Universität für Bodenkultur Wien

Department of Landscape, Spatial and Infrastructure Sciences

Institute of Statistics

Head of Institute: Univ.-Prof. Dipl.-Ing. Dr.techn. Friedrich Leisch

Advisor: Univ.-Prof. Dipl.-Ing. Dr.techn. Friedrich Leisch

PARTITIONING CLUSTERING ALGORITHMS IN FOOD SECURITY MAPPING

Dissertation
for obtaining a doctorate degree
at the University of Natural Resources and Life Sciences Vienna
Institute of Statistics

Submitted by
Weksi Budiaji

Vienna, June 2019

Acknowledgments

During my work on this thesis is was supported by the Ministry of Research, Technology, and Higher Education of Indonesia, Kemenristekdikti, and Österreichischer Austauschdienst, OeAD, under the Indonesia Austria Scholarship Program (IASP).

I would like to thank my supervisor Professor Friedrich Leisch for his many suggestions and discussions. During my work, he inspired me how to tackle problems and generate ideas. Many thanks to Theresa who helped my family a lot when they were in Vienna and assisted me in my every single problem. Many thanks to Tobias, Bernard H, Simona, Atif, Christina, and Dominic as my PhD companions who always set delighted lunch talks and story sharing. Many thanks to Institute of Statistics team members who gave me warm interactions like a family during my project.

I would like to thank my Indonesian friends and my family for their encouragement, support, and wishes. Thanks to my mother, my father, and my sister for their blessing. Special thanks to my beloved wife, Titin Meiti Ade Chendrakasih, for her help and understanding, Bagas Zulfikareksa Budiaji, Elia Fikaraisa Budiaji for their time. Audri Fikaraisa Budiaji, you are a very special gift of Allah SWT from Vienna.

Vienna, June 2019 *Weksi Budiaji*

Preface

This thesis describes the result of PhD research initiated in November 2015 and accomplished in June 2019. The research was done mainly in the Institute of Statistics, University of Natural Resources and Life Sciences Vienna.

This research was stemmed from my passion to explore the food security domain, especially in its methods, measurements, and visualizations. As severe food security mainly happens in developing countries, a continuous monitoring to improve the food security in these countries becomes indispensable. Indonesia as a developing country has developed a food security and vulnerability atlas (FSVA) to monitor the country's food security state. It grouped its regions via a partitioning algorithm based on a numerical data set. If the available data set is a mixed variables data set, how will we create the FSVA? It is my passion to develop tools to solve the barrier of this mixed variables data set such that the result is applicable mutatis mutandis.

The thesis traces existing mixed variables distances and medoid-based partitioning algorithms. A generalized distance function and generalized spatial distance function has been introduced to extend the mixed variable distance choices and impose constraints in the variables, respectively. A medoid-based partitioning algorithm, in addition, has also been developed, namely a simple k-medoids which is *vis-a- vis* with the existing medoid-based partitioning algorithms.

The mixed variables distances and medoid-based algorithms are directly implemented in the R environment via the kmed package. The internal and relative criteria validation and visualization of the partitioning results were also distributed in the package. Thanks to my supervisor, Professor Friedrich Leisch, who is also one of the R core team members, who encouraged me to put the R functions for this research in an R package and deposit it in the cran R. In addition, Josep Richert, an Rstudio ambassador and chief editor of the R Views blog, selected this package as one of the "Top 40 New Package Picks" in February 2018 from the 171 new packages.

Abstract

This thesis discusses medoid-based partitioning algorithm for mixed variables data sets. The partitioning around medoids (PAM) in the Gower distance is a common approach for mixed variables data partitioning whereas varying the distances is required to investigate the data structure in a partitioning algorithm. Thus, a generalized distance function (GDF) was developed in order to extend the distance choices in the mixed variables data. It consists of distance combinations with specified weights.

The GDF is then supplied in a medoid-based partitioning algorithm in order to produce the partitioning results. A medoid-based partitioning algorithm called a fast and simple k-medoids (SFKM), which is claimed to be more efficient than the PAM, has been developed. However, it suffers from empty cluster and local optima problems such that a simple k-medoids (SKM) is developed. The simulations showed that the PAM and SKM were better than the SFKM, while the PAM and SKM were on a par. The SKM, moreover, was more efficient than the PAM when the number of objects (n) is larger than 1000 and the number of clusters (k) is smaller than 10.

In the empirical data set of food security mapping of Banten Province, Indonesia, the GDF and SKM algorithm were applied to group the districts in Banten Province. A generalized spatial distance function (GSDF) was introduced in order to manage the spatial and administrative constraints imposed in the food security mapping due to agricultural extension agents task coverage. The results of both GDF and GSDF were compared applying internal and relative criteria. A bootstrap sampling technique was applied in the latter criteria, producing a measure of stability proportion. The opted result was the four-cluster from the GSDF, namely *available*, *utilize-accessible*, *accessible*, and *available-stable future* clusters.

The R implementation was in the kmed package, which covers the GDFs and medoid-based algorithms. The usage of the package includes cluster validation and visualization via internal and relative criteria, while the partitioning results could be plotted in a modified barplot for interpretation purpose. Moreover, examples of writing the user's own functions were given.

Zusammenfassung

In dieser Arbeit wird ein Medoid-basierter partitionierender Clusteralgorithmus für Datensätze mit mindestens zwei unterschiedlichen Typen von Variablen numerisch, kategoriell oder binär - diskutiert. Ein üblicher Ansatz im Partitionieren um Medoide (PAM) ist die Verwendung der Gower Distanz für gemischte Variablen. Für partitionierende Algorithmen ist es notwendig, unterschiedliche Distanzfunktionen zu verwenden, um die Datenstruktur zu untersuchen. Daher wurde eine generalisierte Distanzfunktion (GDF) entwickelt, um die Auswahlmöglichkeiten im Fall von gemischten Variablen zu erweitern. Die GDF besteht aus Kombinationen von Distanzmaßen mit vordefinierten Gewichten.

Die GDF wird dann im einem Medoid-basierten partitionierenden Clusteralgorithmus verwendet, um eine Partition der Daten zu erhalten. Es wurde ein neuer Algorithmus entwickelt, der SFKM (simple and fast k-mediods), der effizienter als PAM sein soll. Allerdings hat dieser Algorithmus Probleme mit leeren Clustern und lokalen Optima. Daher wurde ein simpler k-Medoid Algorithmus (SKM) entwickelt. Simulationen haben gezeigt, dass PAM und SKM bessere Ergebnisse liefern als SFKM, wobei PAM und SKM gleichauf lagen. SKM war auerdem effizienter als PAM, wenn die Anzahl der Objekte (n) größer als 1000 ist und die Anzahl der Cluster (k) kleiner als 10.

In einem weiteren Kapitel dieser Arbeit wurden GDF und SKM auf einen Datensatz zum Mapping von Lebensmittelsicherheit in der Banten Provinz in Indonesien angewendet, mit dem Ziel, die einzelnen Bezirke der Banten Provinz zu gruppieren. Hier wurde eine generalisierte räumliche Distanzfunktion (GSDF) entwickelt, um die räumlichen und administrativen Beschränkungen zu berücksichtigen, die beim Mapping von Lebensmittelsicherheit entstehen durch die Abdeckung von Aufgaben von einzelnen landwirtschaftlichen Entwicklungsbeamten (agricultural extension agents). Die Ergebnisse von GDF und GSDF wurden mit internen und relativen Kriterien verglichen. In einem weiteren Kriterium wurde eine Bootstrap Sampling Methode verwendet, die ein Maß für den stabilen Anteil liefert. Das gewählte Resultat war eine 4-Cluster Lösung von GSDF, ein Cluster hatte die Eigenschaft *available*, der zweite *utilize-accessible*, der dritte *accessible*, und der vierte *available-stable future*.

Im R Paket kmed wurden die GDFs und die Medoid-basierten Algorithmen implementiert. Die Anwendung des Pakets enthlt auch Cluster Validierung und Visualisierung mittels interner und relativer Kriterien. Des Weiteren können partitionierende

Clusterlösungen mittels modifizierter Barplots visualisiert und interpretiert werden. Zusätzlich werden Beispiele für eigene Anwenderfunktionen bereitgestellt.

Table of contents

1 Introduction **1**

1.1 Food security data . . . 1

1.2 Cluster analysis . . . 2

1.3 Overview of the thesis . . . 4

2 Partitioning Algorithms for Mixed Variable Data **5**

2.1 Distances . . . 5

2.1.1 Gower . . . 5

2.1.2 Wishart . . . 6

2.1.3 Podani . . . 6

2.1.4 Huang . . . 6

2.1.5 Harikumar-PV . . . 6

2.2 Generalized distance function (GDF) . . . 7

2.3 Generalized spatial distance function (GSDF) . . . 8

2.4 Medoids-based algorithms . . . 9

2.4.1 Partitioning around medoids (PAM) . . . 9

2.4.2 K-medoids (KM) . . . 10

2.4.3 Simple and fast k-medoids (SFKM) . . . 11

2.4.4 Rank k-medoids (RKM) . . . 12

2.4.5 Increasing number of cluster in k-medoids (INCKM) . . . 13

2.4.6 Simple k-medoids (SKM) . . . 15

3 Cluster Validation and Visualization **17**

3.1 External criteria . . . 17

3.1.1 Cluster accuracy . . . 17

3.1.2 Cluster purity . . . 18

3.1.3 Rand index . . . 18

3.2 Internal criteria . . . 18

3.2.1 Silhouette width . . . 18

3.2.2 Shadow value . . . 19

3.3 Relative criteria . . . 20

3.3.1 Consensus matrix . . . 20

3.3.2 Reduced size of consensus matrix . . . 21

3.4 Visualization . . . 22
3.4.1 Internal-based criteria . . . 22
3.4.2 Relative-based criteria . . . 25
3.4.3 Combination of external-based and relative-based criteria . . . 28
3.4.4 Location and dispersion . . . 30

4 Implementation in R **32**
4.1 Distance calculation in R . . . 33
4.1.1 Numerical distances . . . 33
4.1.2 Binary and categorical distance . . . 34
4.1.3 Mixed distances . . . 34
4.2 Medoids-based algorithms in R . . . 35
4.2.1 SFKM in R . . . 35
4.2.2 RKM in R . . . 36
4.2.3 INCKM in R . . . 36
4.3 Cluster validation and visualization in R . . . 37
4.3.1 Internal criteria in R . . . 37
4.3.2 Relative criteria in R . . . 38
4.3.3 Visualization of the clustering result in R . . . 41
4.4 Writing your own functions . . . 43
4.4.1 GDF generating . . . 43
4.4.2 SFKM initial medoids . . . 43
4.4.3 Cluster bootstrap . . . 44
4.4.4 Heatmap reordering . . . 45

5 Demonstration **46**
5.1 Artificial numerical data set . . . 46
5.1.1 PAM . . . 46
5.1.2 KM . . . 48
5.1.3 SFKM . . . 48
5.1.4 RKM . . . 49
5.1.5 INCKM . . . 49
5.1.6 SKM . . . 50
5.2 Artificial mixed variable data set . . . 50
5.2.1 Different variable proportion . . . 51
5.2.2 Different number of clusters . . . 51
5.2.3 Different number of variables . . . 52
5.2.4 Different number of objects . . . 53
5.2.5 Algorithms bench marking . . . 53
5.3 Heart disease data set . . . 56

5.4 Global food security data 56
5.5 Sponge data set 60

6 Application on Food Security Mapping **63**
6.1 Summary statistics 64
6.2 Cluster analysis 65
6.2.1 GSDF and SKM 65
6.2.2 GDF and SKM 68
6.3 Visualization 70
6.3.1 Barplot 70
6.3.2 Stripes plot and directed graph 73
6.3.3 Food security mapping 74

7 Conclusions **76**

References **78**

Index of Tables **84**

Table of Figures **85**

Appendix A: R documentation **89**

Appendix B: Vignette of kmed package **118**

Table of abbreviations **146**

Curriculum Vitae **147**

1 Introduction

This thesis is concerned with distance-based partitioning algorithms for food security mapping. The food security data usually consist of mixed variables, such that the focus of the distance-based partitioning algorithms is medoid-based algorithms, which are the most suitable algorithms for mixed variables data set. A list of mixed variable distances as an input of the algorithm is also provided with an additional extension from the common approach, namely generalized distance function (GDF).

To complement the algorithms, the cluster validation and visualization are presented. While the cluster validations include internal, external, and relative criteria, the cluster visualizations are internal-based, relative-based, location and dispersion visualizations. The focus of the validation and visualization is a relative-based criterion where it applies a bootstrap sampling technique.

1.1 Food security data

Definitions of food security have shifted over time. They are expanding and continuously evaluated (Jarosz, 2011). Food security has been enhanced from the secure, sufficient, and suitable supply of food for all persons to become a broader concept involving availability, accessibility, utilization, and stability concepts. Coates (2013) and van Dijk and Meijerink (2014), moreover, have added the food acceptability across many different cultures. Despite this addition, the well-known and commonly endorsed definition of food security dimensions are the availability, accessibility, utilization, and stability dimensions (FAO, 2006; Barrett, 2010).

In the last two decade, the dimensions of food security have been foremost and have become standard indicators to measure a food security state. Applying these dimensions to evaluate the achievement of the Millennium Development Goal (MDG) 1c hunger target, FAO et al. (2015) have reported that a total of 72 developing countries have accomplished a food secure state.

Indonesia, as one of the developing countries, has indicated a positive performance with regard to the food security progress. This improvement can be due to its continuous inspections via a food security and vulnerability atlas (FSVA), utilizing national food security data. FSC and WFP (2015) have mapped FSVA for 2005, 2009, and 2015 to illustrate and monitor the food security state of all regions in Indonesia. The latest FSVA has shown that 58 regions are categorized as being the most vulnerable.

The FSVA has presented a comprehensive mapping of the food security state in Indonesia, however, it excludes all cities. With the region-based measurement in the FSVA, moreover, a lower administrative level, i.e. a district-based, is utterly required because the agriculture extension (AE) agents who are responsible for the food security monitoring, have assignments in a district-based coverage.

The FSVA merely includes numerical variables as the indicator variables as well, while binary or categorical indicator variables are often possible. Accessibility to potable water with yes or no answers, for instance, can be available and is an important variable to contribute the food security dimensions. Thus, mixed variables in food security data are inevitable.

FSC and WFP (2015) have mentioned that Banten Province, as one of the newest provinces, is able to change its status of food security into a more secure state. Due to the cities exclusion in the FSVA, only 50% of the area in Banten Province has been mapped and monitored for its food security state. In this thesis, a comprehensive FSVA of Banten Province is mapped by taking into account the presence of mixed variables, the inclusion of cities, and the lower administrative objects.

1.2 Cluster analysis

Measuring food security subsequently requires pre-survey and post-survey analyses, while the former involves a food security zoning (mapping) (WFP, 2009). The food security mapping task is discriminating areas (objects) based on the food security dimensions such that homogeneous areas are assigned into a group and heterogeneous ones are partitioned. This analysis is identical to a cluster analysis, and in the statistical learning context, it is addressed as a method of unsupervised classification due to the absence of prior labels (Duda et al., 2001; Hastie et al., 2009).

Cluster analysis is an important exploratory tool in data structure investigation. This method is able to discriminate objects into groups where each object within the group is similar (homogeneous) to each other and objects between groups are distinct (heterogeneous) to one another (Kaufman and Rousseeuw, 1990; Gan et al., 2007). The process of separating homogeneous group into a distinctive part is referred to a dissection (Everitt et al., 2011). Thus, we can divide the steps of cluster analysis into two consecutive parts, the similarity quantification and the segregating process (algorithm).

A homogeneous (similarity) quantification between objects can based on their distance in a distance-based clustering algorithm, the more similar objects have the closer distance and vice versa. Some common distances such as Euclidean and Manhattan are applicable for objects that have numerical variables, while the objects having binary or categorical variables, Matching and Jaccard similarity can be selected. If the objects have mixture of numerical, binary, and categorical variables, on the other hand,

the Gower (1971) similarity measures can be applied. Other distances for numerical, binary, and categorical variables can refer to Xu and Wunsch II (2005); Everitt et al. (2011); dos Santos and Zárate (2015).

A distance-based algorithm to build clusters is opted after the similarity measure is defined. A rough but commonly agreed-upon taxonomy of clustering algorithm/ technique for a distance-based method is hierarchical vs partitioning algorithms (Jain et al., 1999; Xu and Wunsch II, 2005). The hierarchical clustering algorithms are irreversible processes that repeat the cycle of either merging all objects into a cluster or separating a cluster of all objects into an individual.

On the contrary, the partitioning clustering algorithms produce diverse partitions that are evaluated based on some defined criteria, for instance by minimizing the distance between the objects and the cluster centers. This algorithm assigns the objects to the closest predefined centers and re-estimates the centers based on the current members. The process is repeated until convergence. A popular choice for a partitioning algorithm is k-means and partitioning around medoids (Hastie et al., 2009; Leisch, 2006).

A partitioning clustering algorithm is a technique to group objects based on some designed objectives. It projects objects that consist of multidimensional characteristics into a single discrete variable, namely the cluster membership (Leisch, 2008). One of the popular partitioning algorithm is k-means (Hartigan and Wong, 1979), which Wu et al. (2008) has mentioned as the one of top ten algorithms in data mining. The k-means, in addition, applies Euclidean distance.

Although the k-means is considered as a fast algorithm, the result of k-means greatly relies on its initialization. If "bad" objects are selected in the initialization step, an empty cluster can occur (Pakhira, 2009). The k-means algorithm has local optima of the objective function, i.e. minimize the distance between the objects and the cluster centers, so that numerous local optima could emerge depending on the choice of the starting values (Steinley, 2003).

The k-means algorithm is irrelevant when the variables of inclusion are non-numerical variables, i.e. binary, categorical or mixed variables, because "means" as the center of the clusters is unavailable and Euclidean distance is not applicable. Moreover, in a mixed variable data set, modified k-means algorithms have been developed to manage this problem by redefining the cluster center as means-like, i.e. a prototype/ central value. The prototypes can be the combinations of the means/ median (numerical) and mode (binary/ categorical), or proportional distribution (categorical) (Huang, 1997; Yin and Tan, 2005; Ahmad and Dey, 2007; Bushel et al., 2007; Ji et al., 2013; Liu et al., 2016).

With the modified k-means algorithms, the new cluster center definitions vary depending on the distances applied in each class of variable of the mixed variable data. However, if the cluster centers are a set of selected objects (medoids), the medoids are

always the most centrally located objects irrespective of the distance and algorithm applied. In addition, any distance for numerical, binary, categorical, or mixed variables can be easily applied in a medoid-based algorithm.

1.3 Overview of the thesis

This thesis focuses on medoid-based partitioning algorithms, validation, and visualization for mixed variables data. The algorithms are then applied to create a district-based FSVA of Banten Province.

Chapter 2 discusses the existing partitioning algorithms that is applicable for mixed variables data. It also provides a distance function to manage with mixed variables data. Applying this distance function, distance weights and combinations can be easily adjusted.

Chapter 3 presents the validation and visualization of the cluster result in order to obtain a suitable number of clusters. A cluster heatmap of stability measured is presented and combined with cluster indices to complement the cluster results. The heatmap can be converted as well into a directed graph figure.

Chapter 4 applies the partitioning algorithms, validation, and visualization in R software environment (R Core Team, 2015). The focus is applying the kmed package (Budiaji, 2019). This package is helpful to calculate mixed variables distances, apply some partitioning algorithms, and evaluate the cluster results.

Chapter 5 provides demonstration studies of the partitioning algorithms in two types of simulated mixed variable data sets. Two mixed variable data sets from the UCI website are also analyzed.

In Chapter 6, food security mapping is presented. Because districts in the map are spatially contiguous, a constrained clustering is required in the analysis. It can be easily perform via a generalized spatial distance function (GSDF) explained in Chapter 2.

Chapter 7 summarizes the main finding of the thesis. The appendices contain the R documentation (Appendix A) and vignette of the kmed package (Appendix B).

2 Partitioning Algorithms for Mixed Variable Data

This chapter introduces the distances and partitioning algorithms for mixed variables data. In a partitioning algorithm, each object is set to have one and only one cluster membership based on the similarity/ distance measure. Compared to distance measures in either numerical or categorical variables, there are limited alternatives for mixed variables distance. Although the most common measure and medoid-based algorithm for mixed variables data is Gower (1971) and partitioning around medoids (PAM) (Kaufman and Rousseeuw, 1990), respectively, some other measures and algorithms are also presented.

2.1 Distances

2.1.1 Gower

Gower (1971) similarity can calculate mixed variable data by adding the numerical, binary, and categorical variables. The Gower similarity between object i and object j is defined as

$$s_{ij} = \frac{\sum_{l=1}^{p} \omega_{ijl} s_{ijl}}{\sum_{l=1}^{p} \omega_{ijl}}, \tag{2.1}$$

$$\begin{aligned} s_{ijl} &= 1 - \frac{|x_{il} - x_{jl}|}{R_l}, \; or \\ s_{ijl} &\in \{0, 1\}, \end{aligned} \tag{2.2}$$

where ω_{ijl} is the j-th weight of object i in the variable l, x_{il} is the value of the object i on the l-th (numerical) variable, R_l is the range of the l-th (numerical) variable, and $s_{ijl} = 1$ if the objects i and j are similar in the l-th (binary or categorical) variable and 0 otherwise. The Gower distance is then obtained by subtracting 1 with Equation 2.1, i.e. $d_{ij} = 1 - s_{ij}$. Equation 2.1 is also valid with at least a non missing value in the p variables. However, when there is a missing value in the object i or j on the l-th variable, for example, ω_{ijl} becomes 0. It implies that if there is a missing value, how to calculate the distance among the objects can be different due to the difference of $\sum_{l=1}^{p} \omega_{ijl}$.

2.1.2 Wishart

Wishart (2003) has modified the Gower similarity to incorporate it in a k-means algorithm. The difference it has with the Gower similarity is in the numerical variable where the Wishart distance applies a variance weight instead of a range in the numerical variables, and a squared distance component. The Wishart distance is calculated by

$$d_{ij} = \sqrt{\sum_{l=1}^{p} \omega_{ijl} \left(\frac{x_{il} - x_{jl}}{\delta_{ijl}} \right)^2}, \tag{2.3}$$

where $\delta_{ijl} = \sigma_l$ when l is a numerical or ordinal variable, or $\delta_{ijl} = 1$ when l is a binary variable, and $\delta_{ijl} = 1$ for $x_{il} = x_{jl}$ or $\delta_{ijl} = x_{il} - x_{jl}$ for $x_{il} \neq x_{jl}$ when l is a categorical variable.

2.1.3 Podani

Podani (1999) has proposed an alternative distance for mixed variable data. Although the Podani distance has a similar form to Equation 2.3, the δ_{ijl} becomes the range (R_l) when l is a numerical or ordinal variable. Like the Gower and Wishart distances, the Podani distance can calculate a distance with missing values.

2.1.4 Huang

Another mixed variable distance that combines numerical and categorical variables is Huang (1997) distance. The Huang distance between object i and object j can be computed by

$$d_{ij} = \sum_{r=1}^{P_n} (x_{ir} - x_{jr})^2 + \gamma \sum_{s=1}^{P_c} \delta_c(x_{is} - x_{js}), \tag{2.4}$$

where p_n is the number of numerical variables, p_c is the number of categorical variables, γ is the weight for the categorical distance, δ_c is the categorical distance, and $\delta_c(x_{is}, x_{js})$ is the categorical distance between object i and object j in the variable s. It is suggested that γ is replaced by the average standard deviation of the numerical variables and the categorical distance is the mismatch coefficient.

2.1.5 Harikumar-PV

Harikumar and PV (2015) has proposed a vector based distance of numerical, binary, and categorical distances. The Harikumar-PV distance is defined as

$$d_{ij} = \sum_{r=1}^{P_n} |x_{ir} - x_{jr}| + \sum_{s=1}^{P_c} \delta_c(x_{is} - x_{js}) + \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt}), \tag{2.5}$$

where p_b is the number of binary variables, and $\delta_b(x_{it}, x_{jt})$ is the binary distance between object i and object j in the variable t. The binary distance in the Harikumar-PV distance is Hamming distance, while the categorical distance applies a co-occurrence distance, a distance that is calculated based on the distribution of other categorical variables.

2.2 Generalized distance function (GDF)

We propose a generalized distance function (GDF) to calculate mixed variable distance. The GDF approach is similar to Huang (1997), Ahmad and Dey (2007), McCane and Albert (2008), and Harikumar and PV (2015) approaches, where numerical, binary, and categorical distances are combined. The GDF is defined as

$$d_{ij} = \left(\alpha \sum_{r=1}^{p_n} \delta_n(x_{ir}, x_{jr}) + \beta \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt}) + \gamma \sum_{s=1}^{p_c} \delta_c(x_{is}, x_{js})\right)^{\omega}, \tag{2.6}$$

where ω, α, β and γ are the weights for the whole distance function, numerical, binary, and categorical variables respectively, δ_n is the numerical distance.

Applying the GDF (Equation 2.6), the distance is adjustable to any type of data and flexible with any combination of distances and weights. The aforementioned distances, moreover, can be modified into the GDF function such that they are equal. Thus, the Gower, Wishart, Podani, Huang, and Harikumar-PV distances become

$$\begin{aligned}
d_{ij} &= \frac{1}{p_n + p_b + p_c}\left(\sum_{r=1}^{p_n} \frac{1}{R_r} |x_{ir} - x_{jr}| + p_b \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt}) + p_c \sum_{s=1}^{p_c} \delta_c(x_{is}, x_{js})\right), \\
d_{ij} &= \left(\frac{1}{p_n + p_b + p_c}\left(\sum_{r=1}^{p_n} \frac{1}{s_r^2} (x_{ir} - x_{jr})^2 + p_b \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt}) + p_c \sum_{s=1}^{p_c} \delta_c(x_{is}, x_{js})\right)\right)^{\frac{1}{2}}, \\
d_{ij} &= \left(\sum_{r=1}^{p_n} \frac{1}{R_r^2} (x_{ir} - x_{jr})^2 + p_b \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt}) + p_c \sum_{s=1}^{p_c} \delta_c(x_{is}, x_{js})\right)^{\frac{1}{2}}, \\
d_{ij} &= \sum_{r=1}^{p_n} (x_{ir} - x_{jr})^2 + \frac{\sum_{r=1}^{p_n} s_r}{p_n} \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt}) + \frac{\sum_{r=1}^{p_n} s_r}{p_n} \sum_{s=1}^{p_c} \delta_c(x_{is}, x_{js}), \\
d_{ij} &= \sum_{r=1}^{p_n} |x_{ir} - x_{jr}| + \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt}) + \sum_{s=1}^{p_c} \delta_c(x_{is}, x_{js}),
\end{aligned}$$

respectively. Table 2.1 shows the reformulation of some mixed variables data into the GDF structure.

Table 2.1: Mixed variable distances in the GDF formulation

GDF	ω	α	β	γ	$\delta_n(x_{ir}, x_{jr})$	$\delta_b(x_{it}, x_{jt})$	$\delta_c(x_{is}, x_{js})$
Gower	1	$\frac{1}{p_n+p_b+p_c}$	$\frac{p_b}{p_n+p_b+p_c}$	$\frac{p_c}{p_n+p_b+p_c}$	M rw	SM	SM
Wishart	$\frac{1}{2}$	$\frac{1}{p_n+p_b+p_c}$	$\frac{p_b}{p_n+p_b+p_c}$	$\frac{p_c}{p_n+p_b+p_c}$	SE vw	SM	SM
Podani	$\frac{1}{2}$	1	p_b	p_c	SE r^2w	SM	SM
Huang	1	1	$\overline{s_n}$	$\overline{s_n}$	SE	H	H
Harikumar-PV	1	1	1	1	M	H	CoC

p_n = Number of numerical variables, p_b = Number of binary variables,
p_c = Number of categorical variables, $\overline{s_n}$ = mean of standard deviation of numerical variables, E = Euclidean, H = Hamming, M = Manhattan,
SE = Squared Euclidean, SM = Simple Matching, CoC = Co-occurrence,
rw = range weighted, r^2w = squared range weighted, vw = variance weighted

2.3 Generalized spatial distance function (GSDF)

When spatial constraints occur in the clustering task, they can be taken into account as either weights or other numerical variables included in the clustering algorithm. The former is preferable due to its distinction between variables and spatial spaces. Then, the weights of the spatial constraint can be obtained by a neighborhood system. The neighborhood system generates a connecting scheme (contiguity matrix) such as the Delanuay triangulation, Gabriel graph, relative neighborhood, minimum spanning tree, maximum distance, and semi-variogram (Legendre and Legendre, 2012; Pawitan and Huang, 2003; Simbahan and Dobermann, 2006).

While the input of the medoids-based algorithm is a distance matrix where a GDF (Section 2.2) can be applied, a Hadamard product of distance and contiguity matrices from the variables and spatial information, respectively, is applicable for a clustering algorithm (Legendre and Legendre, 2012). In the GDF form, we call the Hadamard product of the distance and contiguity matrices as a generalized spatial distance function (GSDF). The GSDF is then defined as

$$\begin{aligned}
d_{ij} &= \theta_{ij}\left(\alpha\sum_{r=1}^{p_n}\delta_n(x_{ir},x_{jr}) + \beta\sum_{t=1}^{p_b}\delta_b(x_{it},x_{jt}) + \gamma\sum_{s=1}^{p_c}\delta_c(x_{is},x_{js})\right)^{\omega} \\
&= \left(\theta_{ij}^{\frac{1}{\omega}}\left(\alpha\sum_{r=1}^{p_n}\delta_n(x_{ir},x_{jr}) + \beta\sum_{t=1}^{p_b}\delta_b(x_{it},x_{jt}) + \gamma\sum_{s=1}^{p_c}\delta_c(x_{is},x_{js})\right)\right)^{\omega} \\
&= \left(\theta_{ij}^{\frac{1}{\omega}}\,\alpha\sum_{r=1}^{p_n}\delta_n(x_{ir},x_{jr}) + \theta_{ij}^{\frac{1}{\omega}}\,\beta\sum_{t=1}^{p_b}\delta_b(x_{it},x_{jt}) + \theta_{ij}^{\frac{1}{\omega}}\,\gamma\sum_{s=1}^{p_c}\delta_c(x_{is},x_{js})\right)^{\omega} \\
&= \left(\alpha_{ij}^{*}\sum_{r=1}^{p_n}\delta_n(x_{ir},x_{jr}) + \beta_{ij}^{*}\sum_{t=1}^{p_b}\delta_b(x_{it},x_{jt}) + \gamma_{ij}^{*}\sum_{s=1}^{p_c}\delta_c(x_{is},x_{js})\right)^{\omega},
\end{aligned}$$

where α_{ij}^*, β_{ij}^*, and γ_{ij}^* are the numerical, binary, and categorical weights considering the spatial constraint, θ_{ij}. Then, the weights of mixed variable distances, i.e. θ_{ij} (Table

2.2), replace a constant of 1 in Table 2.1.

Table 2.2: Spatial weights of mixed variable distances

	ω	α^*	β^*	γ^*
Gower	1	$\frac{\theta_{ij}}{p_n+p_b+p_c}$	$\frac{\theta_{ij}\ p_b}{p_n+p_b+p_c}$	$\frac{\theta_{ij}\ p_c}{p_n+p_b+p_c}$
Wishart	$\frac{1}{2}$	$\frac{\theta_{ij}}{p_n+p_b+p_c}$	$\frac{\theta_{ij}\ p_b}{p_n+p_b+p_c}$	$\frac{\theta_{ij}\ p_c}{p_n+p_b+p_c}$
Podani	$\frac{1}{2}$	θ_{ij}	$\theta_{ij}\ p_b$	$\theta_{ij}\ p_c$
Huang	1	θ_{ij}	$\theta_{ij}\ \overline{s_n}$	$\theta_{ij}\ \overline{s_n}$
Harikumar-PV	1	θ_{ij}	θ_{ij}	θ_{ij}

2.4 Medoids-based algorithms

2.4.1 Partitioning around medoids (PAM)

A well-known choice of a medoids-based algorithm is partitioning around medoids (PAM) (Kaufman and Rousseeuw, 1990). For mixed variable data, the common practice is to calculate Gower (1971) distance and run the PAM algorithm from this distance matrix. The PAM algorithm is

1. Select a set of initial medoids, $\mathcal{M}_k$, arbitrarily.

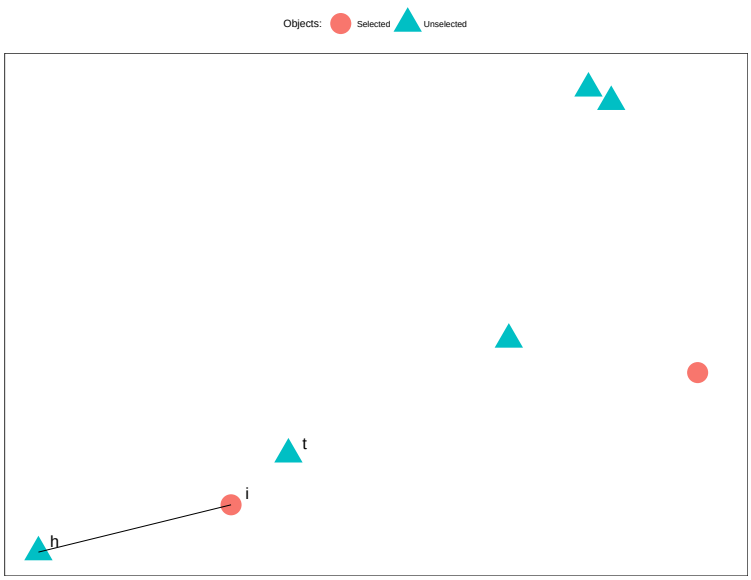


Figure 2.1: Swapping step in the PAM algorithm

2. For each medoid m and each non-medoid o, swap m and o and calculate the total swapping cost. For example, for each pair of non-medoid h and medoid i (Figure 2.1), the total swapping cost, TC_{ih}, can be computed. The swapping cost due to object t (non-medoid) can be calculated by

$$C_{tih} = d(t, h) - d(t, i),$$

where $d(t, h)$ is the distance between object t and h. Then, TC_{ih} is computed by

$$TC_{ih} = \sum_{\forall o} C_{oih}. \tag{2.7}$$

3. Select the configuration that has a lowest TC, i.e. if TC_{ih} in Equation 2.7 is negative ($TC_{ih} < 0$), i is replaced by h.

