, B.G.: Alternative EM methods for nonparametric finite mixture models. *Biometrika* **88**, 535–550 (2001)
84. Ranalli, M., Rocci, R.: Mixture models for ordinal data: a pairwise likelihood approach. *Stat. Comput.* **26**, 529–547 (2016)
85. Raciati, S., Virola, C., Wit, E.C.: Mixture model with multiple allocations for clustering spatially correlated observations in the analysis of ChIP-Seq data. *Biom. J.* **59**, 1301–1316 (2017)

86. Redner, R.A., Walker, H.C.: Mixture densities, maximum likelihood and the EM algorithm. *SIAM Rev.* **26**, 195–239 (1984)
87. Schwarz, G.: Estimating the dimension of a model. *Ann. Stat.* **6**, 461–464 (1978)
88. Scrucca, L.: Dimension reduction for model-based clustering. *Stat. Comput.* **20**, 471–484 (2010)
89. Scrucca, L.: Graphical tools for model-based mixture discriminant analysis. *Adv. Data Anal. Class.* **8**, 147–165 (2014)
90. Scrucca, L., Fop, M., Murphy, T.B., Raftery, A.E.: mclust 5: Clustering, classification and density estimation using Gaussian finite mixture models. *R J.* **8**(1), 205–233 (2016)
91. Scrucca, L., Raftery, A.E.: Improved initialisation of model-based clustering using a Gaussian hierarchical partition. *Adv. Data Anal. Class.* **9**, 447–460 (2015)
92. Seidel, W., Mosler, K., Alker, M.: A cautionary note on likelihood ratio tests in mixture models. *Ann. Inst. Stat. Math.* **52**, 481–487 (2000)
93. Soromenho, G.: Comparing approaches for testing the number of components in a finite mixture model. *Comput. Stat.* **9**, 65–78 (1994)
94. Teicher, H.: Identifiability of mixtures. *Ann. Math. Stat.* **32**, 244–248 (1961)
95. Teicher, H.: Identifiability of finite mixtures. *Ann. Math. Stat.* **34**, 1265–1269 (1963)
96. Tiedeman, D.V.: On the study of types. In: Sells, S.B. (ed.) *Symposium on Pattern Analysis*, Randolph Field, TX: Air University, U.S.A.F. School of Aviation Medicine, pp. 1–14 (1955)
97. Titterington, D.M., Smith, A.F.M., Makov, U.E.: *Statistical Analysis of Finite Mixture Distributions*. Wiley, Chichester (1985)
98. Vermunt, J.K., Magidson, J.: Technical guide for Latent GOLD 4.0: Basic and advanced (2005). <https://www.statisticalinnovations.com>
99. Wei, G., Tanner, M.: A monte-carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *J. Amer. Stat. Assoc.* **85**, 699–704 (1990)
100. White, A., Murphy, T.B.: BayesLCA: An R package for Bayesian latent class analysis. *J. Stat. Softw.* **61**, 1–28 (2014)
101. Wickham, H.: *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York (2016)
102. Wolfe, J.H.: Object cluster analysis of social areas. Master's thesis, University of California, Berkeley (1963)
103. Wu, C.F.J.: On convergence properties of the EM algorithm. *Ann. Stat.* **11**, 95–103 (1983)
104. Yakowitz, S.J., Spragins, J.D.: On the identifiability of finite mixtures. *Ann. Math. Stat.* **39**, 209–214 (1968)
105. Young, D., Benaglia, T., Chauveau, D., Hunter, D.: mixtools: Tools for Analyzing Finite Mixture Models. R package version 1.2.0 (2020). <https://CRAN.R-project.org/package=mixtools>

# Chapter 7

## Issues in Gaussian Model-Based Clustering



### 7.1 Introduction

When applying GMMs for clustering, we need to consider several issues. In fact, finite mixtures, considered as semiparametric/nonparametric estimators of an unknown density, suffer for high-dimensional data and deviations from the nominal model due to model misspecification, e.g., component-specific covariance, skewness, or outliers. These problems may lead to potentially singular estimates, biased parameter estimates, and unreliable clustering results leading to overestimation of the number of clusters. For the sake of simplicity, considerations in this chapter are restricted to some solutions proposed over the past decade to deal with these issues. We focus on Finite Mixture Models (FMMs) for continuous data in high-dimensional settings and with non-Gaussian component-specific distributions. The latter can be useful in situations where the clusters exhibit heavy tails, non-elliptical shapes, and/or outliers. Finally, we briefly mention some further approaches to these issues.

### 7.2 High Dimensionality in Gaussian Mixture Models

Applications to various domains often entail high-dimensional data, with hundreds or thousands of variables for each unit. Unfortunately, model-based clustering approaches may be over-parameterized in high-dimensional spaces, and they may therefore suffer from the so-called curse of dimensionality [13]. In fact, the number of parameters to be estimated usually depends on the dimension of the observed space. Thus, the larger the sample size, the more efficient the estimation of the model parameters. If the number of units is small when compared to the number of parameters (or variables), we may have problems in calculating the inverse of the covariance matrix and therefore obtain unstable solutions. In the next sections, we describe different approaches from the specialized literature.

### 7.2.1 Regularization and Tandem Analysis Strategies

A simple way to deal with the curse of dimensionality is to place it as a computational problem in inverting the covariance matrix and to solve it by numerically regularizing the estimates of the covariance matrices  $\hat{\Sigma}_g$ . A simple way to regularize  $\Sigma_g$  consists in introducing a ridge regularization based on a parameter  $\varepsilon_g > 0$ ,

$$\tilde{\Sigma}_g = \hat{\Sigma}_g + \varepsilon_g \mathbf{I}. \quad (7.1)$$

To tune the regularization parameter  $\varepsilon_g$ , standard model selection criteria may be used, but further regularization techniques have been proposed in the supervised clustering context; see [74] for a comprehensive overview.

An alternative way is to proceed via a sequential procedure performing a dimensional reduction before using a clustering algorithm. Principal Component Analysis (PCA) can be applied to project the observed variables onto a reduced subspace, while a clustering algorithm may be used to allocate units to homogeneous clusters on the basis of Principal Components (PCs). The hope when using PCA prior to clustering is that PCA may extract the essential information on the cluster structure that is contained in the data. Since PCs are uncorrelated and ordered, the first few PCs, which contain most of the variability in the data, are used in cluster analysis [49]. This procedure, referred to as tandem analysis [48], provides PCs and an optimal classification, minimizing two different target functions that may work in opposite directions. That is, PCA may identify components that do not contribute much to detect the clustering structure in the data. To overcome this issue, several authors have proposed methodologies for simultaneous classification and dimensional reduction that will be recalled in Sect. 7.2.2.

### 7.2.2 Parsimonious Gaussian Mixture Models

A different, related, way to look at the issue of high dimensionality is to consider a parsimonious model specification. The first example of parsimonious models is proposed in [9] by parameterizing the covariance matrix in terms of its eigendecomposition (see Eq. 6.34, Sect. 6.6). However, if the number of variables  $p$  is large relative to the sample size  $n$ , it may not be possible to use this decomposition to build an appropriate model for the component-specific covariance matrix; even if this is possible, the results may not be valid. In such cases, it would be useful to assume that the component-specific covariance matrix is diagonal (e.g.,  $\Sigma_g = \lambda_g \mathbf{I}_p$  or  $\Sigma_g = \lambda \mathbf{I}_p$ ) leading to spherical clusters with axes aligned to those of the feature space.

As mentioned in Sect. 7.2.1, an alternative way to the sequential procedure consists in simultaneously performing classification and dimension reduction (within each cluster). One of the most popular proposals in this field is the Mixtures of Factor

Analyzers (MFA) introduced in [39] and extended in [64]. Under the MFA model, conditional on  $z_{ig} = 1$ ,  $\mathbf{x}_i$  is modeled as

$$\mathbf{x}_i = \boldsymbol{\mu}_g + \boldsymbol{\Lambda}_g \mathbf{u}_{ig} + \mathbf{e}_{ig}, \quad (7.2)$$

with probability  $\pi_g$ . The term  $\boldsymbol{\mu}_g$  is a component-specific mean vector,  $\boldsymbol{\Lambda}_g$  is a component-specific factor loadings matrix of order  $(p \times q)$  ( $q < p$ ),  $\mathbf{u}_{ig}$  is a  $q$ -dimensional vector of component-specific latent variables (factors), which are assumed to be independent and identically distributed variates  $\text{MVN}_q(\mathbf{0}, \mathbf{I}_q)$ , and  $\mathbf{I}_q$  denotes the identity matrix of order  $q$ . The error terms  $\mathbf{e}_{ig}$  are independent and identically distributed Gaussian component-specific random variables with mean  $\mathbf{0}$  and diagonal covariance matrix  $\mathbf{D}_g = \text{diag}(d_{1g}^2, \dots, d_{pg}^2)$ , which are assumed to be independent of  $\mathbf{u}_{ig}$ . Conditional on belonging to the  $g$ -th component, the density is

$$f(\mathbf{x}_i | \boldsymbol{\theta}_g) = \phi(\mathbf{x}_i | \boldsymbol{\mu}_g, \boldsymbol{\Lambda}_g \boldsymbol{\Lambda}'_g + \mathbf{D}_g), \quad (7.3)$$

with probability  $\pi_g$ . According to this assumption, the marginal density of  $\mathbf{x}_i$  is

$$f(\mathbf{x}_i | \boldsymbol{\Psi}) = \sum_{g=1}^k \pi_g \phi(\mathbf{x}_i | \boldsymbol{\mu}_g, \boldsymbol{\Lambda}_g \boldsymbol{\Lambda}'_g + \mathbf{D}_g), \quad (7.4)$$

and the overall set of model parameters is

$$\boldsymbol{\Psi} = \{\pi_1, \dots, \pi_{k-1}, \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_1, \dots, \boldsymbol{\Lambda}_k, \mathbf{D}_1, \dots, \mathbf{D}_k\}. \quad (7.5)$$

An EM-type algorithm can be used for parameter estimation. Regarding the choice of the number of factors  $q$ , a possible approach consists in looking at the specific proportion of the total variance in the original data that is explained by the first  $q$  factors. Simulation studies show that BIC is a good tool to select the number of factors [58]. Different approaches for selecting the number of factors are discussed in Sect. 3 of [66].

It is worth notice that MFA is, in essence, a dimensional reduction of a GMM, where within each component a factor model is fitted to the data, with weights given by the posterior probabilities [60]. Since the covariance matrix for each component is specified through the lower dimensional factor loading matrix, the model has  $[pq - q(q-1)/2]k + pk$  rather than  $kp(p+1)/2$  parameters for modeling the cluster-specific covariance. Note that the term  $q(q-1)/2$  represents the number of constraints that are imposed to make  $\boldsymbol{\Lambda}' \mathbf{D}^{-1} \boldsymbol{\Lambda}$  diagonal and to guarantee identifiability of  $\boldsymbol{\Lambda}$ . If  $q$  is sufficiently smaller than  $p$ ,  $\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g \boldsymbol{\Lambda}'_g + \mathbf{D}_g$  imposes stronger restrictions on covariance matrices reducing the number of free parameters.

Notice that the original constraint is  $\mathbf{D}_g = \mathbf{D}$  [39]; successively, in [63, 64] the extension to the unconstrained MFA has been provided. In [92], a Mixture of Probabilistic Principal Component Analyzers (MPPCA) has been developed; it is closely related to MFA but for unequal, isotropic error component-specific

matrices  $\mathbf{D}_g = d_g \mathbf{I}_p$  and, thus, with  $[pq - q(q - 1)/2]k + k$  free parameters in the component-specific covariance. Further work extends MFA model developing a collection of Parsimonious Gaussian Mixture Models (PGMMs) that can be obtained by constraining  $\boldsymbol{\Lambda}_g = \boldsymbol{\Lambda}$ ,  $\mathbf{D}_g = \mathbf{D}$ , and  $\mathbf{D}_g = d_g \mathbf{I}_p$  [69, 70]. Table 7.1 reports all such models with the corresponding number of free parameters associated with the component-specific covariance matrices. In [71], PGMMs are further extended by considering the reparametrization  $\mathbf{D}_g = \omega_g \boldsymbol{\Delta}_g$ , where  $\omega_g \in \mathbb{R}^+$  and  $\boldsymbol{\Delta}_g = \text{diag}\{\delta_1, \dots, \delta_p\}$  such that  $|\boldsymbol{\Delta}_g| = 1$ , adding more models leading to the Expanded Parsimonious Gaussian Mixture Models (EPGMMs). Table 7.2 contains all the members of the EPGMM family. Parameter estimation for both the PGMM and the EPGMM families can be carried out by using an AECM algorithm [73]. Details are given in [63, 70, 72]. The selection phase for PGMM/EPGMM can be performed by using standard information criteria as in general mixture models. How-

**Table 7.1** Set of eight PGMM models with  $k$  clusters,  $p$  variables, and  $q$  factors

Model ID	$\boldsymbol{\Sigma}_g$	Parameters for $\boldsymbol{\Sigma}_g$
CCC	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + d \mathbf{I}_p$	$pq - q(q - 1)/2 + 1$
CCU	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \mathbf{D}$	$pq - q(q - 1)/2 + p$
CUC	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + d_g \mathbf{I}_p$	$pq - q(q - 1)/2 + k$
CUU	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \mathbf{D}_g$	$pq - q(q - 1)/2 + kp$
UCC	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + d \mathbf{I}_p$	$k(pq - q(q - 1)/2) + 1$
UCU	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \mathbf{D}$	$k(pq - q(q - 1)/2) + p$
UUC	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + d_g \mathbf{I}_p$	$k(q - q(q - 1)/2) + k$
UUU	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \mathbf{D}_g$	$k(pq - q(q - 1)/2) + kp$

**Table 7.2** Set of 12 EPGMM models with  $k$  clusters,  $p$  variables, and  $q$  factors

Model ID	$\boldsymbol{\Sigma}_g$	Parameters for $\boldsymbol{\Sigma}_g$
CCCC	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \omega \mathbf{I}_p$	$pq - q(q - 1)/2 + 1$
CCCU	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \omega \boldsymbol{\Delta}$	$pq - q(q - 1)/2 + p$
CCUC	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \omega_g \mathbf{I}_p$	$pq - q(q - 1)/2 + k$
CUUU	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \omega_g \boldsymbol{\Delta}_g$	$pq - q(q - 1)/2 + kp$
UCCC	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \omega \mathbf{I}_p$	$k(pq - q(q - 1)/2) + 1$
UCCU	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \omega \boldsymbol{\Delta}$	$k(pq - q(q - 1)/2) + p$
UCUC	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \omega_g \mathbf{I}_p$	$k(q - q(q - 1)/2) + k$
UUUU	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \omega_g \boldsymbol{\Delta}_g$	$k(pq - q(q - 1)/2) + kp$
CCUU	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \omega_g \boldsymbol{\Delta}$	$(pq - q(q - 1)/2) + (k + (p - 1))$
UCUU	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \omega_g \boldsymbol{\Delta}$	$k(pq - q(q - 1)/2) + (k + (p - 1))$
CUCU	$\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \omega \boldsymbol{\Delta}_g$	$(pq - q(q - 1)/2) + (1 + k(p - 1))$
UUCU	$\boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \omega \boldsymbol{\Delta}_g$	$k(pq - q(q - 1)/2) + (1 + k(p - 1))$

ever, recently, it has been observed [14] that, even for moderately large values of  $p$ , the BIC could fail to select  $k$  and  $q$  for the members of the PGMM family.

An alternative approach, called Mixture of Common Factor Analyzers (MCFA) model, has been proposed in [7]. It is particularly useful when  $p$ ,  $k$ , or both are large, and, consequently, MFA or EPGMM may not be parsimonious enough. This essentially further constrains the component-specific means and covariance matrices to reduce the number of free parameters. Notice that this model is quite restrictive, and it is not recommended for general use [65]. The MCFA model is a generalization of the Mixtures of Common Uncorrelated Factor Spherical-error Analyzers (MCUFSAs) [100, 101], where the covariance of the noise term is constrained to be spherical and the component-specific covariance matrices of the factors to be diagonal. The MCUFSA is therefore more parsimonious than the MCFA. A similar approach referred to as the Heteroscedastic Mixture Factor Analyzers (HMFA) has been introduced in [75], where the only difference is in the definition of the common loadings matrix (it does not need to have orthonormal columns as in the MCFA model).