4. Repeat steps 2-3 and 4 until the selected medoids do not change.

Due to the sensitivity of means towards outliers and noise, the PAM is more robust than the k-means. However, the PAM has a high complexity resulted from examining all of the $k(n-k)$ combinations of intermediate solutions in steps 2 and 3 of each iteration (Rangel et al., 2016).

2.4.2 K-medoids (KM)

The k-medoids (KM) algorithm adapts the k-means algorithm, with the cluster centers represented by a set of medoids instead of the means. Reynolds et al. (2006) have presented the k-medoids algorithm as

1. Select a set of initial medoids, $\mathcal{M}_k$, as many as k from all object X_n at random.

2. Assign the label/ membership of each object to the closest medoid $\mathcal{M}_k$ by preserving the cluster label $l(x_n)$ fixed.

3. Update the new position of $\mathcal{M}_k$ medoids by finding the object within the cluster that minimizes the sum of distance between this object with the other objects in the cluster.

$$\begin{aligned} m_g \quad := \quad & \underset{g \in K}{argmin} \sum_{n:l(x_n)=m_g} d(x_n, m_g), \\ & g = 1, 2, 3, \ldots, k. \end{aligned} \tag{2.8}$$

   Figure 2.2 shows that a candidate of medoid is scanned from all objects within cluster, i.e. setting the cluster membership fixed.

4. Repeat steps 2 and 3 until the medoids are fixed.

5. Assign all objects to the closest medoids.

The KM algorithm is an optimization problem that equals to a binary linear programming formulation (Gordon and Vichi, 1998). Compared to the PAM, it gains efficiency in terms of speed. However, due to similarity to the k-means algorithm, it suffers from the aforementioned problems such as an empty cluster and local optima depending on the starting values.

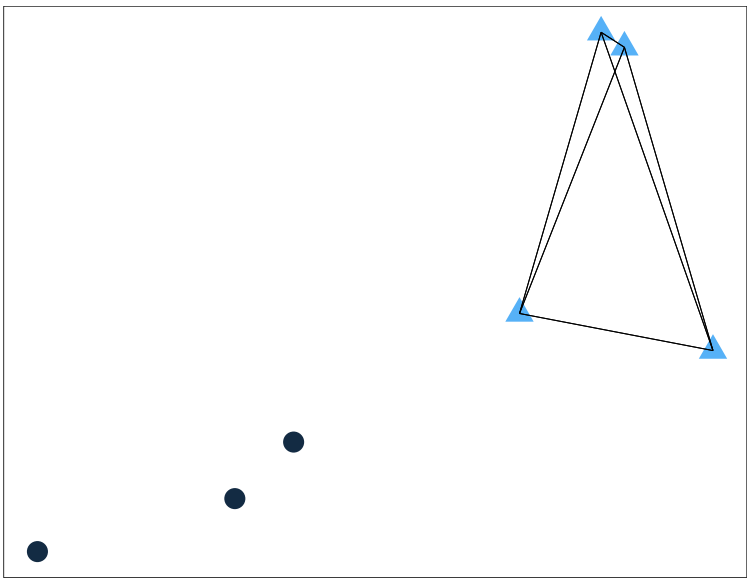


Figure 2.2: Updating a medoid in the KM algorithm

2.4.3 Simple and fast k-medoids (SFKM)

A simple and fast k-medoids (SFKM) algorithm has been proposed by Park and Jun (2009). The SFKM algorithm is similar to the KM algorithm. The differences are in the first step where the SFKM does not select the initial medoids arbitrarily, and in the stopping rule of the algorithm. The initial medoids are selected based on an ordered v_j where it is calculated by

$$v_j = \sum_{i=1}^{n} \frac{d_{ij}}{\sum_{l=1}^{n} d_{il}}, \quad j = 1, 2, 3, \ldots, n. \tag{2.9}$$

If the number of clusters is k, the first k's are selected as initial medoids. While the KM algorithm stops when the previous medoids are identical, the SFKM is quit when the sum of distance of the objects to their medoids are equal to the previous step. The distance between a medoid and objects within a cluster is defined by

$$E = \sum_{g=1}^{k} \sum_{x_n \in C_g} d(x_n, m_g), \tag{2.10}$$

where C_g is group g, which has a medoid m_g.

Figure 2.3 shows the first and second steps in the SFKM algorithm. The initial medoids of v_1 and v_2, which are sorted from Equation 2.9, are selected where they are the first and second objects that have the closest distance to the other objects. The Equation 2.9 implies that all objects are sorted from the center space to the outer. The first k initial medoids are the k most centrally located objects.

The SFKM algorithm has been applied in both the numerical and categorical variable data set. With the medoids as cluster centers, the SFKM is suitable as well for a

mixed variable data set. Although, it is argued that this algorithm is simple and fast, how to manage with the local optima and empty clusters are not discussed.

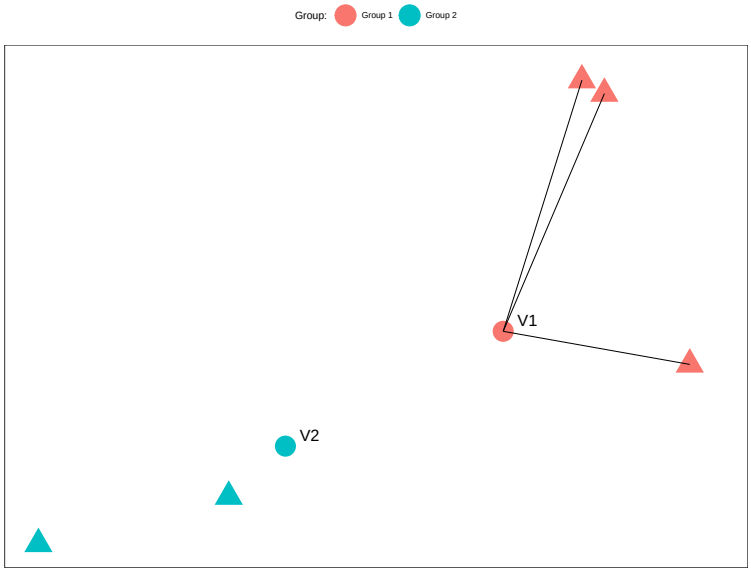


Figure 2.3: First and second steps in the SFKM algorithm

2.4.4 Rank k-medoids (RKM)

Zadegan et al. (2013) have presented a ranked k-medoids (RKM) as an algorithm that is argued as solving the local optima problem in the SFKM algorithm. The RKM algorithm is

1. Transform the n x n distance matrix into a rank matrix $\mathbf{R}$. The rank values in the matrix $\mathbf{R}$ can be calculated by either a row-based or column-based rank from the distance matrix. Thus, the matrix $\mathbf{R}$ is an asymmetric matrix.

2. Select k initial medoids randomly.

3. Assign the label/ membership of a b group of objects to the closest medoid based on the values of the matrix $\mathbf{R}$. The choice of b is arbitrary.

4. Update the medoids in the b group by finding the maximum hostility. The hostility of object i is calculated by

$$h_i = \sum_{X_j \in Y} r_{ij}, \tag{2.11}$$

   where Y is a set of objects as many as b.

5. Repeat steps 3 and 4 until the maximum iteration is reached.

6. Assign all objects to the closest medoids.

Figure 2.4 illustrates initial medoids selection with the three closest objects ($b = 4$) to the initial medoids clustered. There is no restriction in the choice of b. However, when b is large, for instance $b = n$, all objects overlap such that an empty cluster can emerge. In addition, there are two distances involved, the original and asymmetric ($\mathbf{R}$) distances. For the latter distance, a preference of two objects when they have an equal rank has to be taken into account as well.

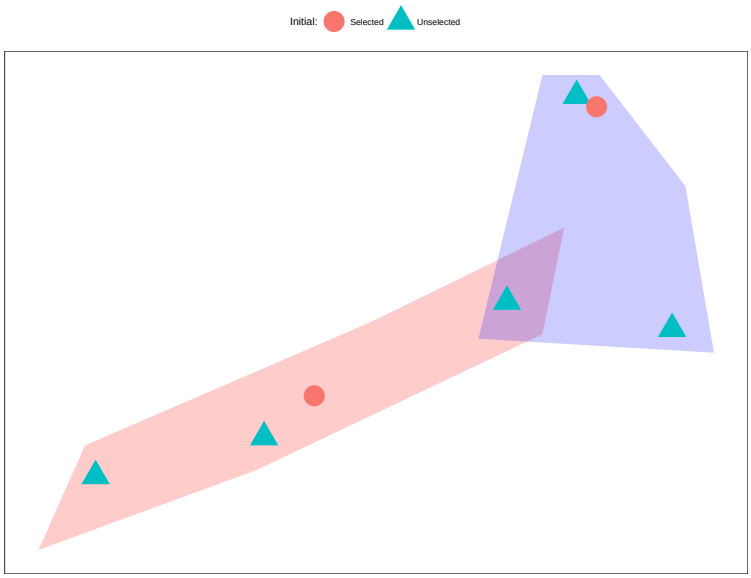


Figure 2.4: First and second steps in the RKM algorithm ($b = 4$)

2.4.5 Increasing number of cluster in k-medoids (INCKM)

An increasing number of clusters in k-medoids (INCKM) has been presented by Yu et al. (2018), who has modified the initial medoids of the KM/ SFKM. The initial medoids are selected from a set of 2^m possible medoids, where $m = \lceil log_2\, k \rceil$, i.e. the smallest integer that is greater than or equal to $log_2\, k$. The INCKM algorithm is originally intended for numerical variable with the Euclidean distance such that the object mean, variances of both data set and all objects are easily calculated. Because it is a medoid-based algorithm, we could redefine the object mean and variances into the centrally located object (medoid) and an average deviation of all objects to the medoid, respectively. With this new definition, the INCKM is applicable for mixed variable data.

The algorithm to select m_g initial medoids is

1. Select the most centrally located objects, v_1, via Equation 2.9.

2. Calculate the average deviation of the data set and the deviation of each object.

The average deviation of the data set can be computed via

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} d(O_i, v_1)}, \tag{2.12}$$

while the average deviation of each object is calculated by applying

$$\sigma_i = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n} d(O_i, O_j)}. \tag{2.13}$$

3. Find the candidate medoids O_c. An O_c object is an object that has an average deviation (Equation 2.13) smaller or equal to the average deviation of the data set (Equation 2.12). It is calculated by

$$O_c = \{X_i | \sigma_i \leq \alpha\sigma, i = 1, 2, \ldots, n\}, \tag{2.14}$$

   where α is a stretch factor defined by the user.

4. The first initial medoids m_1 is an O_c object that minimizes the distance to the all O_c, while the second initial medoids m_2 is an O_c object that maximizes the distance of m_1 to the all O_c objects.

$$\begin{aligned} m_1 &:= \underset{X_i \in S_m}{argmin} \sum_{\forall S_m} d(x_i, m_1). \\ m_2 &:= \underset{X_i \in S_m}{argmax} \sum_{\forall S_m} d(x_i, m_1). \end{aligned}$$

5. If $k = 2$, m_1 and m_2 are the initial medoids.

6. Assign all objects to the closest medoids m_1 or m_2 into two clusters.

7. Select m_3 and m_4 by maximizing the distance between m_1 and m_3 within cluster 1, and the distance between m_2 and m_4 within cluster 2, respectively. If k equals to 4, m_1, m_2, m_3, and m_4 are the initial medoids. However, if k equals to 3, the initial medoids are m_1 and m_2 and one of m_3 or m_4. When $d(m_1, m_3)$ is greater than $d(m_2, m_4)$, m_3 is selected, and otherwise.

8. Repeat steps 6 and 7 until the set of initial medoids m_p as many as k.

Figure 2.5 shows the initial medoids that are selected applying the increasing number of cluster in k-medoids algorithm. If $k = 3$, the medoids are m_1, m_2, and m_3 because $d(m_1, m_3) > d(m_2, m_4)$. The INCKM is equivalent to applying the KM algorithm twice, i.e. in the initial medoids selection and in the clustering algorithm. Thus, the possibility of local optima and empty clusters in the INCKM increases.

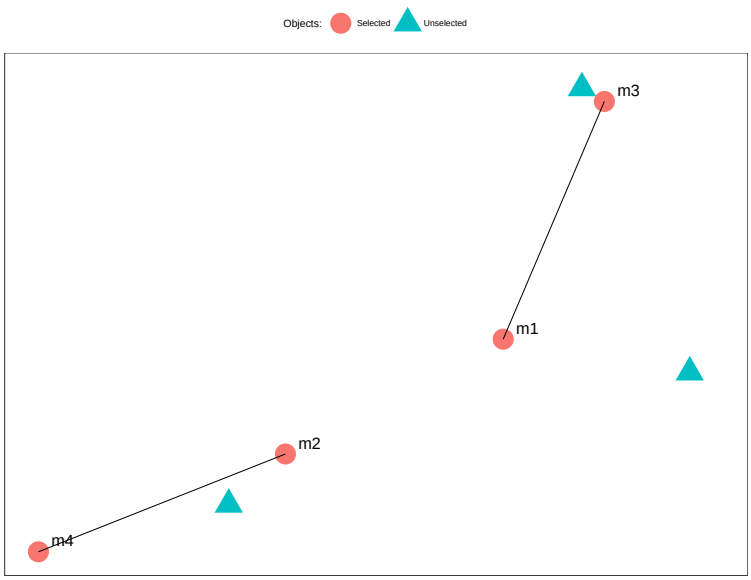


Figure 2.5: Initial medoids in the INCKM algorithm ($k = 4$)

2.4.6 Simple k-medoids (SKM)

We develop a simple k-medoids (SKM) algorithm to manage the local optima and empty cluster in both the KM and SFKM algorithms. To avoid the local optima, several initializations are applied as Steinley and Brusco (2007) recommendation. Preserving the label when non-unique medoids are initialized is a way out for an empty cluster. The initial medoids, moreover, only retains an object that is equivalent to v_1 as a fixed initial medoid in each initialization.

The SKM algorithm is

1. Select a set of initial medoids $\mathcal{M}_k$. The first initial medoids is the most centrally located object. As Equation 2.9 is a standardized version, v_1 is selected by excluding the denominator, i.e. the unstandardized version. The $k - 1$ initial medoids are randomly selected.

2. Assign all objects to the closest medoid $\mathcal{M}_k$. If non-unique exists, objects are assigned to only one of the non-unique medoids.

3. Update the new set medoids as in the KM algorithm. The non-unique medoids preserve its label except the non-unique medoids that has more than one object as the member.

4. Calculate the sum of within cluster distance (E) via Equation 2.10.

5. Repeat steps 2-4 as many as a pre-determined number of iterations or until E is equal to the previous E.

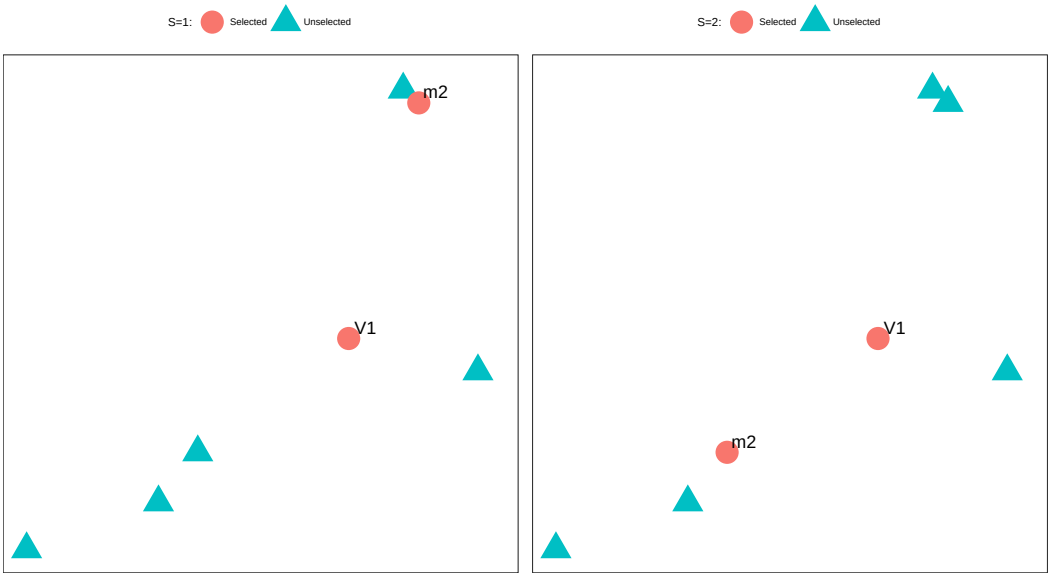


Figure 2.6: Initial medoids in the SKM algorithm ($s = 2$)

6. Repeat steps 1-5 s times to produce s times initialization (s seeding). The final medoids are a set of medoids that has a minimum value of E. Figure 2.6 illustrates two different seedings. In both seedings, the most centrally located object always becomes the one of the initial medoids.

7. Assign all objects to the closest final medoids.

3 Cluster Validation and Visualization

The partitioning process in cluster analysis involves an absence of pre-defined class, thus named an unsupervised technique. The final partition then requires validation measures to find the best partition that fits the underlying data. The cluster validation can be classified into three types, namely external, internal, and relative criteria (Theodoridis and Koutroubas, 2008; Webb and Copsey, 2011). The difference among these three criteria is the available information in the process of cluster validation such that it is hard to compare these criteria in the same framework (Arbelaitz et al., 2013).

3.1 External criteria

The external criteria validate the partition result with the known class label ("gold standard") (Handl et al., 2005). This is useful when the objective of evaluation is a clustering algorithm on a benchmark data set or a controlled environment. The external criteria validation examines if the objects are structured by random or not (Halkidi et al., 2001). A clustering accuracy (Ji et al., 2013) and cluster purity (Handl et al., 2005; Wu et al., 2009) are examples of the external criteria.

Table 3.1: Table of true class vs partitioning result

Partition result	True class C_1	C_2	...	C_k	$\sum$
P_1	n_{11}	n_{12}	...	n_{1k}	$n_{1.}$
P_2	n_{21}	n_{22}	...	n_{2k}	$n_{2.}$
...	...	...	...	...	...
P_k	n_{k1}	n_{k2}	...	n_{kk}	$n_{k.}$
$\sum$	$n_{.1}$	$n_{.2}$	...	$n_{.k}$	$n_{..}$

3.1.1 Cluster accuracy

Assuming the diagonal values of the contingency table of partitioning result vs true class (Table 3.1) are optimum, the accuracy rate of the clustering algorithm can be calculated by

$$A = \sum_{i=1}^{k} \frac{n_{ii}}{n_{..}}, \tag{3.1}$$

where n_{ii} is the diagonal values of row i column i, and $n_{..}$ is the sum of all values.

3.1.2 Cluster purity

The cluster purity is computed by

$$P = \sum_{i=1}^{k} \frac{n_{i.}}{n_{..}} (\underset{j \in k}{max} \frac{n_{ij}}{n_{i.}}), \tag{3.2}$$

where $n_{i.}$ is the sum of row i, and n_{ij} is the value of row i column j. When the diagonal values of Table 3.1 are also maximum, A is equal to P.

3.1.3 Rand index

Let B denotes the number of all pairs of data points which are either assigned to the same clusters by both partitions or to different clusters by both partitions. On the other hand, let D denotes the number of all pairs of data points which are assigned to a cluster in a partition, yet to a different cluster by another partition. The rand index (Rand, 1971) is then measured by

$$RI = \frac{B}{B + D}. \tag{3.3}$$

3.2 Internal criteria

When either the true class or gold standard is absent, which is always likely to happen in a real data set, the internal validation is relevant. Since the class label is unknown, the internal validation uses the intrinsic information of the data to measure the quality of partition. It usually measures compactness and separation of the cluster (Arbelaitz et al., 2013). The compactness examines the cluster homogeneity, i.e. within-cluster variance, while the separation assesses the degree of separation between clusters. Charrad et al. (2014) has listed 19 internal indices that have been implemented in R and SAS software. One of the popular indices is silhouette width (Rousseeuw, 1987) by which the compactness and separation are non-linearly combined (Brock et al., 2008).

3.2.1 Silhouette width

Silhouette width is the average silhouette value of each object. The silhouette value of object i can be calculated by

$$sv(i) = \frac{b_i - a_i}{max(a_i, b_i)}, \text{ where} \tag{3.4}$$

$$a_i = \frac{1}{n(C(i))} \sum_{j \in C(i)} d(i,j) \text{ and } b_i = \underset{C_k \in \mathcal{C} \setminus C(i)}{min} \sum_{j \in C_k} \frac{d(i,j)}{n(C_k)}.$$

$C(i)$ is the cluster that belongs to object i, a_i is the average distance between object i and all objects within cluster $C(i)$, and b_i is the average distance between object i and all objects within the nearest neighbor cluster. The value of silhouette is within $[-1, 1]$ where 1 indicates a well-separated cluster, conversely, -1 indicates a poorly-separated cluster.

3.2.2 Shadow value

A similar measure to silhouette, called shadow value, which is based on the first and second closest centroids, has been developed (Leisch, 2006, 2010). The shadow value can be computed via

$$sh(i) = \frac{2\, d(i, c(i))}{d(i, c(i)) + d(i, \tilde{c}(i))}, \tag{3.5}$$

where $c(i)$ and $\tilde{c}(i)$ is the first and second-closest centroid from object i, respectively. The average shadow value of objects where cluster x is the closest and cluster y is the second closest is then obtained by

$$sh_{xy} = \frac{\sum_{i \in A_{xy}} sh(i)}{A_x}, \tag{3.6}$$

where $A_{xy} = \{i \in X_N | c(i) = c_x, \tilde{c}(i) = c_y\}$. If sh_{xy} is close to 1, it indicates the clusters are poorly-separated.

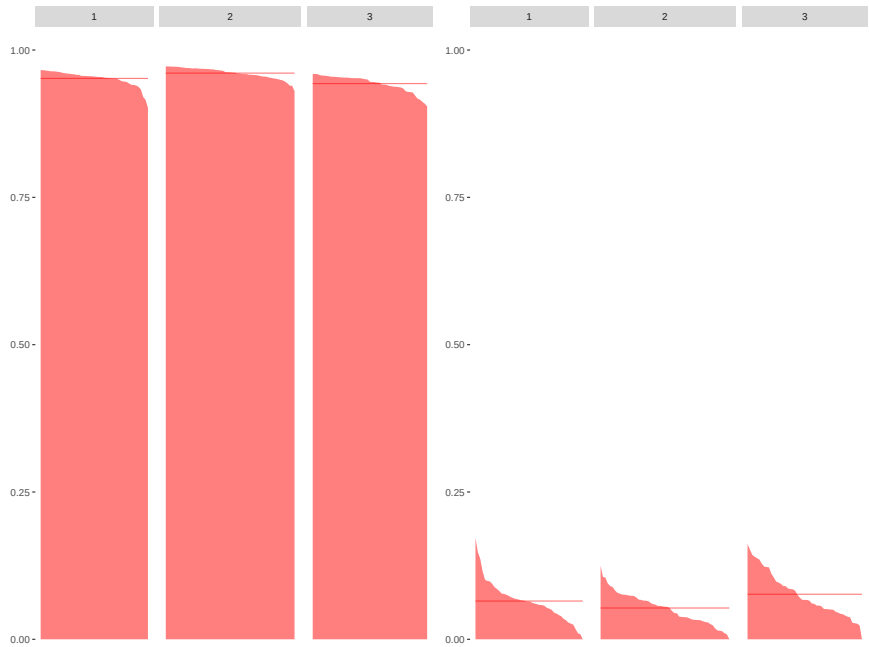


Figure 3.1: Silhouette (left) and shadow value (right) of well-separated clusters

Figure 3.1 shows well-separated clusters applying silhouette and shadow values, while Figure 3.2 shows the opposite. When the clusters are well-separated, the silhou-

ette graphs (Figure 3.1) have high values. At the same case, the shadow values have small values. Thus, the silhouette and shadow value plot are interpreted in a different way.

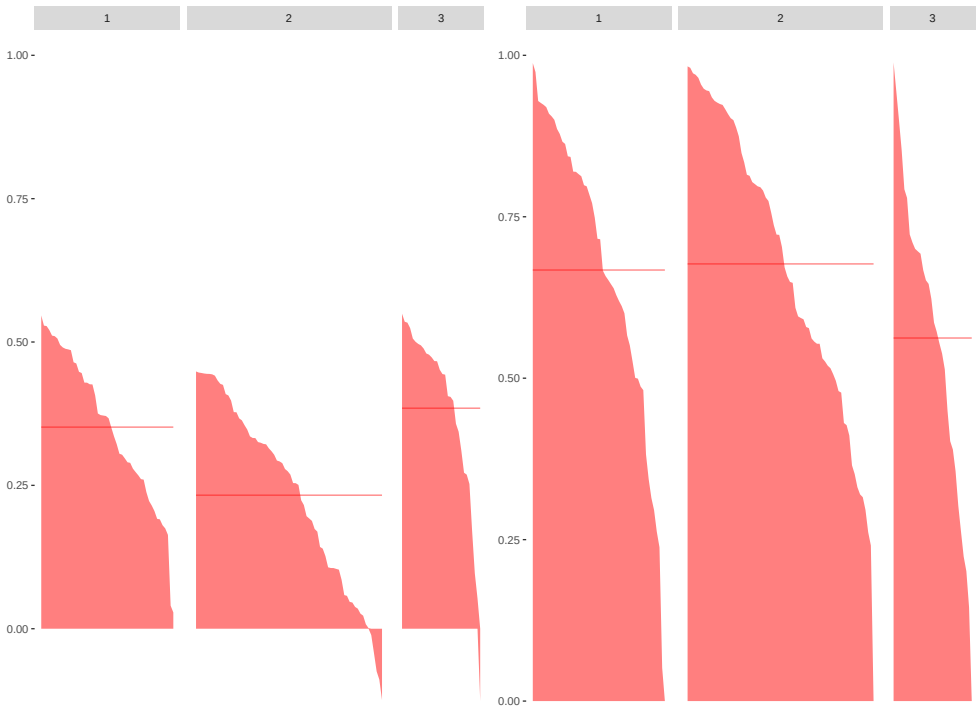


Figure 3.2: Silhouette (left) and shadow value (right) of poor-separated clusters

3.3 Relative criteria

Another form of cluster validation is a stability based-validation, which does not require compactness assumption. It relies on sets of replicated samples via a replication technique like the cross validation method. The bootstrap method, i.e. sampling with replacement, is also an alternative method to asses the stability of clusters (Jain and Moreau, 1987; Fang, 2012), where the natural structure of data exists if the cluster results are consistent across replications (Dolnicar and Leisch, 2010).

3.3.1 Consensus matrix

The consistency of cluster results of a clustering algorithm across replications can be put in a $n\,x\,n$ agreement/ consensus matrix. Because the consensus between objects i and j is equal to that of between objects j and i, the consensus matrix is a symmetric matrix. To create an ordered consensus matrix, another clustering algorithm is required (Monti et al., 2003; Benson-Putnins et al., 2011) such that a heatmap is produced.

Figure 3.3 shows heatmaps of consensus matrix of well and poorly sepaterated clusters. Because they rely on another clustering algorithm, a hierarchical clustering

for instance, to rearrange the objects in the consensus matrix, a problem in a branch of cluster trees occurs that is called a swing problem (Weinstein, 2008), which it is also described as a seriation problem (Wilkinson and Friendly, 2009).

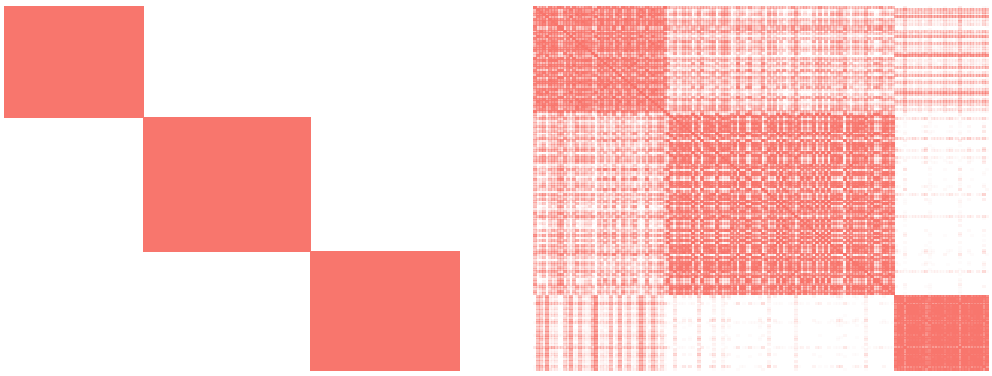


Figure 3.3: Consensus matrix heatmap of well-separated (left) and poor-separated (right) clusters

3.3.2 Reduced size of consensus matrix

We introduce a reduced size of consensus matrix, which has a $k\,x\,k$ dimension instead of $n\,x\,n$. It eliminates the swing/ seriation problem by applying the Hungarian method to optimize the relabeling result. Compared to the original consensus matrix ($n\,x\,n$), a reduced dimension of a consensus matrix has a smaller size of dimension, i.e. $k\,x\,k$, where k is the number of clusters. The values in the matrix represent proportions of stable and unstable objects when the objects are sampled twice in a pair of replicated data sets. To create a $k\,x\,k$ reduced size of consensus matrix from b bootstrap samples, the algorithm is

1. Assign all objects (population) into k clusters via a clustering algorithm.

2. Draw b bootstrap samples from the data.

3. Applying the clustering algorithm partitioned into k clusters, assign memberships of the selected objects for each b bootstrap samples.

4. Evaluate the membership label of the b bootstrap samples by referring to the population label in the step 1. To optimize the relabeling result, a linear programming method can be applied. However, because the Hungarian method is more efficient than linear programming (Hornik, 2017), the former is preferred to the latter.

5. Calculate the percentage of objects that remain in the same cluster or leaves to any different cluster when an object is taken twice in a pair of bootstrap

samples. As a result, square matrices $\mathbf{Q}$ as many as $\binom{b}{2}$ are produced. At the beginning, $\mathbf{Q}$ is a column vector with the length k. Then, it transforms into a $k\,x\,k$ squared matrix where $\mathbf{Q}_{(1,2)}$ and $\mathbf{Q}_{(2,1)}$ have opposite meaning. The former is the proportion of objects leaving from cluster 1 to cluster 2, while the latter means the proportion of objects moving from cluster 2 to cluster 1. Thus, matrix $\mathbf{Q}$ is read per row. If matrix $\mathbf{Q}$ is formed from a row vector in the beginning, it is read per column instead.

6. Calculate the weighted sum of $\mathbf{Q}_i$ matrices.

$$\mathbf{V} = \sum_{i=1}^{\binom{b}{2}} \frac{1}{\binom{b}{2}} \mathbf{Q}_i.$$

The matrix $\mathbf{V}$ can be a symmetric square matrix, if the proportion of objects leaving cluster i to cluster j is equal to the proportion of objects moving from cluster j to cluster i for all i and j $(i = j = 1, 2, \ldots k)$. The diagonal values of $\mathbf{V}$ always represent the proportion of stable objects, while the off diagonal values are the unstable proportion.