Several alternative approaches for dealing with high-dimensional data have been proposed, just to mention a few examples: High-dimensional GMMs (HD-GMM) [18, 19], the Least Absolute Shrinkage and Selection Operator (LASSO)-penalized-likelihood method [14], and the Mixture of Contaminated Gaussian Factor Analyzers (MCGFA) [82] implemented via the package **mcgfa** [16], among others. However, they are out of the scope of this book. The interested reader may refer to [17, 65] for a review.

The next section presents an application of the function `pgmmEM` of the package **pgmm** [67], which implements the EPGMM models and an AECM algorithm for estimation. Model selection can be made by means of BIC (default) or ICL: the higher the better. The strategy is similar to GMMs.

1. Fix the range of the number of clusters  $k$ .
2. Fix the range of the number of factors  $q$ .
3. Fit the model.

### 7.2.3 ***pgmm***

The dataset **Economics**, previously analyzed in [87] and available in the package **datasetsICR** [40], refers to  $n = 55$  Italian Economics faculties evaluated on 24 indicators splitted in 4 dimensions for the academic year 2009/2010. The relevant dimensions are listed below (see also Table 7.3 for the variable description):

- Productivity (P1, P2, P3A, P3B, P4A, P4B);

**Table 7.3** Economics data: variable description

<i>Productivity indicators</i>	
P1	Rate of persistence between the first and the second academic year
P2	Achieved credits
P3(A)	Rate of regular students enrolled in the 3-year bachelor-level programs
P3(B)	Rate of regular students enrolled in the 2-year master-level programs
P4(A)	Rate of regular graduated students in the 3-year bachelor-level programs
P4(B)	Rate of regular graduated students in the 2-year master-level programs
<i>Teaching indicators</i>	
D1	Permanent professors per credits
D2	Permanent professors per enrolled student
D3	Seats per enrolled student in the academic year 2009/2010
D4	Seats per student enrolled in the academic year 2008/2009
D5	Researchers to professors ratio
D6	Monitored teaching activities
<i>Research indicators</i>	
R1	Financed research units per professor
R2	Average funding per research unit
R3	Submitted projects per professor
R4	Success rate in the PRIN program
R5	Average funding for international research per professor
R6	Financed research projects per professor
R7	Average funding for FIRB project
<i>Internationalization indicators</i>	
I1	Outgoing student mobility
I2	Incoming student mobility
I3	Host universities
I4	International opportunities
I5	Courses with double or joint title

- Teaching (D1, D2, D3 D4, D5, D6);
- Research (R1, R2, R3, R4, R5, R6, R7);
- Internationalization (I1, I2, I3, I4, I5).

Moreover, the variable University\_Type offers information on the University type.

```
> library(datasetsICR)
> data("Economics")
> University_Type <- Economics[, 13]
> table(University_Type)
```

```
University_Type
Private Public
    7       48
```

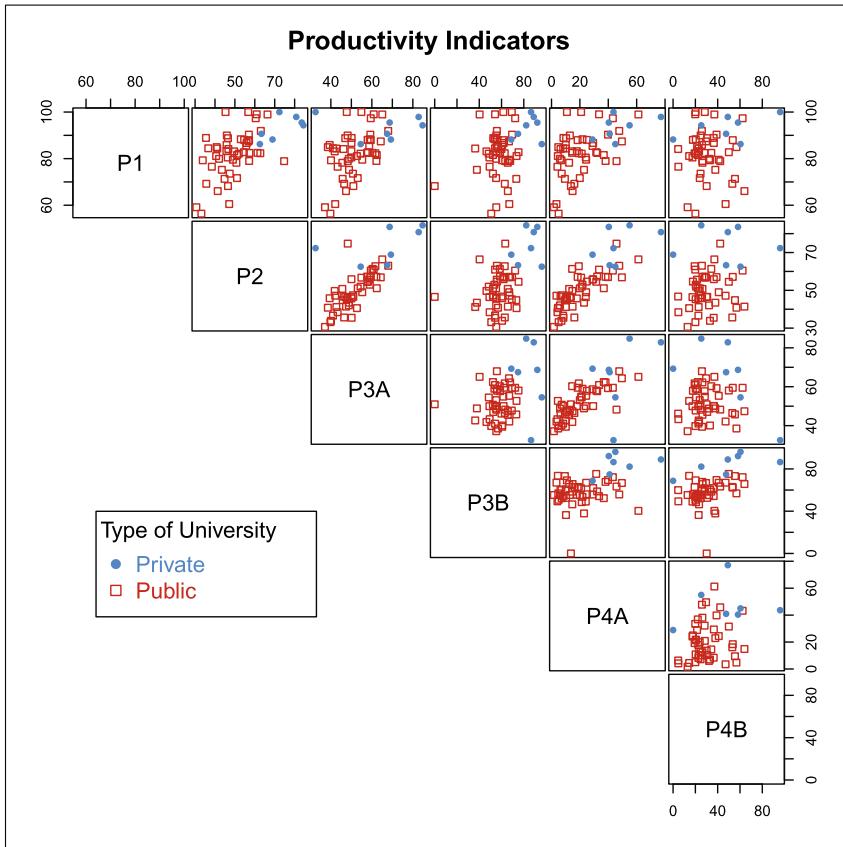
Since most of the considered institutions lack in internationalization and research, we consider only the first two dimensions (productivity and teaching) including  $p = 12$  variables. Figures 7.1 and 7.2 show the matrices of scatterplots for productivity and teaching indicators, respectively. They can be obtained by the following code.

```
> library(mclust)
> clp <- clPairs(Economics[, 1:6], University_Type,
+                   lower.panel = NULL,
+                   main = "Productivity Indicators")
> clPairsLegend(0.1, 0.4, class = clp$class,
+                 col = clp$col, pch = clp$pch,
+                 title = "Type of University")
> clp2 <- clPairs(Economics[, 7:12], University_Type,
+                    lower.panel = NULL,
+                    main = "Teaching Indicators")
> clPairsLegend(0.1, 0.4, class = clp2$class,
+                 col = clp2$col, pch = clp2$pch,
+                 title = "Type of University")
```

Most of the variables show high variability (but D1 and D2), providing evidence of some heterogeneity in the sample. To fit the EPGMM models described in Table 7.2, the function `pgmmEM` of the package **pgmm** is used. The arguments `rG` and `rq` are used to set the range for the number of clusters  $k$  and factors  $q$ , respectively. By default `rG = 1:2` and `rq = 2`. Notice that the number of factors `rq` must satisfy  $(p - q)^2 > p + q$ . Expert users may consider `relax = TRUE` to relax this constraint. First, we standardize the data. By using the default starting value option, the maximum number of clusters and factors is 5 (more clusters cause problems of invertibility of covariance matrices). The code is shown below.

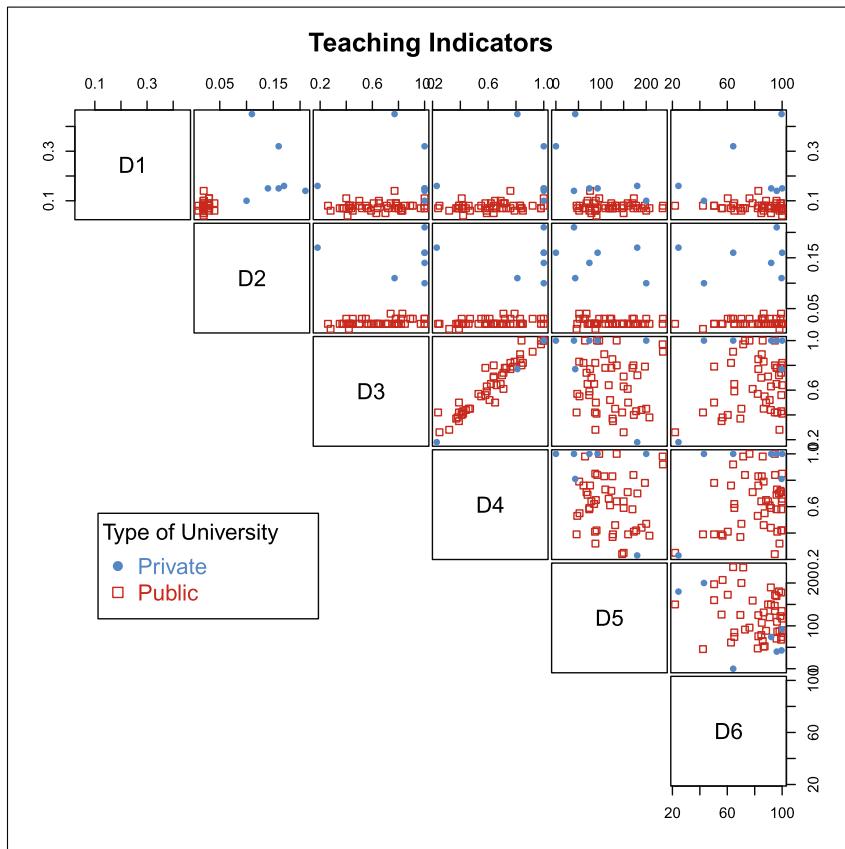
```
> library(pgmm)
> default_pgmm <- pgmmEM(scale(Economics[, 1:12]),
+                           rG = 1:5, rq = 1:5)
Based on k-means starting values, the best model (BIC)
for the range of factors and components used is a UUU
model with q = 5 and G = 2. The BIC for this model is
-1397.965.
```

Alternative options may be specified. For instance, the argument `ic1 = TRUE` may be given to use ICL instead of BIC to select the best model. We may change the starting values of the AECM algorithm moving from the default  $k$ -Means (`zstart = 2`) to random (`zstart = 1`) or custom starts (`zstart = 3`).



**Fig. 7.1** Economics data: matrix of scatterplots for productivity indicators

When `zstart = 1`, the number of random starts (by default three) can be set through the argument `loop`. If `zstart = 3`, the function `pgmmEM` expects the object `zlist` to be specified, that is, a list comprising vectors of initial partitions, such that each vector `zlist[[g]]` gives the starting values for a certain partition with a number of clusters equal to `g`. The argument `modelSubset` allows to specify one or more models to be used (by default all 12 models). This option is useful if one is interested in considering only a specific set of model(s). For further details, see `help(pgmmEM)`. In the following, several different options are given.



**Fig. 7.2** Economics data: matrix of scatterplots for teaching indicators

```

> # Change model selection criterion
> ICL_pgmm <- pgmmEM(scale(Economics[, 1:12]),
+                         rG = 1:5, rq = 1:5, icl = TRUE)
Based on k-means starting values, the best model (ICL)
for the range of factors and components used is a UUU
model with q = 5 and G = 2. The ICL for this model is
-1397.965.
> # Initialization with 100 random starts
> Random_pgmm <- pgmmEM(scale(Economics[, 1:12]),
+                           rG = 1:5, rq = 1:5, zstart = 1,
+                           loop = 100)
Based on 100 random starts, the best model (BIC) for
the range of factors and components used is a CUU
model with q = 1 and G = 5. The BIC for this model is
587.4393.
> # Initialization with hierarchical clustering starts
> hcl <- hclust(dist(scale(Economics[, 1:12])))

```

```

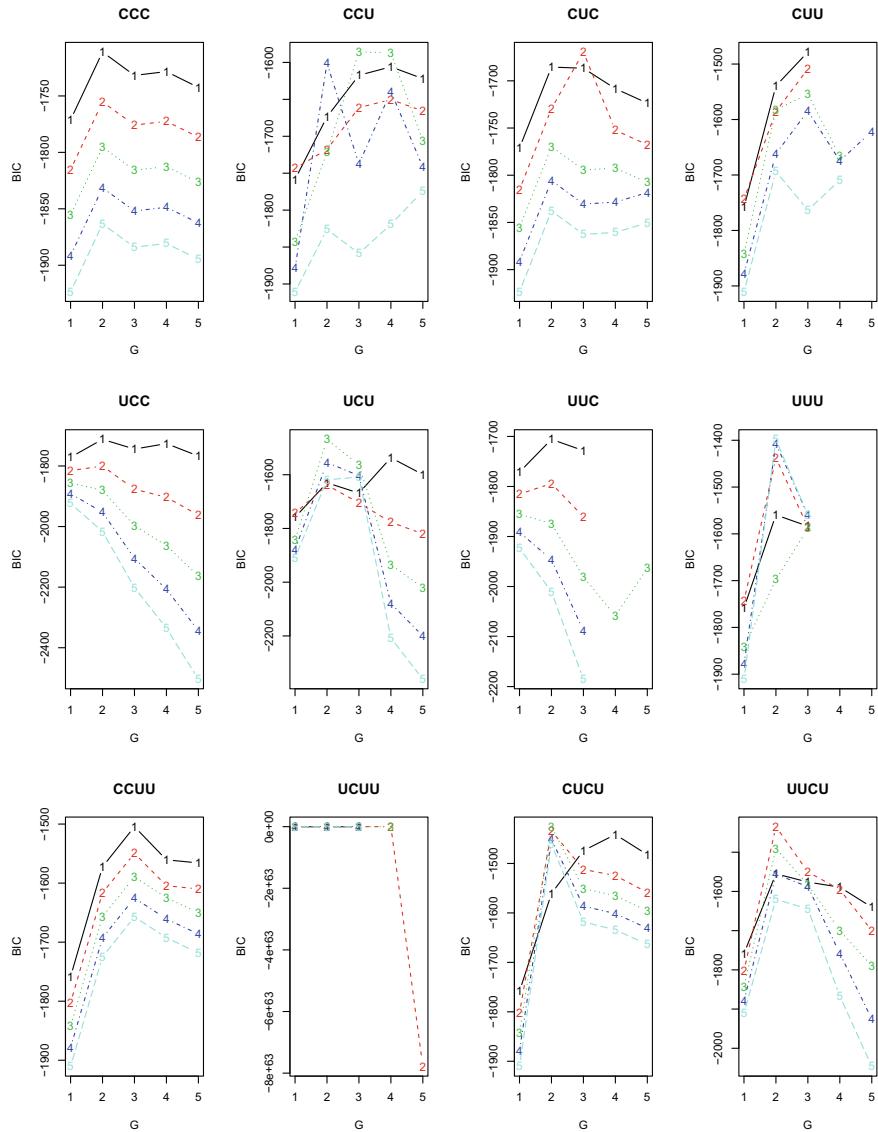
> z<-list()
> for(g in 1:6){z[[g]] <- cutree(hcl, k = g)}
> Hier_pgmm <- pgmmEM(scale(Economics[, 1:12]),
+                         rG = 1:2, rq = 1:4,
+                         zstart = 3, zlist = z)
Based on custom starting values, the best model (BIC)
for the range of factors and components used is a CUCU
model with q = 2 and G = 2. The BIC for this model is
-1407.189.
> # Initialization with model-based clustering starts
> z <- list()
> for(g in 1:6){
+   a <- Mclust(Economics[, 1:12], G = g)
+   z[[g]] <- a$classification
+   z[[g]] <- as.integer(z[[g]])
+ }
> Mclust_pgmm <- pgmmEM(scale(Economics[, 1:12]),
+                          rG = 1:5, rq = 1:6, zstart = 3,
+                          zlist = z)
Based on custom starting values, the best model (BIC)
for the range of factors and components used is a
UUU model with q = 5 and G = 2. The BIC for this model
is -1399.55.
> # Initialization with DIANA starts
> library(cluster)
> diana.res <- diana(x = scale(Economics[, 1:12]),
+                      diss = FALSE)
> z <- list()
> for(g in 1:6){
+   z[[g]] <- cutree(diana.res, k = g)
+ }
> diana_pgmm <- pgmmEM(scale(Economics[, 1:12]),
+                         rG = 1:2, rq = 1:7, zstart = 3,
+                         zlist = z)
Based on custom starting values, the best model (BIC)
for the range of factors and components used is a CUU
model with q = 3 and G = 2. The BIC for this model is
-1483.157.