3.4 Visualization

The clustering process of both hierarchical and partitioning algorithm can be visualized in a plot. The common graph for the hierarchical algorithm is a dendogram, while a clustergram is able to illustrate both algorithms (Schonlau, 2004). Although the latter depicts the character of hierarchical (nested) and partitioning (merged and partitioned) algorithms well, neither well nor poor separated clusters are discernible. The well and poorly separated clusters are presented in two different visualization techniques, i.e. internal-based and relative-based criteria.

3.4.1 Internal-based criteria

Neighborhood graph

A neighborhood graph visualization is drawn based on an internal criterion, i.e. the shadow values (Equation 3.6) as edge weights, in an undirected graph (Leisch, 2006). The edges correspond to the either well or poorly separated clusters. The thinner the lines, the better the clusters are separated and vice versa. Figure 3.4 shows a well-separated cluster in a neighborhood graph (left), while Figure 3.5 illustrates a poorly-separated clusters. If the line in the neighborhood graph is thin, it indicates the clusters having good separation.

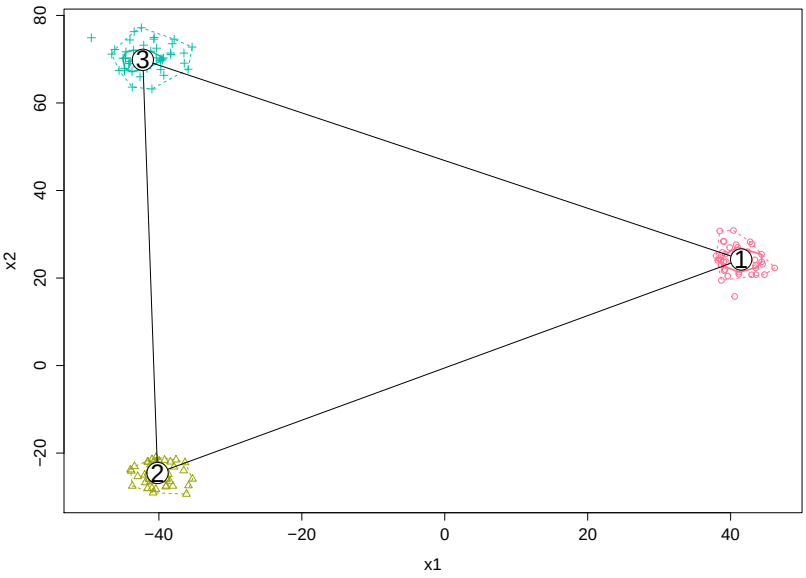


Figure 3.4: Shadow value plot of well-separated clusters in neighborhood graph

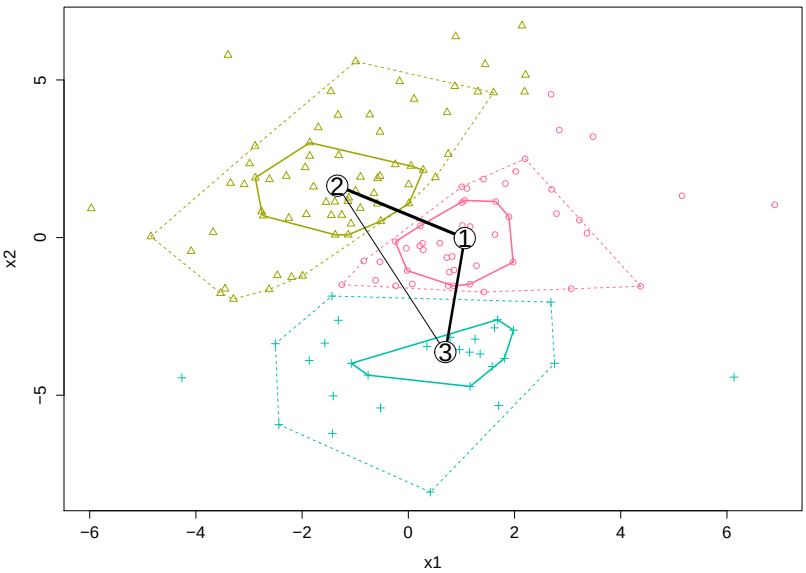


Figure 3.5: Shadow value plot of poor-separated clusters in neighborhood graph

Stripe plot

A stripe plot, moreover, is drawn based on the first and second closest centroid (Leisch, 2008). While the neighborhood graph has two axes representing two selected variables, the stripe plot has a cluster number in an axis, and a distance in the other axis. If clusters have good separation, it is indicated by non-overlap stripes. Figure 3.6 and 3.7 show well and poorly separated clusters illustrated in a stripe plot, respectively.

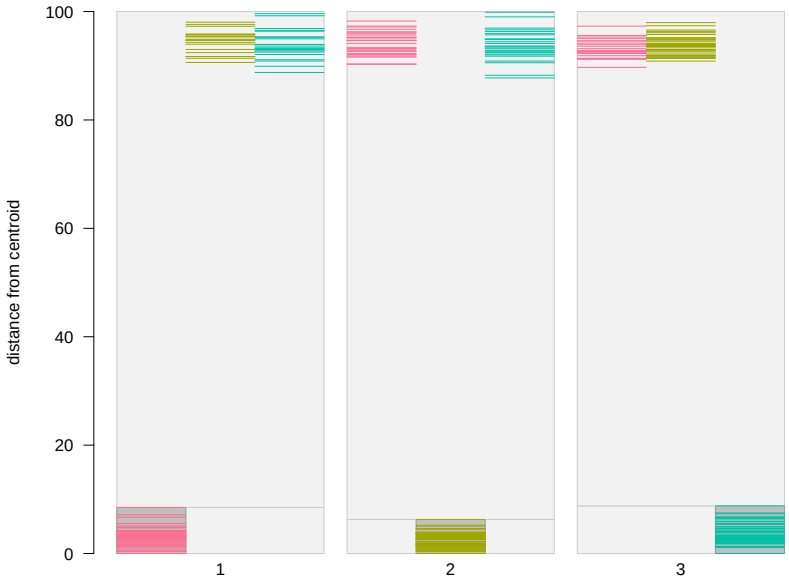


Figure 3.6: A stripe plot of well-separated clusters

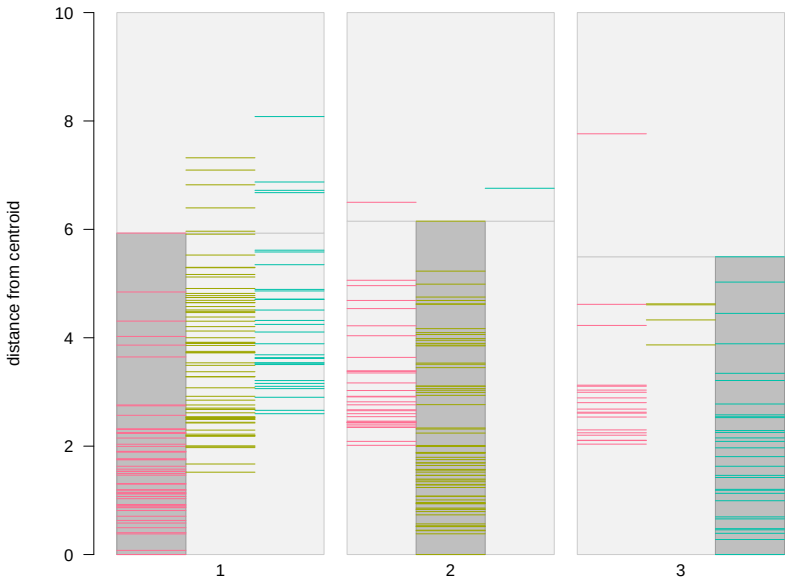


Figure 3.7: A stripe plot of poorly-separated clusters

Modified stripe plot

When the data set is mixed variables data, a modified stripe plot is proposed. Adapted from the original stripe plot, a modified stripe plot easily replaces the centroids into medoids. The results of the modified stripe plots in both the well and poorly separated clusters (Figure 3.8 and 3.9) are similar to the stripe plots. The only difference is that the former is based on medoids.

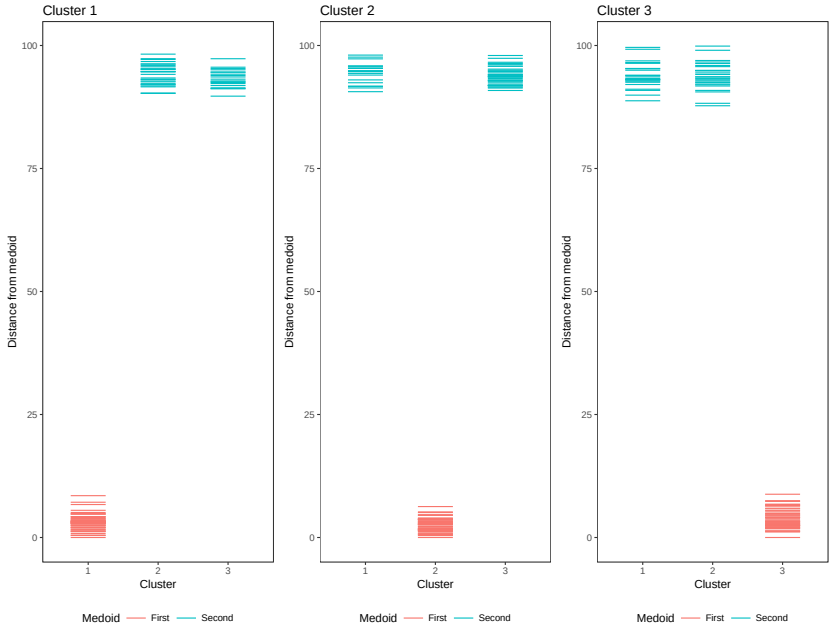


Figure 3.8: A modified stripe plot of well-separated clusters

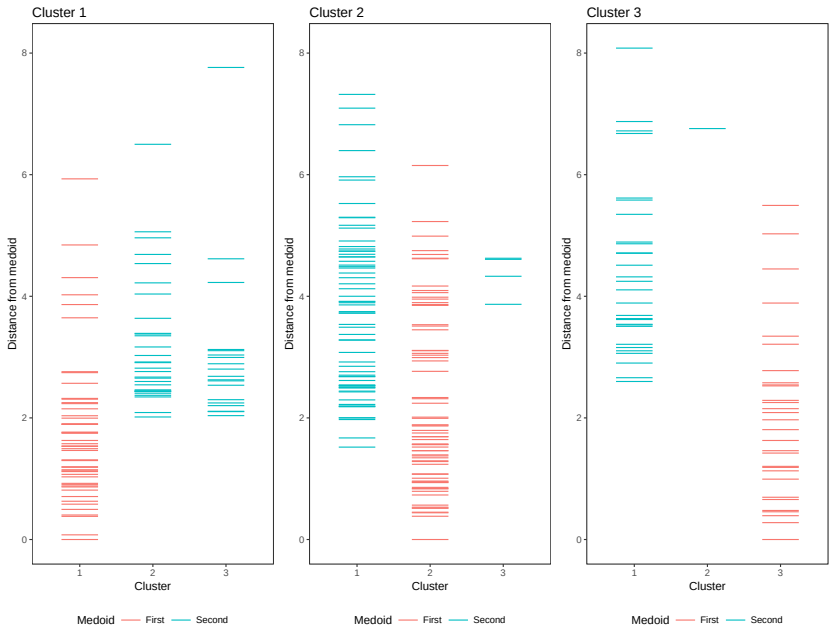


Figure 3.9: A modified stripe plot of poor-separated clusters

3.4.2 Relative-based criteria

The heatmap of consensus matrix in Section 3.3.1 represents a relative-based criteria visualization. It is a visualization of a $n\,x\,n$ consensus matrix. On the other hand, the reduced size of the consensus matrix (Section 3.3.2) is a summary version of the the $n\ x\ n$ consensus matrix. Because the reduced size of the consensus matrix has much smaller dimension ($k\ x\ k$) than the original consensus matrix, the matrix $\mathbf{V}$ can be visualized directly in a squared matrix (Figure 3.10).

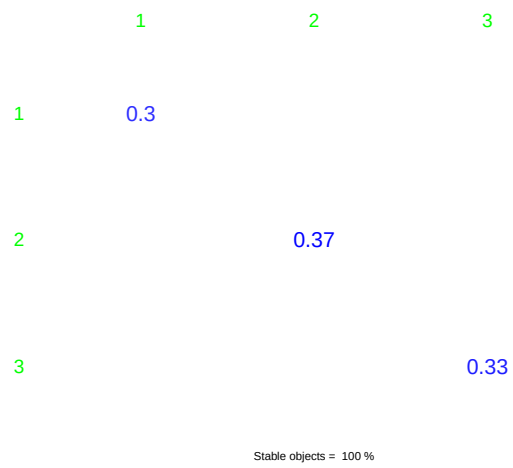


Figure 3.10: Direct visualization of matrix **V** of well-separated clusters

Heatmap-like pattern

A matrix can be visualized in a pattern such as squares, circles, bars, or pies (Murdoch and Chow, 1996; Friendly, 2002). Its values can also be transformed by applying either a linear or nonlinear transformation (Hahsler and Hornik, 2011) to produce a shaded patterns of plot for visualization purpose. Thus, a matrix **V** can be visualized in a pattern such that it is similar to a heatmap.

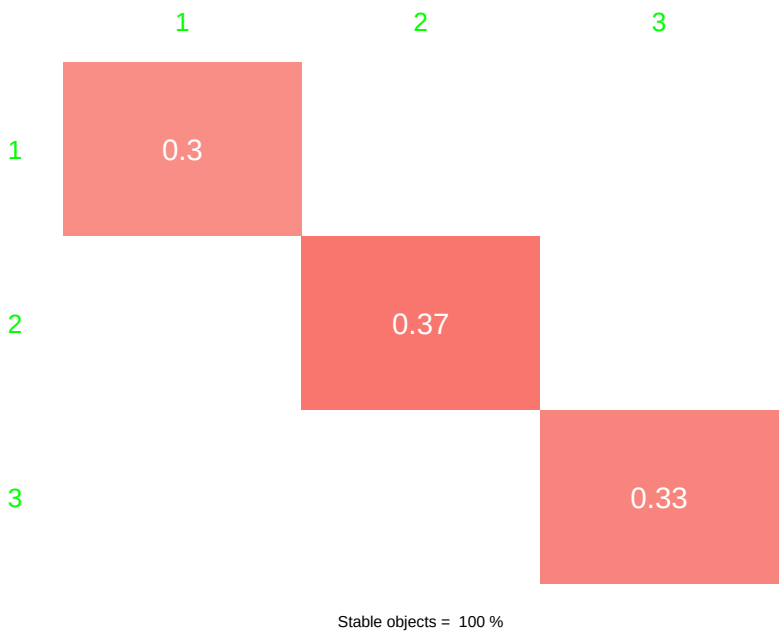


Figure 3.11: Reduced size of consensus matrix heatmap of well-separated clusters

Figure 3.11 shows a shaded squared patterns of matrix **V** by applying a non-linear transformation. Each element of matrix **V** is transform into

$$v_{ij} = \frac{v_{ij} - min\left(\forall_{i,j}\, v_{ij}\right)}{max\left(\forall_{i,j}\, v_{ij}\right) - min\left(\forall_{i,j}\, v_{ij}\right)}.$$

The best clustering result based on relative criteria is achieved when the sum of diagonal values are 1. While Figure 3.11 illustrates well-separated clusters, Figure 3.12 shows the contrary.

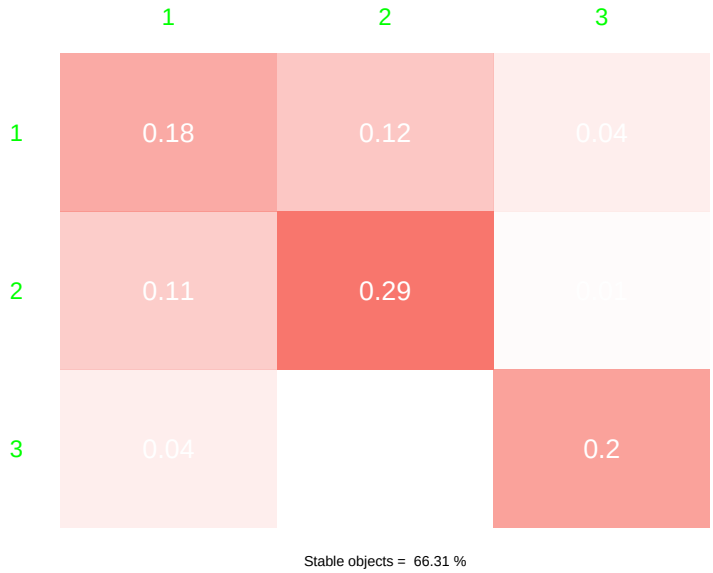


Figure 3.12: Reduced size of consensus matrix heatmap of poor-separated clusters

Although Figure 3.11 and 3.3 (left) are similar, they are read and constructed differently. The proportions of the stable/ unstable objects are visible in the reduced size of the consensus matrix, in which they can be easily replaced by the actual number of objects. While the $n\ x\ n$ consensus matrix describes the unstable objects twice in both the upper and lower of the off-diagonal values, i.e. indicated by a symmetric squared matrix, the reduced size of the $k\ x\ k$ consensus matrix only presents once. Hence, it has to be read either by row or column depending on the column or row vector at the beginning, respectively. In the grid measure, moreover, both heatmaps have different sizes, where the original consensus matrix has $n\ x\ n$ grids compared to only $k\ x\ k$ grids in the reduced size of the consensus matrix.

Directed graph

The reduced size of consensus matrix $\mathbf{V}$ can be directly plotted in 2-dimension as a directed graph. The graph elements can be decorated with attributes (Kolaczyk and Csardi, 2014) in order to comply with the associated values of the matrix $\mathbf{V}$. Some attribute elements are introduced as follows to visualize the matrix $\mathbf{V}$ in a network graph.

1. The number of nodes is as many as the number of clusters (k).

2. The size of the diameter nodes is proportional to the diagonal element values. This corresponds to the size of the stable objects.

3. The non-zero elements in the off-diagonal matrix are equal in number to the number of edges in the graph.

4. Because of a non-symmetric matrix, the edges are directed.

5. The weight for the edges is proportional to the off diagonal element values. This represents the number of unstable objects. If the weight is less than $\frac{1}{n}$, the edge is absent.

To set the nodes and edges in a 2-dimensional space, it requires a graph layout. Battista et al. (1994) have listed many graph layouts with their respective drawing algorithms. One of them is a tree in which a star layout is formed when the degree distribution of the nodes achieves a lower bound (Kincaid and Phillips, 2011). A k-star layout is depicted by a node in the center and $k-1$ equidistant nodes. The largest proportion of the stable objects can be set as the center of the nodes.

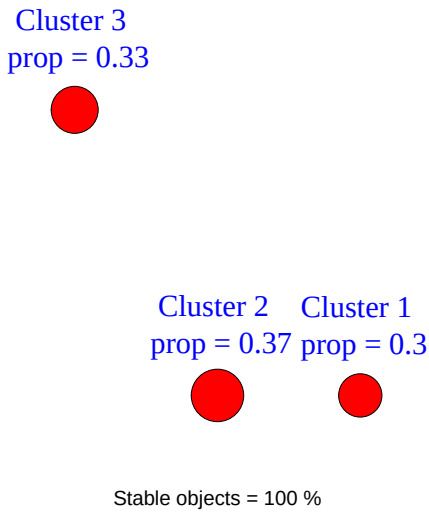


Figure 3.13: A directed graph of well separated clusters

Figure 3.13 shows a well-separated cluster presented in a directed graph opposing to Figure 3.14. The best-separated clusters are achieved when the edge is absent, while poorly-separated clusters are indicated by many edges with thicker lines. Although the neighborhood and directed graphs depict similar style, i.e. topology-network, they are created differently. They are based on the internal-based and relative-based criteria, respectively.

3.4.3 Combination of external-based and relative-based criteria

In order not to rely solely on the consensus matrix and stability measure (Senbabaoglu et al., 2014; Hennig, 2007), relative criteria can be combined with external criteria. The external criteria can also be extracted simultaneously when the relative criteria algorithm to obtain the reduced size of consensus matrix (Section 3.3.2) is run. Then,

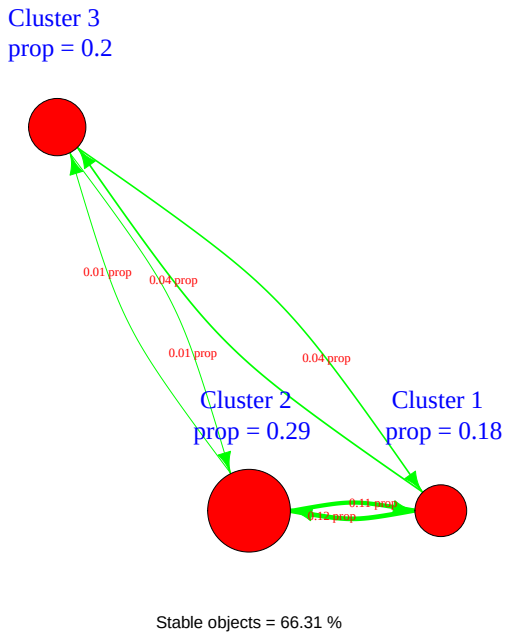


Figure 3.14: A directed graph of poor separated clusters

a kernel density evaluation of the external criteria can be plotted side by side with the reduced size of consensus matrix. Hence, the reduced size of consensus matrix and the kernel density become a good pair to visualize the bootstrap summary.

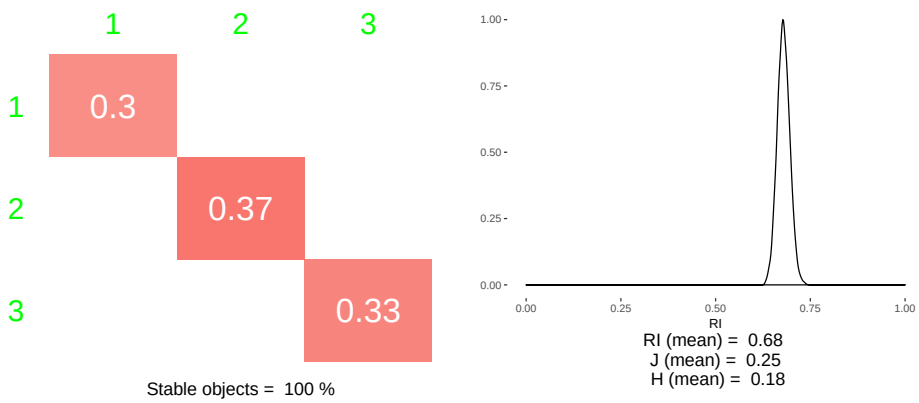


Figure 3.15: Combination of relative-based and internal-based criteria of well-separated clusters

Figure 3.15 shows the combination plot of a reduced size of the cluster consensus matrix in a squared pattern and a rand index kernel density. The best separated

clusters are indicated by a maximum proportion (100%) of stable objects and and a strong peak at one.

3.4.4 Location and dispersion

In order to interpret the cluster result of a numerical variable data set, Leisch (2008) has proposed a barplot with the measures of the location and dispersion of each cluster. This barplot can also be added by some markers such that a marked barplot is produced (Dolnicar and Leisch, 2014). The marker tags the mean of the population and within cluster (Figure 3.16). If the data set is an ordinal variables data set, on the other hand, Brentari et al. (2016) have performed a rank-based boxplot for the location and dispersion measures of the clusters.

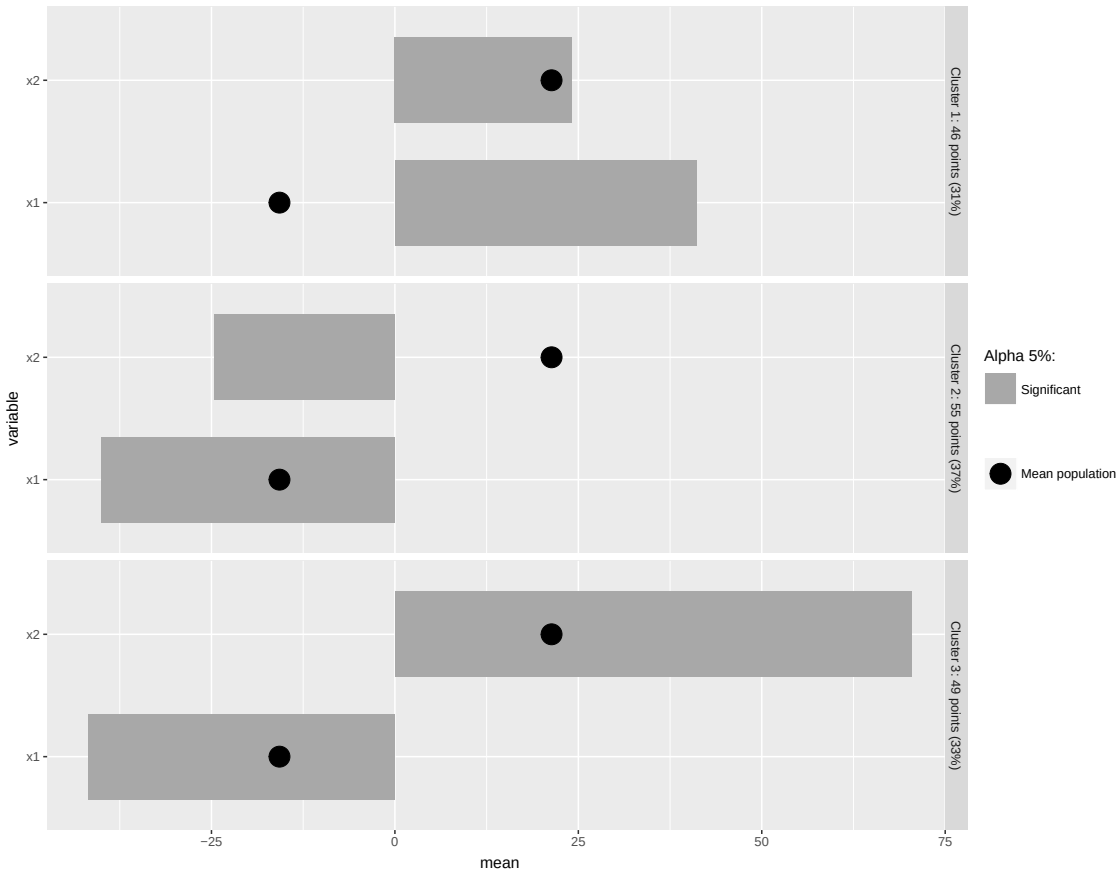


Figure 3.16: A marked barplot of numerical data set ($k = 3$)

While the barplot is intended for numerical data sets, we modify the barplot to adapt the mixed variable data. Two rules are introduced to create a modified barplot:

1. The numerical variables are re-scaled such that the minimum and maximum values are 0 and 1, respectively.

2. While a mean is applied for the numerical variables, a proportion measure is calculated in binary/ categorical variables to replace the mean.

Figure 3.17 shows a modified barplot of the mixed variable data set partitioned into two clusters. The population means (numerical variables) are indicated by dots, while the population proportion (binary and categorical variables) are denoted by triangles (t). If a variable has a triangle ($t = 1$), it is either a binary variable or a categorical variable that has two classes. Moreover, when the number of triangles is more than two ($t > 1$), the variables are categorical variables with $t + 1$ categories.

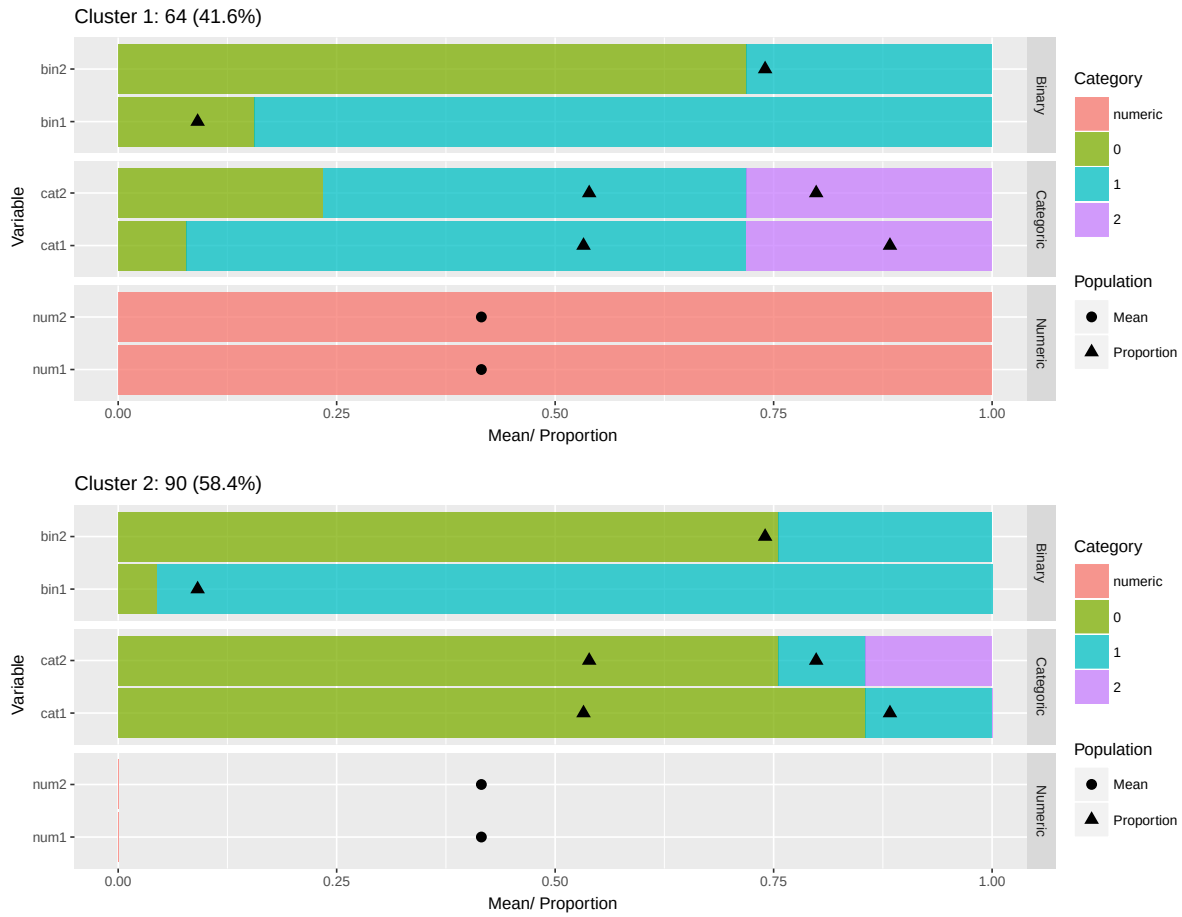


Figure 3.17: A modified marked barplot of mixed variable data set ($k = 2$)

4 Implementation in R

The GDF functions and medoids-based algorithms (Chapter 2), and cluster validation and visualization (Chapter 3) are implemented in the R software (R Core Team, 2015). This chapter discusses the main functions in the kmed package (Budiaji, 2019) with the presented codes and results between two horizontal lines. The details of implementation are required in order to extend the package usage by writing the user's own function, especially when implementing the GDF, developing the k-medoids initialization algorithm, applying bootstrap and heatmap reordering algorithms. For the implementation in R, a small data set, i.e. 100 objects, is generated by the clusterGeneration package (Qiu and Joe, 2015) with the genRandomClust function.

```
library(kmed)
set.seed(2018)
randclust <- clusterGeneration::genRandomClust(3, sepVal = 0.3, numNonNoisy
    = 6, numReplicate = 1, clustszind = 3, clustSizes =
    as.numeric(table(sample(1:3, 100, replace = TRUE))),outputDatFlag=FALSE,
    outputLogFlag=FALSE, outputEmpirical=FALSE, outputInfo=FALSE)
dat1 <- randclust$datList$test_1
dat1 <- data.frame(dat1)
rownames(dat1) <- 1:nrow(dat1)
classdat1 <- randclust$memList$test_1
dat1$x3 <- dat1$x3 < median(dat1$x3)
dat1$x4 <- dat1$x4 < median(dat1$x4)
## Quantile categorization
intv <- function(vec, class) {
  nbase <- (1:(class-1))/class
  nq <- numeric(length(nbase))
  for (i in 1:length(nq)) {
    nq[i] <- quantile(vec, nbase[i])
  }
  res <- c(min(vec), nq, max(vec))
  res[1] <- res[1]-1
  return(res)
}
dat1$x5 <- as.factor(cut(dat1$x5, intv(dat1$x5, 3), labels = (1:3)))
dat1$x6 <- as.factor(cut(dat1$x6, intv(dat1$x6, 4), labels = (1:4)))
```

The generated data set in this chapter has three clusters with two variables of each numerical, binary, and categorical variables. Cluster 1, 2, and 3 have 39, 34, and 27 objects, respectively. The binary and categorical variable are generated via a quantile categorization (Section 5.2). While the categorical variables have three and four classes, each binary variable has equally distributed categories.

```
classdat1
##  [1] 1 1 1 2 2 1 1 1 1 2 2 1 1 3 3 1 2 2 3 3 2 1 2 3 2 3 1 2 3 2 1 3 3
## 2 3 2 2 1 1 1 3 1 2 1 3 3 1 2 3
## [50] 2 3 3 1 1 2 1 2 1 1 3 2 2 2 3 1 1 1 1 1 2 2 3 3 1 1 3 2 2 1 1 2 1
## 2 1 3 3 3 2 2 2 3 3 2 2 3 2 1 1
## [99] 1 1
table(classdat1)
## classdat1
##  1  2  3
## 39 34 27
summary(dat1)
##      x1               x2              x3             x4          x5     x6
## Min.   :-6.8230  Min.   :-9.0950  Mode :logical  Mode :logical  1:34   1:25
## 1st Qu.:-2.1477  1st Qu.:-3.7587  FALSE:50       FALSE:50       2:33   2:25
## Median : 0.2452  Median : 0.7726  TRUE :50       TRUE :50       3:33   3:25
## Mean   : 0.2024  Mean   : 0.1101  NA's :0        NA's :0               4:25
## 3rd Qu.: 2.2374  3rd Qu.: 3.8407
## Max.   : 6.0427  Max.   : 8.4472
```

4.1 Distance calculation in R

4.1.1 Numerical distances

In the kmed package, there are five numerical distances implemented, namely Manhattan weighted by range (mrw), squared Euclidean weighted by range (ser), squared Euclidean weighted by squared range (ser.2), squared Euclidean weighted by variance (sev), and un-weighted squared Euclidean (se). To implement the numerical distance, the distNumeric function can be called. Then, the method argument controls the desired distance, for instance squared Euclidean weighted by range (ser).

```
numdist <- distNumeric(data.matrix(dat1[,1:2]), data.matrix(dat1[,1:2]),
   method = "ser")
round(numdist[1:6,1:6],2)
##       1    2    3     4     5    6
##1   0.00 2.53 6.24 12.52  11.47 1.04
##2   2.53 0.00 0.82  7.37   5.35 1.20
```

```
##3   6.24 0.82 0.00   6.48   3.95 3.49
##4  12.52 7.37 6.48   0.00   0.42 6.37
##5  11.47 5.35 3.95   0.42   0.00 5.65
##6   1.04 1.20 3.49   6.37   5.65 0.00
```

4.1.2 Binary and categorical distance

The cooccur function implements the co-occurrence distance for binary and categorical variables. The other distance is the simple matching (matching) that is commonly applied for binary and categorical variables. The difference between the cooccur and matching functions is that the latter applies a pair distance such that the inputs have to be two matrices.

```
bindist <- cooccur(dat1[,3:4])
bindist[1:6,1:6]
##      [,1] [,2] [,3] [,4] [,5] [,6]
##[1,] 0.00 0.44 0.00 0.44 0.44 0.88
##[2,] 0.44 0.00 0.44 0.88 0.88 0.44
##[3,] 0.00 0.44 0.00 0.44 0.44 0.88
##[4,] 0.44 0.88 0.44 0.00 0.00 0.44
##[5,] 0.44 0.88 0.44 0.00 0.00 0.44
##[6,] 0.88 0.44 0.88 0.44 0.44 0.00
catdist <- matching(dat1[,3:4], dat1[,3:4])
catdist[1:6,1:6]
##     1   2   3   4   5   6
##1 0.0 0.5 0.0 0.5 0.5 1.0
##2 0.5 0.0 0.5 1.0 1.0 0.5
##3 0.0 0.5 0.0 0.5 0.5 1.0
##4 0.5 1.0 0.5 0.0 0.0 0.5
##5 0.5 1.0 0.5 0.0 0.0 0.5
##6 1.0 0.5 1.0 0.5 0.5 0.0
```

4.1.3 Mixed distances

The mixed distances like the Gower (gower), Wishart (wishart), Podani (podani), Huang (huang), Harikumar-PV (harikumar), Ahmad-Dey (ahmad) are directly available in the kmed package via the distmix function. The distmix function requires a column id of each class of variables. If the results of the distmix are compared to the daisy function in the cluster package (Maechler et al., 2017), they are equal.

```
distdat1 <- distmix(dat1, method = 'gower', idnum=1:2, idbin=3:4, idcat=5:6)
round(distdat1[1:6,1:6],2)
```

```
##     1    2    3    4    5    6
##1 0.00 0.59 0.47 0.53 0.50 0.55
##2 0.59 0.00 0.55 0.83 0.80 0.40
##3 0.47 0.55 0.00 0.48 0.45 0.78
##4 0.53 0.83 0.48 0.00 0.03 0.64
##5 0.50 0.80 0.45 0.03 0.00 0.61
##6 0.55 0.40 0.78 0.64 0.61 0.00
gowdat1 <- cluster::daisy(dat1, metric = "gower", type = list(symm = 3:4))
sum(distdat1-as.matrix(gowdat1) > 1e-10)
## [1] 0
```

4.2 Medoids-based algorithms in R

There are three medoids-based algorithms directly implemented in the kmed package, namely the SFKM (fastkmed), RKM (rankkmed), and INCKM (inckmed). For the KM and SKM algorithms, the fastkmed function can be indirectly applied (Section 4.4.2). On the other hand, the PAM has been implemented in the cluster package with the pam function and k-medoids from a linear programming perspective has been applied in the clue package (Hornik, 2005, 2017) with the kmedoids function.

4.2.1 SFKM in R

The generated data set, dat1, is partitioned via the SFKM algorithm with the Gower GDF. The fastkmed function requires arguments of the distance matrix, number of clusters, and number of iterations (10 as a default). With the dat1 data set, the SFKM results in 16% missed classification rate. The medoid objects are objects number 37, 72, and 82. The sum of the within cluster distance is calculated as 22.4.

```
sfkmdat1 <- fastkmed(distdat1, 3, iterate = 50)
resfkm <- sfkmdat1$cluster
table(classdat1, resfkm)
##        resfkm
##classdat1 1  2  3
##       1  1 32  6
##       2 32  1  1
##       3  5  2 20
(1+6+1+1+5+2)/100
## [1] 0.16
sfkmdat1$medoid
## [1] 37 82 72
```

```
sfkmdat1$minimum_distance
## [1] 22.40216
```

4.2.2 RKM in R

Next, the RKM algorithm can be also applied directly in R. Similar to the fastkmed function, the rankkmed requires the same arguments with an additional argument m to calculate a hostility measure. If m is set to 5, the hostility is calculated based on the five closest objects from the medoids. For the dat1 data set, the RKM partitioning algorithm produces 22% missed classification rate with the objects number 18, 31, and 51 as the medoids. The sum of the within cluster distance is higher than the SFKM algorithm. It determines that the SFKM is better than the RKM for the dat1 data set.

```
rkmdat1 <- rankkmed(distdat1, 3, m = 5, iterate = 50)
resrkm <- rkmdat1$cluster
table(classdat1, resrkm)
##        resrkm
##classdat1 1  2  3
##       1  2 29  8
##       2 33  0  1
##       3  8  3 16
(2+8+0+1+8+3)/100
##[1] 0.22
rkmdat1$medoid
##[1] "18" "31" "51"
rkmdat1$minimum_distance
##[1] 24.95745
```

4.2.3 INCKM in R

The other medoid-based algorithm that is directly applied is the inckmed function that implements the INCKM algorithm. Instead of m, the inckmed requires alpha as an additional argument. It is a stretch factor that specifies a range to find medoid candidates. Although the missed classification rate of the INCKM algorithm is equal to the SFKM algorithm, i.e. 16%, its sum of the within cluster distance is lower than the SFKM indicating that the INCKM is better.

```
inckmdat1 <- inckmed(distdat1, 3, alpha = 1.1, iterate = 50)
resinckm <- inckmdat1$cluster
```

```
table(classdat1, resinckm)
##        resinckm
##classdat1 1  2  3
##       1  2 35  2
##       2  7  0 27
##       3 22  3  2
(2+2+7+0+3+2)/100
## [1] 0.16
inckmdat1$medoid
## [1] 49 82 37
inckmdat1$minimum_distance
## [1] 21.80374
```

4.3 Cluster validation and visualization in R

The kmed package implements internal and relative criteria validation. Both criteria can be visualized in a graph. The clustering results, moreover, are visualized via a biplot of the principle component and barplot.

4.3.1 Internal criteria in R

Silhouette

The sil function calculates silhouette values of each object and produces a silhouette plot. It requires a distance matrix, medoid id, and cluster membership, while a title of the plot is an option. The silhouette plot of the dat1 data set partitioned via the INCKM algorithm is shown in Figure 4.1 (left). In cluster 1, there are some objects that have negative silhouette values.

```
sildat1 <- sil(distdat1, inckmdat1$medoid, inckmdat1$cluster)
sildat1$result[c(1:5),]
##    silhouette cluster
##1  0.220040523     2
##2  0.467714497     2
##3 -0.009763089     1
##4  0.397216287     3
##5  0.413070884     3
```

Shadow value

Figure 4.1 (right) also shows a shadow value plot of the dat1 data set produced by the csv function. The required arguments for this function are identical to the sil function.

Based on the silhouette and shadow value plots, both plots depict poorly-separated clusters when the dat1 data set is partitioned via the INCKM algorithm.

```
shadat1 <- csv(distdat1, inckmdat1$medoid, inckmdat1$cluster)
shadat1$result[c(1:5),]
##    shadval cluster
##1 0.8024029     2
##2 0.5110660     2
##3 0.9357375     1
##4 0.6596323     3
##5 0.6274620     3
gridExtra::grid.arrange(sildat1$plot, shadat1$plot, ncol = 2)
```

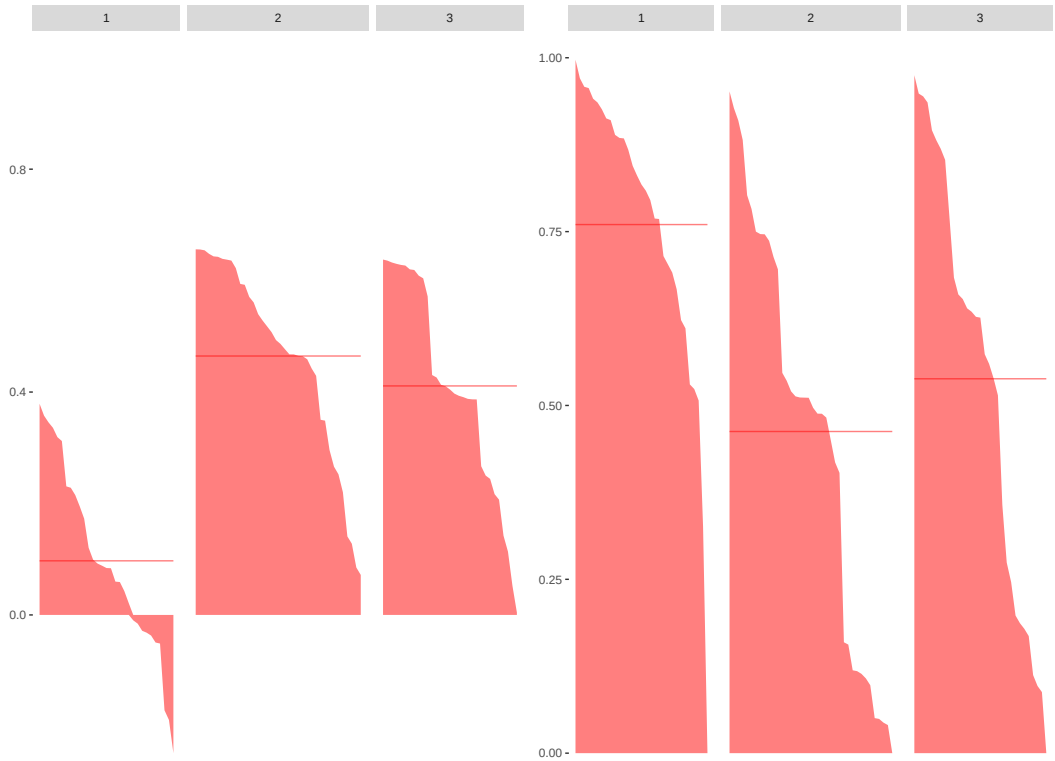


Figure 4.1: Silhouette (left) and shadow value (right) plots of the generated data set (dat1)

4.3.2 Relative criteria in R

To apply relative criteria validation via a heatmap of the $n\ x\ n$ consensus matrix , there are three steps. First, a matrix of bootstrap replicates is created. Then, this matrix is transformed into a consensus (agreement) matrix. The last step is drawing the consensus matrix in a heatmap image.

The bootstrap matrix

The clustboot function can be applied to create a bootstrap matrix. There are four arguments in the function, i.e. a distance matrix, number of clusters, algorithm, and number of bootstrap replicates. For the algorithm argument, it is a user defined function, i.e. a function within an argument, that has to be created before hand. It has two input arguments, namely a distance matrix and number of clusters. Meanwhile, the output is a vector of class membership. The default argument for the algorithm provided in the kmed package is the SFKM algorithm, which is defined as the fastclust function.

```
# the default function for the bootstrap/ reorder matrix provided in the
# package. A re-declaration of the function is unnecessary.
# fastclust <- function(x, nclust) {
#  res <- fastkmed(x, nclust, iterate = 50)
#  return(res$cluster)
# }

# step 1: a matrix of bootstrap replicates via inckm algorithm
# an inckm algorithm as an input in the algorithm argument is developed
algorinckm <- function(x, nclust) {
  res <- inckmed(x, nclust, alpha = 1.1, iterate = 50)
  return(res$cluster)
}
inckmboot <- clustboot(distdat1, nclust=3, algorithm = algorinckm, nboot=50)
colnames(inckmboot) <- paste("bootstrap sample", 1:ncol(inckmboot))
rownames(inckmboot) <- 1:nrow(inckmboot)
inckmboot[1:6, 1:3]
##  bootstrap sample 1 bootstrap sample 2 bootstrap sample 3
##1                  1                  2                  2
##2                  1                  2                  2
##3                  2                  1                  1
##4                  0                  0                  3
##5                  2                  0                  3
##6                  1                  2                  2
```

The consensus matrix

The consensus matrix is produced by the consensusmatrix function based on the previous bootstrap matrix. The consensus matrix has to be ordered such that objects having a similar consensus index are close to each other. A user has to define how the consensus matrix is ordered. This ordering task can be carried out by creating

an ordering function, for instance a ward linkage algorithm ordering. This function is then supplied in the reorder argument of the consensusmatrix function. The reorder argument, moreover, is similar to the algorithm argument in the clustboot function. The default reorder function is the fastclust function.

```
# step 2: a consensus matrix
# a ward linkage algorithm as an input in the reorder argument is generated
wardorder <- function(x, nclust) {
  res <- hclust(x, method = "ward.D2")
  member <- cutree(res, nclust)
  return(member)
}
consensusinckm <- consensusmatrix(inckmboot, nclust = 3, reorder =
    wardorder)
consensusinckmalt <- consensusmatrix(inckmboot, nclust = 3, reorder =
    algorinckm)
consensusinckm[1:6, 1:6]
##          1         1         1         1         1         1
##1 1.0000000 0.7142857 0.8823529 0.6000000 0.7391304 0.7894737
##1 0.7142857 1.0000000 0.7500000 0.8823529 0.8947368 0.5625000
##1 0.8823529 0.7500000 1.0000000 1.0000000 0.9000000 0.8421053
##1 0.6000000 0.8823529 1.0000000 1.0000000 1.0000000 0.7058824
##1 0.7391304 0.8947368 0.9000000 1.0000000 1.0000000 0.7222222
##1 0.7894737 0.5625000 0.8421053 0.7058824 0.7222222 1.0000000
consensusinckmalt[1:6, 1:6]
##          1         1         1         1         1         1
##1 1.0000000 1.0000000 1.0000000 0.9565217 1.0000000 0.9583333
##1 1.0000000 1.0000000 0.9473684 1.0000000 1.0000000 1.0000000
##1 1.0000000 0.9473684 1.0000000 1.0000000 0.9545455 1.0000000
##1 0.9565217 1.0000000 1.0000000 1.0000000 0.9375000 0.9500000
##1 1.0000000 1.0000000 0.9545455 0.9375000 1.0000000 0.9500000
##1 0.9583333 1.0000000 1.0000000 0.9500000 0.9500000 1.0000000
```

The heatmap

The consensus matrix can be plotted in a heatmap. The clustheatmap function does it directly when the input is a consensus matrix. An argument of the heatmap title can be added in the clustheatmap function as well. A squared block diagonal image of the heatmap indicates that the evaluated clustering algorithm is suitable to group the data. Figure 4.2 shows a heatmap of the generated data set (dat1) partitioned by the INCKM algorithm. The consensus matrix yielded is then reordered by the ward linkage and INCKM algorithms. The results of the two different reordering algorithms

depict two similar patterns of heatmap.

```
# step 3: a heatmap image
plotheat1 <- clustheatmap(consensusinckm, "Ward reorder")
plotheat2 <- clustheatmap(consensusinckmalt, "INCKM reorder")
gridExtra::grid.arrange(plotheat1, plotheat2, ncol = 2)
```

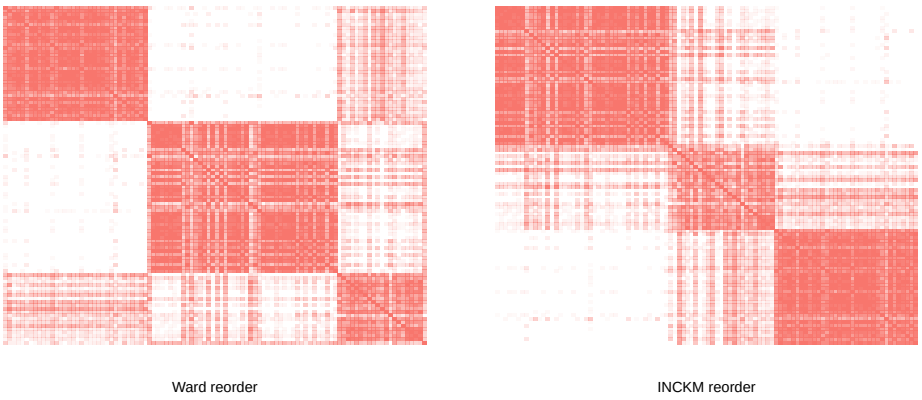


Figure 4.2: Heatmap image of generated data set (dat1) partitioned via the INCKM algorithm with the ward linkage (left) and INCKM (right) to reorder the consensus matrix

4.3.3 Visualization of the clustering result in R

Principle component plot

If variables in the data set are all numerical variables, the pcabiplot function can be applied to obtain a graph of two principle components (pc) in its axes and colored objects based on the clustering result. This function has arguments to control the axes and colored objects. Figure 4.3 shows a pca biplot of iris data set grouped by the SFKM algorithm with Manhattan range weighted GDF.

```
pcadat <- prcomp(iris[,1:4], scale. = TRUE)
mrwiris <- distNumeric(data.matrix(iris[,1:4]), data.matrix(iris[,1:4]),
    method = "mrw")
sfkmiris <- fastkmed(mrwiris, 3, iterate = 50)
pcabiplot(pcadat, colobj = sfkmiris$cluster+1, o.size = 3)
```

Barplot

The summary of variables based on the clustering result can be plotted in a barplot. The barplotnum function creates a barplot of clustering results from numerical variables data set. It requires the original data, and class memberships. It also produces a significance test of each variable, i.e. t-test between means of the population and

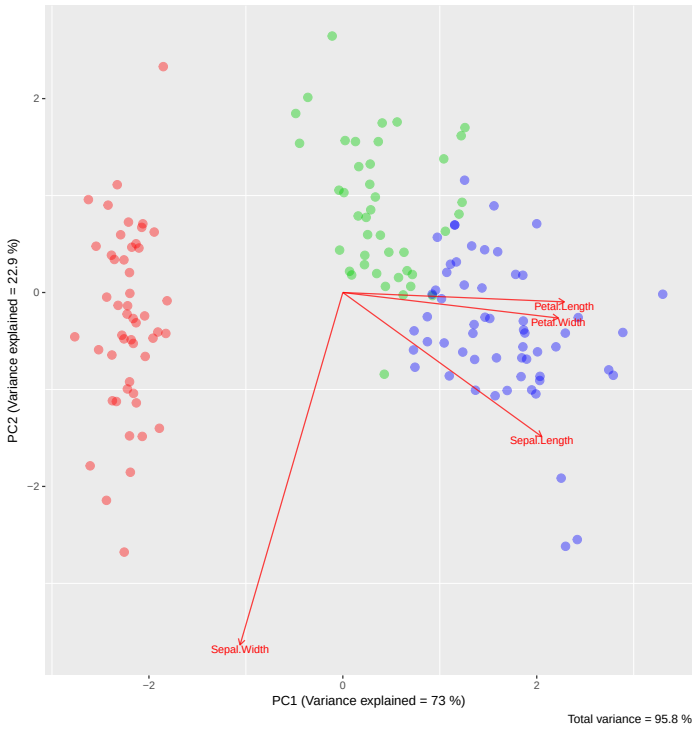


Figure 4.3: PCA biplot of the iris data set with colored objects via the SFKM algorihtm with the Manhattan range weighted GDF

within a cluster. The layout of the barplot is set in the nc argument. Figure 4.4 shows a barplot of the iris data set via the SFKM algorithm with Manhattan range weighted GDF. The layout of Figure 4.4 is set nc = 1.

```
barplotnum(iris[,1:4], sfkmiris$cluster)
```

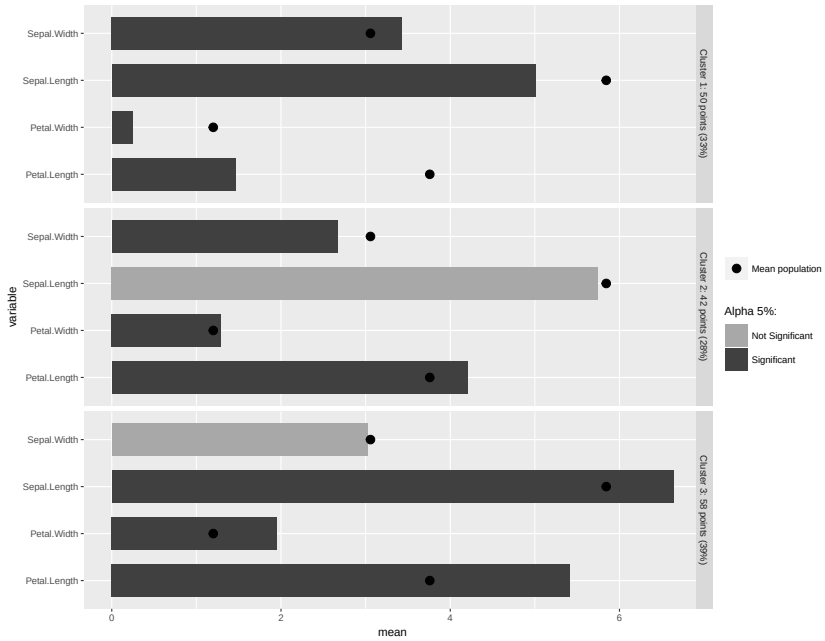


Figure 4.4: Barplot of the iris data set via the SFKM algorithm

4.4 Writing your own functions

4.4.1 GDF generating

The GDF (Section 2.2) can be generated manually to vary the distance options in the mixed variable data. Esimma, for example, which combines Euclidean and simple matching distances, can be easily implemented in R. The collection of numerical, binary, and categorical distances can be obtained from proxy (Meyer and Buchta, 2016) and nomclust (Sulc and Rezankova, 2016) packages. The latter package is an implementation of some categorical distances, such as Morlini and Zani (2012) and Lin (1998), that have been presented by Boriah et al. (2008). With these additional two packages, a user can vary the GDF by combining the numerical, binary, and categorical distances. The result of the SFKM algorithm via the combination of Mahalanobis (numerical), Tanimoto (binary), and Morlini (categorical), for instance, produces a 15% missed classification rate, which is slightly lower than the Gower GDF (Section 4.2.1).

```
#combine Mahalanobis, Tanomoto, and Morlini
matamor <- as.matrix(proxy::dist(dat1[,1:2], method = "Mahalanobis")) +
  as.matrix(proxy::dist(dat1[,3:4], method = "Tanimoto")) +
  nomclust::morlini(dat1[,5:6])
diag(matamor) <- 0
sfkmmata <- fastkmed(matamor, 3, iterate = 50)
resfkm2 <- sfkmmata$cluster
table(classdat1, resfkm2)
##        resfkm2
##classdat1 1  2  3
##       1  0 38  1
##       2 32  0  2
##       3  7  5 15
(0+1+0+2+7+5)/100
## [1] 0.15
sfkmmata$medoid
## [1] 37 82 64
sfkmmata$minimum_distance
## [1] 171.4651
```

4.4.2 SFKM initial medoids

In the fastkmed function, the init argument is set to be NULL where the standard SFKM algorithm is applied. This argument is flexible such that other algorithms can emerge by defining the desired initial medoids, for instance the KM algorithm, which set the initial medoids randomly. It produces 24% missed classification rate. The

INCKM algorithm, which is denoted by the inckmed function in the kmed package, in addition, also utilizes the init argument flexibility.

```
kminit <- sample(1:nrow(matamor), 3)
kmmata <- fastkmed(matamor, 3, iterate = 50, init = kminit)
reskm <- kmmata$cluster
table(classdat1, reskm)
##        reskm
##classdat1 1  2  3
##       1  0  7 32
##       2 34  0  0
##       3 12 10  5
(0+7+0+0+12+5)/100
## [1] 0.24
kmmata$medoid
## [1] 37 91 42
kmmata$minimum_distance
## [1] 166.985
```

4.4.3 Cluster bootstrap

The bootstrap matrix, which is created by the clustboot function, has the algorithm argument that can be defined by a user (Section 4.3.2). It guarantees that any hard clustering algorithms is applicable. It has to be consisted of two input arguments, i.e. a distance matrix and number of clusters and a vector output of cluster memberships. A hard clustering from other packages, the pam function from the cluster package for example, is applicable as long as the user-defined function has correct input and output arguments.

```
# A user defined funtion with a distance matrix and number of cluster
# input arguments and a vector of class membership output.
algorpam <- function(x, nclust){
  algor <- cluster::pam(x, nclust, diss = TRUE)
  res <- algor$clustering
  return(res)
}
pamboot <- clustboot(distdat1, nclust = 3, algorithm = algorpam, nboot = 50)
colnames(pamboot) <- paste("bootstrap sample", 1:ncol(pamboot))
rownames(pamboot) <- 1:nrow(pamboot)
pamboot[1:6, 1:3]
##  bootstrap sample 1 bootstrap sample 2 bootstrap sample 3
##1                  1                  1                  1
##2                  1                  1                  1
```

```
##3                0                2                2
##4                2                2                3
##5                2                2                3
##6                1                1                1
```

4.4.4 Heatmap reordering

The heatmap reordering has a similar case to the cluster bootstrap matrix. The consensusmatrix function, which is intended to produce a heatmap, has the reorder argument that is adjustable to be defined by a user. The format of this user defined function is similar to the algorithm argument in the cluster bootstrap matrix. The difference is that the reorder is only valid for any distance-input clustering algorithm due to the nxn consensus/ agreement matrix as an input. Figure 4.5 shows a heatmap of the simulated data set (dat1) grouped by the INCKM algorithm and reordered by the PAM algorithm. It produces a similar heatmap pattern to Figure 4.2.

```
consensuspam <- consensusmatrix(inckmboot, nclust = 3, reorder = algorpam)
clustheatmap(consensuspam, "PAM reorder")
```

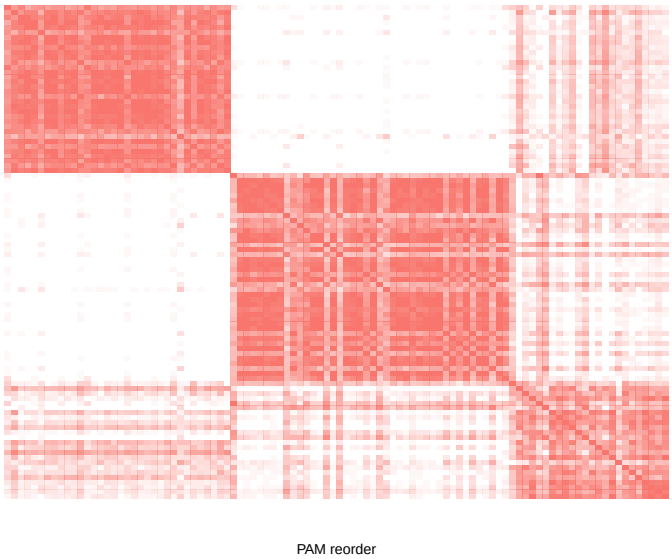


Figure 4.5: Heatmap of the generated data set (dat1) partitioned via the INCKM algorithm with the PAM algorithm to reorder the consensus matrix

5 Demonstration

Two types of data sets are presented in this chapter. The first types are artificial data sets from the Qiu and Joe (2006a) cluster generation algorithm. This algorithm generates only numerical data sets, such that the categorization of a numerical variable to a binary/ categorical variable is required in order to obtain a mixed variable data set. On the other hand, the second types of the data sets are real data sets from the machine learning repository of UCI (Lichman, 2013) and The Economist Intelligent Unit (2017). All data sets are then analyzed via six different k-medoids algorithms. The GDF also varies, applying the Gower, Wishart, Podani, Huang, Harikumar-PV GDF. Two distances derived from the GDF formula are introduced by varying the distance combinations (Table 5.1).

Table 5.1: Two distances from the GDF formulation

GDF	ω	α	β	γ	$\delta_n(x_{ir}, x_{jr})$	$\delta_b(x_{it}, x_{jt})$	$\delta_c(x_{is}, x_{js})$
Esimma	1	1	1	1	E	SM	SM
Marweco	1	1	1	1	M rw	CoC	CoC

E = Euclidean, CoC = Co-occurrence, M = Manhattan,
SM = Simple Matching, rw = range weighted

5.1 Artificial numerical data set

In the data simulation, the Qiu and Joe (2006a) algorithm is applied to generate an artificial data set consisting of two numerical variables partitioned into four clusters. The algorithm has a separation index (Qiu and Joe, 2006b) between -1 and 1, such that the more dispersed cluster has a separation index closer to 1. The separation index of the simulated data set is set to be 0.1, 0.3, 0.5, and 0.7. Then, each data set has 200 objects (Figure 5.1).

The simulated data set is partitioned via six k-medoids algorithms with seven GDF's (Tabel 2.1 and 5.1). Due to involving only numerical variables, the seven GDF's are adjusted (Table 5.2). Then, the rand index (Equation 3.3) is applied to evaluate the clustering results.