```

For a specified initialization strategy, we can provide a plot comparing the BIC values for all the analyzed models. For instance, Fig. 7.3 compares the BIC values for the 12 PGMM models based on  $k$ -Means starting values through the following command.

```
> plot(default_pgmm)
```

Notice that by modifying the initialization strategy of the AECM algorithm, the available models, that is, the ones which can be estimated, may change. This issue is usually related to singularity of some cluster-specific covariance matrix. For instance, the hierarchical initialization reduces to  $k = 2$  and  $q = 4$  the number of clusters and



**Fig. 7.3** Economics data: BIC plots for the 12 PGMM models fitted by pgmmEM (default starting value option) for  $k = 1, \dots, 5$

factors, respectively. The standard model-based clustering initialization increases the range of the possible number of factors. The output of pgmmEM is an object of class pgmm containing the information about the best model (the one with the highest value of BIC/ICL) and the BIC/ICL values for each of the possible models. In this example, we are mainly interested in the classification (map). We report the following cross-tabulations and the corresponding Adjusted Rand Index (ARI, [47]) using University\_Type for comparison.

```
> names(default_pgmm)
[1] "map"          "model"        "g"            "q"            "bic"
[6] "zhat"         "load"         "noisev"
[9] "plot_info"    "summ_info"
> table(default_pgmm$map, University_Type)
  University_Type
    Private Public
1       7      0
2       0     48
> adjustedRandIndex(default_pgmm$map, University_Type)
[1] 1
> table(ICL_pgmm$map, University_Type)
  University_Type
    Private Public
1       7      0
2       0     48
> adjustedRandIndex(ICL_pgmm$map, University_Type)
[1] 1
> table(Random_pgmm$map, University_Type)
  University_Type
    Private Public
1       0      4
2       6      0
3       0     33
4       0     10
5       1      1
> round(adjustedRandIndex(Random_pgmm$map,
+                           University_Type), 2)
[1] 0.32
> table(Hier_pgmm$map, University_Type)
  University_Type
    Private Public
1       0     48
2       7      0
> adjustedRandIndex(Hier_pgmm$map, University_Type)
[1] 1
> table(Mclust_pgmm$map, University_Type)
  University_Type
    Private Public
1       0     48
2       7      0
> adjustedRandIndex(Mclust_pgmm$map, University_Type)
[1] 1
> table(diana_pgmm$map, University_Type)
```

```

University_Type
  Private  Public
1         2      48
2         5       0
> round(adjustedRandIndex(diana_pgmm$map ,
+           University_Type), 2)
[1] 0.78

```

As it can be noticed, BIC and ICL give identical results by using a  $k$ -Means initialization strategy. All the initialization strategies agree in partitioning the universities in  $k = 2$  clusters (even though DIANA leads to a different partition,  $n_1 = 50, n_2 = 5$ ) but random initialization strategy which suggests  $k = 5$  clusters. Notice that model-based clustering and hierarchical clustering as opposed to  $k$ -Means strategies give a perfect partition of the universities in Private and Public (Cluster 1 are Public and Cluster 2 are Private, ARI = 1), the solutions differ in terms of the number of factors (for  $k$ -Means and model-based clustering,  $q = 5$ ; for hierarchical clustering,  $q = 2$ ). The best model seems to be CUU (constrained factor loading matrix and unconstrained error matrix) with a random initialization,  $k = 5$  clusters and  $q = 1$  factor (BIC = 587.44). However, it is likely a spurious solution due to an almost empty cluster (only two universities). Thus, we choose as best solution the one corresponding to BIC = -1397.97 (the second highest value), i.e., the model UUU with  $q = 5$  and  $k = 2$ . This solution corresponds to the above-mentioned perfect partition of the universities into Private and Public (ARI = 1).

A final look at cluster-specific loading matrices gives us information on the relationship between the indicators and the factors within each cluster.

```

> round(default_pgmm$load[[1]], 2)
     [,1]   [,2]   [,3]   [,4]   [,5]
[1,]  0.26  0.33 -0.03  0.04 -0.11
[2,]  0.01  0.36 -0.33 -0.21 -0.19
[3,] -0.84 -0.26 -1.08 -0.08 -0.50
[4,]  0.34 -0.10 -0.01 -0.29 -0.10
[5,]  0.39 -0.02 -0.50  0.09 -0.24
[6,]  1.26  0.30  0.71 -0.09 -0.25
[7,]  0.39  0.30 -1.33  0.38 -0.37
[8,] -0.21  0.51  0.21 -0.23 -0.11
[9,] -0.98 -0.69 -0.36 -0.33 -0.25
[10,] -0.92 -0.66 -0.37 -0.31 -0.25
[11,]  0.63 -0.30  0.66  0.06  0.58
[12,] -0.91 -0.09 -0.53  0.03 -0.44
> round(default_pgmm$load[[2]], 2)
     [,1]   [,2]   [,3]   [,4]   [,5]
[1,] -0.58 -0.04 -0.10  0.21 -0.23
[2,] -0.53  0.34 -0.10  0.10 -0.15
[3,] -0.54  0.27 -0.12  0.09 -0.16
[4,] -0.04  0.20 -0.12 -0.25  0.04
[5,] -0.57  0.44 -0.11  0.11 -0.15
[6,]  0.02  0.34 -0.07 -0.23  0.08
[7,] -0.04  0.02 -0.03 -0.05 -0.01

```

```
[8 ,] -0.02 -0.04 -0.02 -0.01 -0.01
[9 ,] -0.53 -0.48 -0.35 -0.30 -0.24
[10,] -0.56 -0.38 -0.38 -0.34 -0.23
[11,]  0.30 -0.03 -0.08 -0.39  0.14
[12,] -0.28 -0.19  0.01  0.26 -0.16
```

As it can be noticed, indicators D1 and D2 have almost null factor loadings for all factors within the Public university cluster, while they are associated with Factors 3 and 2 within the Private university cluster, respectively. The remaining indicators are associated with Factor 1 especially within the Public university cluster. We are not really interested to give an interpretation of factors; thus, a solution with  $q = 5$  factors allows us to get a more flexible model. However, if it is not the case, the solution with  $q = 2$  factors corresponding to the third best model (CUCU model with  $\text{BIC} = -1407.19$ ) may be used. The corresponding loading matrix (constant across clusters) is as follows.

```
> round(Hier_pgmm$load, 2)
      [,1]   [,2]
[1,] -0.43 -0.25
[2,] -0.62 -0.24
[3,] -0.58 -0.31
[4,] -0.12 -0.02
[5,] -0.74 -0.23
[6,] -0.17  0.15
[7,] -0.03 -0.04
[8,]  0.02 -0.05
[9,]  0.09 -0.87
[10,] 0.04 -0.84
[11,] 0.33  0.13
[12,] -0.13 -0.27
```

We observe a clearer relationship between most of the productivity indicators (unless P4B having no clear association with any factor), D5 and Factor 1, D3, D4, D6 and Factor 2. D1 and D2 have almost zero factor loadings for all factors as for the Public university cluster of the previous solution. Here, the two factors can be interpreted more easily than before as productivity status and research availability. Thus, we could choose to take this solution losing relatively little in terms of BIC ( $-1397.95 + 1407.19 = 9.24$ ).

Finally, we observe that, as for the function **Mclust** of the package **mclust** [88] and the function **mixmodCluster** of the package **Rmixmod** [51], the function **pgmmEM** allows to perform model-based classification by specifying the argument **class**. The latter is a vector with length equal to the number of units, where the  $i$ -th entry is either zero if the component membership of the  $i$ -th unit is unknown or it corresponds to the component the unit is allocated to. In this example, we can assume to know only two-thirds of the universities type and we want to classify the remaining to assess whether the indicators are really discriminant. Note that the lowest value in **rG** cannot be smaller than **max(class)**.

```

> levels(University_Type) <- c(1, 2)
> University_Type <- as.integer(University_Type)
> University_Type
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[24] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[47] 2 2 1 1 1 1 1 1 1 1
> for(i in 1:dim(Economics)[1]){
+                               if(i%%3 == 0)
+                               {University_Type[i] = 0}
+ }
> z <- list()
> University_Type
[1] 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2
[24] 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2 2 0 2
[47] 2 0 1 1 0 1 1 0 1
> for(g in 1:6){
+           a <- Mclust(scale(Economics[, 1:12]),
+                         G = g)
+           z[[g]] <- a$classification
+           z[[g]] <- as.integer(z[[g]])
+         }
> classif_pgmm <- pgmmEM(scale(Economics[, 1:12]),
+                           rG = 2:3, rq = 1:7,
+                           class = University_Type,
+                           zstart = 3, zlist = z)
Based on the labelled and unlabelled data provided,
the best model (BIC) for the range of factors and
components used is a UUU model with q = 6 and G = 2.
The BIC for this model is -1226.675.
> cls_ind <- (University_Type == 0)
> table(University_Type)
  University_Type
  0   1   2
  18   5  32
> table(University_Type[cls_ind],
+        classif_pgmm$map[cls_ind])
  1   2
  0   2  16

```

As it can be noticed by comparing the two tables above, the model is able to properly classify the 18 universities.

### 7.3 Departures from Gaussian Mixture Models

A model is said to be robust if it is not sensitive to departures from the nominal model, in our case, a component-specific Gaussian density. Using GMMs for clustering, and interpreting components as clusters, implies a specific geometric shape for clusters. In real-life applications, one may encounter clusters with more general shapes, e.g., asymmetric, or data contaminated by bad points (outliers, spurious points, noise). Thus, using GMMs may lead to overestimate the number of clusters since one or more

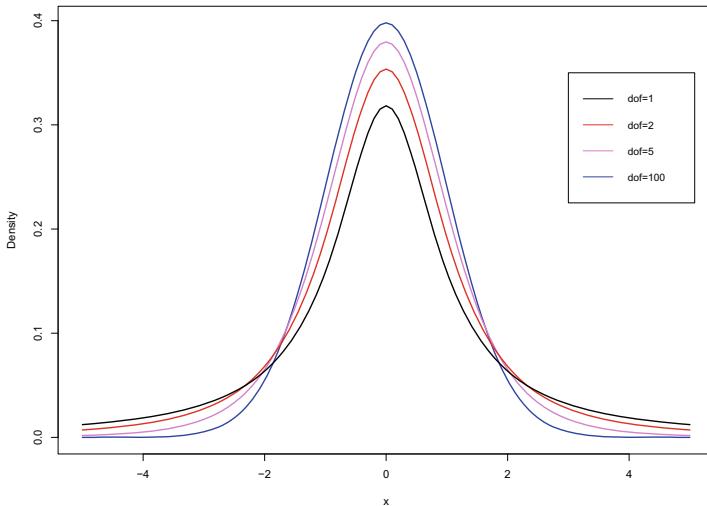
clusters may be represented by mixtures of Gaussian components or outliers can be identified as separate clusters. Clearly, these problems can produce poor parameter estimates. A good approach to obtain clusters with more flexible shapes consists in using mixture models with non-Gaussian component-specific densities. In the following sections, we review a number of model-based clustering approaches for multivariate continuous data with non-Gaussian component-specific distributions.

### 7.3.1 Mixtures of $t$ Distributions

The most natural departure from GMMs is the mixture of multivariate Student's  $t$  distributions [62, 79]. The  $t$  distribution has heavier tails than the Gaussian and provides a robust approach in the presence of outliers. When compared to the Gaussian, the  $t$  distribution is characterized by an additional parameter, the number of degrees of freedom (dof),  $\nu$ , which represents a robustness tuning parameter: as  $\nu$  tends to infinity, the  $t$  distribution approaches the Gaussian distribution. Thus, the smaller the  $\nu$ , the higher the protection against outliers. Figure 7.4 shows the density function of the  $t$  distribution for different values of  $\nu$ , obtained by the following code.

```
> curve(dt(x, 100), -5, 5, col = "blue",
+       ylab = "Density", lwd = 2)
> curve(dt(x, 5), -5, 5, add = TRUE, col = "violet",
+       lwd = 2)
> curve(dt(x, 2), -5, 5, add = TRUE, col = "red",
+       lwd = 2)
> curve(dt(x, 1), -5, 5, add = TRUE, col = "black",
+       lwd = 2)
> legend(3, 0.35, legend = c("dof = 1", "dof = 2",
+                            "dof = 5", "dof = 100"),
+         col = c("black", "red", "violet", "blue"),
+         lty = 1, cex = 0.9)
```

The component-specific density of the  $t$  distribution depends on the mean vector  $\mu_g$ , the scale matrix  $\Sigma_g$ , and the degrees of freedom  $\nu_g$ . Many parsimonious model families based on mixtures of  $t$  distributions have been proposed in the literature [1, 3, 57, 61, 89]. The latest version can probably be found in [2, 4]. Borrowing the idea of [9] on the eigendecomposition of the scale matrix and constraining the dof to be constant among clusters, they introduce a family of 28 models (32 if we consider the univariate models). This is implemented in the package **teigen** [4]. Note that  $\lambda_g$  cannot be interpreted as the volume because the cluster volume is a combination of  $\nu_g$  and  $\lambda_g$  (for details, see [63]). A summary of the multivariate models available in **teigen** is given in Table 7.4. Similar to **mclust**, the package **teigen** has a nomenclature for reporting model constraints. An ECM algorithm is used for parameter estimation.



**Fig. 7.4** Density function of the  $t$  distribution for different values of dof

### 7.3.2 Mixtures of Skew-Normal Distributions

While mixtures of  $t$  distributions have been introduced for handling heavy tails and outliers, skewness can be captured by using skewed component-specific distributions. Different types of skewed distributions may be considered. For instance, it is well known that household incomes are usually (positively) skewed: most of the citizens fall in low-average income group, while only few individuals may have a very high income and this makes the right tail heavier. Again, human longevity has a (negatively) skewed distribution: most people live beyond their middle age, or even older; however, some individuals die very young increasing the left tail of the distribution. Several mixtures are suitable for capturing such a skewness in the observed data. One of the most used is the mixture of skew-normal distributions [54, 55]. Originally introduced in [6], the multivariate skew-normal distribution has known several versions over the years: restricted, unrestricted, extended, and generalized, just to mention the most representative; see [52] for an overview. The idea is that the skew-normal distribution generalized with an additional shape parameter may provide a flexible approach to model asymmetric data. The Gaussian distribution is obviously a specific case of this class. In the original works [54, 55], model parameters have been estimated by the Monte Carlo EM (MCEM) algorithm; only later an analytical form for the E-step which avoids the need for a Monte Carlo integration has been derived [53].

**Table 7.4** Set of 28 parsimonious  $t$  distribution mixture models with  $k$  clusters and  $p$  variables. “C” denotes constrained, “U” denotes unconstrained, and “I” denotes the identity matrix

Model	$\lambda_g$	$\mathbf{D}_g$	$\mathbf{A}_g$	$v_g$	Free covariance and dof parameters
CIIC	C	I	I	C	$1 + 1$
CIIU	C	I	I	U	$1 + k$
UIIC	U	I	I	C	$k + 1$
UIIU	U	I	I	U	$k + k$
CICC	C	I	C	C	$p + 1$
CICU	C	I	C	U	$p + k$
UICC	U	I	C	C	$(p - 1) + k + 1$
UICU	U	I	C	U	$(p - 1) + k + k$
CIUC	C	I	U	C	$kp(k - 1) + 1$
CIUU	C	I	U	U	$kp(k - 1) + k$
UIUC	U	I	U	C	$kp + 1$
UIUU	U	I	U	U	$kp + k$
CCCC	C	C	C	C	$[p(p + 1)/2] + 1$
CCCU	C	C	C	U	$[p(p + 1)/2] + k$
UCCC	U	C	C	C	$[p(p + 1)/2] + (k - 1) + 1$
UCCU	U	C	C	U	$[p(p + 1)/2] + (k - 1) + k$
CUCC	C	U	C	C	$k[p(p + 1)/2] - (k - 1)p + 1$
CUCU	C	U	C	U	$k[p(p + 1)/2] - (k - 1)p + k$
UUCC	U	U	C	C	$k[p(p + 1)/2] - (k - 1)(p - 1) + 1$
UUCU	U	U	C	U	$k[p(p + 1)/2] - (k - 1)(p - 1) + k$
CCUC	C	C	U	C	$[p(p + 1)/2] + (k - 1)(p - 1) + 1$
CCUU	C	C	U	U	$[p(p + 1)/2] + (k - 1)(p - 1) + k$
CUUC	C	U	U	C	$k[p(p + 1)/2] - (k - 1) + 1$
CUUU	C	U	U	U	$k[p(p + 1)/2] - (k - 1) + k$
UCUC	U	C	U	C	$k[p(p + 1)/2] + (k - 1)p + 1$
UCUU	U	C	U	U	$k[p(p + 1)/2] + (k - 1)p + k$
UUUC	U	U	U	C	$k[p(p + 1)/2] + 1$
UUUU	U	U	U	U	$k[p(p + 1)/2] + k$

### 7.3.3 Handling Kurtosis and Skewness via Finite Mixture Models

Literature on kurtosis and skewness in the mixture framework is huge, and a systematic and comprehensive review is out of the scope of this book. However, to orient the reader, a brief overview will be given in the following pages. We separate such approaches into two main families handling kurtosis or both kurtosis and skewness. As far as the first category is entailed, we may refer to mixtures of normal-variance mixtures (also known as mixtures of Gaussian scale mixtures) [41] that include as

specific cases mixtures of  $t$  distributions, mixtures of contaminated Gaussian distributions [84] and mixtures of Pearson type VII distributions [91]. Other examples of the first category are mixtures of multivariate leptokurtic-normal distributions [8], mixtures of power exponential distributions [25, 102], mixtures of multiple scaled  $t$  distributions [31], and mixtures of multiple scaled contaminated normal distributions [85]. Notice that the latter two proposals allow for a variable-specific degree of kurtosis. Moreover, most of the above-mentioned proposals handle leptokurtosis, while only the proposals in [25, 102] deal also with platykurtosis. On the other hand, one of the first proposals handling both kurtosis and skewness is surely the skew- $t$  mixture model [53, 56, 77, 80, 95, 96]. Thanks to the skewness parameter that accommodates asymmetric data and the dof parameter that allows for heavy tails, skew- $t$  mixtures represent a general model including skew-normal,  $t$ , and Gaussian mixtures as special cases. Parsimonious mixtures of multivariate skew-normal and skew- $t$  distributions are discussed in [96]. Other examples of mixtures handling both kurtosis and skewness include generalized hyperbolic mixtures [20], shifted asymmetric Laplace mixtures [34], normal inverse Gaussian mixtures [50, 78, 90], variance-gamma mixtures [68], and multivariate contaminated shifted asymmetric Laplace mixtures [76]. Recently, a family of mixtures of multivariate skewed power exponential distributions that account for varying tail weight (heavy, Gaussian, or light), peakedness (thinner or thicker than Gaussian), and skewness has been proposed in [26]. Several packages deal with the above-cited models; **EMMIXskew** [97] focuses on mixtures of multivariate skew-normal and skew- $t$  distributions, **mixSPE** [27] on mixtures of multivariate power exponential and multivariate skewed power exponential distributions, **mixsmsn** [81] on mixtures of normal,  $t$ , skew-normal, skew-contaminated normal, skew-slash, and skew- $t$ , and **pkgMixGHD** [93, 94] on mixtures of generalized hyperbolic distributions and their extensions, **MixSAL** [35] on mixtures of shifted asymmetric Laplace distributions.