5.1.1 PAM

When the artificial data set is partitioned via the PAM algorithm, the rand indices that are produced from all seven GDF's are consistently high (Table 5.3). This indicates

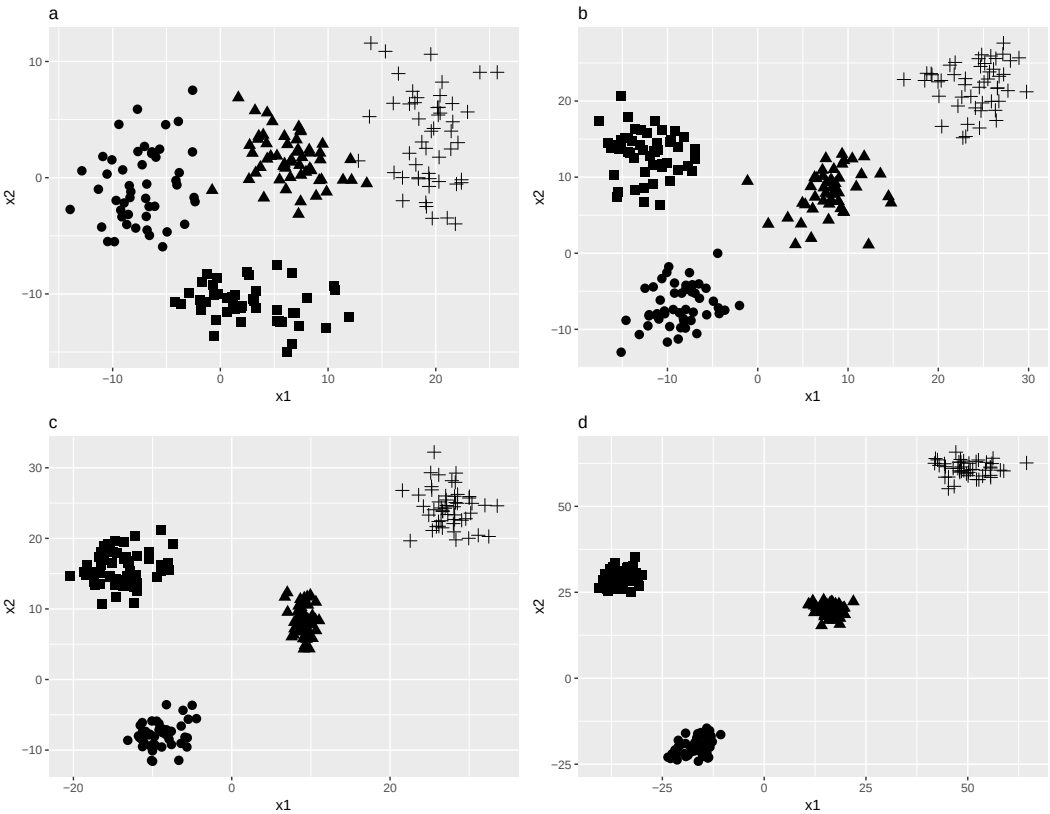


Figure 5.1: The simulated data set with separation values: 0.1 (a), 0.3 (b), 0.5 (c), and 0.7 (d)

Table 5.2: Adjusted numerical variable distances

GDF	ω	α	$\delta_n(x_{ir}, x_{jr})$
Gower	1	$\frac{1}{p_n}$	Manhattan weighted by range
Wishart	1	$\frac{1}{p_n}$	Squared Euclidean weighted by variance
Podani	$\frac{1}{2}$	1	Squared Euclidean weighted by squared range
Huang	1	1	Squared Euclidean
Harikumar-PV	1	1	Manhattan
Esimma	1	1	Euclidean
Marweco	1	1	Manhattan weighted by range

Table 5.3: Rand indices of the simulated data via the PAM algorithm

Data	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
Data a	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Data b	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Data c	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00

that the PAM produces a very good clustering result for all of the data sets in any GDF.

5.1.2 KM

For each GDF, the KM algorithm is repeated five times. The results of the KM algorithm for the simulated data produce varied rand indices (Table 5.4). This indicates a strong effect of medoid initialization, where different initials produce different results.

Table 5.4: Rand indices of the simulated data via the KM algorithm

Replicate	Data	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
1	Data a	0.99[a]	0.98	0.98	0.99	0.99	0.99	0.82
	Data b	1.00	0.83	0.84	1.00	0.84	1.00	1.00
	Data c	0.86	1.00	1.00	1.00	1.00[b]	1.00	1.00
	Data d	1.00	1.00	0.57	0.83	0.84	1.00	0.83
2	Data a	0.81[a]	0.98	0.98	0.99	0.82	0.99	0.98
	Data b	0.84	1.00	0.84	1.00	1.00	1.00	0.84
	Data c	1.00	1.00	0.85	1.00	1.00[b]	1.00	1.00
	Data d	1.00	1.00	0.84	0.81	1.00	0.84	0.85
3	Data a	0.83[a]	0.98	0.99	0.99	0.99	0.99	0.99
	Data b	1.00	1.00	1.00	1.00	1.00	1.00	0.84
	Data c	1.00	1.00	1.00	0.79	0.84[b]	0.83	0.83
	Data d	0.84	1.00	1.00	1.00	0.83	0.85	1.00
4	Data a	0.82[a]	0.82	0.98	0.82	0.83	0.99	0.98
	Data b	1.00	1.00	1.00	1.00	1.00	1.00	0.84
	Data c	0.83	0.84	1.00	0.84	0.59[b]	0.86	0.84
	Data d	1.00	1.00	0.86	1.00	0.59	0.84	1.00
5	Data a	0.98[a]	0.81	0.83	0.81	0.99	0.99	0.99
	Data b	1.00	1.00	1.00	1.00	1.00	0.84	1.00
	Data c	0.83	1.00	0.85	0.84	1.00[b]	1.00	0.84
	Data d	0.83	0.84	0.85	1.00	1.00	0.84	1.00

Examples of rand indices that vary in different replications in:
[a] Gower and [b] Harikumar-PV GDFs

5.1.3 SFKM

Similar to the results of the PAM algorithm, the SFKM algorithm has also consistent indices for all GDF (Table 5.5). For these simulated data sets, although the *a* data set has the lowest value compared to the other three data sets, the SFKM algorithm produces robust results.

Table 5.5: Rand indices of the simulated data via the SFKM algorithm

Data	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
Data a	0.81	0.81	0.81	0.81	0.81	0.81	0.81
Data b	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Data c	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00

5.1.4 RKM

In the RKM algorithm, a hostility parameter has to be defined. For the simulated data sets, they are assigned as 10, 20, 30, 40, and 50. If the hostility parameter is defined as 20, for instance, the hostility is measured based on 20 closest objects. Table 5.6 shows that the rand indices of the RKM algorithm are varied for all of the data sets.

Table 5.6: Rand indices of the simulated data via the RKM algorithm

Hostility	Data	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
10	Data a	0.83	0.96	0.83	0.90	0.82	0.85	0.85
	Data b	1.00	0.88	0.85	0.68^b	0.91	0.82	0.92
	Data c	0.84	0.84^a	0.84	0.84	0.84	0.91	0.84
	Data d	0.79	0.85	0.78	0.87	0.83	0.84	0.85
20	Data a	0.86	0.81	0.82	0.77	0.84	0.86	0.93
	Data b	0.74	0.83	0.91	1.00^b	0.85	0.72	0.67
	Data c	0.79	0.85^a	0.84	0.86	0.86	0.85	0.84
	Data d	0.90	0.83	0.82	1.00	0.59	0.82	0.81
30	Data a	0.81	0.85	0.83	0.84	0.79	0.73	0.71
	Data b	0.88	0.83	0.87	0.85^b	0.85	0.80	0.83
	Data c	0.84	$0.85\ ^a$	0.84	0.78	0.83	0.84	0.87
	Data d	0.85	0.80	0.84	1.00	0.80	0.81	0.84
40	Data a	0.61	0.92	0.88	0.85	0.81	0.95	0.61
	Data b	0.90	0.84	0.67	1.00^b	0.80	0.98	0.90
	Data c	1.00	1.00^a	1.00	1.00	0.85	0.83	0.84
	Data d	0.83	1.00	0.83	0.81	0.85	0.88	0.90
50	Data a	0.74	0.84	0.78	0.72	0.97	0.77	0.77
	Data b	0.96	0.93	0.97	0.97^b	0.87	0.97	0.96
	Data c	0.76	0.61^a	0.78	0.78	0.79	0.88	0.76
	Data d	0.85	0.85	0.79	0.90	0.90	0.90	0.90

Examples of rand indices that vary in different hostility settings in:
^{a} Wishart and ^{b} Huang GDFs

5.1.5 INCKM

When the hostility parameter has to be specified in the RKM algorithm, the INCKM algorithm has an alpha parameter to determine a stretch factor. It measures how far the range of the medoid candidate is calculated. In these data sets, the stretch factors are defined as 1.1, 1.5, 2, 3, and 5. Table 5.7 shows that the INCKM algorithm produces varied rand indices. Although they are consistently high in the small stretch factor, i.e. 1.1, they varies after a particular stretch value. The Gower GDF, for instance, the rand indices vary when the stretch factor greater than 2, while, it is consistent in any stretch factor for the Esimma GDF.

Table 5.7: Rand indices of the simulated data via the INCKM algorithm

Stretch	Data	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
1.1	Data a	0.98	0.98	0.99	0.99	0.84	0.82	0.98
	Data b	1.00^a	1.00	1.00	1.00	1.00	1.00^b	1.00
	Data c	1.00	0.86	1.00	1.00	1.00	1.00	1.00
	Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1.5	Data a	0.82	0.98	0.99	0.99	0.82	0.81	0.82
	Data b	1.00^a	1.00	1.00	1.00	1.00	1.00^b	1.00
	Data c	0.83	0.84	1.00	1.00	0.83	1.00	0.83
	Data d	0.84	1.00	1.00	1.00	0.84	1.00	0.84
2	Data a	0.82	0.98	0.82	0.81	0.82	0.99	0.82
	Data b	0.84^a	0.86	1.00	1.00	0.83	1.00^b	0.84
	Data c	0.83	0.83	0.83	0.83	0.83	0.83	0.83
	Data d	0.84	0.84	0.84	1.00	0.84	0.84	0.84
3	Data a	0.82	0.98	0.82	0.83	0.82	0.81	0.82
	Data b	0.84^a	0.86	0.86	0.86	0.83	1.00^b	0.84
	Data c	0.83	0.83	0.83	0.83	0.83	0.83	0.83
	Data d	0.84	0.84	0.84	0.84	0.84	0.84	0.84
5	Data a	0.82	0.98	0.82	0.81	0.82	0.81	0.82
	Data b	0.84^a	0.85	0.86	0.86	0.83	1.00^b	0.84
	Data c	0.83	0.83	0.83	0.83	0.83	0.83	0.83
	Data d	0.84	0.84	0.84	0.84	0.84	0.84	0.84

Examples of rand indices that vary in different stretch factor settings in:
^{a} Gower and ^{b} Esimma GDFs

5.1.6 SKM

An important parameter in the SKM algorithm is s (seeding), which determines how many times an initialization process is repeated. For these data sets, five different seedings are applied, namely 10, 20, 40, 60, and 80. Table 5.8 shows that the rand indices for all GDF's and seedings have consistently high results. Similar to the PAM and SFKM algorihtms, the SKM algorithm performs well.

5.2 Artificial mixed variable data set

The Qiu and Joe (2006a) algorithm generates a numerical variable data set only. Then, in order to obtain a binary/ categorical variable, a numerical variable is categorized via a quantile categorization. The categorization of a numerical variable into a binary/ categorical variable is obtained by dividing the numerical variable into c-number of classes with $100/c$ % quantile as the cut point. Thus, a binary variable is achieved when a median of the numerical variable is applied as a cutoff point.

The artificial mixed variable data set consists of four different settings with separation value equal to 0.5. The variants include the variables proportion, number of clusters, number of variables, and number of objects. Due to the consistency results of the PAM, SFKM, and SKM algorithms (Section 5.1), these algorithm are included

Table 5.8: Rand indices of the simulated data via the SKM algorithm

Seeding	Data	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
10	Data a	0.98	0.98	0.99	0.99	0.99	0.99	0.98
	Data b	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data c	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00
20	Data a	0.98	0.98	0.99	0.99	0.99	0.99	0.98
	Data b	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data c	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00
40	Data a	0.98	0.98	0.99	0.99	0.99	0.99	0.98
	Data b	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data c	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00
60	Data a	0.98	0.98	0.99	0.99	0.99	0.99	0.98
	Data b	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data c	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00
80	Data a	0.98	0.98	0.99	0.99	0.99	0.99	0.98
	Data b	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data c	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Data d	1.00	1.00	1.00	1.00	1.00	1.00	1.00

in the mixed variable data set simulation. Although the INCKM algorithm is also consistent with a stretch factor of 1.1, it is impractical for the simulation settings because a suitable stretch factor has to be defined for each simulated data set, which can be different from a data set to another data set. Thus, each simulated data set is partitioned applying seven GDF's (Table 2.1 and 5.1) by the PAM, SFKM, and SKM algorithms. To compare the algorithm performance, the rand indices are calculated and averaged via a subsetting strategy (Hennig, 2007) in 50 replications. Then, they are presented in a table.

5.2.1 Different variable proportion

In the variable proportion experiments, the artificial mixed variable data have four different settings. They are dominated by numerical, binary, categorical with 3 classes of categories, and categorical with 5 classes of categories (Table 5.9). In all settings, the SFKM algorithm produces the lowest rand index compared to the PAM and SKM algorithms, except in the data set dominated by binary variables when the Huang, Harikumar, or Esimma GDFs are applied. Moreover, the all rand indices of the PAM and SKM are not statistically different at the 0.05 alpha level.

5.2.2 Different number of clusters

The numbers of clusters (k) generated are 3, 4, 8, 10. In any number of clusters, there are always rand indices of the SFKM algorithm that are lower than the other two

Table 5.9: Rand indices of the different proportion of variables

Domination	Algorithm	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
Numerical	PAM	0.87	0.92	0.84	0.92	0.95	0.94	0.92
	SFKM	0.80^a	0.84^a	0.81^a	0.90^a	0.92^a	0.91^a	0.85^a
	SKM	0.87	0.92	0.84	0.92	0.95	0.94	0.92
Binary	PAM	0.80	0.78	0.79	0.73	0.76	0.71	0.84
	SFKM	0.76^a	0.73^a	0.73^a	0.73	0.74	0.71	0.80^a
	SKM	0.79	0.76	0.79	0.73	0.75	0.71	0.84
Categorical	PAM	0.88	0.97	0.86	0.99	0.99	0.98	0.96
$c = 3$	SFKM	0.83^a	0.89^a	0.80^a	0.94^a	0.94^a	0.93^a	0.91^a
	SKM	0.89	0.98	0.87	0.99	0.99	0.98	0.96
Categorical	PAM	0.86	0.97	0.85	0.99	0.99	0.98	0.94
$c = 5$	SFKM	0.78^a	0.83^a	0.76^a	0.97^a	0.97^a	0.95^a	0.86^a
	SKM	0.85	0.97	0.85	0.99	0.99	0.98	0.94

Significantly different ($\alpha = 0.05$) to: ^{a} PAM and SKM

Table 5.10: Rand indices of the different number of clusters

k	Algorithm	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
3	PAM	0.98	0.99	0.98	0.95	0.96	0.95	1.00
	SFKM	0.96^a	0.94^a	0.85^a	0.94	0.95	0.94	0.93^a
	SKM	0.98	0.99	0.98	0.95	0.96	0.95	1.00
4	PAM	0.89	1.00	0.85	1.00	1.00	1.00	1.00
	SFKM	0.83^a	0.94^a	0.81^a	0.98^a	0.97^a	0.98^a	0.90^a
	SKM	0.89	1.00	0.85	1.00	1.00	1.00	1.00
8	PAM	0.88	0.94	0.87	0.94	0.94	0.94	0.94
	SFKM	0.85^a	0.91^a	0.85^a	0.94	0.94	0.94	0.91^a
	SKM	0.88	0.93	0.87	0.94	0.93	0.94	0.93
10	PAM	0.88	0.97	0.88	0.95	0.95	0.95	0.94
	SFKM	0.86^a	0.94^a	0.87^a	0.95	0.94^a	0.94	0.90^a
	SKM	0.88	0.96	0.87	0.95	0.95	0.95	0.93

Significantly different ($\alpha = 0.05$) to: ^{a} PAM and SKM

algorithms (Table 5.10). Similar to the simulation with different variable proportions, the SFKM algorithm is under performed compared to the other two algorithms.

5.2.3 Different number of variables

The algorithm of the simulated data generates four different numbers of variables. They are 6, 8, 10, and 14 mixed variables. Table 5.11 shows that the SFKM algorithm always performs the worst. As the number of variables increases, the rand index of the Gower GDF decreases. For the other six GDF's, on the other hand, the rand indices substantially decrease when the number of variables is more than 10 ($p > 10$).

Table 5.11: Rand indices of the different number of variables

p	Algorithm	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
6	PAM	0.94	0.99	0.94	0.99	0.99	0.99	0.94
	SFKM	0.91^a	0.93^a	0.88^a	0.98^a	0.95^a	0.96^a	0.89^a
	SKM	0.94	0.99	0.94	0.99	0.99	0.99	0.94
8	PAM	0.89	1.00	0.87	1.00	1.00	1.00	0.97
	SFKM	0.86^a	0.97^a	0.81^a	0.99	0.96^a	0.99^a	0.93^a
	SKM	0.89	1.00	0.87	1.00	1.00	1.00	0.97
10	PAM	0.83	0.98	0.78	1.00	1.00	1.00	0.99
	SFKM	0.76^a	0.88^a	0.71^a	0.96^a	0.94^a	0.97^a	0.88^a
	SKM	0.82	0.99	0.78	1.00	1.00	1.00	0.99
14	PAM	0.84	0.87	0.82	0.85	0.84	0.85	0.95
	SFKM	0.76^a	0.78^a	0.75^a	0.79^a	0.79^a	0.80^a	0.85^a
	SKM	0.84	0.87	0.82	0.85	0.83	0.85	0.95

Significantly different ($\alpha = 0.05$) to: ^{a} PAM and SKM

5.2.4 Different number of objects

Four different numbers of objects are generated from 50 to 500 objects. Similar to the other experiment settings, the rand index of the SFKM algorithm is always lower than the two other algorithms (Table 5.12). A rand index of the SFKM is only higher than two other algorithms when it has 500 objects in the Podani GDF.

Table 5.12: Rand indices of the different number of objects

n	Algorithm	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
50	PAM	0.88	1.00	0.83	1.00	1.00	1.00	0.99
	SFKM	0.81^a	0.88^a	0.77^a	0.95^a	0.90^a	0.93^a	0.86^a
	SKM	0.88	1.00	0.83	1.00	1.00	1.00	0.99
100	PAM	0.87	0.91	0.84	0.87	0.87	0.88	0.91
	SFKM	0.83^a	0.87^a	0.79^a	0.85	0.85^a	0.88	0.89^a
	SKM	0.87	0.91	0.84	0.86	0.88	0.88	0.91
200	PAM	0.87	1.00	0.86	1.00	1.00	1.00	0.97
	SFKM	0.85^a	0.95^a	0.80^a	0.96^a	0.93^a	0.98^a	0.91^a
	SKM	0.87	1.00	0.86	1.00	1.00	1.00	0.97
500	PAM	0.87	0.98	0.82	1.00	1.00	1.00	0.97
	SFKM	0.84^a	0.89^a	0.85^a	0.99^a	0.95^a	0.98^a	0.89^a
	SKM	0.88	0.98	0.83	1.00	1.00	1.00	0.97

Significantly different ($\alpha = 0.05$) to: ^{a} PAM and SKM

5.2.5 Algorithms bench marking

Due to the previous experiment results, which are favorable to the PAM and SKM algorithms, the bench marking (evaluation) is applied for the PAM and SKM algorithms only. The evaluations perform in a different number of clusters (k), objects (n), with a fixed seeding/ initialization ($s = 20$). The k's are 2, 4, 7, and 10 clusters, while the n's are 50, 200, 1000, and 3000 objects.

Figure 5.2 and 5.3 show a comparison between the PAM and SKM for $k = 2$ and $k = 4$, respectively. They illustrate a similar result where the PAM requires slightly more time (in microseconds) than the SKM, along with the increasing number of objects. When k equals to 7, moreover, the difference of the PAM and SKM is pronounced (in microseconds) for the number of objects n larger than 1000 (Figure 5.4).

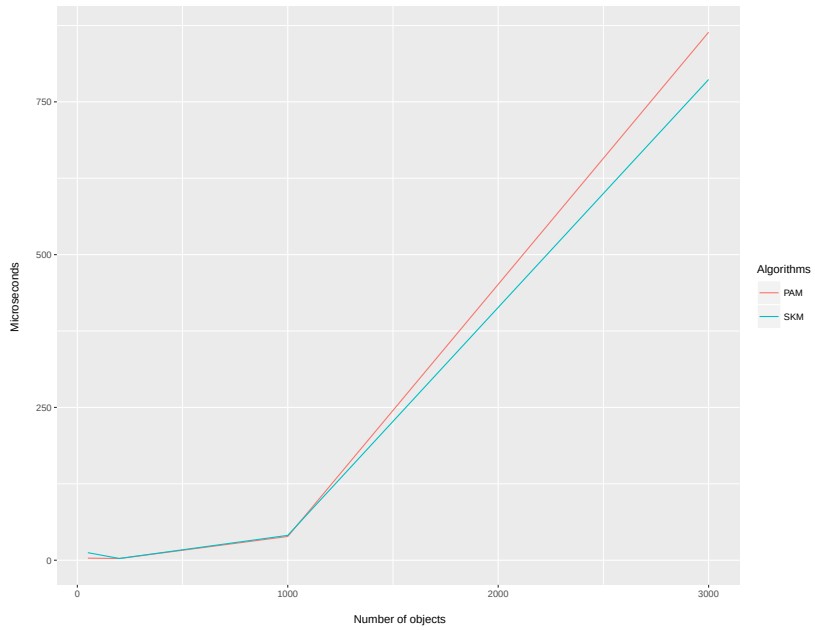


Figure 5.2: Benchmarking of the PAM and SKM ($k = 2$, $s = 20$)

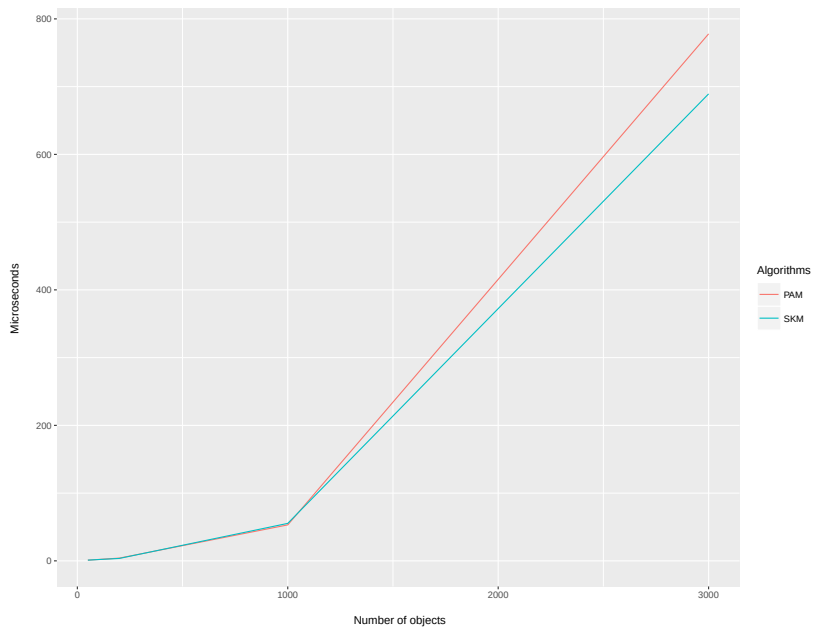


Figure 5.3: Benchmarking of the PAM and SKM ($k = 4$, $s = 20$)

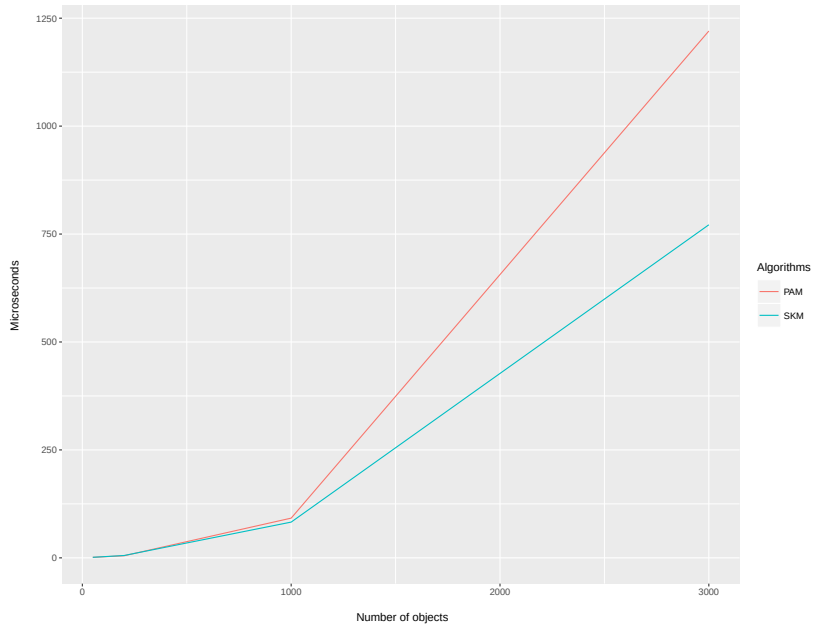


Figure 5.4: Benchmarking of the PAM and SKM ($k = 7$, $s = 20$)

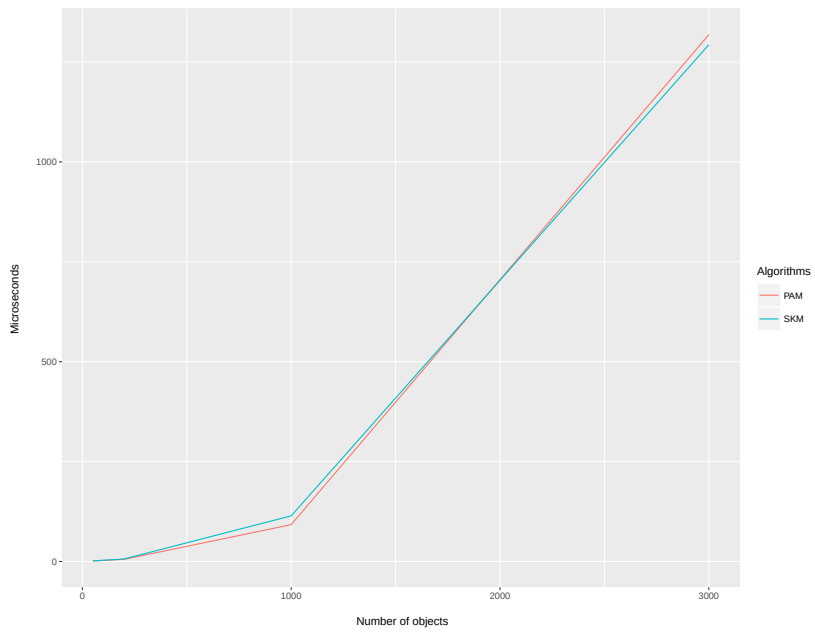


Figure 5.5: Benchmarking of the PAM and SKM ($k = 10$, $s = 20$)

If the number of clusters (k) equals to 10, on the other hand, the running time of the PAM and SKM is similar (Figure 5.5). In any number of objects, they require a similar amount of time to complete the partitioning task. Hence, when the number of clusters is small ($k < 10$) and the number of objects is large ($n > 1000$), the SKM algorithm (with $s = 20$) requires less time than the PAM algorithm.

5.3 Heart disease data set

The heart disease data set is a data set from the machine learning repository of UCI (Lichman, 2013). This data set has a grouping variable where 297 patients are grouped into two classes according to the presence or absence of heart disease. The data set is intended for a classification task, albeit we can apply this data to evaluate the medoid-based algorithm performance.

The heart disease data set is a mixed variable data set that consists of 13 variables, which are composed of six numerical, three binary, and four categorical variables. The 13 variables are age of the patients (numerical), sex (binary), type of chest pain (categorical), resting blood pressure (numerical), serum cholesterol (numerical), fasting blood sugar (binary), resting electrocardiographs (categorical), maximum heart rate (numerical), exercise included angina (binary), ST depression induced by exercise relative to rest (numerical), slope of the peak exercise ST segment (categorical), number of major vessel (numerical), heart rate (categorical).

The results of applying six medoid-based algorithms in seven GDFs show that the Gower, Podani, and Marweco GDFs have similar results where five algorithms produce a high accuracy rate (Table 5.13). It implies that when a weighted of range is applied in the numerical variables of the heart disease data set, the accuracy rate achieves the maximum value for this data set, i.e. 79-81%.

Table 5.13: Accuracy rate of the heart disease data

	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
PAM	**0.80**	0.74	**0.80**	0.59	0.57	0.56	**0.79**
KM	**0.81**	0.75	**0.80**	0.58	0.54	0.58	**0.79**
SFKM	0.59	0.75	**0.80**	0.59	0.56	0.56	**0.79**
RKM	**0.79**	0.64	0.57	0.58	0.68	0.59	0.57
INCKM	**0.80**	**0.79**	**0.80**	0.59	0.57	0.58	**0.79**
SKM	**0.80**	0.74	**0.80**	0.59	0.57	0.56	**0.79**

High values of the accuracy rate are achieved in the Gower, Podani, and Marweco GDFs. These GDFs apply a range weighted in their numerical variables.

5.4 Global food security data

The global food security data consist of 113 countries and 27 indicator variables from The Economist Intelligent Unit (2017). The 27 indicator variables are then summarized into four variables, namely affordability, availability, quality and safety, natural resources and resilience. All variables are scored by food-security panel specialists from 0 to 100 where 100 is the most favorable toward food security. Hence, the four food security variables are numeric.

The task for this data set is to group the countries based on their food security indices such that similar countries in terms of food security characteristics are composed in the same cluster. Due to all variables being numerical, a projection of the first two principal components are applied prior the application of clustering algorithms to explore the data structure. Figure 5.6 shows that the developed countries are in the upper right of the plot with favorable food security in all variables, while the developing countries show the opposite. The first exploration from the principle components plot indicates that the countries can be partitioned into two or three groups.

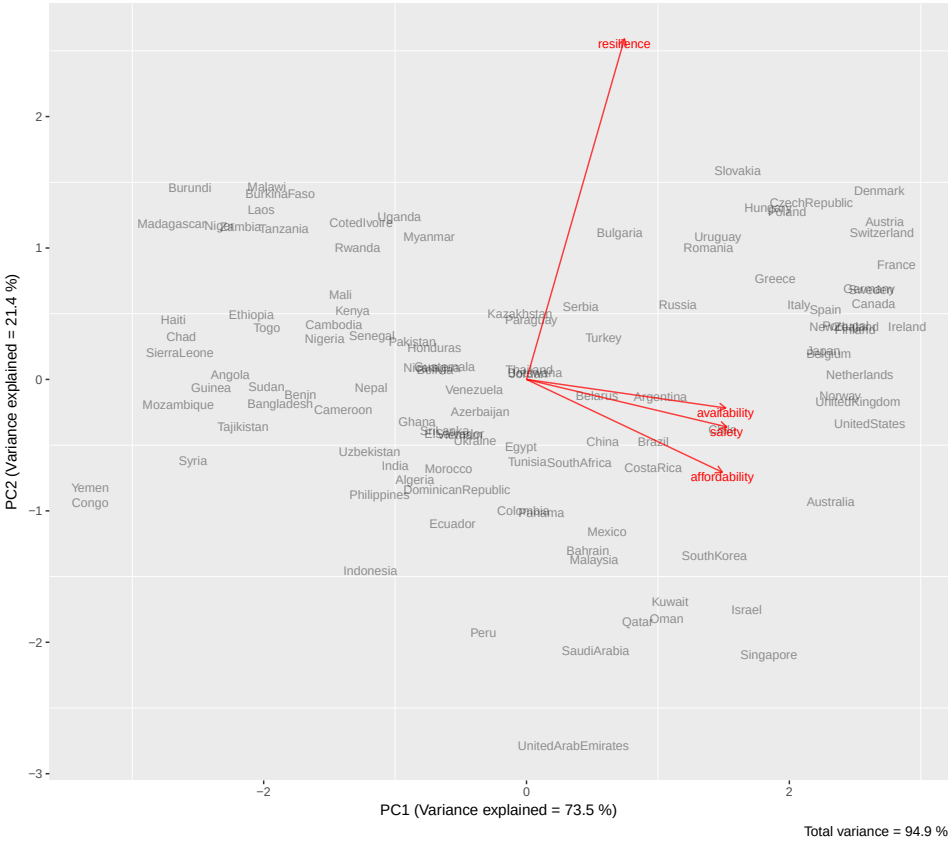


Figure 5.6: The first and second principal components plot of the global food security data

If external-based criteria are applied in the 50 bootstrap replicates, the means of rand index $k = 3$ is slightly better than that of $k = 2$ (Table 5.14 and 5.15). In the Huang GDF, which has the highest rand index in both $k = 2$ and $k = 3$, the SKM algorithm has the highest rand index. Thus, for further evaluation, only the SKM with $k = 2$ and $k = 3$ of the Huang GDF is evaluated.

Applying internal-based criteria visualization, in addition, the stripe plot of $k = 2$ and $k = 3$ shows that the separation of the clusters are similar, i.e. not-well separated (Figure 5.7 and 5.8). Combining relative-based and external-based criteria, the stability proportion of $k = 3$ and $k = 2$ is also similar (Figure 5.9 and 5.10). Hence, taken into account the similarity in the stripe plot and stability proportion, the number of clusters is 3 due to its higher rand index.

Table 5.14: Rand indices of the global food security data for $k = 2$

	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
PAM	**0.62**	0.61	0.61	**0.62**	0.61	0.61	0.60
KM	0.60	0.60	0.60	0.60	0.61	0.61	0.60
SFKM	**0.62**	0.59	**0.62**	**0.62**	**0.62**	**0.62**	0.59
RKM	0.57	0.57	0.58	0.58	0.56	0.57	0.57
INCKM	0.61	0.61	0.61	**0.62**	0.61	0.61	0.61
SKM	**0.62**	0.61	0.61	**0.62**	**0.62**	**0.62**	0.61

0.62 is the highest rand index

Table 5.15: Rand indices of the global food security data for $k = 3$

	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
PAM	0.66	0.66	0.66	0.66	**0.67**	0.66	0.66
KM	0.65	0.64	0.65	0.66	0.65	0.66	0.65
SFKM	0.65	0.65	0.66	0.66	0.66	0.66	0.64
RKM	0.62	0.62	0.62	0.63	0.63	0.62	0.62
INCKM	0.66	0.66	**0.67**	0.66	**0.67**	0.66	0.65
SKM	0.66	0.65	**0.67**	**0.67**	0.66	**0.67**	0.66

0.67 is the highest rand index

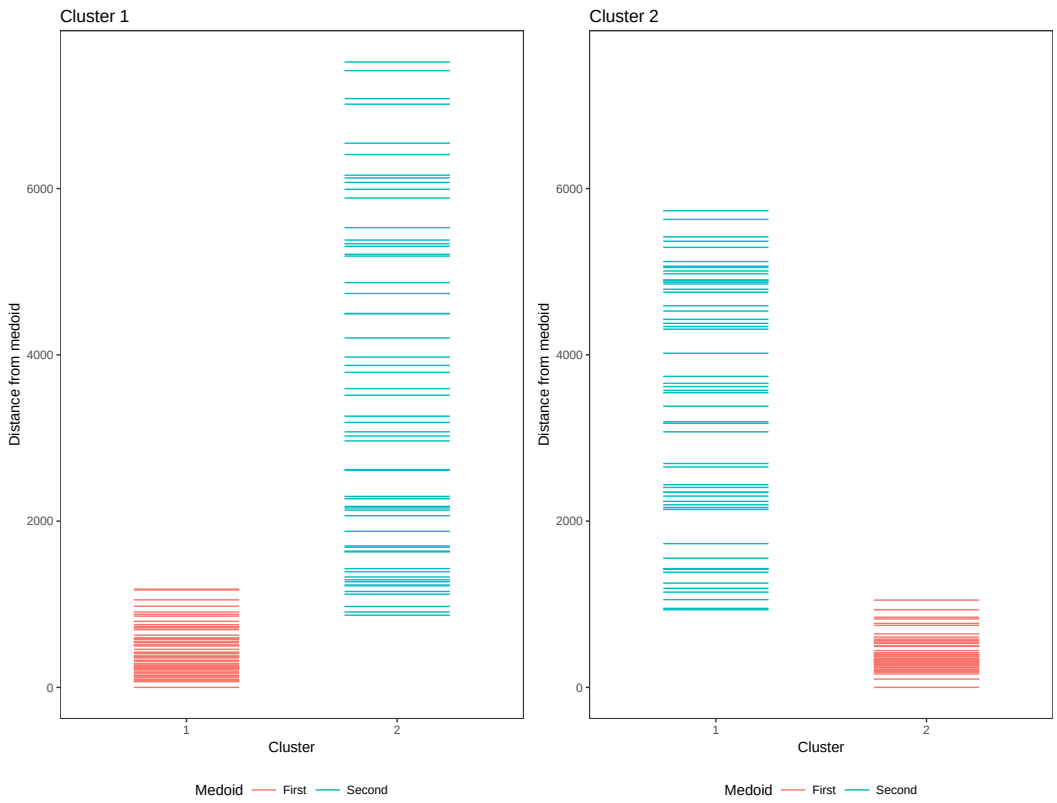


Figure 5.7: Stripe plot of the global food security data via the SKM algorithm with the Huang GDF ($k = 2$)

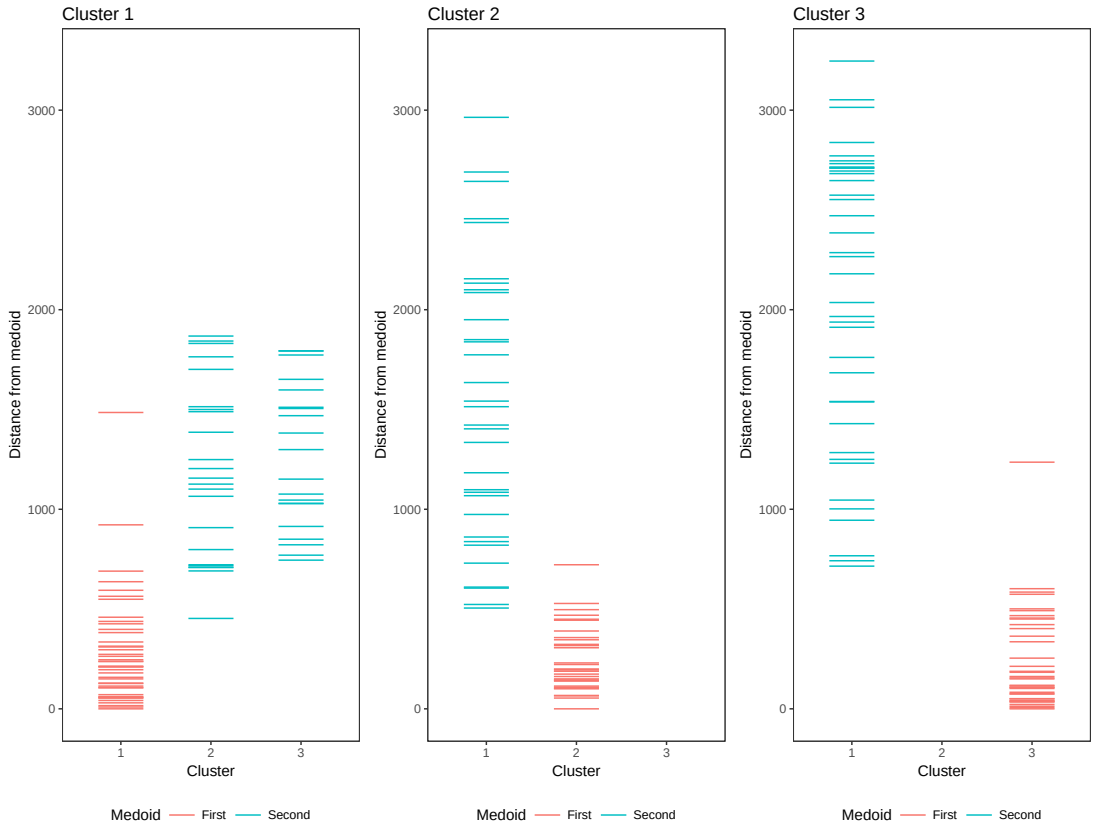


Figure 5.8: Stripe plot of the global food security data via the SKM algorithm with the Huang GDF ($k = 3$)

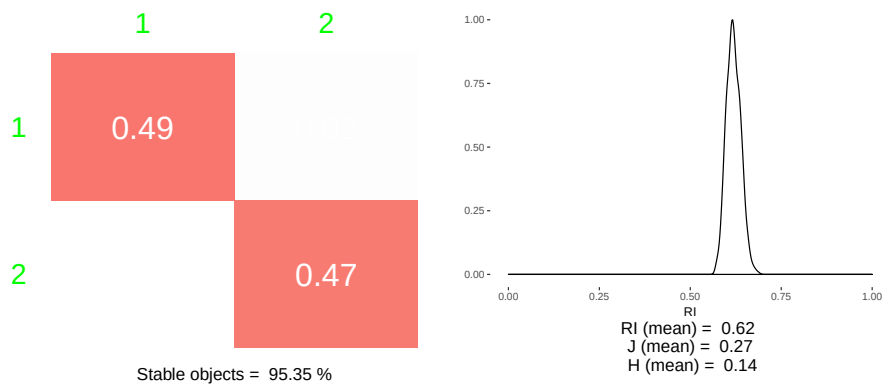


Figure 5.9: Combination plots of the global food security data via the SKM algorithm with the Huang GDF ($k = 2$)

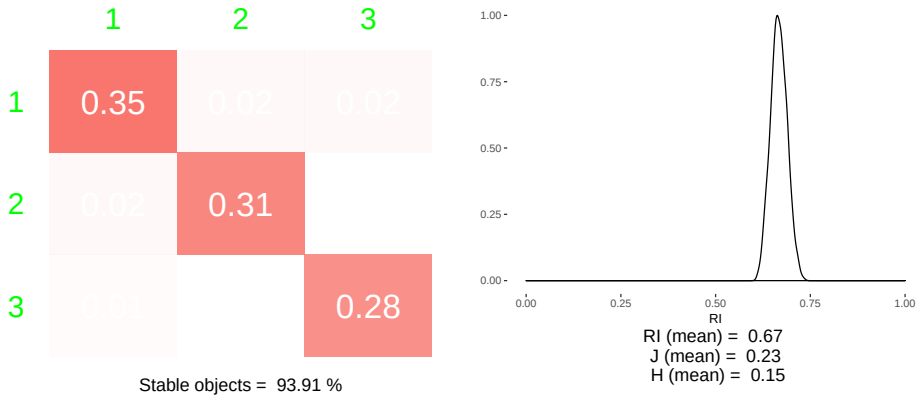


Figure 5.10: Combination plots of the global food security data via the SKM algorithm with the Huang GDF ($k = 3$)

5.5 Sponge data set

Like the heart disease data set, the marine sponge data set is a data set from the UCI machine learning repository. This data represent marine sponges consisting of 45 mixed variables, and 76 objects. Due to many missing values in the 39th variable, this variable is excluded. Upon the 39th variable exclusion, the number of numerical, binary, and categorical variables becomes 3, 15, and 26, respectively. Similar to the global food security data set, the task for the sponge data set is a clustering task because the class memberships of the sponges are absent.

Due to the small number of objects in the data set, and the high number of clusters that are examined, i.e. 2 to 15, the KM, SFKM, and INCKM algorithms are excluded. An empty cluster is possible to arise in those algorithms. The RKM algorithm is also excluded due to its sensitivity of the m-parameter to calculate the hostility index in this data set. Thus, the algorithms applied in this data set are the PAM and SKM algorithms.

The silhouette index as an internal criterion is applied to asses the number of clusters in the seven GDF's. The maximum value of the average silhouette value of each object is achieved in the Marweco GDF in the two clusters (Table 5.16). Table 5.16 shows that the average value of silhouette indices in any GDF decreases slightly. However, in the Esimma GDF, it increases when $k = 14$. With the internal criterion, it has still uninformative measure.

Table 5.16: Average value of silhouette indices of the sponge data set

k	Algorithm	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
2	PAM	0.46	0.30	0.28	**0.54**	**0.54**	**0.50**	**0.56**
	SKM	0.46	0.30	0.28	**0.54**	**0.53**	**0.50**	**0.56**
3	PAM	0.35	0.22	0.21	0.41	0.45	0.41	0.42
	SKM	0.35	0.22	0.20	0.41	0.45	0.42	0.42
4	PAM	0.25	0.16	0.15	0.36	0.36	0.37	0.34
	SKM	0.26	0.16	0.15	0.28	0.31	0.34	0.34
5	PAM	0.22	0.13	0.17	0.21	0.22	0.37	0.36
	SKM	0.22	0.13	0.13	0.21	0.22	0.37	0.36
6	PAM	0.23	0.15	0.16	0.22	0.24	0.35	0.37
	SKM	0.23	0.12	0.14	0.31	0.19	0.36	0.27
7	PAM	0.25	0.16	0.16	0.21	0.26	0.36	0.28
	SKM	0.21	0.15	0.16	0.17	0.25	0.36	0.28
8	PAM	0.29	0.17	0.17	0.23	0.25	0.36	0.30
	SKM	0.28	0.15	0.15	0.19	0.25	0.40	0.27
9	PAM	0.31	0.15	0.19	0.25	0.28	0.38	0.32
	SKM	0.25	0.15	0.13	0.23	0.32	0.36	0.33
10	PAM	0.29	0.17	0.18	0.28	0.26	0.40	0.33
	SKM	0.22	0.14	0.18	0.21	0.22	0.38	0.28
11	PAM	0.29	0.18	0.20	0.31	0.27	0.43	0.35
	SKM	0.20	0.16	0.19	0.21	0.24	0.42	0.24
12	PAM	0.32	0.19	0.19	0.30	0.28	0.45	0.33
	SKM	0.31	0.17	0.21	0.24	0.26	0.42	0.31
13	PAM	0.32	0.20	0.20	0.30	0.29	0.46	0.35
	SKM	0.24	0.16	0.16	0.24	0.26	0.37	0.28
14	PAM	0.30	0.22	0.20	0.32	0.30	**0.50**	0.33
	SKM	0.27	0.14	0.18	0.24	0.29	0.47	0.24
15	PAM	0.30	0.22	0.20	0.32	0.33	**0.53**	0.36
	SKM	0.27	0.18	0.18	0.23	0.31	0.36	0.24

Candidates of k are selected when the average value is at least 0.50

To add information for deciding the number of clusters in the marine sponge data set, a relative criterion via bootstrap samples is taken into consideration as well. Table 5.17 shows that the stability proportions are equal to or larger than 80% when the number of clusters is 2 to 7, 13, 14, and 15. Hence, based on the nature of the marine sponge species and relative criteria evaluation, the simplest and most suitable number of clusters is 7, when applying the Esimma GDF and PAM algorithm.

Figure 5.11 shows a directed graph of the sponge data set. Cluster 3 and 5 are closer to each other than to the other clusters, while cluster 6 is also close to cluster 5. The both clusters have only 1 unstable object. The most distinctive clusters are cluster 1, 2, 4, and 7.

Table 5.17: Stability proportion of the sponge data set

k	Algorithm	Gower	Wishart	Podani	Huang	Harikumar	Esimma	Marweco
2	PAM	**1.00**	**0.97**	**1.00**	**0.99**	**0.98**	**0.99**	**0.99**
	SKM	**1.00**	**0.97**	**0.99**	**0.99**	**0.97**	**0.99**	**0.99**
3	PAM	**0.87**	0.75	0.79	**0.85**	**0.92**	0.71	0.78
	SKM	**0.84**	0.78	0.79	**0.89**	**0.93**	**0.84**	0.78
4	PAM	0.73	0.73	0.77	0.63	0.72	**0.85**	**0.88**
	SKM	0.78	**0.80**	**0.80**	0.68	0.73	**0.86**	**0.93**
5	PAM	0.72	0.70	0.78	0.70	0.75	0.76	**0.87**
	SKM	0.70	0.67	0.72	0.66	0.67	0.79	**0.85**
6	PAM	0.73	0.67	0.71	0.70	0.70	0.79	**0.84**
	SKM	0.68	0.58	0.64	0.65	0.59	0.74	0.78
7	PAM	0.76	0.61	0.71	0.64	0.67	**0.86**	0.77
	SKM	0.66	0.56	0.64	0.62	0.64	0.73	0.68
8	PAM	0.70	0.60	0.70	0.65	0.66	0.78	0.72
	SKM	0.65	0.54	0.59	0.56	0.58	0.73	0.67
9	PAM	0.67	0.62	0.65	0.66	0.64	0.77	0.69
	SKM	0.62	0.56	0.57	0.55	0.58	0.70	0.62
10	PAM	0.64	0.63	0.59	0.68	0.62	0.78	0.72
	SKM	0.55	0.51	0.59	0.47	0.53	0.68	0.60
11	PAM	0.66	0.65	0.58	0.65	0.59	0.76	0.71
	SKM	0.56	0.53	0.55	0.49	0.53	0.67	0.61
12	PAM	0.67	0.67	0.65	0.61	0.64	0.79	0.72
	SKM	0.50	0.54	0.55	0.53	0.56	0.66	0.54
13	PAM	0.71	0.63	0.66	0.61	0.62	**0.80**	0.74
	SKM	0.55	0.56	0.58	0.52	0.54	0.65	0.58
14	PAM	0.67	0.64	0.65	0.66	0.65	**0.80**	0.73
	SKM	0.53	0.55	0.58	0.51	0.53	0.61	0.55
15	PAM	0.67	0.67	0.67	0.65	0.64	**0.84**	0.76
	SKM	0.54	0.54	0.54	0.50	0.53	0.61	0.54

Candidates of k are selected when the stability proportion is equal to or larger than 0.80

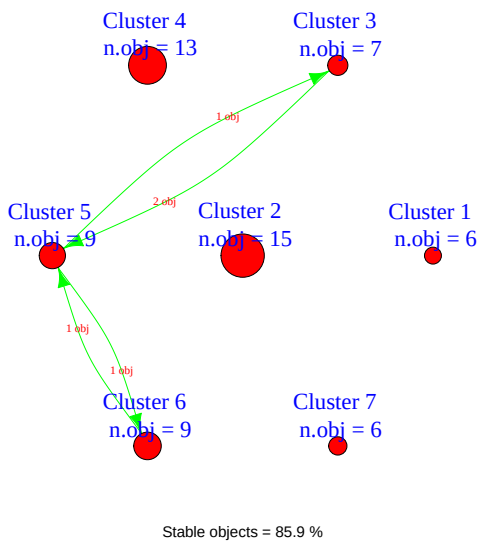


Figure 5.11: Directed graph of sponge data set via the PAM algorithm with the Esimma GDF ($k = 7$)

6 Application on Food Security Mapping

Due to the lack of food security mapping at the district level, this chapter discusses an application of a medoid-based algorithm via both the GSDF and GDF in food security mapping. The mapping is based on the food security zoning data set of Banten Province, Indonesia, where it is a mixed variable data set. The analysis varies the GSDF and GDF, yet fixes the SKM algorithm due to its greater efficiency than the PAM algorithm (Section 5.2.5).

The food security zoning data set has six numerical, two binary and two categorical variables. The data consist of 154 districts in the Banten Province, Indonesia. All ten variables were taken from the village potency data (PODES) collected by the Statistical Bureau of Indonesia. They represent the indicator variables of the food security dimensions that were commonly endorsed by FAO amplifying the food availability, accessibility, utilization, and stability aspects (van Dijk and Meijerink, 2014; Coates, 2013; FAO, 2006).

Table 6.1: Variables in the food security data set

variable	type	definition (computation)
nccons[a]	numerical	normative cereal consumption (cereal production/ n persons)
poor[b]	numerical	poor households ratio (n poor households/ n households)
electricity[b]	numerical	no electricity households (n households without electricity/ n households)
illiterate[c]	numerical	literacy activities (literacy activities within district/ all literacy activities)
water[c]	binary	potable water (1 is absent, 0 otherwise)
health[c]	categorical	nearest health service (3 categories: 1 (< 5 km), 2 (5-10 km), and 3 (> 10 km))
disaster[d]	numerical	drought accidents (drought accidents within district/ all drought accidents)
convert[d]	binary	land conversion (1 is present, 0 otherwise)
market[b]	categorical	nearest market (3 categories: 1 (< 10 km), 2 (10-20 km), and 3 (> 20 km))
industry[c]	numerical	food and beverages small business (n food and beverages small business within district/ all food and beverages small business)

[a]Availability indicator, [b]Accessibility indicator,
[c]Utilization indicator, [d]Stability indicator

The food availability is indicated by the normative cereal consumption per capita. The second dimension, the food accessibility is represented by the poor household ratio, ratio of households that do not have any access to the electricity and to the market facility. Next, the illiteracy activities, absence of potable water, health facility and small business of the food and beverages denote the food utilization. Last, the occurrence of drought and presence of land conversion indicate the food stability dimension. Table 6.1 summarizes the indicator variables in the food security zoning data.

6.1 Summary statistics

The summary statistics of food security zoning data are presented in Table 6.2. The normative cereal consumption per capita as the availability indicator has a mean 0.354 ton/ year. It is larger than its baseline for insecure food availability conditions, i.e. 0.11 ton/ year (FSC and WFP, 2015). It indicates the districts' food availability is guaranteed.

Table 6.2: Summary statistics of the indicator variables

Indicator	Variable	min	mean	max
Availability	nccons	0.000	0.354	1.600
Accessibility	poor	0.027	0.339	0.718
	electricity	0.000	0.016	0.252
		nearby	middle-range	far-away
	market	0.539	0.260	0.201
		min	mean	max
Utilization	illiterate	0.000	0.006	0.016
	industry	0.000	0.007	0.142
		nearby	middle-range	far-away
	health	0.532	0.351	0.117
		present	absent	
	water	0.909	0.091	
Stability	convert	0.740	0.260	
		min	mean	max
	disaster	0.002	0.006	0.015

In the accessibility indicator variables, the ratio of poor households is 34%, the household without electricity access is 1.6%, and most districts (54%) have easy access to the closest market. The main concern in the accessibility indicators is the number

of poor households where one third of the districts are poor and there is a district with a huge poverty rate (71.8%).

For the indicator variables of utilization, each district organizes only one literacy activity ($0.0064 * 154 = 0.92$) and has one food and beverages small business on average. A lot of districts have nearby health facility locations (53.2%) and 140 out of 154 (91%) have potable water access. These four utilization indicator variables suggest that most districts are able to utilize food properly.

Last, for the stability indicator variables, it is about 74% of the districts that have a land conversion activity and each district has only one disaster (drought accident) on average. Although the Banten Province is a new growing province, land conversion activities, especially arable land conversion into either settlement or industry, threat food stability. The stability indicator variables indicate that the land conversion threat is higher than the risk of natural disasters.

6.2 Cluster analysis

The first problem encountered before applying a cluster analysis is using either original or standardized numerical variables to calculate a distance. In the food security zoning data, the numerical variables are unstandardized. The reason in using unstandardized variables is that the numerical variables are obtained from a proportional calculation so that they are comparable to one another. An unstandardized variable is preferred for the real data interpretation as well. After deciding to apply the unstandardized numerical variables, the next step is to calculate the distance between districts.

6.2.1 GSDF and SKM

Due to the administrative constraint and the geographical boundary, the distance between districts is computed by the GSDF (Table 2.2). Then, the SKM algorithm is applied in the GSDF.

To obtain the number of clusters, a bootstrap cluster evaluation in various numbers of clusters ($k = 2, 3, \ldots, 9$) of all five GSDF is applied. Table 6.3 shows that the all indices are similar in each GSDF. The cluster stability proportion, on the other hand, indicates that the Huang GSDF has the most stable objects in $k = 2$ and 3 (Table 6.4). In four clusters, moreover, the Wishart GSDF has a high stability proportion, i.e. 90%. For further analysis, only two, and three clusters in the Huang GSDF and four clusters in the Wishart GSDF are compared via their both kernel density and cluster consensus matrix heatmap.

Figure 6.1 shows that the kernel density of rand index (right) for two clusters has only a peak, which is good, and its stability proportion (the sum of the diagonal values of the heatmap) is high (94%). Moreover, with three clusters (Figure 6.2), its rand index kernel density has a peak and the stability proportion is relatively good (82%).

Table 6.3: Cluster indices summary of all GSDF

GSDF	ri	jaccard	hubert	k^a	k^b	k^c
Gower	0.72	0.27	0.11	9	2	2
Wishart	0.72	0.25	0.13	8	2	4
Podani	0.72	0.25	0.10	9	2	3
Huang	0.71	0.27	0.14	9	2	2
Harikumar-PV	0.72	0.25	0.08	9	2	2

k is the corresponding number of clusters from the ^{a} rand, ^{b} jaccard, and ^{c} hubert indices in the left columns

Table 6.4: Cluster stability proportion of all GSDF

GSDF	2	3	4	5
Gower	**0.90**	0.78	0.68	0.53
Wishart	**0.85**	0.75	**0.90**	0.79
Podani	**0.86**	0.79	0.62	0.56
Huang	**0.94**	**0.82**	0.69	0.55
Harikumar-PV	**0.81**	0.71	0.67	0.55

Candidates of k are selected when the stability proportion is at least 0.80. $k = 2$ and 3 in the Huang GDF and $k = 4$ in the Wishart GDF are preferred due to its maximum value.

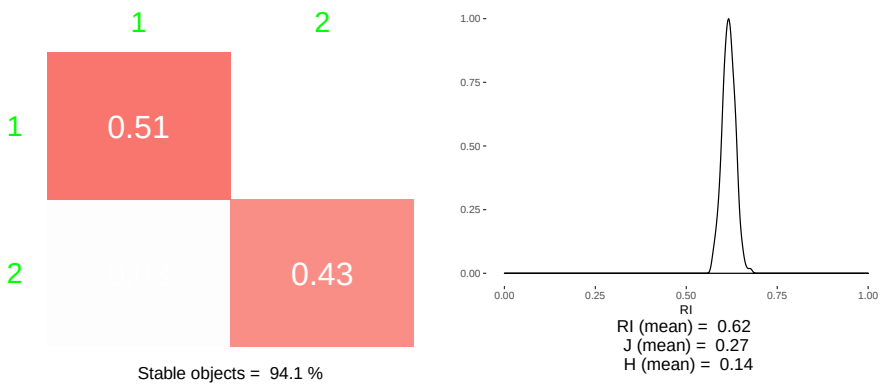


Figure 6.1: The cluster consensus heatmap (left) and kernel density (right) of the Huang GSDF for two clusters

The four clusters of the Wishart GSDF produce also a high stability proportion (90%) and a peak figure in the kernel density (Figure 6.3). Considering the evaluation using stability proportion, kernel density, and technical application, four clusters applying the Wishart GSDF are selected.

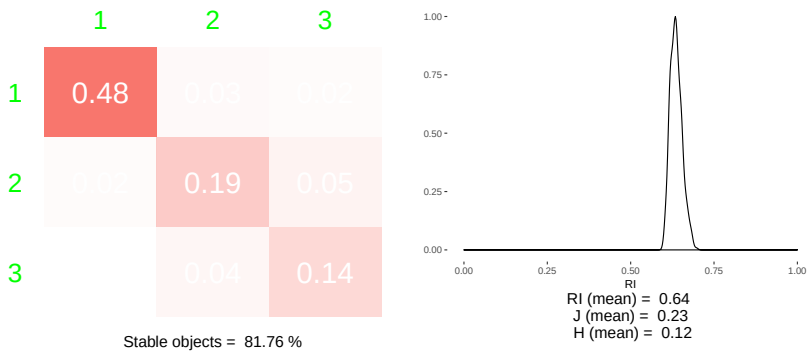


Figure 6.2: The cluster consensus heatmap (left) and kernel density (right) of the Huang GSDF for three clusters

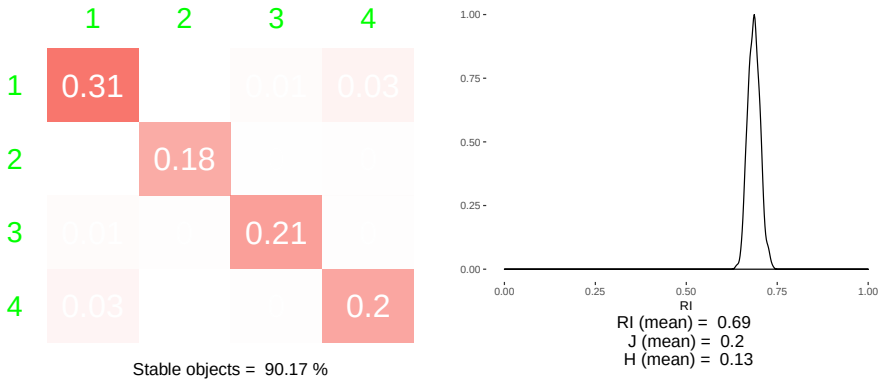


Figure 6.3: The cluster consensus heatmap (left) and kernel density (right) of the Wishart GSDF for four clusters

6.2.2 GDF and SKM

The clustering result by applying the GSDF and SKM is compared to the GDF and SKM in which spatial constraints are excluded. Table 6.5 shows that all indices in the GDF are similar to that of the GSDF in Table 6.3. Compared to the stability proportions of the GSDF (Table 6.4), however, the stability proportions of the GDF in the two to four clusters are the best in the Gower, Harikumar-PV and Wishart GDF, respectively (Table 6.6).

Table 6.5: Cluster indices summary of all GDF

GDF	ri	jaccard	hubert	k^a	k^b	k^c
Gower	0.72	0.28	0.15	9	2	2
Wishart	0.72	0.26	0.11	9	2	2
Podani	0.72	0.27	0.13	9	2	2
Huang	0.71	0.28	0.15	9	2	2
Harikumar-PV	0.72	0.26	0.15	9	2	3

k is the corresponding number of clusters from the ^{a} rand, ^{b} jaccard, and ^{c} hubert indices in the left columns

Table 6.6: Cluster stability proportion of all GDF

GDF	2	3	4	5
Gower	**0.96**	0.74	0.63	0.67
Wishart	**0.89**	0.77	**0.80**	0.63
Podani	**0.93**	0.71	0.73	0.66
Huang	**0.94**	**0.85**	**0.80**	0.69
Harikumar-PV	0.78	**0.93**	0.79	0.75

Candidates of k are selected when the stability proportion is at least 0.80. $k = 2$ in the Huang GDF, $k = 3$ in the Harikumar-PV GDF, and $k = 4$ in the Wishart GDF are preferred due to its maximum value.

For the cluster consensus heatmap, high proportions of the stable objects are indicated by an apparent diagonal heatmap in both the Gower and Harikumar-PV GDF's (Figure 6.4 (left) and 6.5 (left)). They also have smooth kernel density consisting only a peak (Figure 6.4 (right) and 6.5 (right)). Moreover, the Wishart GDF with four clusters has a similar figure (Figure 6.6) thought it has a lower stability proportion, i.e 80%. With respect to the stability proportion, kernel density, and technical application, four clusters are selected. For the reason of the technical application, especially for the extension agents, a visualization of cluster distributions is evaluated spatially between GSDF and GDF.

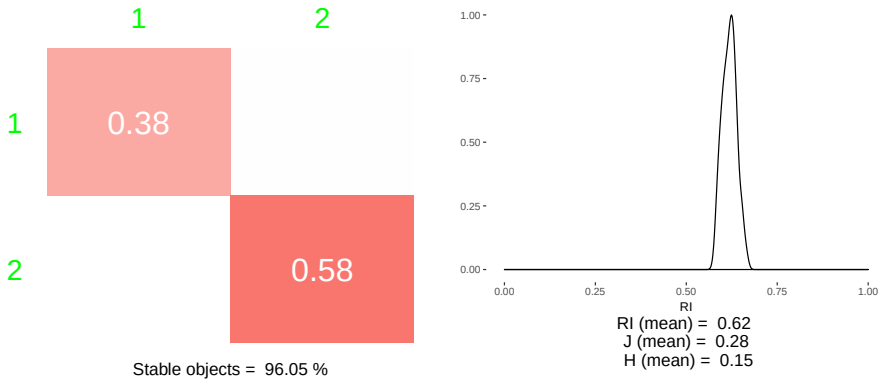


Figure 6.4: The cluster consensus heatmap (left) and kernel density (right) of the Gower GDF for two clusters

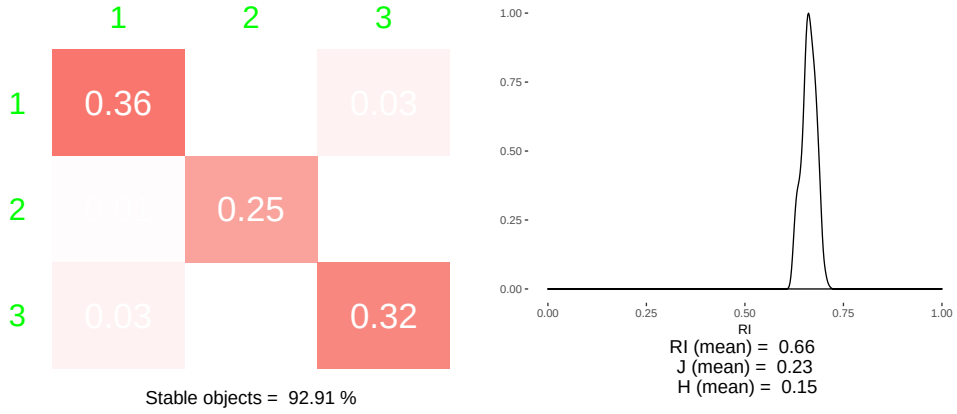


Figure 6.5: The cluster consensus heatmap (left) and kernel density (right) of the Harikumar-PV GDF for three clusters

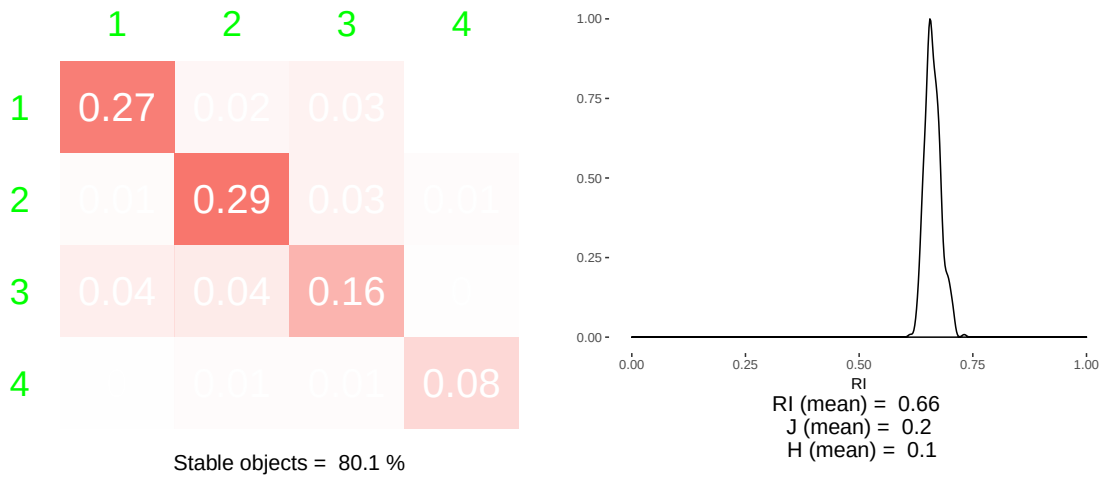


Figure 6.6: The cluster consensus heatmap (left) and kernel density (right) of the Wishart GDF for four clusters

6.3 Visualization

The distributions of clusters via both the GSDF and GDF are mapped in Figure 6.7 and 6.8, respectively. The GDF map is more spread than GSDF, as expected. Thus, for further analysis, four clusters of the Wishart GSDF is selected.