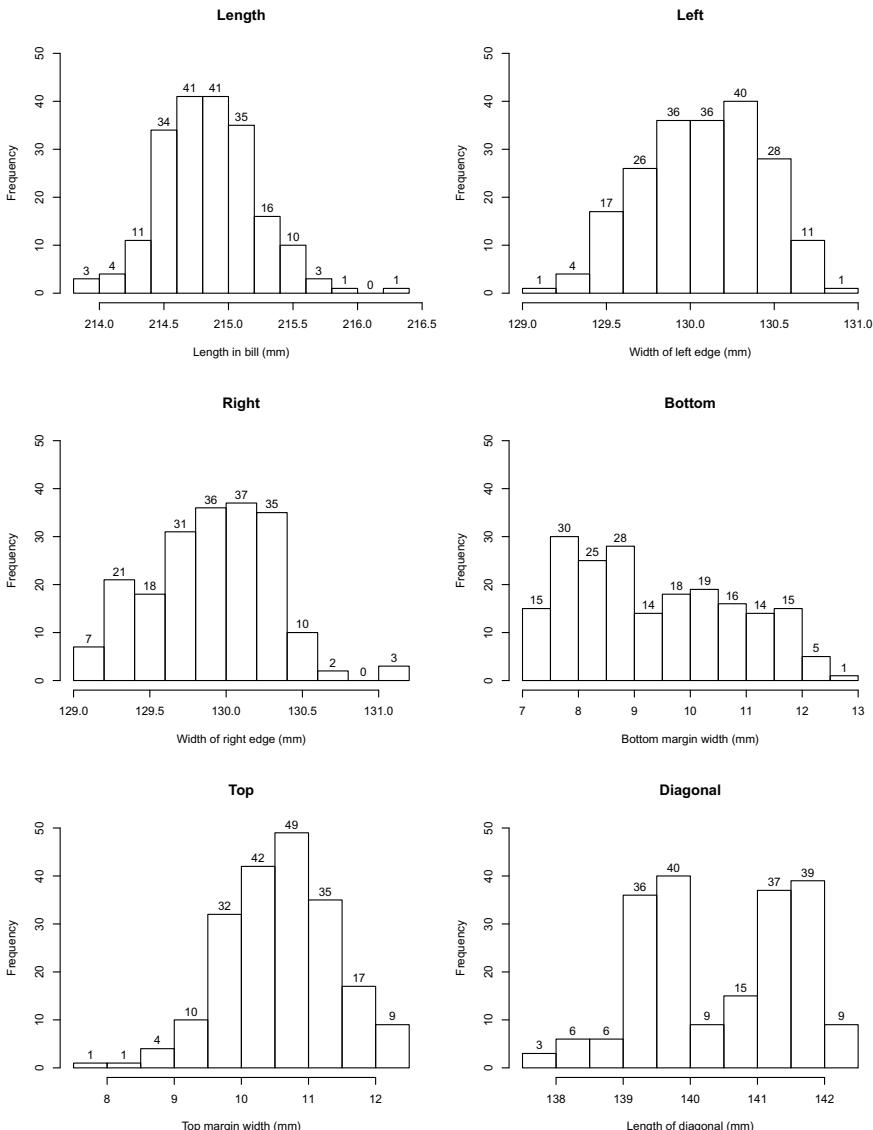
### 7.3.4 *teigen* and *mixsmsn*

As an illustration of non-Gaussian model-based clustering, we fit mixtures of  $t$ , skew-normal, and skew- $t$  distributions to the dataset `banknote` by using the package **teigen** (for mixtures of  $t$  distributions) and the package **mixsmsn** (for all of them). The data are available in **mclust** and was originally analyzed in [30, 59]. It contains the following  $p = 6$  physical measurements and information on the status of 100 genuine and 100 counterfeit old-Swiss 1000-franc banknotes:

- Status: The status of the banknote (genuine or counterfeit);
- Length: Length of bill (mm);
- Left: Width of left edge (mm);
- Right: Width of right edge (mm);
- Bottom: Bottom margin width (mm);

- Top: Top margin width (mm);
- Diagonal: Length of diagonal (mm).

Figure 7.5 shows the histograms of the  $p = 6$  features obtained by the following code.



**Fig. 7.5** banknote data: histograms of the  $p = 6$  physical measurements

```

> library(mclust)
> data("banknote")
> str(banknote)
'data.frame': 200 obs. of 7 variables:
 $ Status   : Factor w/ 2 levels "counterfeit",...: ...
 $ Length   : num  215 215 215 215 215 ...
 $ Left      : num  131 130 130 130 130 ...
 $ Right     : num  131 130 130 130 130 ...
 $ Bottom    : num  9 8.1 8.7 7.5 10.4 9 7.9 7.2 ...
 $ Top       : num  9.7 9.5 9.6 10.4 7.7 10.1 9.6 ...
 $ Diagonal  : num  141 142 142 142 142 ...
> attach(banknote)
> par(mfrow = c(3, 2))
> hist(Length, main = "Length", label = TRUE,
+       xlab = "Length in bill (mm)", ylim = c(0, 50))
> hist(Left, main = "Left", label = TRUE,
+       xlab = "Width of left edge (mm)",
+       ylim = c(0, 50))
> hist(Right, main = "Right", label = TRUE,
+       xlab = "Width of right edge (mm)",
+       ylim = c(0, 50))
> hist(Bottom, main = "Bottom", label = TRUE,
+       xlab = "Bottom margin width (mm)",
+       ylim = c(0, 50))
> hist(Top, main = "Top", label = TRUE,
+       xlab = "Top margin width (mm)",
+       ylim = c(0, 50))
> hist(Diagonal, main = "Diagonal", label = TRUE,
+       xlab = "Length of diagonal (mm)",
+       ylim = c(0, 50))

```

The histograms show that, marginally, all the features present asymmetric distributions, and Right and Diagonal have a bimodal distribution with asymmetric components. Our task consists of assessing the relative performance of the standard GMM compared with three non-Gaussian mixtures. First, we fit a mixture of  $t$  distributions to the dataset by using the function `teigen` of the package **teigen**. We should standardize the data matrix since variables are on different scales, and the model is not scale-invariant; standardization is, however, the default in `teigen`. The most important arguments of `teigen` for our analysis are summarized below.

- **x:** the data matrix represented by a numeric matrix, data frame, or vector (for univariate data).
- **Gs:** the number of clusters to fit (default: 1:9).
- **models:** a character vector indicating the models to fit.
  - "all": (default) runs 28 `teigen` models.
  - "dfunconstrained": runs the 14 unconstrained dof models.
  - "dfconstrained": runs the 14 constrained dof models.
  - "mclust": runs the 10 **mclust** models using the Gaussian distribution rather than the  $t$  distributions.

- "gaussian": is similar to "mclust" but includes four additional mixture models (see [4] for more details).
- **init**: a character string indicating the initialization method: "kmeans" (default), "hard" ( $z_{ig} \in \{0, 1\}$ ) and "soft" ( $z_{ig} \in [0, 1]$ ) random, and "emem" (i.e., the emEM strategy [15], see Sect. 6.3.3). Alternatively, a list of initializing classifications such that `init[[G]]` contains the initializing vector for all the  $k$  considered.
- **scale**: logical indicating whether the function should standardize the data (default: TRUE).
- **dfstart**: the initialized value for the dof (default is 50).
- **dfupdate**: string or logical indicating how the dof should be estimated (the default is "approx"; alternatively, "numeric" can be specified which makes use of `uniroot()`). If FALSE, the value from `dfstart` is used, and the dof values are not updated. If TRUE, "numeric" will be used for back-compatibility.

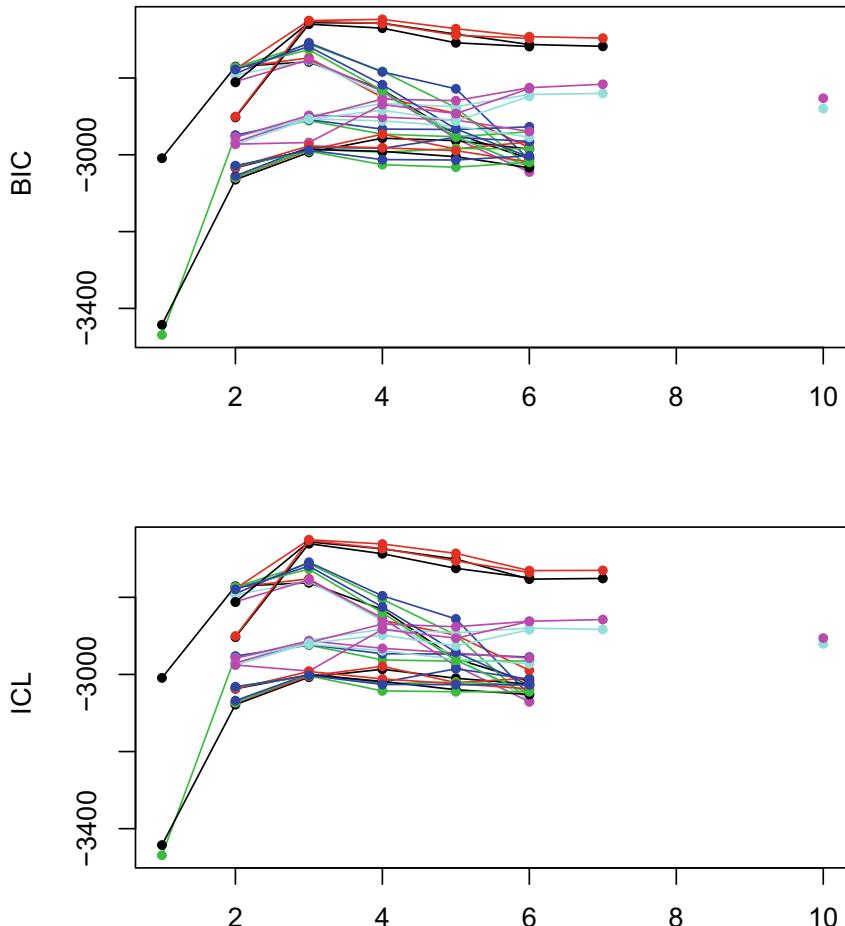
In the following, we run all the available teigen models for  $G_s = 1:10$  clusters initialized with different strategies and print the corresponding cross-tabulations and ARI indices. By setting `verbose = FALSE`, the output is not displayed.

```
> library(teigen)
> # Soft random starting values
> set.seed(13)
> teigen_soft <- teigen(banknote[, -1], Gs = 1:10,
+                         init = "soft", verbose = FALSE)
> # Hierarchical starting values
> hbank <- hclust(dist(scale(banknote[, -1])))
> initial <- lapply(1:10, function(i) cutree(hbank,
+                                         k = i))
> teigen_hier <- teigen(banknote[, -1], Gs = 1:10,
+                         init = initial,
+                         verbose = FALSE)
> # k-means starting values
> teigen_k <- teigen(banknote[, -1], Gs = 1:10,
+                      verbose = FALSE)
> # emem initialization
> set.seed(653)
> teigen_emem <- teigen(banknote[, -1], Gs = 1:10,
+                         init = "emem", ememargs =
+                         list(numstart = 100, iter = 5,
+                              model = "UUUU", init = "soft"),
+                         verbose = FALSE)
> teigen_soft$bestmodel
[1] "The best model (BIC of -2647.35) is CCCC with G=4"
> teigen_soft$iclresults$bestmodel
[1] "The best model (ICL of -2650.73) is CCCC with G=3"
> teigen_hier$bestmodel
[1] "The best model (BIC of -2647.33) is CCCC with G=4"
> teigen_hier$iclresults$bestmodel
[1] "The best model (ICL of -2651.35) is UCCC with G=3"
> teigen_k$bestmodel
```

```
[1] "The best model (BIC of -2647.48) is CCCC with G=4"
> teigen_k$iclresults$bestmodel
[1] "The best model (ICL of -2662.29) is CCCC with G=4"
> teigen_emem$bestmodel
[1] "The best model (BIC of -2647.32) is CCCC with G=4"
> teigen_emem$iclresults$bestmodel
[1] "The best model (ICL of -2650.65) is CCCC with G=3"
> table(teigen_soft$classification, Status)
  Status
  counterfeit genuine
  1          15      1
  2          85      0
  3           0     24
  4           0     75
> round(adjustedRandIndex(teigen_soft$classification,
+                           Status), 2)
[1] 0.68
> table(teigen_soft$iclresults$class, Status)
  Status
  counterfeit genuine
  1          15      1
  2           0     99
  3          85      0
> round(adjustedRandIndex(teigen_soft$iclresults$class,
+                           Status), 2)
[1] 0.86
> table(teigen_hier$classification, Status)
  Status
  counterfeit genuine
  1           0     24
  2           0     75
  3          15      1
  4          85      0
> round(adjustedRandIndex(teigen_hier$classification,
+                           Status), 2)
[1] 0.68
> table(teigen_hier$iclresults$class, Status)
  Status
  counterfeit genuine
  1          15      1
  2           0     99
  3          85      0
> round(adjustedRandIndex(teigen_hier$iclresults$class,
+                           Status), 2)
[1] 0.86
> table(teigen_k$classification, Status)
  Status
  counterfeit genuine
  1           0     75
  2           0     24
  3          85      0
  4          15      1
> round(adjustedRandIndex(teigen_k$classification,
+                           Status), 2)
```

```
[1] 0.6789772
> table(teigen_k$iclresults$class, Status)
  Status
  counterfeit genuine
  1          0      75
  2          0      24
  3         85      0
  4         15      1
> round(adjustedRandIndex(teigen_k$iclresults$class,
+                           Status), 2)
[1] 0.68
> round(adjustedRandIndex(teigen_emem$classification,
+                           Status), 2)
[1] 0.68
> table(teigen_emem$iclresults$class, Status)
  Status
  counterfeit genuine
  1          0      99
  2         85      0
  3         15      1
> round(adjustedRandIndex(teigen_emem$iclresults$class,
+                           Status), 2)
[1] 0.86
> table(teigen_emem$classification, Status)
  Status
  counterfeit genuine
  1          0      24
  2         15      1
  3          0      75
  4         85      0
```

If we select the best model by BIC, all the initializations give the same partition with  $k = 4$  clusters corresponding to the model CCCC (constrained scale matrix volume, orientation, shape, and constrained dof). On the other hand, selecting the best model by ICL, all the initializations give the same partition with  $k = 3$  clusters (corresponding to the model CCCC) unless  $k$ -Means which selects the same BIC-based solution of  $k = 4$  clusters. Moreover, the hierarchical initialization selects the model UCCC as the best one. The fact that all the initializations give similar results is quite surprising since, as we saw in Chap. 6, the EM algorithm is usually strongly sensitive to starting values. For this reason, the default  $k$ -Means initialization uses 50 random starting points and completely random initialization methods are usually not recommended. If we want to have deterministic results, we should initialize the EM algorithm by using a deterministic clustering method such as MBHAC for **mclust** (see Sect. 6.3.3). Figure 7.6 shows the BIC and ICL values for each model by using an emem initialization strategy. It is obtained by the following code.