6.3.1 Barplot

In order to interpret the cluster result of numerical variable data set, Leisch (2008) has proposed a barplot with the location and dispersion of each clusters. This barplot can also be added by some markers such that a marked barplot is produced (Dolnicar and Leisch, 2014). The marker tags the means of the population and within cluster. If the data are ordinal variables data set, Brentari et al. (2016) have performed a rank-based boxplot for the location and dispersion of the clusters.

Due to a mixed variable data set, the barplot is modified to assist the food security data set interpretation. Two rules are added in the modified barplot. First, the numerical variables are re-scaled such that the minimum and maximum values are 0 and 1, respectively. Then, a proportion measure of each category applies for the binary and categorical/ ordinal variables.

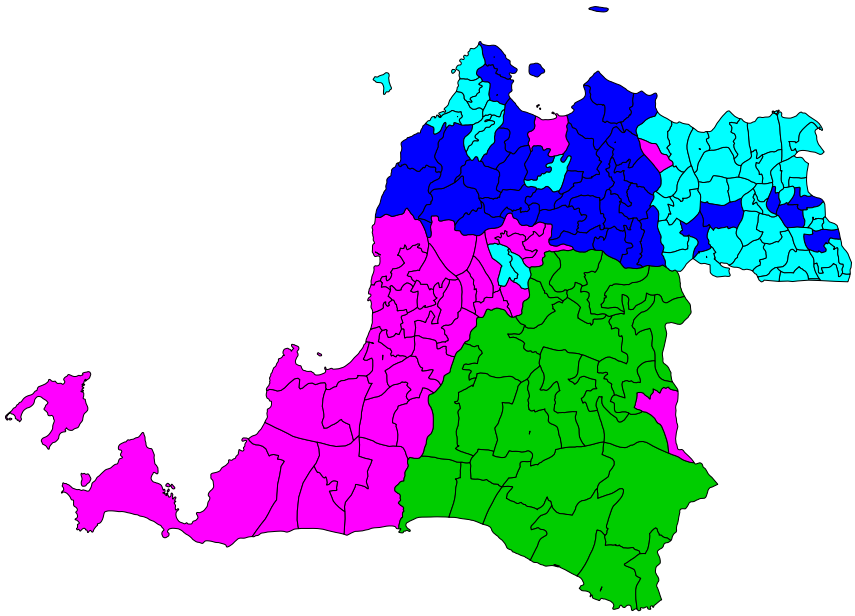


Figure 6.7: The clusters distribution via the GSDF

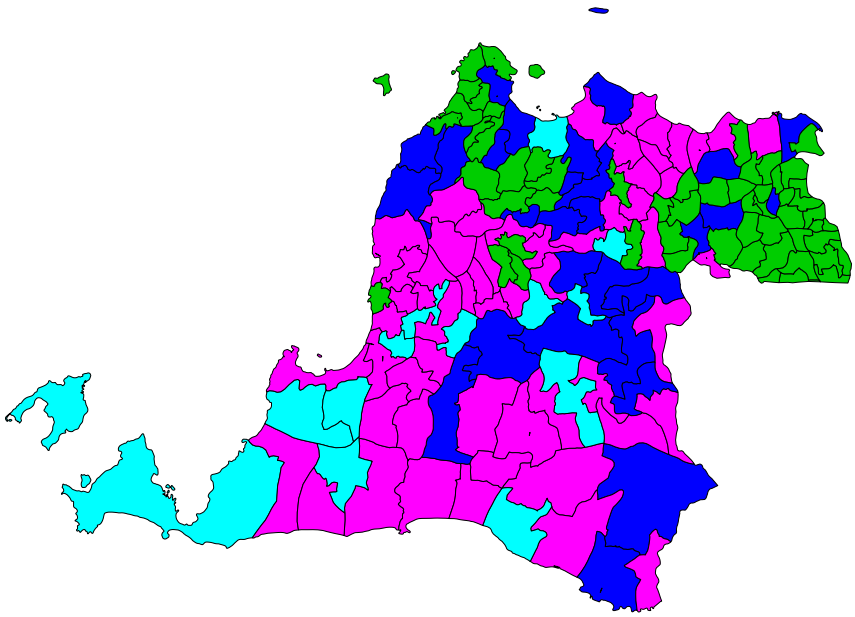


Figure 6.8: The clusters distribution via the GDF

Figure 6.9 shows the modified barplot of the food security data set for four clusters. The population means (numerical variables) are indicated by dots, while the population proportion (binary and categorical variables) are denoted by triangles. If a variable has a triangle, it is a binary/ categorical variable, which has two classes of categories. Moreover, when the number of triangles are two, the variables are categorical variables with three categories.

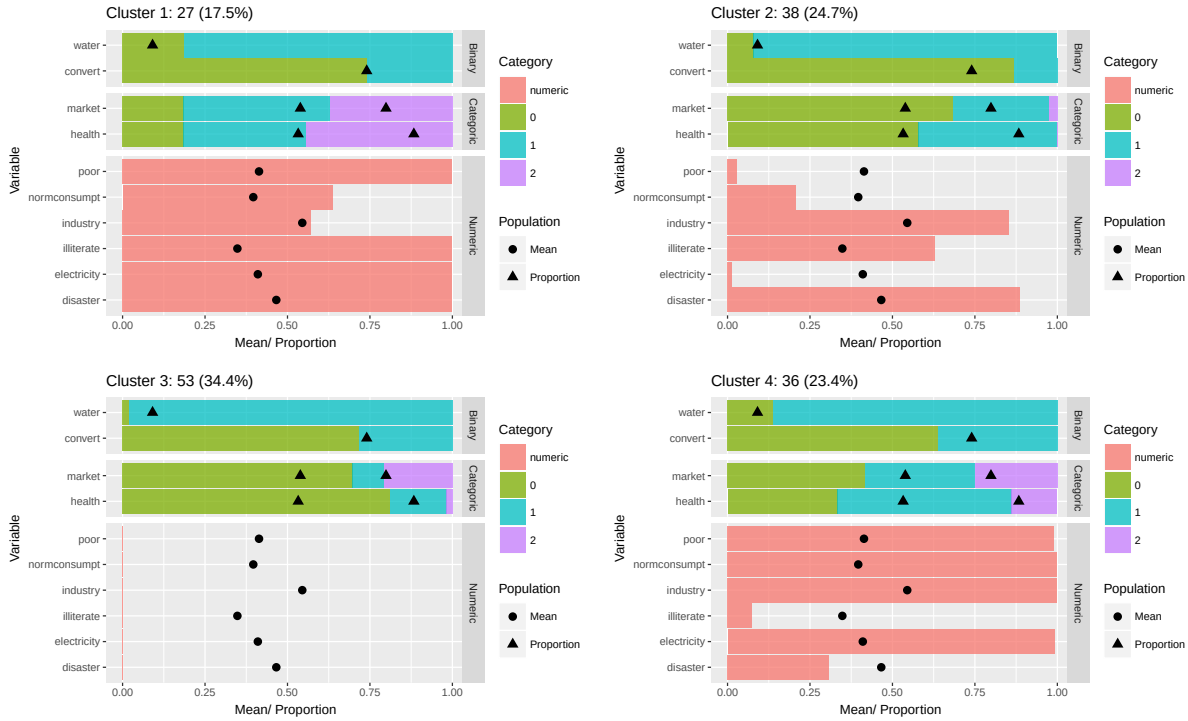


Figure 6.9: The modified barplot of food security data (mixed variables) for four clusters

Cluster 1 (17.5%): available. It has higher normative consumption than the population means meaning that its food availability is assured. However, the number of the natural disasters, the ratio of the poor household, and the proportion of the household without electricity are higher than that of the population. Moreover, the access to the market and health facility is limited. Thus, cluster 1 is only strong in the available dimension.

Cluster 2 (24.7%): utilize-accessible. It has a good indicator of food utilization due to a high number of small business of the food and beverages and illiteracy activities. The proportions of the poor household and household without electricity, which assign as the food accessibility indicator variables, are lower than that of the population. They indicate that the food accessibility is guaranteed. However, its food stability is fragile because the number of natural disasters and land conversions is high. Hence, it is a cluster with the specialty in the food utilization and accessibility dimension.

Cluster 3 (34.4%): accessible. Similar to cluster 2, the food accessibility is the strength of this cluster. With a low ratio of poor household and household without

electricity, the accessibility is assured. Although it performs poorly in the availability, stability, and utilization aspects, due to possessing the best market facility, this cluster becomes the best cluster of accessibility towards food.

Cluster 4 (23.4%): available-stable future. Like cluster 1, cluster 4 guarantees the food availability. In addition, the access to the market and health facility is limited. The number of natural disasters and land conversion in this cluster, on the other hand, is less than that of the population. It indicates the future stability of food. Thus, the food availability and stability is the strength of this cluster.

6.3.2 Stripes plot and directed graph

While a modified barplot (Figure 6.9) describes each variable, to visualize the cluster structures, both the internal and relative criteria based validation can be applied. A stripe plot of centroid-based clustering, which adopts the shadow values, has been proposed by Leisch (2008) where it applies the first and second closest centroids. Adapted from the stripes plot, a modified stripe plot has been developed by replacing the centroids into medoids context. Figure 6.10 shows the modified stripes plot of the food security data in four clusters. From the internal criterion, all clusters are not well-separated. They are indicated by overlapped lines between the first and second closest medoids.

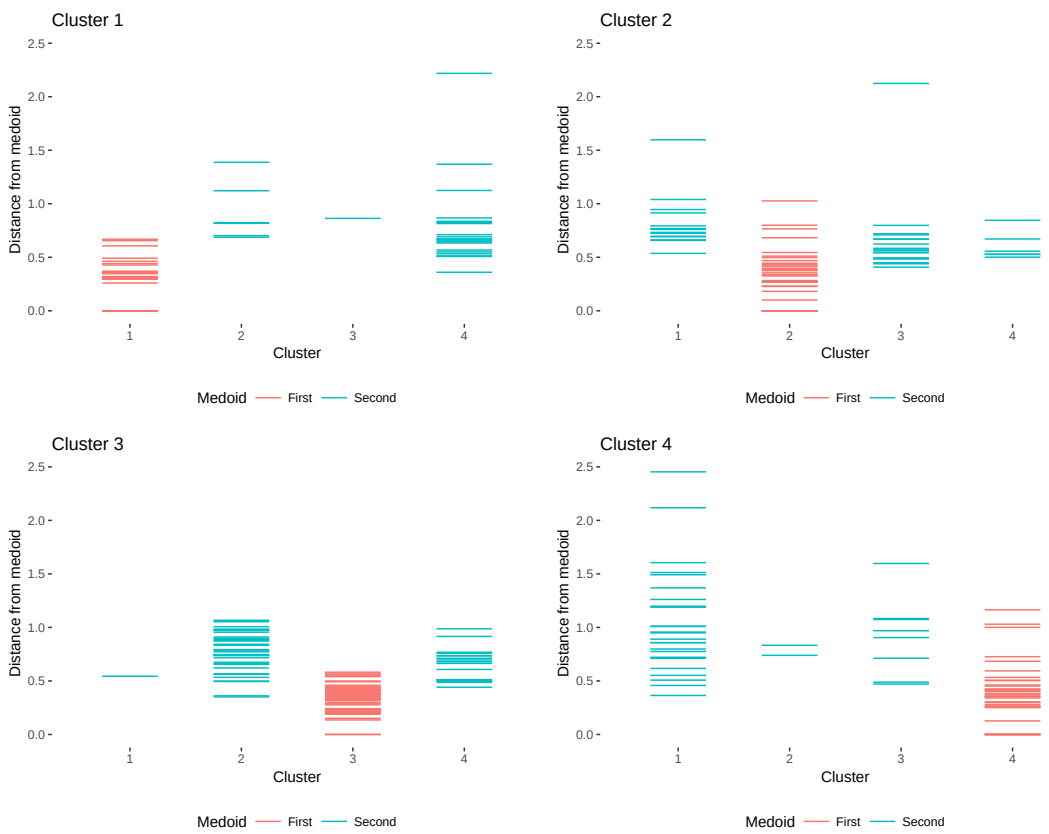


Figure 6.10: The modified stripes plot of food security data (mixed variables) for four clusters

On the other hand, a relative-based criteria validation produces a directed graph of the food security data (Figure 6.11). It shows that cluster 2 (*utilize-accessible* cluster) is the most stable cluster, i.e. 28 districts (18%). However, cluster 1 and 4 have eight unstable districts where four districts tends to move from cluster 1 (*available*) to cluster 4 (*available-stable future*) and four districts do the other away directions. It implies that four districts are able to improve their food stability in the future, while other four districts have their future food stability status at risk due to the shift into cluster 1. Cluster 1 (*available*) and cluster 3 (*accessible*), moreover, have four unstable districts. Two districts have a risk to loss of food production, while the other two districts gain more access towards food.

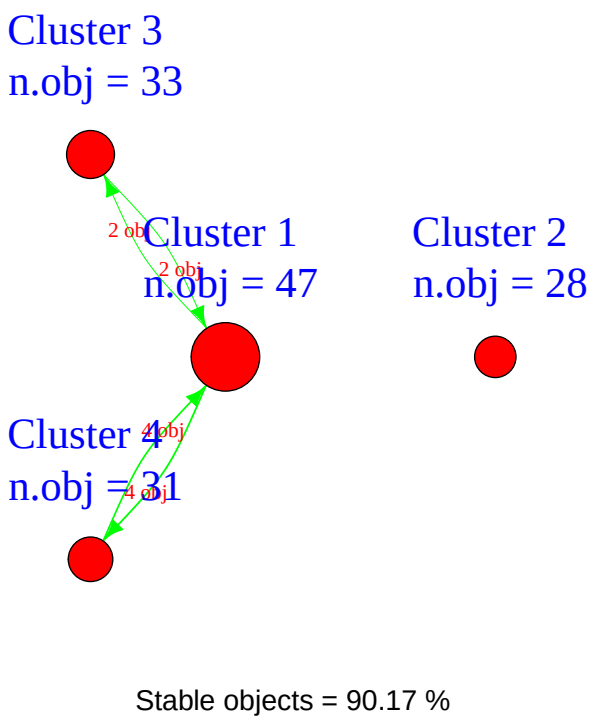


Figure 6.11: The directed graph of food security data (mixed variables) for four clusters

6.3.3 Food security mapping

Combining the cluster results and possible cluster specific activities for the AE agents to focus on, a summary is drawn in a food security map. In cluster 1, AE agents should concentrate on two activities, namely the prevention of erosion/ landslide in the arable land to improve the food stability and the community development activities to reduce poor households. AE agents in the cluster 2 can focus only on the erosion/ landslide prevention, while in the cluster 3, they provide activities to promote literacy and the small business of the food and beverages. Last, cluster 4 has to support the community development and literacy activities. Figure 6.12 presents the food security map with the corresponding activities of each cluster.

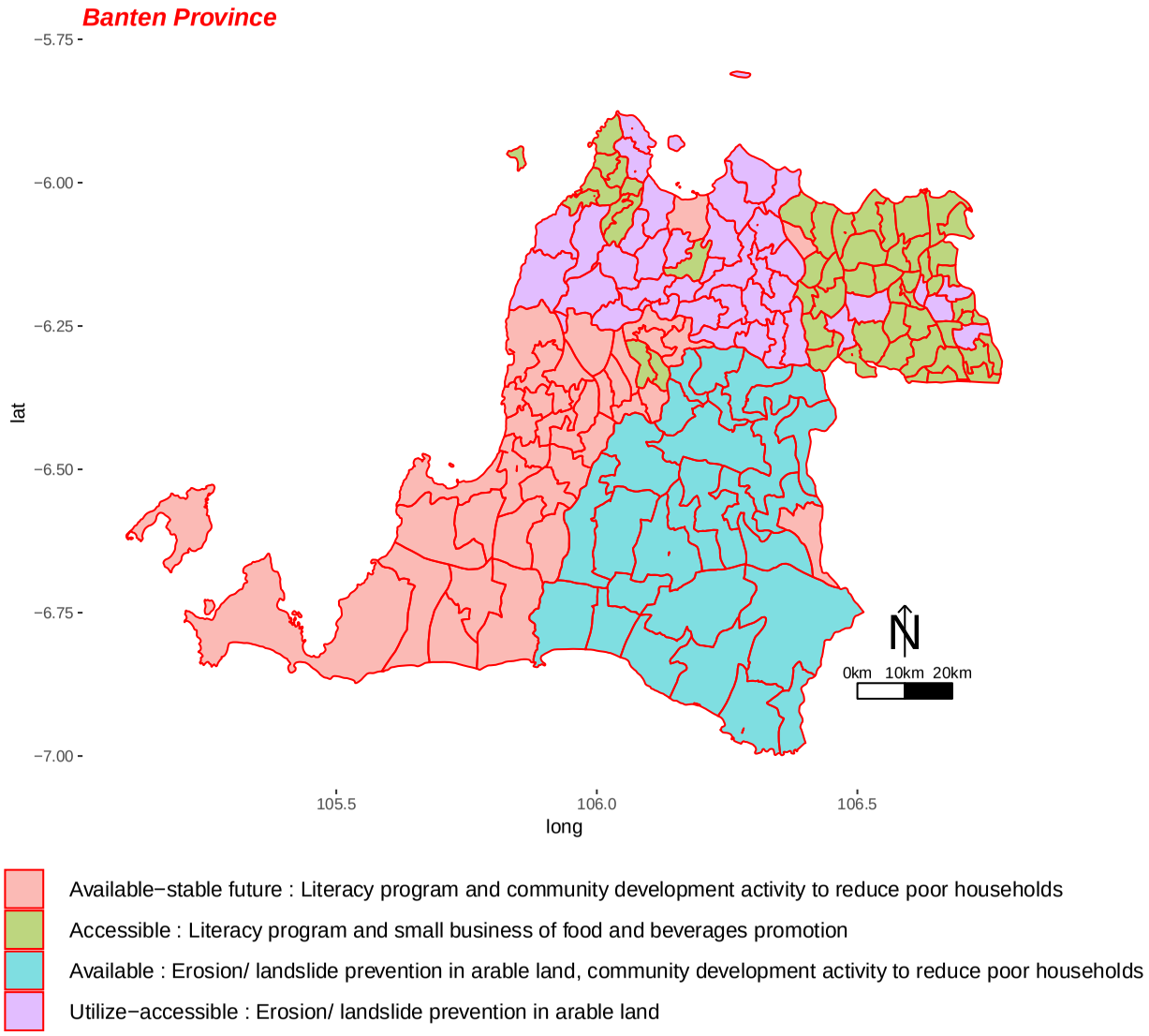


Figure 6.12: The food security map of the Banten Province

7 Conclusions

As food security mapping is identical to a partitioning analysis, a proper partitioning algorithm for food security data is important due to the presence of mixed variables in the data set. A common strategy for the partitioning of the mixed variables data set is applying the Gower distance and followed by the partitioning around medoid (PAM) algorithm. However, because different distances result in different optima in a partitioning algorithm, variation of distances is required to explore the suitable result of the partitioning algorithm.

This dissertation addresses the generalization of a distance function to manage variations of the applicable distances in mixed variables data. The generalized distance function (GDF) has been formulated to adapt the existing mixed variables distances and produce new variations of the distances. By introducing a spatial constraint in the weight of the GDF, it becomes a generalized spatial distance function (GSDF) where it is also applicable for food security mapping *vis-a-vis* with the GDF.

In the part of the medoid-based partitioning algorithm, the simple and fast k-medoids (SFKM) has been reported as more efficient than the PAM, albeit having similar results. However, a simulation comparison between the SFKM and PAM produced an under-performing SFKM such that a simple k-medoids (SKM) has been developed to improve the SFKM performance. The SKM algorithm, in addition, has comparable results to the PAM algorithm, and is more efficient than the PAM algorithm when the number of cluster is small ($k < 10$) and the number of objects is high ($n > 1000$).

To evaluate the partitioning results, visualization techniques of both the internal-based and relative-based criteria can depict well or poorly separated clusters. For the internal-based criterion, the modified stripes show non-overlap stripes in well-separated clusters. Meanwhile, a reduced size of consensus matrix produces a high stability proportion if the clusters are well separated. A modified barplot visualization, moreover, assists the partitioning result interpretation, especially for variable explanatory within a cluster.

As shown in the empirical case of food security mapping in Banten Province, Indonesia, the SKM algorithm was applied to group the districts in Banten Province via both the GDF and GSDF. The GSDF result was favorable compared to that of the GDF because the spread of the districts from the GSDF was more technically applicable for the easiness of monitoring by the extension agents. There were 4 clusters, namely

available, *utilize-accessible*, *accessible*, and *available-stable future* clusters. Possible activities for those clusters were promoting small business and literacy activities, literacy and poor household reducing activities, and erosion/ landslide prevention, respectively.

The kmed package, moreover, implements the GDF and medoid-based algorithms in the R environment. There are five direct implementations of the GDF, i.e. Gower, Wishar, Podani, Huang, Harikumar-PV, and Ahmad-Dey, via the distmix function. With the support of the other packages, such as the proxy and nomclust packages, the GDF is not limited to only these five GDF's, but also possible for other GDF's by varying the combination of the numerical, binary, and categorical distances.

While the SFKM algorithm initial medoids are fixed, the initial medoids in the fastkmed function, which is an implementation of the SFKM algorithm in the kmed package, are flexible. With this flexibility, any initial medoids method is applicable. Moreover, the implementation of the k-medoids (KM) and increasing the number of clusters k-medoids (INCKM) algorithms in the kmed package apply this flexibility. The SKM algorithm implementation in the kmed package has also been developed based on the fastkmed function such that in the near future, the development of more methods in the initial medoids selection can be adapted easily in the fastkmed function.

References

Ahmad, A. and L. Dey (2007). A k-mean clustering algorithm for mixed numeric and categorical data. *Data and Knowledge Engineering 63*, 503–527.

Arbelaitz, O., I. Gurrutxaga, J. Muguerza, J. Perez, and I. Perona (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition 46*, 243–256.

Barrett, C. B. (2010). Measuring food insecurity. *Science 327*, 825.

Battista, G. D., P. Eades, R. Tamassia, and I. G. Tollis (1994). Algorithm for drawing graphs: An annotated bibliography. *Computational Geometry 4*, 235–282.

Benson-Putnins, D., M. Bonfardin, M. E. Magnoni, and D. Martin (2011). Spectral clustering and visualization: A novel clustering of fisher's iris data. *SIAM Undergraduate Research Online 4*.

Boriah, S., V. Chandola, and V. Kumar (2008). Similarity measures for categorical data: A comparative evaluation. In *Society for Industrial and Applied Mathematics - 8th SIAM International Conference on Data Mining 2008, Proceedings in Applied Mathematics 130*, Volume 1, pp. 243–254.

Brentari, E., L. Dancelli, and M. Manisera (2016). Clustering ranking data in market segmentation: a case study on the Italian McDonald's customers preferences. *Journal of Applied Statistics 43*, 1959–1976.

Brock, G., V. Pihur, S. Datta, and S. Datta (2008). clValid: An R package for cluster validation. *Journal of Statistical Software 25*(4).

Budiaji, W. (2019). *kmed: Distance-Based k-Medoids*. R package version 0.3.0.

Bushel, P., R. Wolfinger, and G. Gibson (2007). Simultaneous clustering of gene expression data with clinical chemistry and pathological evaluations reveals phenotypic prototypes. *BMC Systems Biology 1*, 1–20.

Charrad, M., N. Ghazzali, V. Boiteau, and A. Niknafs (2014). NbClust: An R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software 61*(6), 1–36.

Coates, J. (2013). Build it back better: Deconstructing food security for improved measurement and action. *Global Food Security 2*(3), 188–194.

Dolnicar, S. and F. Leisch (2010). Evaluation of structure and reproducibility of cluster solutions using the bootstrap. *Marketing Letters 21*, 83–101.

Dolnicar, S. and F. Leisch (2014). Using graphical statistics to better understand market segmentation solutions. *International Journal of Market Research 56*, 207–230.

dos Santos, T. and L. Zárate (2015). Categorical data clustering: What similarity measure to recommend? *Expert System with Application 42*, 1247–1260.

Duda, R., P. Hart, and D. Stork (2001). *Pattern Classification* (2 ed.). New York, USA: John Wiley and Sons.

Everitt, B., S. Landau, M. Leese, and D. Stahl (2011). *Cluster Analysis* (5 ed.). West Sussex, UK: John Wiley.

Fang, Y.and Wang, J. (2012). Selection of the number of clusters via the bootstrap method. *Computational Statistics and Data Analysis 56*, 468–477.

FAO (2006). Food security. *Policy Brief June*(2), 1–4.

FAO, IFAD, and WFP (2015). *The State of Food Insecurity in the World 2015. Meeting the 2015 international hunger targets: taking stock of uneven progress.* Rome, Italy: FAO.

Friendly, M. (2002). Corrgrams: Exploratory displays for correlation matrices. *The American Statistician 56*(4), 316–324.

FSC and WFP (2015). *Food Security and Vulnerability Atlas of Indonesia.* Jakarta, Indonesia: FSC and WFP.

Gan, G., C. Ma, and J. Wu (2007). *Data Clustering, Theory, Algorithm, and Application.* Philadelphia, USA: The American Statistical Association and The Society for Industrial and Applied Mathematics.

Gordon, A. and M. Vichi (1998). Partitions of partitions. *Journal of Classification 15*, 265–285.

Gower, J. (1971). A general coefficient of similarity and some of its properties. *Biometrics 27*, 857–871.

Hahsler, M. and K. Hornik (2011). Dissimilarity plots: A visual exploration tool for partitional clustering. *Journal of Computational and Graphical Statistics 20*(2), 335–354.

Halkidi, M., Y. Batistakis, and M. Vazirgiannis (2001). On clustering validation techniques. *Journal of Intelligent Information Systems 17*(2/3), 107–145.

Handl, J., J. Knowles, and D. Kell (2005). Computational cluster validation in post-genomic data analysis. *Bioinformatics 21*(15), 3201–3212.

Harikumar, S. and S. PV (2015). K-medoid clustering for heterogeneous data sets. *JProcedia Computer Science 70*, 226–237.

Hartigan, J. and M. Wong (1979). A k-means clustering algorithm. *Journal of the Royal Statistical Society 28*, 100108.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Element of Statistical Learning: Data Mining, Inference, and Prediction.* New York, USA: Springer.

Hennig, C. (2007). Cluster-wise assessment of cluster stability. *Computational Statistics and Data Analysis 52*, 258–271.

Hornik, K. (2005). A clue for cluster ensembles. *Journal of Statistical Software 14*(12).

Hornik, K. (2017). *clue: Cluster ensembles.* R package version 0.3-53.

Huang, Z. (1997). Clustering large data sets with mixed numeric and categorical values. In *The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 21–34.

Jain, A. and J. Moreau (1987). Bootstrap technique in cluster analysis. *Pattern Recognition 20*, 547–568.

Jain, A., M. Murty, and P. Flynn (1999). Data clustering: A review. *ACM Computing Surveys 31*, 264–323.

Jarosz, L. (2011). Defining world hunger: Scale and neoliberal ideology in international food security policy discourse. *Food Culture and Society an International Journal of Multidisciplinary Research 14*(1).

Ji, J., T. Bai, C. Zhou, C. Ma, and W. Z. (2013). An improved k-prototypes clustering algorithm for mixed numeric and categorical data. *Neurocomputing 120*, 590–596.

Kaufman, L. and P. Rousseeuw (1990). *Finding Groups in Data.* New York, USA: John Wiley and Sons.

Kincaid, R. K. and D. Phillips (2011). Network topology measures. *Wiley Interdisciplinary Reviews: Computational Statistics 3*, 557–565.

Kolaczyk, E. D. and G. Csardi (2014). *Statistical Analysis of Network Data with R.* New York, USA: Springer.

Legendre, P. and L. Legendre (2012). *Numerical Ecology* (3rd Edition ed.), Volume 24. Amsterdam, The Netherlands: Elsevier.

Leisch, F. (2006). A toolbox for k-centroids cluster analysis. *Computational Statistics and Data Analysis 51*, 526–544.

Leisch, F. (2008). *Handbook of Data Visualization*, Chapter Visualizing cluster analysis and finite mixture models, pp. 561–587. Springer Handbooks of Computational Statistics. Springer Verlag.

Leisch, F. (2010). Neighborhood graphs, stripes and shadow plots for cluster visualization. *Statistics and Computing 20*, 457–469.

Lichman, M. (2013). Uci machine learning repository. http://archive.ics.uci.edu/ml.

Lin, D. (1998). An information-theoretic definition of similarity. In *ICML '98: Proceedings of the 15th International Conference on Machine Learning*, San Francisco, pp. 296–304.

Liu, S., L. Shen, and D. Huang (2016). A three-stage framework for clustering mixed data. *WSEAS Transactions on Systems 15*, 1–10.

Maechler, M., P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik (2017). *cluster: Cluster Analysis Basics and Extensions*. R package version 2.0.6 — For new features, see the 'Changelog' file (in the package source).

McCane, B. and M. Albert (2008). Distance functions for categorical and mixed variables. *Pattern Recognition Letters 29*, 986–993.

Meyer, D. and C. Buchta (2016). *proxy: Distance and Similarity Measures*. R package version 0.4-16.

Monti, S., P. Tamayo, J. Mesirov, and T. Golub (2003). Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Machine Learning 52*, 91–118.

Morlini, I. and S. Zani (2012). A new class of weighted similarity indices using polytomous variables. *Journal of Classification 29*(2), 199–226.

Murdoch, D. J. and E. D. Chow (1996). A graphical display of large correlation matrices. *The American Statistician 50*(2), 178–180.

Pakhira, M. K. (2009). A modified k-means algorithm to avoid empty clusters. *International Journal of Recent Trends in Engineering 1*, 221–226.

Park, H. and C. Jun (2009). A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications 36*, 3336–3341.

Pawitan, Y. and J. Huang (2003). Constrained clustering of irregularly sampled spatial data. *Journal of Statistical Computation and Simulation 73*, 853–865.

Podani, J. (1999). Extending gower's general coefficient of similarity to ordinal characters. *Taxon 48*, 331–340.

Qiu, W. and H. Joe (2006a). Generation of random clusters with specified degree of separation. *Journal of Classification 23*, 31534.

Qiu, W. and H. Joe (2006b). Separation index and partial membership for clustering. *Computational Statistics and Data Analysis 50*, 585–603.

Qiu, W. and H. Joe (2015). *clusterGeneration: Random Cluster Generation (with Specified Degree of Separation)*. R package version 1.3.4.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66*(336), 846–850.

Rangel, E., W. Hendrix, A. Agrawal, W. Liao, and A. Choudhary (2016). Agoras: A fast algorithm for estimating medoids in large datasets. *Procedia Computer Science 80*, 1159–1169.

Reynolds, A. P., G. Richards, B. De La Iglesia, and V. J. Rayward-Smith (2006). Clustering rules: A comparison of partitioning and hierarchical clustering algorithms. *Journal of Mathematical Modelling and Algorithms 5*, 475–504.

Rousseeuw, P. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics 20*, 53–65.

Schonlau, M. (2004). Visualizing non-hierarchical and hierarchical cluster analyses with clustergrams. *Computational Statistics 19*, 95–111.

Senbabaoglu, Y., G. Michailidis, and J. Z. Li (2014). Critical limitations of consensus clustering in class discovery. *Scientific Reports 4*, 6207.