**Fig. 7.6** banknote data: BIC (up) and ICL (down) plots for the 28 models fitted by `teigen` (`emem` initialization strategy) for  $k = 1, \dots, 10$

```
> par(mfrow = c(2, 1), mai = c(1, 1, 0.1, 1))
> matplot(t(teigen_emem$allbic), ylab = "BIC",
+           type = "o", pch = 20, lty = 1)
> matplot(t(teigen_emem$iclresults$allicl),
+           ylab = "ICL", type = "o", pch = 20, lty = 1)
```

Notice that the BIC values for  $k = 3$  and  $k = 4$  are quite similar, while the difference becomes slightly more visible for ICL values in  $k = 3$  and  $k = 4$ .

We compute the following cross-tabulation.

```
> table(teigen_emem$iclresults$class,
+        teigen_emem$classification)
  1   2   3   4
1 24   0 75   0
2   0   0   0 85
3   0 16   0   0
```

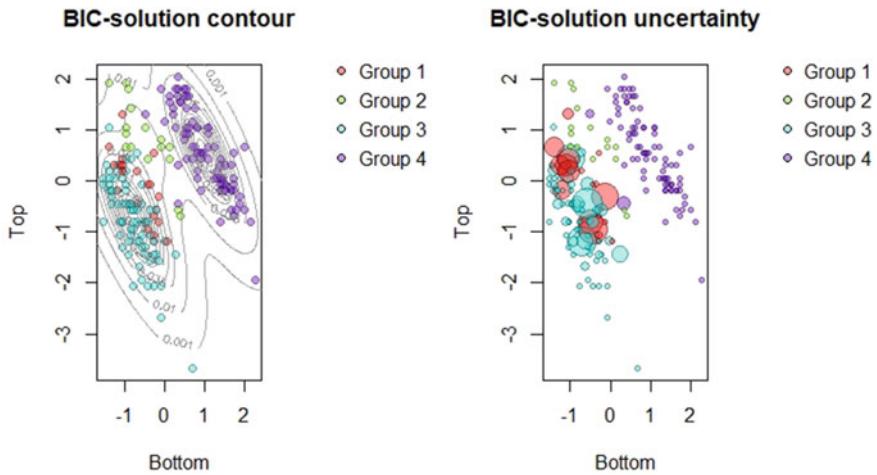
It is clear that the BIC-based solution uses one extra component: it splits the ICL-based Cluster 1 into two clusters (Clusters 1 and 3) as it is also confirmed by the fitted model parameters.

```
> teigen_emem$parameters$pig
[1] 0.123 0.089 0.373 0.424
> teigen_emem$parameters$mean
     [,1]    [,2]    [,3]    [,4]
[1,] 0.928  0.706  0.136 -0.647
[2,] 0.420  0.974  0.718 -0.487
[3,] -0.012 -0.904 -0.850 -0.819
[4,] -0.314  0.404  0.555  0.100
     [,5]    [,6]
[1,] -0.228  0.643
[2,]  0.832 -1.719
[3,] -0.721  1.002
[4,]  0.560 -0.746
> teigen_emem$parameters$sigma
, , 1
     [,1]    [,2]    [,3]    [,4]
[1,] 0.758  0.163  0.156  0.016
[2,] 0.163  0.449  0.268  0.065
[3,] 0.156  0.268  0.498  0.032
[4,] 0.016  0.065  0.032  0.253
[5,] 0.002 -0.022 -0.005 -0.289
[6,] 0.086  0.035  0.057 -0.023
     [,5]    [,6]
[1,] 0.002  0.086
[2,] -0.022 0.035
[3,] -0.005 0.057
[4,] -0.289 -0.023
[5,] 0.562 -0.006
[6,] -0.006 0.096
.....
> teigen_emem$parameters$df
[1] 25.085 25.085 25.085 25.085
> teigen_emem$iclresults$parameters$pig
[1] 0.495 0.425 0.080
> teigen_emem$iclresults$parameters$mean
     [,1]    [,2]    [,3]    [,4]
[1,] 0.210 -0.537 -0.621 -0.784
[2,] -0.315  0.404  0.554  0.999
```

```
[3,]  0.419  0.972  0.719 -0.489
      [,5]  [,6]
[1,] -0.601  0.920
[2,]  0.560 -0.747
[3,]  0.830 -1.713
> teigen_emem$iclresults$parameters$sigma
, , 1
      [,1]  [,2]  [,3]  [,4]
[1,]  0.824  0.288  0.233  0.029
[2,]  0.288  0.652  0.394  0.085
[3,]  0.233  0.394  0.570  0.043
[4,]  0.029  0.085  0.043  0.254
[5,]  0.041  0.044  0.037 -0.281
[6,]  0.059 -0.011  0.030 -0.027
      [,5]  [,6]
[1,]  0.041  0.051
[2,]  0.044 -0.010
[3,]  0.037  0.030
[4,] -0.281 -0.027
[5,]  0.578 -0.021
[6,] -0.021  0.106
...
> teigen_emem$iclresults$parameters$df
[1] 21.725 21.725 21.725
```

Considering the ICL solution, we observe that Cluster 1 is composed almost exclusively by genuine banknotes, Cluster 2 is composed exclusively by counterfeit banknotes, and Cluster 3 contains 15 counterfeit banknotes and 1 genuine banknote. It may indicate that these 16 banknotes have quite different features. On the other hand, the BIC solution splits the genuine cluster of the ICL solution into two clusters (Cluster 1 and Cluster 3) of 24 and 75 banknotes, respectively. We can produce a bivariate marginal contour plot for the `teigen` solution, where the user specifies the desired variates as the resolution of the contours and the uncertainty plot; the dot size is proportional to uncertainty in the classification. The options `xmarg` and `ymarg` specify the variables to plot. Figures 7.7 and 7.8 show the estimated density and uncertainty plots (dimensions 4 and 5) for the BIC and ICL solutions. For the BIC solution, the points at the boundary of Clusters 1 and 3 have the highest uncertainty. This also explains why they are merged together when  $k = 3$  clusters are considered. The following code is used.

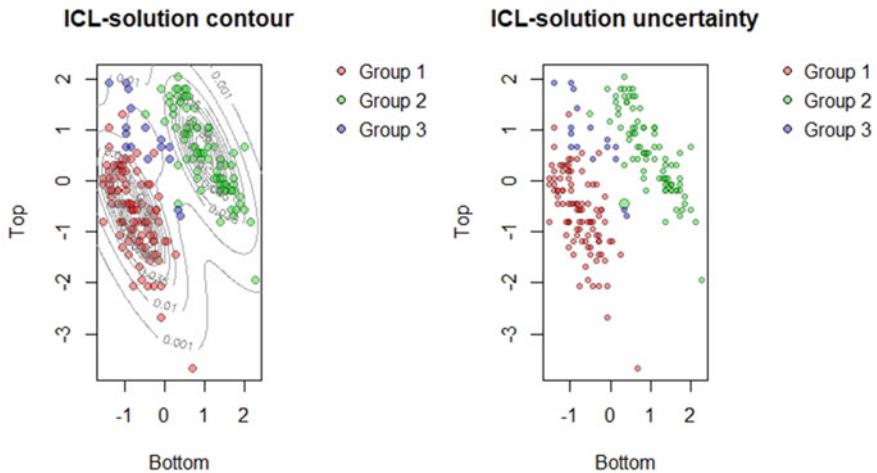
```
> teigen_bankICL <- teigen_emem
> teigen_bankICL$parameters <-
+              teigen_bankICL$iclresults$parameters
> teigen_bankICL$fuzzy <-
+              teigen_bankICL$iclresults$fuzzy
> teigen_bankICL$G <- teigen_bankICL$iclresults$G
> teigen_bankICL$bestmodel <-
+              teigen_bankICL$iclresults$bestmodel
> teigen_bankICL$classification <-
```



**Fig. 7.7** banknote data: estimated density (on the left) and uncertainty (on the right) corresponding to the BIC-best solution ( $k = 4$ ) fitted by `teigen`. Points colored by cluster membership. The larger the dot, the higher the uncertainty

```
+           teigen_bankICL$iclr$classification
> par(mfrow = c(1, 2))
> plot(teigen_emem, what = "contour", xmarg = 4, 5,
+       main = "BIC-solution contour",
+       draw.legend = FALSE)
> plot(teigen_emem, what = "uncertainty",
+       xmarg = 4, 5,
+       main = "BIC-solution uncertainty",
+       draw.legend = FALSE)
> par(mfrow = c(1, 2))
> plot(teigen_bankICL, what = "contour", xmarg = 4, 5,
+       main = "ICL-solution contour",
+       draw.legend = FALSE)
> plot(teigen_bankICL, what = "uncertainty",
+       xmarg = 4, 5,
+       main = "ICL-solution uncertainty",
+       draw.legend = FALSE)
```

Note that the function `teigen` allows for running models in parallel by specifying the argument `parallel.cores = TRUE`. In such a case, the function `detectCores` from the package **parallel** [86] detects the number of cores available and use all of them. In order to compare `teigen` results with the ones obtained by GMM, we specify the same initialization, tolerance levels, and convergence criterion that `Mclust` uses as default and we rerun `teigen` and `Mclust`.



**Fig. 7.8** banknote data: estimated density (on the left) and uncertainty (on the right) corresponding to the ICL-best solution ( $k = 3$ ) fitted by teigen. Points colored by cluster membership. The larger the dot, the higher the uncertainty

```

> sdata <- scale(banknote[, -1])
> mclustinit <- list()
> hcfit <- hcVVV(data = sdata)
> for(i in 1:5) {
+   mclustinit[[i]] <- hclass(hcfit, i)
+ }
> fitt <- teigen(sdata, Gs = 1:5, init = mclustinit,
+                   convstyle = "lop",
+                   eps = c(sqrt(.Machine$double.eps),
+                           1.e-5), verbose = FALSE)
> summary(fitt)
----- Summary for teigen -----
----- RESULTS -----
Loglik:          -1193.778
BIC:             -2647.173
ICL:             -2661.682
Model:            CCCC
Groups:           4
Clustering Table:

  1   2   3   4
16  75  24  85
> fitg <- Mclust(sdata, G = 1:5,
+                   initialization = list(hcfit))
> summary(fitg)
-----
Gaussian finite mixture model fitted by EM algorithm
-----

```

```

Mclust VEE (ellipsoidal, equal shape and orientation)
  model with 4 components:

log-likelihood   n  df          BIC          ICL
  -1191.595  200  51  -2653.405  -2666.898

Clustering table:
  1  2  3  4
16 99 47 38

> table(fitg$classification, Status)
  Status
  counterfeit genuine
  1           15      1
  2            0     99
  3           47      0
  4           38      0

> round(adjustedRandIndex(fitg$classification,
+                           Status), 2)
[1] 0.68

> table(fitt$classification, Status)
  Status
  counterfeit genuine
  1           15      1
  2            0     75
  3            0     24
  4           85      0

> round(adjustedRandIndex(fitt$classification,
+                           Status), 2)
[1] 0.68

> table(fitt$classification, fitg$classification)
  1  2  3  4
1 16  0  0  0
2  0 75  0  0
3  0 24  0  0
4  0  0 47 38

> round(adjustedRandIndex(fitt$classification,
+                           fitg$classification))
[1] 0.60

```

According to BIC, both algorithms suggest  $k = 4$  clusters, but the corresponding partitions are different, as also confirmed by the ARI value. However, we can observe that the 16 banknotes are clustered in a cluster apart and that GMM splits the cluster of 85 banknotes obtained by the teigen model and, vice versa, the teigen model splits the cluster of 99 banknotes obtained by the GMM.

In order to fit mixtures of skew-normal and skew- $t$  distributions, we use the function `smsn.search` of the package **mixsmsn**. Specifically, it searches for the best fitting Finite Mixture of Scale Mixtures of Skew-Normals (FMSMSN) with  $k$  taking values from `g.min` to `g.max`. Options are "t", "Skew.t", "Skew.nc", "Skew.slash", "Skew.normal", and "Normal". Available model selection criteria are "bic" (default), "aic", "edc", and "icl"; the kurtosis parameter is represented by `nu`. The initialization of the EM algorithm is

based on *k*-Means by default, but the user can pass a list with alternative parameter values for the argument `kmeans.param`. For more details on the input arguments, see `help(smsn.search)`; the proper definition of FMSMSN is described in [10, 11, 81].

For illustration purposes, we fit mixtures of `Skew.normal` and `Skew.t` distributions but we also fit mixtures of `Normal` and `t` distributions setting `g.min = 1` and `g.max = 10` to allow for direct comparability. The adopted information criteria are `bic` and `ic1` (the smaller the best).

```
> library(mixsmsn)
> Normal <- smsn.search(banknote[, -1], nu = 3,
+                         g.min = 1, g.max = 5,
+                         family = "Normal",
+                         criteria = "bic",
+                         error = 0.0001,
+                         iter.max = 100,
+                         calc.im = FALSE,
+                         uni.Gama = FALSE,
+                         kmeans.param = NULL)
> NormalICL <- smsn.search(banknote[, -1], nu = 3,
+                            g.min = 1, g.max = 5,
+                            family = "Normal",
+                            criteria = "ic1",
+                            error = 0.0001,
+                            iter.max = 100,
+                            calc.im = FALSE,
+                            uni.Gama = FALSE,
+                            kmeans.param = NULL)
> t <- smsn.search(banknote[, -1], nu = 3,
+                    g.min = 1, g.max = 5,
+                    family = "t", criteria = "bic",
+                    error = 0.0001,
+                    iter.max = 100,
+                    calc.im = FALSE,
+                    uni.Gama = FALSE,
+                    kmeans.param = NULL)
> tICL <- smsn.search(banknote[, -1], nu = 1,
+                       g.min = 1, g.max = 5,
+                       family = "t", criteria = "ic1",
+                       error = 0.0001, iter.max = 100,
+                       calc.im = FALSE,
+                       uni.Gama = FALSE,
+                       kmeans.param = NULL)
> skew_norm <- smsn.search(banknote[, -1], nu = 3,
+                           g.min = 1, g.max = 5,
+                           family = "Skew.normal",
+                           criteria = "bic",
+                           error = 0.0001,
+                           iter.max = 100,
+                           calc.im = FALSE,
+                           uni.Gama = FALSE,
```

```

+           kmeans.param = NULL)
> skew_normICL <- smsn.search(banknote[, -1], nu = 3,
+                               g.min = 1, g.max = 5,
+                               family = "Skew.normal",
+                               criteria = "icl",
+                               error = 0.0001,
+                               iter.max = 100,
+                               calc.im = FALSE,
+                               uni.Gama = FALSE,
+                               kmeans.param = NULL)
> skew_t <- smsn.search(banknote[, -1], nu = 3,
+                        g.min = 1, g.max = 5,
+                        family = "Skew.t",
+                        criteria = "bic",
+                        error = 0.0001,
+                        iter.max = 100,
+                        calc.im = FALSE,
+                        uni.Gama = FALSE,
+                        kmeans.param = NULL)
> skew_tICL <- smsn.search(banknote[, -1], nu = 3,
+                            g.min = 1, g.max = 5,
+                            family = "Skew.t",
+                            criteria = "icl",
+                            error = 0.0001,
+                            iter.max = 100,
+                            calc.im = FALSE,
+                            uni.Gama = FALSE,
+                            kmeans.param = NULL)
> Normal$best.model$bic
[1] 1699.32
> NormalICL$best.model$icl
[1] 1699.45
> t$best.model$bic
[1] 1698.32
> tICL$best.model$icl
[1] 1698.54
> skew_norm$best.model$bic
[1] 1730.99
> skew_normICL$best.model$icl
[1] 1731.01
> skew_t$best.model$bic
[1] 1711.18
> skew_tICL$best.model$icl
[1] 1711.35

```

For each mixture model, we may obtain a different solution due to local maxima. We adopt a multiple random-starting point strategy to mitigate the problem (results not shown here). Table 7.5 summarizes the values of the information criteria for each mixture model. We can see that the Normal and  $\tau$  mixture models have the best performances according to both criteria (they have very close values) corresponding to the same partition into  $k = 3$  clusters (cluster sizes are 17, 99, and 84). On the

**Table 7.5** banknote data: BIC and ICL information criteria corresponding to the different mixture types and the number of clusters

Mixture type	$k$	BIC	ICL
Normal	3	1699.320	1699.445
t	3	1698.321	1698.540
Skew.normal	2	1730.989	1731.011
Skew.t	2	1711.184	1711.346

other hand, the `Skew.normal` mixture model has the highest value followed by the `Skew.t` mixture model. Nonetheless, in this case, the `Normal` and `t` mixture models should not be selected because a limited increase in fit is achieved at the cost of a much higher number of parameters. Therefore, the most valuable models balancing fit and parsimony appear to be `skew-normal` and `skew-t`.

Finally, looking at the following cross-tabulations and ARI values, we can see that the worst models in terms of information criteria correspond to the best models in terms of recovering the known partition (ARI is about 0.98). It is likely due to the fact that the two information criteria tend to overpenalize and fit and partition recovering are not coherent task.

```
> table(Normal$best.model$group, Status)
  Status
    counterfeit genuine
  1          16      1
  2          84      0
  3           0     99
> round(adjustedRandIndex(Normal$best.model$group,
+                           Status), 2)
[1] 0.85
> table(NormalICL$best.model$group, Status)
  Status
    counterfeit genuine
  1          16      1
  2           0     99
  3          84      0
> round(adjustedRandIndex(NormalICL$best.model$group,
+                           Status), 2)
[1] 0.85
> table(t$best.model$group, Status)
  Status
    counterfeit genuine
  1          16      1
  2          84      0
  3           0     99
> round(adjustedRandIndex(t$best.model$group,
+                           Status), 2)
[1] 0.85
> table(tICL$best.model$group, Status)
```

```

Status
counterfeit genuine
1           84      0
2           0      99
3          16      1
> round(adjustedRandIndex(tICL$best.model$group,
+     Status), 2)
[1] 0.85
> table(skew_norm$best.model$group, Status)
  Status
  counterfeit genuine
  1           0      99
  2          100      1
> round(adjustedRandIndex(skew_norm$best.model$group,
+     Status), 2)
[1] 0.98
> table(skew_normICL$best.model$group, Status)
  Status
  counterfeit genuine
  1          100      1
  2           0      99
> round(adjustedRandIndex(
+     skew_normICL$best.model$group, Status), 2)
[1] 0.98
> table(skew_t$best.model$group, Status)
  Status
  counterfeit genuine
  1          100      1
  2           0      99
> round(adjustedRandIndex(skew_t$best.model$group,
+     Status), 2)
[1] 0.98
> table(skew_tICL$best.model$group, Status)
  Status
  counterfeit genuine
  1          100      1
  2           0      99
> round(adjustedRandIndex(skew_tICL$best.model$group,
+     Status), 2)
[1] 0.98

```

## 7.4 Dealing with Outliers

Several approaches have been proposed to deal with outliers in model-based clustering. A straightforward way consists of adding a noise component to the mixture model to represent points that are not consistent with any of the component-specific densities. This is similar to the introduction of a noise cluster in the fuzzy framework (see Sect. 5.8). In [9, 28], a uniform mixture component is added on the convex hull of the data and such proposals are further extended in [32, 33]. Related works include

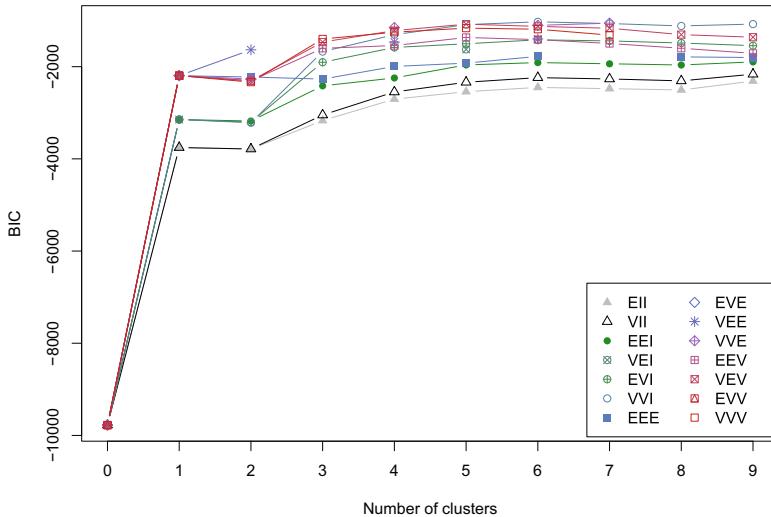
[22] where uniform mixture components are added and [42] where the uniform distribution is replaced by an improper uniform component, an idea further developed in [23, 24, 45].

To give a better insight behind these strategies, let us describe the simplest and most known proposal consisting in adding a further component, described by uniform distribution on the whole data region. This corresponds to the assumption that outliers are generated by a homogeneous Poisson process on the data region. The GMM becomes

$$f(\mathbf{x}_i | \Psi) = \frac{\pi_0}{V} + \sum_{g=1}^k \pi_g f_g(\mathbf{x}_i | \theta_g), \quad (7.6)$$

where  $\pi_0$  represents the expected proportion of outliers in the data and  $V$  is the volume of the data region. Specifically,  $V$  can be defined as the volume of the smallest axis-aligned hypercube containing the data or as the volume of the smallest hypercube aligned with the PCs of the data. In the package **mclust**, the default value for  $V$  is the minimum of these two quantities. The functions `Mclust` and `mclustBIC` allow for specifying a logical or numeric vector (`noise`) in the `initialization` argument indicating an initial guess about noisy data. If numeric, the entries should correspond to row indices. If supplied, a noise term will be added to the model in the estimation. In the following, the code for clustering the dataset `customers` [5] included in the package **datasetsICR** [40] regarding clients of a wholesale distributor (already presented in Sects. 2.6.1 and 6.8.1), where 150 noisy units are added according to the concept of Poisson noise, is listed.

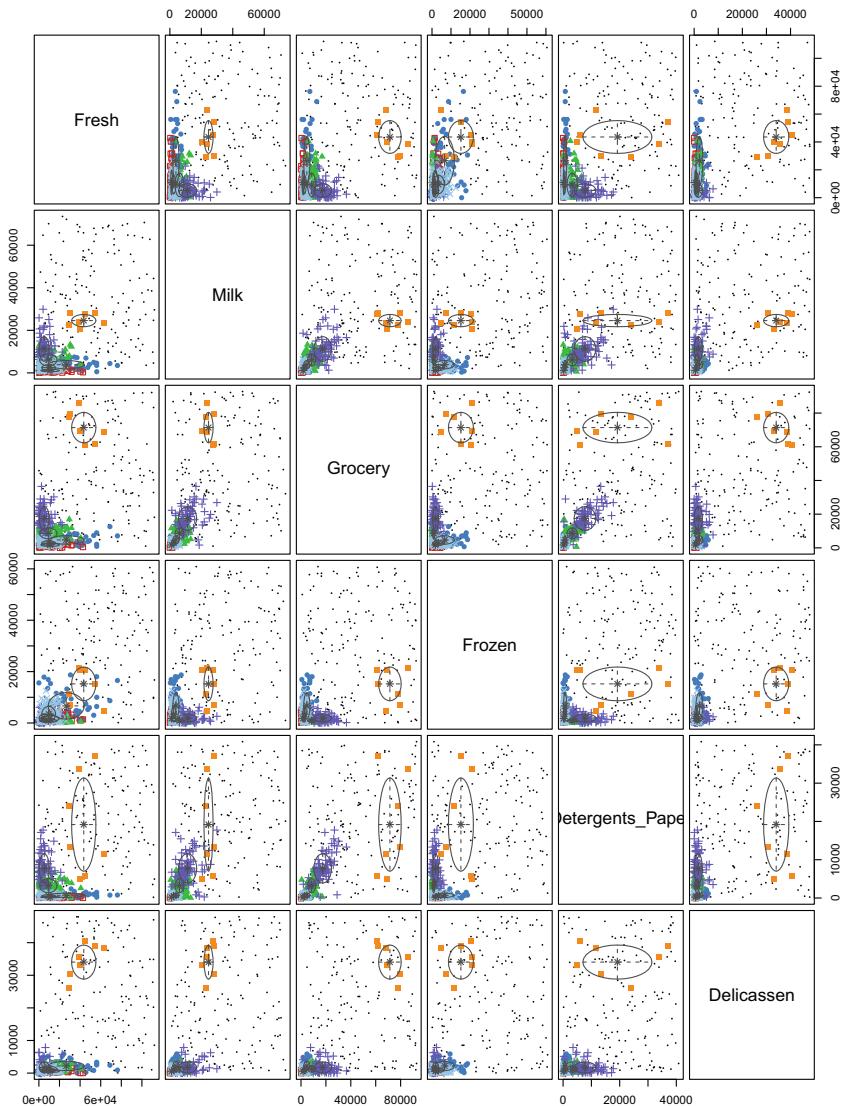
```
> library(datasetsICR)
> data("customers")
> Noise <- 150
> set.seed(0)
> poissonNoise <- apply(apply(customers[, 3:8], 2,
+                                range), 2, function(x, n)
+                                runif(n, min = x[1] - .1,
+                                      max = x[2] + .1), n = Noise)
> set.seed(0)
> noiseInit <- sample(c(TRUE, FALSE),
+                       size = nrow(customers) + Noise,
+                       replace = TRUE, prob = c(3, 1))
> datanoise <- rbind(customers[, 3:8], poissonNoise)
> library(mclust)
> mod <- Mclust(scale(datanoise),
+                  initialization =
+                  list(noise = noiseInit))
> plot(mod, what = "classification")
> plot(mod, what = "uncertainty")
> custNbic <- mclustBIC(data = scale(datanoise),
+                         initialization =
+                         list(noise = noiseInit))
> plot(custNbic, xlab = "Number of clusters")
> mod$modelName
```



**Fig. 7.9** customers data with artificial Poisson noise: BIC plots for the 14 GMMs fitted by `mclustBIC` with noise component for  $k = 0, \dots, 9$

```
[1] "VVI "
> mod$G
[1] 6
> table(mod$classification)
  0   1   2   3   4   5   6
164  53  84 102  73   7 107
```

Figure 7.9 shows the corresponding BIC plots. As we can see, if no clusters are found (i.e., the dataset consists only of Poisson noise) the situation is depicted in the plot as 0 mixture components. BIC chooses model VVI with  $k = 6$  clusters and a noise component. Figure 7.10 displays the corresponding partition. Note that BIC tends to overestimate the number of outliers (164 instead of 150), including the ones that are on the border in each cluster. This is confirmed by Fig. 7.11 as the uncertainty is quite high on the cluster borders. Note that we used a random initial estimate for noise just for illustrative purposes. However, the estimation results may be quite sensitive to the initialization; a good way to initialize the assignment to the noise component is via a nearest neighbor cleaning method [21]. This essentially denotes points as outliers if they are substantially far from the others. The method is implemented in the function `NNclean` of the package `prabclus` [46]. Since the method is not scale-invariant, data should be standardized if variables are measured according to different units of measurements. An extension of the nearest neighbor cleaning method is the Nearest Neighbor Variance Estimation method (NNVE) [98]: it adds simulated noise to the data before applying the nearest neighbor cleaning method.



**Fig. 7.10** customers data with artificial Poisson noise: clustering results corresponding to the best GMM (model=VVI,  $k = 6$  and a noise component) fitted by Mclust with default starting value option and noise component



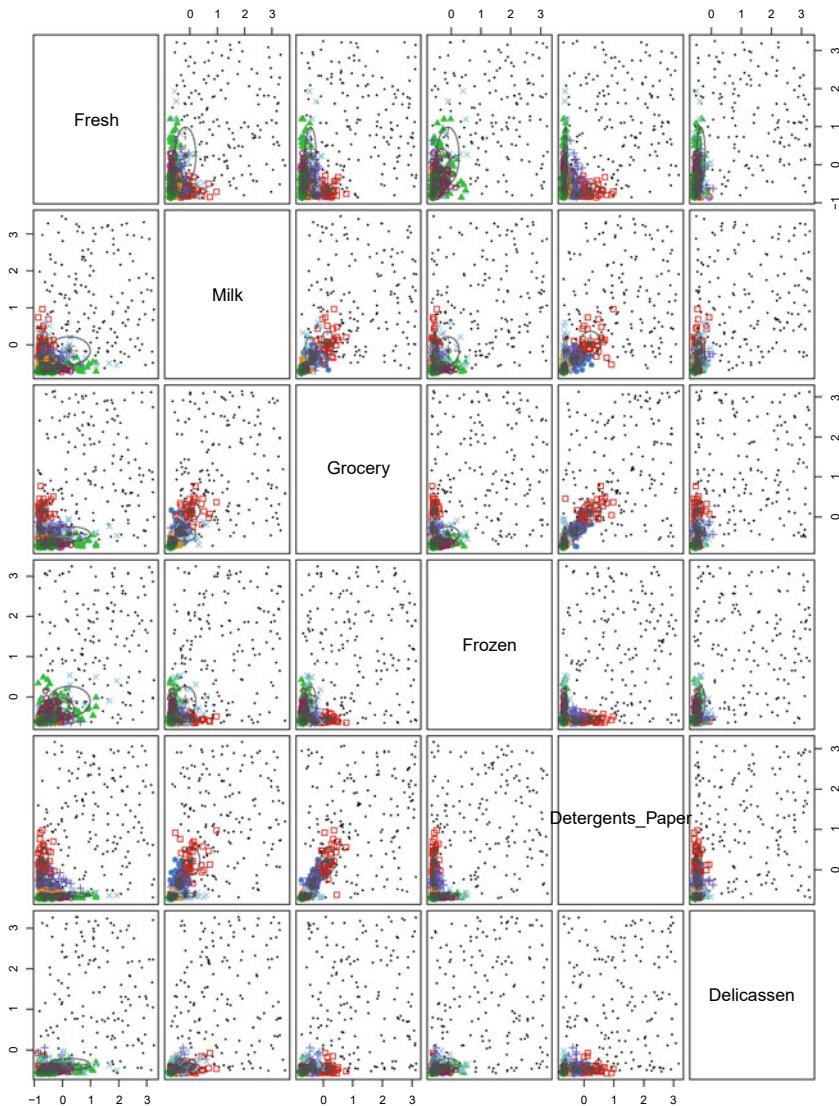
**Fig. 7.11** *customers* data with artificial Poisson noise: uncertainty plot corresponding to the best GMM (model=VVI,  $k = 6$  and a noise component) fitted by *Mcclus* with default starting value option and noise component

The method is implemented in the function `cov.nnve` of the package **covRobust** [99]. The application to the dataset `customers` is obtained through the following code, and the corresponding clustering results are displayed in Figs. 7.12 and 7.13.

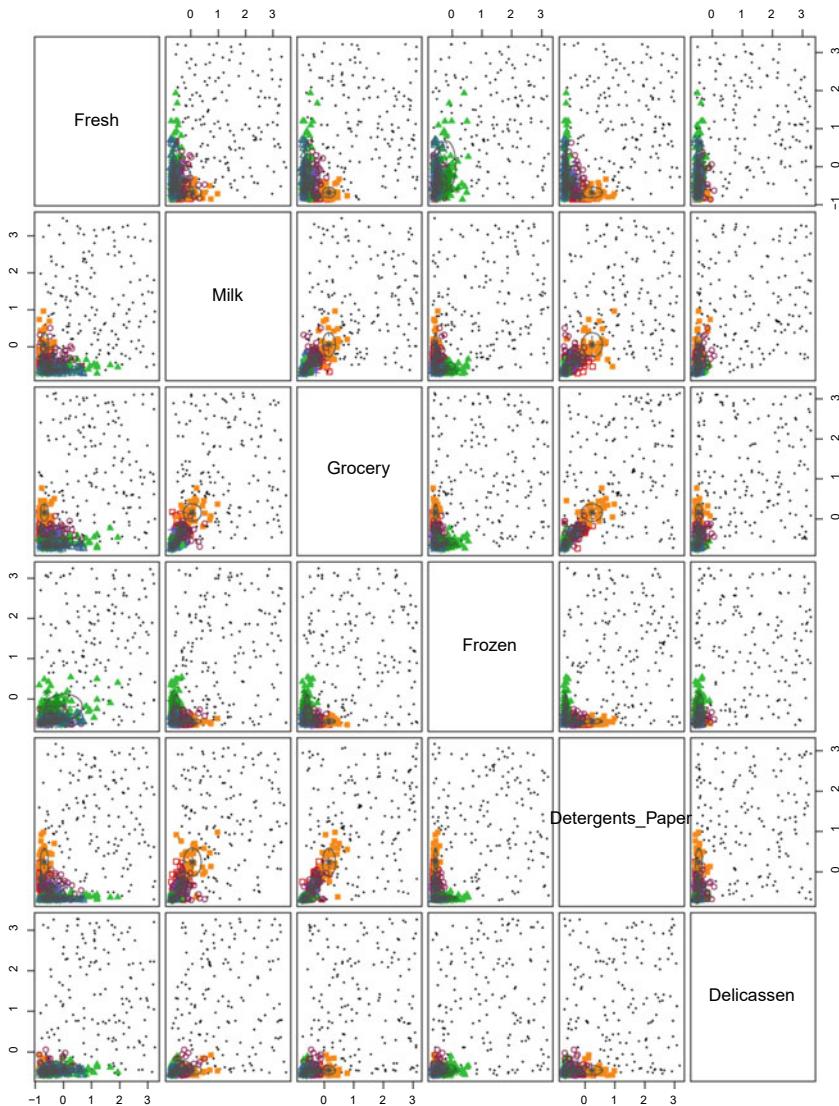
```
> # Noise detection
> library(prabclus)
> NNcleandet <- NNclean(scale(datanoise), k = 12)
> modNNclean <- Mclust(scale(datanoise),
+                         initialization = list(noise
+                         = (NNcleandet$z == 0)))
> plot(modNNclean, what = "classification")
> library(covRobust)
> # Robust covariance estimation
> NNVE <- cov.nnve(scale(datanoise))
> modNNVE <- Mclust(scale(datanoise),
+                      initialization = list(noise
+                      = (NNVE$classification == 0)))
> plot(modNNVE, what = "classification")
> table(modNNclean$classification)
  0   1   2   3   4   5   6   7   8 
167  50  41  68  67  54  17  74  52 
> table(modNNVE$classification)
  0   1   2   3   4   5   6   7   8   9 
170  48  44  56  43  38  56  36  55  44
```

Both methods identify a few more points as outliers than the random method (167 and 170 vs. 164) even though the difference is not so big.

Among alternative approaches to deal with outliers, it is worth mentioning the popular trimming approach which aims to identify and remove outliers rather than model them with flexible mixture components. It should be noted that the proportion of trimming is strictly linked with the weight of the noise component. Originally proposed in [37, 38], the idea of trimming consists of excluding units identified as outliers when estimating model parameters. The trimming gives robust estimates by allowing for a pre-specified proportion of bad points; it can be considered as a robust EM-type algorithm for mixture models with an additional trimming step. The proposal in [38] has been applied to GMM and implemented in the package **tclust** [36]. Clearly, its goal is rather different from the methods discussed before which try to model the noisy data. Finally, a nice approach to deal with bad points is the mixture of contaminated Gaussian distributions mentioned in Sect. 7.3.3 and implemented in the package **ContaminatedMixt** [83]. This model can be seen as a special case of the hierarchical mixture model, where each of the  $k$  components at the top layer is itself a mixture of components. The peculiarity of the proposal is that the percentage of bad points does not need to be specified a priori. Note that, being defined as a mixture of mixture, it is not identifiable without further constraints. A combination between the trimming approach and the mixture of contaminated Gaussian distributions has been proposed by [29].



**Fig. 7.12** customers data with artificial Poisson noise: clustering results corresponding to the best GMM ( $k = 8$ ) fitted by Mclust with noise component initialized by the nearest neighbor cleaning method



**Fig. 7.13** *customers* data with artificial Poisson noise: clustering results corresponding to the best GMM ( $k = 9$ ) fitted by Mclust with noise component initialized by the NNVE method

## 7.5 Cluster or Component?

As it has been already mentioned in Sect. 6.1, if we use GMMs for clustering purposes, each mixture component is interpreted as a cluster. However, if clusters (or some of them) are not well described by such a parametric shape, penalized-likelihood criteria may likely overestimate the number of clusters even though a good density estimation is possible. In the previous sections, we introduced several mixture models that give a more flexible shape to clusters by means of non-Gaussian component-specific densities. Another widely used approach to deal with clusters with a non-Gaussian distribution consists of keeping the GMMs solution and merge mixture components that are close together based on a given criterion. In [43], the following procedure is applied.

1. Set the number of clusters equal to the number of components estimated by GMMs.
2. Find the pair of clusters most promising to merge.
3. Apply a stopping rule to decide whether to merge them to form a new cluster. If merged, go to step 2; otherwise, stop the procedure and take the current partition as the final solution.

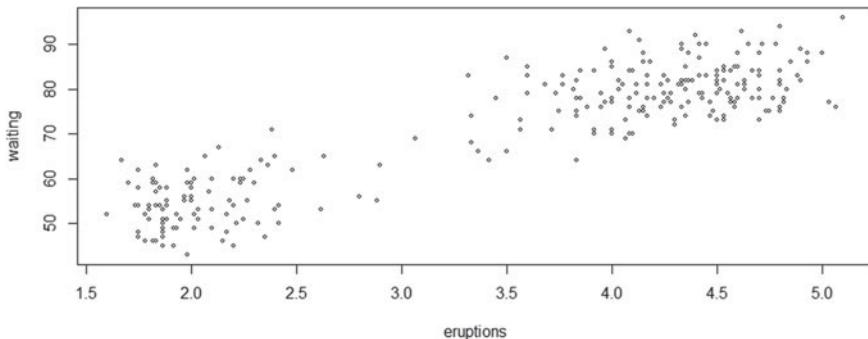
Different choices of GMMs, stopping rules, and methods for finding the most promising pair of components to merge may lead to different merging methods. However, in the literature, particular attention has been paid to the selection of pairs to merge and to the stopping rule.

To give an example, we focus on the approach proposed by [12]. Its basic idea consists in choosing the pair of clusters to merge at each stage as the one that minimizes the increase in entropy over all pairs, where the entropy is computed by using the posterior probabilities

$$\text{Entr}(k) = - \sum_{i=1}^n \sum_{g=1}^k w_{ig} \log(w_{ig}). \quad (7.7)$$

Taking into account that  $\text{Entr}(k) = 0$  for a clustering with no uncertainty (i.e.,  $w_{ig} \in \{0, 1\}, \forall i, g$ ) while the maximum level is attained when the clustering is uninformative (i.e.,  $w_{ig} = 1/k, \forall i, g$ ), the merging process stops when a merge leads to a large increase in entropy. This approach is implemented by the function `clustCombi` of **mclust**. See [43] for an overview of other criteria in the same framework, which are implemented in the function `mergenormals` of the package **fpc** [44].

As an illustration of the entropy-based merging method implemented in **mclust**, we consider a toy example involving the well-known Old Faithful dataset, which provides the waiting time between subsequent eruptions (`waiting`) and the duration of the eruptions (`eruptions`) for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. The dataset `faithful` included in the package **mclust** can be read and data plotted (Fig. 7.14) as follows.

**Fig. 7.14** faithful data: scatterplot

```
> library(mclust)
> data("faithful")
> plot(faithful, cex = 0.5)
```

Now, we fit a GMM by using default options of the function `Mclust`.

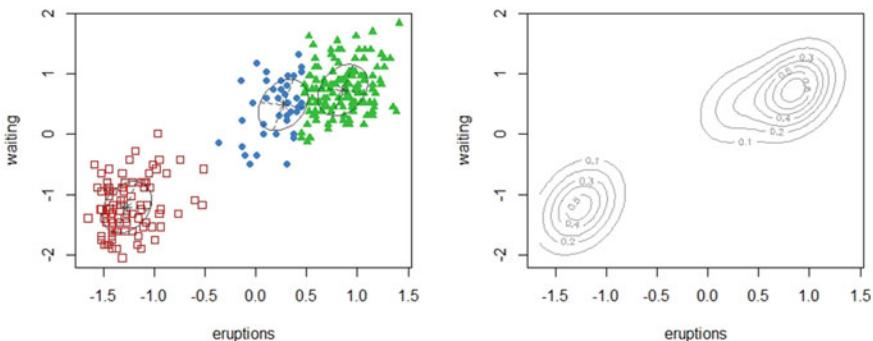
```
> mod2 <- Mclust(scale(faithful))
> summary(mod2)
-----
Gaussian finite mixture model fitted by EM algorithm
-----
Mclust EEE (ellipsoidal, equal volume, shape and
    orientation) model
with 3 components:
  log-likelihood   n  df      BIC      ICL
    -380.5144  272 11  -822.6927 -866.328
Clustering table:
  1   2   3
 41  97 134
> par(mfrow = c(1, 2))
> plot(mod2, what = "classification")
> plot(mod2, what = "density")
> modICL <- mclustICL(scale(faithful))
> summary(modICL)
Best ICL values:
          VEV ,2      VVV ,2      VEE ,2
ICL     -829.137  -831.087  -831.785
ICL diff    0.000    -1.951    -2.649
> par(mfrow = c(1, 2))
> plot(Mclust(scale(faithful), 2, modelNames = "VEV"),
+       what = "classification")
> plot(Mclust(scale(faithful), 2, modelNames = "VEV"),
+       what = "density")
```

According to BIC, the best model is a three-component mixture with common covariance matrix (EEE). From Fig. 7.15, it is clear that one component is used to model the cluster of units having both low duration and low waiting times, while two components are needed to approximate the skewed distribution of the units with larger duration and waiting times. On the other hand, ICL selects as the best model a two-component mixture (Fig. 7.16) with unequal volume and orientation and common shape clusters (VEV). The latter criterion seems to perform better than BIC, since from a visual inspection, the true number of clusters seems to be equal to  $k = 2$ . However, note that the ICL-based solution merges the two components at the top right of Fig. 7.15 in one cluster represented by a single Gaussian distribution, losing in terms of fit with respect to the BIC-based solution.

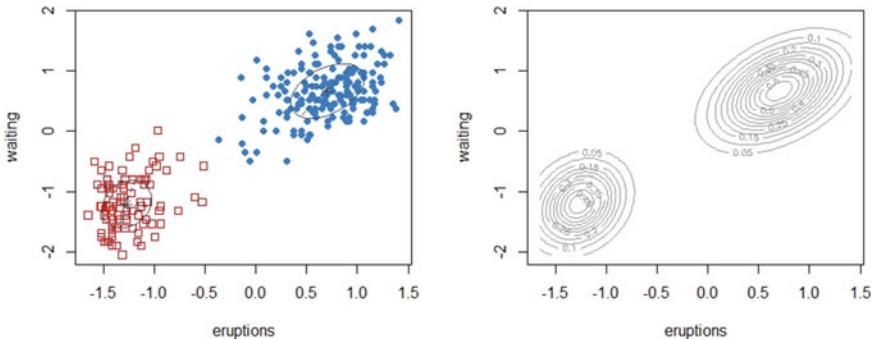
Now, we apply entropy-based merging by the function `clustCombi` which inputs the object returned by `Mclust`. Moreover, we use the function `clustCombiOptim` to select the optimal number of clusters by combining mixture components based on entropy-based merging method.

```
> output <- clustCombi(mod2)
> plot(output, what = "classification")
> combiOptim <- clustCombiOptim(output)
```

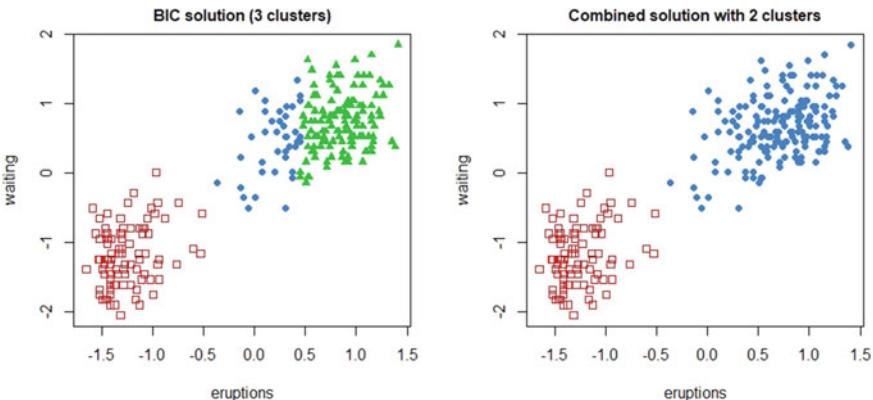
Figure 7.17 shows the results of the merging mixture components, as one goes from the BIC solution with  $k = 3$  clusters to  $k = 2$  clusters. The selected two-cluster solution obtained by means of `clustCombiOptim` combines the two components at the top right of the plot, as desired. The number of clusters is the same selected by ICL, but the resulting densities are different. In the ICL solution, the two components at the top right are represented by a single Gaussian distribution, which is clearly not fully satisfactory. On the other hand, in the merged two-cluster solution, the component at the top right is represented by a two-component Gaussian mixture,



**Fig. 7.15** *faithful* data: clustering results (on the left) and density estimate (on the right) corresponding to the best GMM ( $k = 3$ , `model=EEE`,  $\text{BIC} = -822.69$ ) fitted by `Mclust` with default starting value options



**Fig. 7.16** faithful data: clustering results (on the left) and density estimate (on the right) corresponding to the ICL-best GMM ( $k = 2$ , model=VEV, ICL=  $-829.14$ ) fitted by `mclustICL` with default starting value options



**Fig. 7.17** faithful data: results from the entropy-based merging method applied to the solution having  $k = 3$ , model=EEE, BIC=  $-822.69$

retaining the true, skewed nature of the resulting cluster. Of course, this is just a toy example with a small number of clusters. In more complex situations, we can also make use of some entropy plots which may help select the “proper” number of clusters. For more details, see [12].

## References

1. Andrews, J.L., McNicholas, P.D.: Extending mixtures of multivariate t-factor analyzers. *Stat. Comput.* **21**, 361–373 (2011)
2. Andrews, J.L., McNicholas, P.D.: Model-based clustering, classification, and discriminant analysis via mixtures of multivariate  $t$ -distributions: the tEIGEN family. *Stat. Comput.* **22**, 1021–1029 (2012)

3. Andrews, J.L., McNicholas, P.D., Subedi, S.: Model-based classification via mixtures of multivariate  $t$ -distributions. *Comput. Stat. Data Anal.* **55**, 520–529 (2011)
4. Andrews, J.L., Wickins, J.R., Boers, N.M., McNicholas, P.D.: teigen: An R package for model-based clustering and classification via the multivariate  $t$  distribution. *J. Stat. Softw.* **83**, 1–32 (2018)
5. Abreu, N.: Analise do perfil do cliente Recheio e desenvolvimento de um sistema promocional. Mestrado em Marketing, ISCTE-IUL, Lisbon (2011)
6. Azzalini, A., Dalla Valle, A.: The multivariate skew-normal distribution. *Biometrika* **83**, 715–726 (1996)
7. Baek, J., McLachlan, G.J., Flack, L.: Mixtures of factor analyzers with common factor loadings: applications to the clustering and visualisation of high-dimensional data. *IEEE T. Pattern Anal.* **32**, 1298–1309 (2009)
8. Bagnato, L., Punzo, A., Zozia, M.G.: The multivariate leptokurtic-normal distribution and its application in model-based clustering. *Can. J. Stat.* **45**, 95–119 (2017)
9. Banfield, J.D., Raftery, A.E.: Model-based Gaussian and non-Gaussian clustering. *Biometrics* **49**, 803–821 (1993)
10. Barbosa Cabral, C.R., Lachos, V.H., Prates, M.O.: Multivariate mixture modeling using skew-normal independent distributions. *Comput. Stat. Data Anal.* **56**, 126–142 (2012)
11. Basso, R.M., Lachos, V.H., Barbosa Cabral, C.R., Ghoshc, P.: Robust mixture modeling based on scale mixtures of skew-normal distributions. *Comput. Stat. Data Anal.* **54**, 2926–2941 (2010)
12. Baudry, J.P., Raftery, A.E., Celeux, G., Lo, K., Gottardo, R.: Combining mixture components for clustering. *J. Comput. Graph. Stat.* **19**, 332–353 (2010)
13. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
14. Bhattacharya, S., McNicholas, P.D.: A LASSO-penalized BIC for mixture model selection. *Adv. Data Anal. Class.* **8**, 45–61 (2014)
15. Biernacki, C., Celeux, G., Govaert, G.: Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models. *Comput. Stat. Data An.* **41**, 561–575 (2003)
16. Blostein, M., Punzo, A., McNicholas, P.D.: mcgfa: Mixtures of Contaminated Gaussian Factor Analyzers. R package version 2.2.1 (2019). <https://CRAN.R-project.org/package=mcgfa>
17. Bouveyron, C., Celeux, G., Murphy, T.B., Raftery, A.E.: Model-based Clustering and Classification for Data Science: With Applications in R. Cambridge University Press, Singapore (2019)
18. Bouveyron, C., Girard, S., Schmid, C.: High-dimensional data clustering. *Comput. Stat. Data Anal.* **52**, 502–519 (2007)
19. Bouveyron, C., Girard, S., Schmid, C.: High dimensional discriminant analysis. *Comm. Stat. Theor. Met.* **36**, 2607–2623 (2007)
20. Browne, R.P., McNicholas, P.D.: A mixture of generalized hyperbolic distributions. *Can. J. Stat.* **43**, 176–198 (2015)
21. Byers, S.D., Raftery, A.E.: Nearest neighbor clutter removal for estimating features in spatial point processes. *J. Am. Stat. Assoc.* **93**, 577–584 (1998)
22. Coretto, P.: Robust mixture modelling. In: Proceedings of the Joint Meeting of the SFC and CLADAG. Edizioni Scientifiche Italiane, Napoli, pp. 69–72 (2008)
23. Coretto, P., Hennig, C.: A simulation study to compare robust clustering methods based on mixtures. *Adv. Data Anal. Class.* **4**, 111–135 (2010)
24. Coretto, P., Hennig, C.: Maximum likelihood estimation of heterogeneous mixtures of Gaussian and uniform distributions. *J. Stat. Plan. Infer.* **141**, 462–473 (2011)
25. Dang, U.J., Browne, R.P., McNicholas, P.D.: Mixtures of multivariate power exponential distributions. *Biometrics* **71**, 1081–1089 (2015)
26. Dang, U.J., Gallagher, M.P.B., Browne R.P., McNicholas P.D.: Model-based clustering and classification using mixtures of multivariate skewed power exponential distributions. ArXiv e-prints (2019). <https://arxiv.org/abs/1907.01938>

27. Dang, U.J., Gallaugher, M.P.B., Browne, R.P., McNicholas, P.D.: mixSPE: Mixtures of Power Exponential and Skew Power Exponential Distributions for Use in Model-Based Clustering and Classification. R package version 0.1.1 (2019). <https://CRAN.R-project.org/package=mixSPE>
28. Dasgupta, A., Raftery, A.E.: Detecting features in spatial point processes with clutter via model-based clustering. *J. Am. Stat. Assoc.* **93**, 294–302 (1998)
29. Farcomeni, A., Punzo, A.: Robust model-based clustering with mild and gross outliers. *Test* (2020). <https://doi.org/10.1007/s11749-019-00693-z>
30. Flury, B., Riedwyl, H.: Multivariate Statistics: A Practical Approach. Chapman & Hall, London (1988)
31. Forbes, F., Wraith, D.: A new family of multivariate heavy-tailed distributions with variable marginal amounts of tailweights: Application to robust clustering. *Stat. Comput.* **24**, 971–984 (2014)
32. Fraley, C., Raftery, A.E.: How many clusters? Which clustering method? Answers via model-based cluster analysis. *Comput. J.* **41**, 578–588 (1998)
33. Fraley, C., Raftery, A.E.: Model-based clustering, discriminant analysis and density estimation. *J. Am. Stat. Assoc.* **97**, 611–631 (2002)
34. Franczak, B.C., Browne, R.P., McNicholas, P.D.: Mixtures of shifted asymmetric Laplace distributions. *IEEE T. Pattern Anal.* **36**, 1149–1157 (2014)
35. Franczak, B.C., Browne, R.P., McNicholas P.D., Burak, K.L.: MixSAL: Mixtures of Multivariate Shifted Asymmetric Laplace (SAL) Distributions. R package version 1.0 (2018). <https://CRAN.R-project.org/package=MixSAL>
36. Fritz, H., Garcia-Escudero, L.A., Mayo-Iscar, A.: tclust: An R package for a trimming approach to cluster analysis. *J. Stat. Softw.* **47**, 1–26 (2012)
37. Gallegos, M.T., Ritter, G.: A robust method for cluster analysis. *Ann. Stat.* **33**, 347–380 (2005)
38. Garcia-Escudero, L.A., Gordaliza, A., Matran, C., Mayo-Iscar, A.: A general trimming approach to robust cluster analysis. *Ann. Stat.* **36**, 1324–1345 (2008)
39. Ghahramani, Z., Hinton, G.E.: The EM algorithm for factor analyzers. Technical report CRG-TR-96-1, University of Toronto (1997)
40. Giordani, P., Ferraro, M.B., Martella, F.: datasetsICR: Datasets from the Book “An Introduction to Clustering with R”, R package version 1.0 (2020). <https://CRAN.R-project.org/package=datasetsICR>
41. Guerrero-Colon, J.A., Simoncelli, E.P., Portilla, J.: Image denoising using mixtures of Gaussian scale mixtures. In: Proceedings of the 15th IEEE International Conference on Image Processing. IEEE Press, New York (2008)
42. Hennig, C.: Breakdown points for maximum likelihood estimators of location-scale mixtures. *Ann. Stat.* **32**, 1313–1340 (2004)
43. Hennig, C.: Methods for merging Gaussian mixture components. *Adv. Data Anal. Class.* **4**, 3–34 (2010)
44. Hennig, C.: fpc: Flexible Procedures for Clustering. R package version 2.2-5 (2020). <https://CRAN.R-project.org/package=fpc>
45. Hennig, C., Coretto, P.: The noise component in model-based cluster analysis. In: Preisach, C., Burkhardt, H., Schmidt-Thieme, L., Decker, R. (eds.) Data Analysis, Machine Learning and Applications, pp. 127–138. Springer, Berlin (2008)
46. Hennig, C., Hausdorf, B.: prabclus: Functions for Clustering of Presence-Absence, Abundance and Multilocus Genetic Data. R package version 2.3-2 (2020). <https://CRAN.R-project.org/package=prabclus>
47. Hubert, L., Arabie, P.A.: Comparing partitions. *J. Classif.* **2**, 193–218 (1985)
48. Hubert, L., Arabie, P.A.: Iterative projection strategies for the least-squares fitting of graph theoretic structures to proximity data. Research Report RR-94-02, Department of Data Theory, University of Leiden, Leiden (1994)
49. Jolliffe, I.T., Jones, B., Morgan, B.J.: Cluster analysis of the elderly at home: a case study. In: Diday, E., et al. (eds.) Data Analysis and Informatics. North Holland, Amsterdam (1980)

50. Karlis, D., Santourian, A.: Model-based clustering with non-elliptically contoured distributions. *Stat. Comput.* **19**, 73–83 (2009)
51. Langrognet, F., Lebret R., Poli, C., Iovleff, S., Auder, B., Iovleff, S.: Rmixmod: Classification with Mixture Modelling. R package version 2.1.2.2 (2019). <https://CRAN.R-project.org/package=Rmixmod>
52. Lee, S.X., McLachlan, G.J.: EMMIXuskew: an R package for fitting mixtures of multivariate skew  $t$  distributions via the EM algorithm. *J. Stat. Softw.* **55**, 1–22 (2013)
53. Lee, S.X., McLachlan, G.J.: Finite mixtures of multivariate skew  $t$ -distributions: some recent and new results. *Stat. Comput.* **24**, 181–202 (2014)
54. Li, Y., Wessels, L., de Ridder, D., Reinders, M.: Classification in the presence of class noise using a probabilistic kernel Fisher method. *Pattern Recognit.* **40**, 3349–3357 (2007)
55. Lin, T.I.: Maximum likelihood estimation for multivariate skew normal mixture models. *J. Multivar. Anal.* **100**, 257–265 (2009)
56. Lin, T.I.: Robust mixture modeling using multivariate skew  $t$  distributions. *Stat. Comput.* **20**, 343–356 (2010)
57. Lin, T.I., McNicholas, P.D., Ho, H.J.: Capturing patterns via parsimonious  $t$  mixture models. *Stat. Prob. Lett.* **88**, 80–87 (2014)
58. Lopes, H.F., West, M.: Bayesian model assessment in factor analysis. *Stat. Sinica* **14**, 41–67 (2004)
59. Ma, Y., Genton, M.G.: Flexible class of skew-symmetric distributions. *Scand. J. Stat.* **31**, 459–468 (2004)
60. Martella, F.: Classification of microarray data with factor mixture models. *Bioinformatics* **22**, 202–208 (2006)
61. McLachlan, G.J., Bean, R.W., Ben-Tovim Jones, L.: Extension of the mixture of factor analyzers model to incorporate the multivariate  $t$ -distribution. *Comput. Stat. Data An.* **51**, 5327–5338 (2007)
62. McLachlan, G.J., Peel, D.: Robust cluster analysis via mixtures of multivariate  $t$ -distributions. In: *Advances in pattern recognition, Lecture Notes in Computer Science*, vol. 1451, pp. 658–666. Springer, Berlin (1998)
63. McLachlan, G.J., Peel, D.: *Finite Mixture Models*. Wiley, New York (2000)
64. McLachlan, G.J., Peel, D., Bean, R.: Modelling high-dimensional data by mixtures of factor analyzers. *Comput. Stat. Data Anal.* **41**, 379–388 (2003)
65. McNicholas, P.D.: Model-based clustering. *J. Classif.* **33**, 331–373 (2016)
66. McNicholas, P.D.: *Mixture Model-Based Classification*. Chapman & Hall/CRC Press, Boca Raton (2016)
67. McNicholas, P.D., ElSherbiny, A., Jampani, K.R., McDaid, A.F., Murphy, T.B., Banks, L.: pgmm: Parsimonious Gaussian Mixture Models. R package version 1.2.4 (2019). <https://CRAN.R-project.org/package=pgmm>
68. McNicholas, S.M., McNicholas, P.D., Browne, R.P.: A mixture of variance-gamma factor analyzers. In: Ahmed, S.E. (ed.) *Big and Complex Data Analysis: Methodologies and Applications*, pp. 369–385. Springer International Publishing, Cham (2017)
69. McNicholas, P.D., Murphy, T.B.: Parsimonious Gaussian mixture models. Technical Report 05/11, Department of Statistics, Trinity College Dublin, Dublin (2005)
70. McNicholas, P.D., Murphy, T.B.: Parsimonious Gaussian mixture models. *Stat. Comput.* **18**, 285–296 (2008)
71. McNicholas, P.D., Murphy, T.B.: Model-based clustering of microarray expression data via latent Gaussian mixture models. *Bioinformatics* **26**, 2705–2712 (2010)
72. McNicholas, P.D., Murphy, T.B., McDaid, A.F., Frost, D.: Serial and parallel implementations of model-based clustering via parsimonious Gaussian mixture models. *Comput. Stat. Data Anal.* **54**, 711–723 (2010)
73. Meng, X.L., van Dyk, D.A.: The EM algorithm? An old folk-song sung to a fast new tune. *J. R. Stat. Soc. B* **59**, 511–567 (1997)
74. Mkhadri, A., Celeux, G., Nasrollah, A.: Regularization in discriminant analysis: a survey. *Comput. Stat. Data Anal.* **23**, 403–423 (1997)

75. Montanari, A., Viroli, C.: Heteroscedastic factor mixture analysis. *Stat. Model.* **10**, 441–460 (2010)
76. Morris, K., Punzo, A., McNicholas, P.D., Browne, R.P.: Asymmetric clusters and outliers: mixtures of multivariate contaminated shifted asymmetric Laplace distributions. *Comput. Stat. Data Anal.* **132**, 145–166 (2019)
77. Murray, P.M., McNicholas, P.D., Browne, R.B.: A mixture of common skew- $t$  factor analyzers. *Stat.* **3**, 68–82 (2014)
78. O'Hagan, A., Murphy, T.B., Gormley, I.C., McNicholas, P.D., Karlis, D.: Clustering with the multivariate normal inverse Gaussian distribution. *Comput. Stat. Data Anal.* **93**, 18–30 (2016)
79. Peel, D., McLachlan, G.J.: Robust mixture modelling using the  $t$ -distribution. *Stat. Comput.* **10**, 339–348 (2000)
80. Pyne, S., Hua, X., Wang, K., Rossina, E., Lin, T.I., Maiera, L.M., Baecher-Alland, C., McLachlan, G.J., Tamayo, P., Hafler, D.A., De Jager, P.L., Mesirova, J.P.: Automated high-dimensional flow cytometric data analysis. *Proc. Natl. Acad. Sci. USA* **106**, 8519–8524 (2009)
81. Prates, M.O., Cabral, C.R.B., Lachos, V.H.: mixsmsn: Fitting finite mixture of scale mixture of skew-normal distributions. *J. Stat. Softw.* **54**, 1–20 (2013)
82. Punzo, A., Blöstein, M., McNicholas, P.D.: High-dimensional unsupervised classification via parsimonious contaminated mixtures. *Pattern Recognit.* **98**, 107031 (2020)
83. Punzo, A., Mazza, A., McNicholas, P.D.: ContaminatedMixt: Clustering and Classification with the Contaminated Normal. R package version 1.3.4 (2019). <https://CRAN.R-project.org/package=ContaminatedMixt>
84. Punzo, A., McNicholas, P.D.: Parsimonious mixtures of multivariate contaminated normal distributions. *Biom. J.* **58**, 1506–1537 (2016)
85. Punzo, A., Tortora, C.: Multiple scaled contaminated normal distribution and its application in clustering. *Stat. Model.* (2020). <https://doi.org/10.1177/1471082X19890935>
86. R Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna (2020). <https://www.R-project.org>
87. Raponi, V., Martella, F., Maruotti, A.: A biclustering approach to university performances: an Italian case study. *J. Appl. Stat.* **43**, 31–45 (2015)
88. Scrucca, L., Fop, M., Murphy, T.B., Raftery, A.E.: mclust 5: Clustering, classification and density estimation using Gaussian finite mixture models. *R J.* **8**(1), 205–233 (2016)
89. Steane, M.A., McNicholas, P.D., Yada, R.Y.: Model-based classification via mixtures of multivariate  $t$ -factor analyzers. *Commun. Stat.-Simul. C.* **41**, 510–523 (2012)
90. Subedi, S., McNicholas, P.D.: Variational Bayes approximations for clustering via mixtures of normal inverse Gaussian distributions. *Adv. Data. Anal. Classif.* **8**, 167–193 (2014)
91. Sun, J., Kabán, A., Garibaldi, J.M.: Robust mixture clustering using Pearson type VII distribution. *Pattern Recognit. Lett.* **31**, 2447–2454 (2010)
92. Tipping, M.E., Bishop, C.M.: Mixtures of probabilistic principal component analysers. *Neural Comput.* **11**, 443–482 (1999)
93. Tortora, C., ElSherbiny, A., Browne, R.P., Franczak, B.C., McNicholas, P.D., Amos, D.D.: MixGHD: Model-Based Clustering, Classification and Discriminant Analysis Using the Mixture of Generalized Hyperbolic Distributions. R package version 2.3.3 (2019). <https://CRAN.R-project.org/package=MixGHD>
94. Tortora, C., Franczak, B.C., Browne, R.P., McNicholas, P.D.: A mixture of coalesced generalized hyperbolic distributions. *J. Classif.* **36**, 26–57 (2019)
95. Vrbik, I., McNicholas, P.D.: Analytic calculations for the EM algorithm for multivariate skew- $t$  mixture models. *Stat. Prob. Lett.* **82**, 1169–1174 (2012)
96. Vrbik, I., McNicholas, P.D.: Parsimonious skew mixture models for model-based clustering and classification. *Comput. Stat. Data Anal.* **71**, 196–210 (2014)
97. Wang, K., Ng, A., McLachlan G.: EMMIXskew: The EM Algorithm and Skew Mixture Distribution. R package version 1.0.3 (2013). <https://CRAN.R-project.org/package=EMMIXskew>
98. Wang, N., Raftery, A.E.: Nearest neighbor variance estimation (NNVE): robust covariance estimation via nearest neighbor cleaning (with discussion). *J. Am. Stat. Assoc.* **97**, 994–1019 (2002)

99. Wang, N., Raftery, A.E., Fraley, C.: covRobust: Robust Covariance Estimation via Nearest Neighbor Cleaning. R package version 1.1-3 (2017). <https://CRAN.R-project.org/package=covRobust>
100. Yoshida, R., Higuchi, T., Imoto, S.: A mixed factor model for dimension reduction and extraction of a group structure in gene expression data. In: Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference, vol. 8, pp. 161–172 (2004)
101. Yoshida, R., Higuchi, T., Imoto, S., Miyano, S.: Array cluster: an analytic tool for clustering, data visualization and model finder on gene expression profiles. *Bioinformatics* **22**, 1538–1539 (2006)
102. Zhang, J., Liang, F.: Robust clustering using exponential power mixtures. *Biometrics* **66**, 1078–1086 (2010)