Simbahan, G. and A. Dobermann (2006). An algorithm for spatially constrained classification of categorical and continuous soil properties. *Geoderma 136*, 504523.

Steinley, D. (2003). Local optima in k-means clustering: What you dont know may hurt you. *Psychological Methods 8*, 294304.

Steinley, D. and M. Brusco (2007). Initializing k-means batch clustering: A critical evaluation of several techniques. *Journal of Classification 24*, 99–121.

Sulc, Z. and H. Rezankova (2016). *nomclust: Hierarchical Nominal Clustering Package.* R package version 1.00.1011.

The Economist Intelligent Unit (2017). Global food security index 2017: Measuring food security and the impact of resource risks.

Theodoridis, S. and K. Koutroubas (2008). *Pattern Recognition* (4 ed.). San Diego, USA: Elsevier Academic Press.

van Dijk, M. and G. Meijerink (2014). A review of global food security scenario and assessment studies: Results, gaps and research priorities. *Global Food Security 3*, 227–238.

Webb, A. and K. Copsey (2011). *Statistical Pattern Recognition* (3 ed.). West Sussex, UK: John Wiley and Sons.

Weinstein, J. N. (2008). A postgenomic visual icon. *Science 319*, 1772–1773.

WFP (2009). *Comprehensive Food Security and Vulnerability Analysis Guidelines.* Rome, Italy: World Food Programm.

Wilkinson, L. and M. Friendly (2009). The history of the cluster heat map. *The American Statistician 63*(2), 179–184.

Wishart, D. (2003). K-means clustering with outlier detection, mixed variables and missing values. In *Exploratory Data Analysis in Empirical Research: Proceedings of the 25th Annual Conference of the Gesellschaft für Klassifikation e.V., University of Munich, March 14-16, 2001*, Berlin, Heidelberg, pp. 216–226. Springer Berlin Heidelberg.

Wu, J., J. Chen, H. Xiong, and M. Xie (2009). External validation measures for k-means clustering: A data distribution perspective. *Expert Systems with Applications 36*, 6050–6061.

Wu, X., V. Kumar, J. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, Z. Z. Yu, P.S., M. Steinbach, D. Hand, and D. Steinberg (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems 14*, 1–37.

Xu, R. and D. Wunsch II (2005). Survey of clustering algorithm. *IEEE Transactions on Neural Networks 16*, 645–678.

Yin, J. and Z. Tan (2005). Clustering mixed type attributes in large dataset. In Y. Pan, D. Chen, M. Guo, J. Cao, and J. Dongarra (Eds.), *Parallel and Distributed Processing Application*, LNCS 3758, Berlin, pp. 655–661. ISPA: Springer-Verlag.

Yu, D., G. Liu, M. Guo, and X. Liu (2018). An improved k-medoids algorithm based on step increasing and optimizing medoids. *Expert Systems With Applications 92*, 464–473.

Zadegan, S., M. M, and F. Sadoughi (2013). Ranked k-medoids: A fast and accurate rank-based partitioning algorithm for clustering large datasets. *Knowledge-Based Systems 39*, 133–143.

Index of Table

2.1 Mixed variable distances in the GDF formulation 8
2.2 Spatial weights of mixed variable distances 9

3.1 Table of true class vs partitioning result 17

5.1 Two distances from the GDF formulation 46
5.2 Adjusted numerical variable distances 47
5.3 Rand indices of the simulated data via the PAM algorithm 47
5.4 Rand indices of the simulated data via the KM algorithm 48
5.5 Rand indices of the simulated data via the SFKM algorithm 48
5.6 Rand indices of the simulated data via the RKM algorithm 49
5.7 Rand indices of the simulated data via the INCKM algorithm 50
5.8 Rand indices of the simulated data via the SKM algorithm 51
5.9 Rand indices of the different proportion of variables 52
5.10 Rand indices of the different number of clusters 52
5.11 Rand indices of the different number of variables 53
5.12 Rand indices of the different number of objects 53
5.13 Accuracy rate of the heart disease data 56
5.14 Rand indices of the global food security data for $k = 2$ 58
5.15 Rand indices of the global food security data for $k = 3$ 58
5.16 Average value of silhouette indices of the sponge data set 61
5.17 Stability proportion of the sponge data set 62

6.1 Variables in the food security data set 63
6.2 Summary statistics of the indicator variables 64
6.3 Cluster indices summary of all GSDF 66
6.4 Cluster stability proportion of all GSDF 66
6.5 Cluster indices summary of all GDF 68
6.6 Cluster stability proportion of all GDF 68

Table of Figures

2.1 Swapping step in the PAM algorithm 9
2.2 Updating a medoid in the KM algorithm 11
2.3 First and second steps in the SFKM algorithm 12
2.4 First and second steps in the RKM algorithm ($b = 4$) 13
2.5 Initial medoids in the INCKM algorithm ($k = 4$) 15
2.6 Initial medoids in the SKM algorithm ($s = 2$) 16

3.1 Silhouette (left) and shadow value (right) of well-separated clusters . . 19
3.2 Silhouette (left) and shadow value (right) of poor-separated clusters . . 20
3.3 Consensus matrix heatmap of well-separated (left) and poor-separated (right) clusters 21
3.4 Shadow value plot of well-separated clusters in neighborhood graph . . 23
3.5 Shadow value plot of poor-separated clusters in neighborhood graph . . 23
3.6 A stripe plot of well-separated clusters 24
3.7 A stripe plot of poorly-separated clusters 24
3.8 A modified stripe plot of well-separated clusters 25
3.9 A modified stripe plot of poor-separated clusters 25
3.10 Direct visualization of matrix $\mathbf{V}$ of well-separated clusters 26
3.11 Reduced size of consensus matrix heatmap of well-separated clusters . . 26
3.12 Reduced size of consensus matrix heatmap of poor-separated clusters . 27
3.13 A directed graph of well separated clusters 28
3.14 A directed graph of poor separated clusters 29
3.15 Combination of relative-based and internal-based criteria of well-separated clusters 29
3.16 A marked barplot of numerical data set ($k = 3$) 30
3.17 A modified marked barplot of mixed variable data set ($k = 2$) 31

4.1 Silhouette (left) and shadow value (right) plots of the generated data set (dat1) 38
4.2 Heatmap image of generated data set (dat1) partitioned via the INCKM algorithm with the ward linkage (left) and INCKM (right) to reorder the consensus matrix 41

4.3 PCA biplot of the iris data set with colored objects via the SFKM algorihtm with the Manhattan range weighted GDF 42
4.4 Barplot of the iris data set via the SFKM algorithm 42
4.5 Heatmap of the generated data set (dat1) partitioned via the INCKM algorithm with the PAM algorithm to reorder the consensus matrix . . 45

5.1 The simulated data set with separation values: 0.1 (a), 0.3 (b), 0.5 (c), and 0.7 (d) 47
5.2 Benchmarking of the PAM and SKM ($k = 2$, $s = 20$) 54
5.3 Benchmarking of the PAM and SKM ($k = 4$, $s = 20$) 54
5.4 Benchmarking of the PAM and SKM ($k = 7$, $s = 20$) 55
5.5 Benchmarking of the PAM and SKM ($k = 10$, $s = 20$) 55
5.6 The first and second principal components plot of the global food security data 57
5.7 Stripe plot of the global food security data via the SKM algorithm with the Huang GDF ($k = 2$) 58
5.8 Stripe plot of the global food security data via the SKM algorithm with the Huang GDF ($k = 3$) 59
5.9 Combination plots of the global food security data via the SKM algorithm with the Huang GDF ($k = 2$) 59
5.10 Combination plots of the global food security data via the SKM algorithm with the Huang GDF ($k = 3$) 60
5.11 Directed graph of sponge data set via the PAM algorithm with the Esimma GDF ($k = 7$) 62

6.1 The cluster consensus heatmap (left) and kernel density (right) of the Huang GSDF for two clusters 66
6.2 The cluster consensus heatmap (left) and kernel density (right) of the Huang GSDF for three clusters 67
6.3 The cluster consensus heatmap (left) and kernel density (right) of the Wishart GSDF for four clusters 67
6.4 The cluster consensus heatmap (left) and kernel density (right) of the Gower GDF for two clusters 69
6.5 The cluster consensus heatmap (left) and kernel density (right) of the Harikumar-PV GDF for three clusters 69
6.6 The cluster consensus heatmap (left) and kernel density (right) of the Wishart GDF for four clusters 70
6.7 The clusters distribution via the GSDF 71
6.8 The clusters distribution via the GDF 71
6.9 The modified barplot of food security data (mixed variables) for four clusters 72

6.10 The modified stripes plot of food security data (mixed variables) for four clusters 73
6.11 The directed graph of food security data (mixed variables) for four clusters 74
6.12 The food security map of the Banten Province 75

Appendix A: R documentation

Reference manual of kmed package is deposited in cran R and can be obtained via https://cran.r-project.org/web/packages/kmed/kmed.pdf

Package ‘kmed’

June 14, 2019

Type Package

Title Distance-Based K-Medoids

Version 0.3.0

Date 2019-06-14

Author Weksi Budiaji

Maintainer Weksi Budiaji <budiaji@untirta.ac.id>

Description Algorithms of distance-based k-medoids clustering: simple and fast k-medoids, ranked k-medoids, and increasing number of clusters in k-medoids. Calculate distances for mixed variable data such as Gower, Podani, Wishart, Huang, Harikumar-PV, and Ahmad-Dey. Cluster validations apply internal and relative criteria. The internal criteria include silhouette index and shadow values. The relative criterium applies bootstrap procedure producing a heatmap with a flexible reordering matrix algorithm such as ward, complete, or centroid linkages. The cluster result can be plotted in a marked barplot or pca biplot.

Depends R (>= 2.10)

License GPL-3

LazyData TRUE

RoxygenNote 6.1.0

Suggests knitr, rmarkdown

VignetteBuilder knitr

Imports ggplot2

R topics documented:

barplotnum 2
clust4 3
clust5 4
clustboot 5
clustheatmap 7
consensusmatrix 8

cooccur 10
csv 12
distmix 13
distNumeric 16
fastkmed 17
globalfood 18
heart 19
inckmed 20
matching 22
pcabiplot 23
rankkmed 24
shadow 25
sil 26
silhoutte 27
stepkmed 27

Index **28**

`barplotnum` *Barplot of each cluster for numerical variables data set*

Description

This function creates a barplot from a cluster result. A barplot indicates the location and dispersion of each cluster. The x-axis of the barplot is variable's mean, while the y-axis is the variable's name.

Usage

```
barplotnum(dataori, clust, nc = 1, alpha = 0.05)
```

Arguments

`dataori`	An original data set.
`clust`	A vector of cluster membership (see **Details**).
`nc`	A number of columns for the plot of all cluster (see **Details**).
`alpha`	A numeric number to set the significant level (between 0 and 0.2).

Details

This is a marked barplot because some markers are added, i.e. a significant test, a population mean for each (numerical) variable. The significance test applies t-test between the population's mean and cluster's mean in every variable. The alpha is set in between 0 to 20%. If the population mean differs to the cluster's mean, the bar shade in the barplot also differs.

`clust` is a vector with the length equal to the number of objects (n), or the function will be an *error* otherwise. `nc` controls the layout (grid) of the plot. If `nc = 1`, the plot of each cluster is placed in a column. When the number of clusters is 6 and `nc = 2`, for example, the plot has a layout of 3-row and 2-column grids.

Value

Function returns a barplot.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Leisch, F. (2008). Handbook of Data Visualization, Chapter Visualizing cluster analysis and finite mixture models, pp. 561-587. Springer Handbooks of Computational Statistics. Springer Verlag.

Dolnicar, S. and F. Leisch (2014). Using graphical statistics to better understand market segmentation solutions. International Journal of Market Research 56, 207-230.

Examples

```
dat <- iris[,1:4]
memb <- cutree(hclust(dist(dat)),3)
barplotnum(dat, memb)
barplotnum(dat, memb, 2)
```

`clust4` *4-clustered data set*

Description

A dataset containing two variables of 300 objects and their class memberships generated by the **clusterGeneration** package.

Usage

```
clust4
```

Format

A data frame with 300 rows and 3 variables:

x1 X1.

x2 X2.

class Class membership.

Source

Data is generated via the `genRandomClust` function in the **clusterGeneration** package. The code to generate this data set is

set.seed(2016)

randclust <- clusterGeneration::genRandomClust(4, sepVal = 0.001, numNonNoisy = 2, numReplicate = 1, clustszind = 3, clustSizes = as.numeric(table(sample(1:4, 300, replace = TRUE))), outputDatFlag=FALSE, outputLogFlag=FALSE, outputEmpirical=FALSE, outputInfo=FALSE)

clust4 <- as.data.frame(randclust$datList$test_1)

clust4$class <- randclust$memList$test_1

References

Qiu, W., and H. Joe. 2015. ClusterGeneration: Random Cluster Generation (with Specified Degree of Separation).

Qiu, W., and H. Joe. 2006a. Generation of Random Clusters with Specified Degree of Separation. Journal of Classification 23 pp. 315-34.

Qiu, W., and H. Joe. 2006b. Separation Index and Partial Membership for Clustering. Computational Statistics and Data Analysis 50 pp. 585-603.

`clust5` *5-clustered data set*

Description

A dataset containing two variables of 800 objects and their class memberships generated by the **clusterGeneration** package.

Usage

```
clust5
```

Format

A data frame with 800 rows and 3 variables:

x1 X1.

x2 X2.

class Class membership.

Source

Data is generated via the `genRandomClust` function in the **clusterGeneration** package. The code to generate this data set is

set.seed(2016)

randclust <- clusterGeneration::genRandomClust(5, sepVal = 0.2, numNonNoisy = 2, numReplicate = 1, clustszind = 3, clustSizes = as.numeric(table(sample(1:5, 800, replace = TRUE))), outputDatFlag=FALSE, outputLogFlag=FALSE, outputEmpirical=FALSE, outputInfo=FALSE)

clust5 <- as.data.frame(randclust$datList$test_1)

clust5$class <- randclust$memList$test_1

References

Qiu, W., and H. Joe. 2015. ClusterGeneration: Random Cluster Generation (with Specified Degree of Separation).

Qiu, W., and H. Joe. 2006a. Generation of Random Clusters with Specified Degree of Separation. Journal of Classification 23 pp. 315-34.

Qiu, W., and H. Joe. 2006b. Separation Index and Partial Membership for Clustering. Computational Statistics and Data Analysis 50 pp. 585-603.

`clustboot` *Bootstrap replications for clustering alorithm*

Description

This function does bootstrap replications for a clustering algorithm. Any hard clustering algorithm is valid.

Usage

```
clustboot(distdata, nclust = 2, algorithm = fastclust, nboot = 25,
  diss = TRUE)
```

Arguments

`distdata`	A distance matrix (*n x n*)/ *dist* object or a data frame.
`nclust`	A number of clusters.
`algorithm`	A clustering algorithm function (*see* **Details**).
`nboot`	A number of bootstrap replicates.
`diss`	A logical if `distdata` is a distance matrix/ object or a data frame.

Details

This is a function to obtain bootstrap evaluation for cluster results. The `algorithm` argument is a function where this function has two input arguments. The two input arguments are a *distance matrix/ object* or a *data frame*, and *number of clusters*. Then the output is only a *vector of cluster memberships*.

The default `algorithm` is `fastclust` applying the `fastkmed` function. The code of the `fastclust` is

fastclust <- function(x, nclust) {

res <- fastkmed(x, nclust, iterate = 50)

return(res$cluster)

}

For other examples, *see* **Examples**. It applies ward and kmeans algorithms. When kmeans is applied, for example, `diss` is set to be `FALSE` because the input of the `kmclust` and `clustboot` is a data frame instead of a distance.

Value

Function returns a matrix of bootstrap replicates with a dimension of *n x b*, where *n* is the number of objects and *b* is the number of bootstrap replicates.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Dolnicar, S. and Leisch, F. 2010. Evaluation of structure and reproducibility of cluster solutions using the bootstrap. Marketing Letters 21 pp. 83-101.

Examples

```
num <- as.matrix(iris[,1:4])
mrwdist <- distNumeric(num, num, method = "mrw")
ward.D2 <- function(x, nclust) {
res <- hclust(as.dist(x), method = "ward.D2")
member <- cutree(res, nclust)
return(member)
}
kmclust <- function(x, nclust) {
res <- kmeans(x, nclust)
return(res$cluster)
}
irisfast <- clustboot(mrwdist, nclust=3, nboot=7)
head(irisfast)
irisward <- clustboot(mrwdist, nclust=3, algorithm = ward.D2, nboot=7)
head(irisward)
iriskmeans <- clustboot(num, nclust=3, algorithm = kmclust, nboot=7, diss = FALSE)
head(iriskmeans)
```

`clustheatmap` *Consensus matrix heatmap from A consensus matrix*

Description

This function creates a consensus matrix heatmap from a consensus/ agreement matrix. The values of the consensus/ agreement matrix are transformed in order to plot the heatmap.

Usage

```
clustheatmap(consmat, title = "")
```

Arguments

`consmat`	A matrix of consensus/ agreement matrix (*see* **Details**).
`title`	A title of the plot.

Details

This is a function to produce a consensus matrix heatmap from a consensus/ agreement matrix. A matrix produced by the `consensusmatrix` function can be directly provided in the `consmat` argument. The values of the consensus matrix, **A**, are then transformed via a non-linear transformation by applying

$$a_{ij}^{trf} = \frac{a_{ij} - min(a_{..})}{max(a_{..}) - min(a_{..})}$$

where a_{ij} is the value of the consensus matrix in row *i* and column *j*, and $a_{..}$ is the all values of the matrix ($\forall$**A**).

Value

Function returns a heatmap plot.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Monti, S., P. Tamayo, J. Mesirov, and T. Golub. 2003. Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. Machine Learning 52 pp. 91-118.

Hahsler, M., and Hornik, K., 2011. Dissimilarity plots: A visual exploration tool for partitional clustering. Journal of Computational and Graphical Statistics 20(2) pp. 335-354.

Examples

```
num <- as.matrix(iris[,1:4])
mrwdist <- distNumeric(num, num, method = "mrw")
irisfast <- clustboot(mrwdist, nclust=3, nboot=7)
complete <- function(x, nclust) {
res <- hclust(as.dist(x), method = "complete")
member <- cutree(res, nclust)
return(member)
}
consensuscomplete <- consensusmatrix(irisfast, nclust = 3, reorder = complete)
clustheatmap(consensuscomplete)
```

`consensusmatrix` *Consensus matrix from A matrix of bootstrap replicates*

Description

This function creates a consensus matrix from a matrix of bootstrap replicates. It transforms an *n x b* matrix into an *n x n* matrix, where *n* is the number of objects and *b* is the number of bootstrap replicates.

Usage

```
consensusmatrix(bootdata, nclust, reorder = fastclust)
```

Arguments

`bootdata`	A matrix of bootstrap replicate (*n x b*) (*see* **Details**).
`nclust`	A number of clusters.
`reorder`	Any distance-based clustering algorithm function (*see* **Details**).

Details

This is a function to obtain a consensus matrix from a matrix of bootstrap replicates to evaluate the clustering result. The `bootdata` argument can be supplied directly from a matrix produced by the `clustboot` function. The values of the consensus matrix, **A**, are calculated by

$$a_{ij} = a_{ji} = \frac{\#n\ of\ objects\ \mathbf{i}\ and\ \mathbf{j}\ in\ the\ same\ cluster}{\#n\ of\ objects\ \mathbf{i}\ and\ \mathbf{j}\ sampled\ at\ the\ same\ time}$$

where a_{ij} is the agreement index between objects *i* and *j*. Note that due to the agreement between objects *i* and *j* equal to the agreement between objects *j* and *i*, the consensus matrix is a symmetric matrix.

Meanwhile, the `reorder` argument is a function to reorder the objects in both the row and column of the consensus matrix such that similar objects are close to each other. This task can be solved by applying a clustering algorithm in the consensus matrix. The `reorder`

has to consist of two input arguments. The two input arguments are a *distance matrix/ object* and *number of clusters*. The output is only a *vector of cluster memberships*. Thus, the algorihtm that can be applied in the `reorder` argument is the distance-based algorithm with a distance as the input.

The default `reorder` is `fastclust` applying the `fastkmed` function. The code of the `fastclust` is

fastclust <- function(x, nclust) {

res <- fastkmed(x, nclust, iterate = 50)

return(res$cluster)

}

For other examples, *see* **Examples**. It applies centroid and complete linkage algorithms.

Value

Function returns a consensus/ agreement matrix of *n x n* dimension.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Monti, S., P. Tamayo, J. Mesirov, and T. Golub. 2003. Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. Machine Learning 52 pp. 91-118.

Examples

```
num <- as.matrix(iris[,1:4])
mrwdist <- distNumeric(num, num, method = "mrw")
irisfast <- clustboot(mrwdist, nclust=3, nboot=7)
consensusfast <- consensusmatrix(irisfast, nclust = 3)
centroid <- function(x, nclust) {
res <- hclust(as.dist(x), method = "centroid")
member <- cutree(res, nclust)
return(member)
}
consensuscentroid <- consensusmatrix(irisfast, nclust = 3, reorder = centroid)
complete <- function(x, nclust) {
res <- hclust(as.dist(x), method = "complete")
member <- cutree(res, nclust)
return(member)
}
consensuscomplete <- consensusmatrix(irisfast, nclust = 3, reorder = complete)
consensusfast[c(1:5,51:55,101:105),c(1:5,51:55,101:105)]
consensuscentroid[c(1:5,51:55,101:105),c(1:5,51:55,101:105)]
consensuscomplete[c(1:5,51:55,101:105),c(1:5,51:55,101:105)]
```

`cooccur` *Co-occurrence distance for binary/ categorical variables data*

Description

This function calculates the co-occurrence distance proposed by Ahmad and Dey (2007).

Usage

```
cooccur(data)
```

Arguments

`data`	A matrix or data frame of binary/ categorical variables (*see* **Details**).

Details

This function computes co-occurrence distance, which is a binary/ categorical distance, that based on the other variable's distribution (*see* **Examples**). In the **Examples**, we have a data set:

object	x	y	z
1	1	2	2
2	1	2	1
3	2	1	2
4	2	1	2
5	1	1	1
6	2	2	2
7	2	1	2

The co-occurrence distance transforms each category of binary/ categorical in a variable based on the distribution of other variables, for example, the distance between categories 1 and 2 in the x variable can be different to the distance between categories 1 and 2 in the z variable. As an example, the transformed distance between categories 1 and 2 in the z variable is presented.

A cross tabulation between the z and x variables with corresponding (column) proportion is

	1	2	‖	1	2
1	2	1	‖	1.0	0.2
2	0	4	‖	0.0	0.8

A cross tabulation between the z and y variables with corresponding (column) proportion is

	1	2	‖	1	2
1	1	3	‖	0.5	0.6
2	1	2	‖	0.5	0.4

Then, the maximum values of the proportion in each row are taken such that they are 1.0, 0.8, 0.6, and 0.5. The new distance between categories 1 and 2 in the z variable is

$$\delta^z_{1,2} = \frac{(1.0 + 0.8 + 0.6 + 0.5) - 2}{2} = 0.45$$

The constant 2 in the formula applies because the z variable depends on the 2 other variable distributions, i.e the x and y variables. The new distances of each category in the for the x and y variables can be calculated in a similar way.

Thus, the distance between objects 1 and 2 is 0.45. It is only the z variable counted to calculate the distance between objects 1 and 2 because objects 1 and 2 have similar values in both the x and y variables.

The `data` argument can be supplied with either a matrix or data frame, in which the class of the element has to be an integer. If it is not an integer, it will be converted to an integer class. If the `data` of a variable only, a simple matching is calculated. The co-occurrence is absent due to its dependency to the distribution of other variables and a `warning` message appears.

Value

Function returns a distance matrix (*n x n*).

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Ahmad, A., and Dey, L. 2007. A K-mean clustering algorithm for mixed numeric and categorical data. Data and Knowledge Engineering 63, pp. 503-527.

Harikumar, S., PV, S., 2015. K-medoid clustering for heterogeneous data sets. JProcedia Computer Science 70, 226-237.

Examples

```
set.seed(1)
a <- matrix(sample(1:2, 7*3, replace = TRUE), 7, 3)
cooccur(a)
```

`csv` *Centroid shadow value (CSV) index and plot*

Description

This function computes shadow values and shadow value plots of each cluster. The plot presents the mean of the shadow values as well.

Usage

```
csv(distdata, idmedoid, idcluster, title = "")
```

Arguments

`distdata`	A distance matrix (*n x n*) or *dist* object.
`idmedoid`	A vector of id medoids (*see* **Details**).
`idcluster`	A vector of cluster membership (*see* **Details**).
`title`	A title of the plot.

Details

The origin of the shadow value is calculated in the `shadow` function of the **flexclust** package, in which it is based on the first and second closest centroid. The `csv` function in this package modifies the centroid into medoid such that the formula to compute shadow value of object *i* is

$$sh(i) = \frac{2d(i, m(i))}{d(i, m(i)) + d(i, m'(i))}$$

where $d(i, m(i))$ is the distance between object *i* to the first closest medoid and *d(i, m'(i))* is the distance between object *i* to the second closest medoid.

The `idmedoid` argument corresponds to the `idcluster` argument. If the length of `idmedoid` is 3, for example, the `idcluster` has to have 3 unique cluster memberships, or it returns `Error` otherwise. The length of the `idcluster` has also to be equal to *n* (the number of objects).

Value

Function returns a list with following components:

`result` is a data frame of the shadow values for all objects

`plot` is the shadow value plots of each cluster.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

F. Leisch. 2010 Neighborhood graphs, stripes and shadow plots for cluster visualization. Statistics and Computing. vol. 20, pp. 457-469

Examples

```
distiris <- as.matrix(dist(iris[,1:4]))
res <- fastkmed(distiris, 3)
sha <- csv(distiris, res$medoid, res$cluster)
sha$result[c(1:3,70:75,101:103),]
sha$plot
```

`distmix` *Distances for mixed variables data set*

Description

This function computes a distance matrix for a mixed variable data set applying various methods.

Usage

```
distmix(data, method = "gower", idnum = NULL, idbin = NULL,
  idcat = NULL)
```

Arguments

`data`	A data frame or matrix object.
`method`	A method to calculate the mixed variables distance (*see* **Details**).
`idnum`	A vector of column index of the numerical variables.
`idbin`	A vector of column index of the binary variables.
`idcat`	A vector of column index of the categorical variables.

Details

There are six methods available to calculate the mixed variable distance. They are `gower`, `wishart`, `podani`, `huang`, `harikumar`, `ahmad`.

`gower`

The Gower (1971) distance is the most common distance for a mixed variable data set. Although the Gower distance accommodates missing values, a missing value is not allowed in this function. If there is a missing value, the Gower distance from the `daisy` function in the **cluster** package can be applied. The Gower distance between objects *i* and *j* is computed by $d_{ij} = 1 - s_{ij}$, where

$$s_{ij} = \frac{\sum_{l=1}^{p} \omega_{ijl} s_{ijl}}{\sum_{l=1}^{p} \omega_{ijl}}$$

ω_{ijl} is a weight in variable *l* that is usually 1 or 0 (for a missing value). If the variable *l* is a numerical variable,

$$s_{ijl} = 1 - \frac{|x_{il} - x_{jl}|}{R_l}$$

$s_{ijl} \in \{0, 1\}$, if the variable *l* is a binary/ categorical variable.

`wishart`

Wishart (2003) has proposed a different measure compared to Gower (1971) in the numerical variable part. Instead of a range, it applies a variance of the numerical variable in the s_{ijl} such that the distance becomes

$$d_{ij} = \sqrt{\sum_{l=1}^{p} \omega_{ijl} \left(\frac{x_{il} - x_{jl}}{\delta_{ijl}} \right)^2}$$

where $\delta_{ijl} = s_l$ when *l* is a numerical variable and $\delta_{ijl} \in \{0, 1\}$ when *l* is a binary/ categorical variable.

`podani`

Podani (1999) has suggested a different method to compute a distance for a mixed variable data set. The Podani distance is calculated by

$$d_{ij} = \sqrt{\sum_{l=1}^{p} \omega_{ijl} \left(\frac{x_{il} - x_{jl}}{\delta_{ijl}} \right)^2}$$

where $\delta_{ijl} = R_l$ when *l* is a numerical variable and $\delta_{ijl} \in \{0, 1\}$ when *l* is a binary/ categorical variable.

`huang`

The Huang (1997) distance between objects *i* and *j* is computed by

$$d_{ij} = \sum_{r=1}^{P_n} (x_{ir} - x_{jr})^2 + \gamma \sum_{s=1}^{P_c} \delta_c(x_{is} - x_{js})$$

where P_n and P_c are the number of numerical and categorical variables, respectively,

$$\gamma = \frac{\sum_{r=1}^{P_n} s_r^2}{P_n}$$

and $\delta_c(x_{is} - x_{js})$ is the mismatch/ simple matching distance (*see* `matching`) between object *i* and object *j* in the variable *s*.

`harikumar`

Harikumar-PV (2015) has proposed a distance for a mixed variable data set:

$$d_{ij} = \sum_{r=1}^{P_n} |x_{ir} - x_{jr}| + \sum_{s=1}^{P_c} \delta_c(x_{is} - x_{js}) + \sum_{t=1}^{p_b} \delta_b(x_{it}, x_{jt})$$

where P_b is the number of binary variables, $\delta_c(x_{is}, x_{js})$ is the co-occurrence distance (*see* `cooccur`), and $\delta_b(x_{it}, x_{jt})$ is the Hamming distance.

`ahmad`

Ahmad and Dey (2007) has computed a distance of a mixed variable set via

$$d_{ij} = \sum_{r=1}^{P_n}(x_{ir} - x_{jr})^2 + \sum_{s=1}^{P_c}\delta_c(x_{is} - x_{js})$$

where $\delta_c(x_{it}, x_{jt})$ are the co-occurrence distance (*see* `cooccur`). In the Ahmad and Dey distance, the binary and categorical variables are not separable such that the co-occurrence distance is based on the combined these two classes, i.e. binary and categorical variables.

At leas two arguments of the `idnum`, `idbin`, and `idcat` have to be provided because this function calculates the mixed distance. If the `method` is `harikumar`, the categorical variables have to be at least two variables such that the co-occurrence distance can be computed. It also applies when `method = "ahmad"`. The `idbin` + `idcat` has to be more than 1 column. It returns to an `Error` message otherwise.

Value

Function returns a distance matrix (*n x n*).

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Ahmad, A., and Dey, L. 2007. A K-mean clustering algorithm for mixed numeric and categorical data. Data and Knowledge Engineering 63, pp. 503-527.

Gower, J., 1971. A general coefficient of similarity and some of its properties. Biometrics 27, pp. 857-871

Harikumar, S., PV, S., 2015. K-medoid clustering for heterogeneous data sets. JProcedia Computer Science 70, pp. 226-237.

Huang, Z., 1997. Clustering large data sets with mixed numeric and categorical values, in: The First Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 21-34.

Podani, J., 1999. Extending gower's general coefficient of similarity to ordinal characters. Taxon 48, pp. 331-340.

Wishart, D., 2003. K-means clustering with outlier detection, mixed variables and missing values, in: Exploratory Data Analysis in Empirical Research: Proceedings of the 25th Annual Conference of the Gesellschaft fur Klassifikation e.V., University of Munich, March 14-16, 2001, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 216-226.

Examples

```
set.seed(1)
a <- matrix(sample(1:2, 7*3, replace = TRUE), 7, 3)
a1 <- matrix(sample(1:3, 7*3, replace = TRUE), 7, 3)
mixdata <- cbind(iris[1:7,1:3], a, a1)
colnames(mixdata) <- c(paste(c("num"), 1:3, sep = ""),
```

```
                        paste(c("bin"), 1:3, sep = ""),
                        paste(c("cat"), 1:3, sep = ""))
distmix(mixdata, method = "gower", idnum = 1:3, idbin = 4:6, idcat = 7:9)
```

`distNumeric` *A pair distance for numerical variables*

Description

This function computes a pairwise numerical distance between two numerical data sets.

Usage

```
distNumeric(x, y, method = "mrw", xyequal = TRUE)
```

Arguments

`x`	A first data matrix (*see* **Details**).
`y`	A second data matrix (*see* **Details**).
`method`	A method to calculate the pairwise numerical distance (*see* **Details**).
`xyequal`	A logical if `x` is equal to `y` (*see* **Details**).

Details

The `x` and `y` arguments have to be matrices with the same number of columns where the row indicates the object and the column is the variable. This function calculate all pairwise distance between rows in the `x` and `y` matrices. Although it calculates a pairwise distance between two data sets, the default function computes all distances in the `x` matrix. If the `x` matrix is not equal to the `y` matrix, the `xyequal` has to be set `FALSE`.

The `method` available are `mrw` (Manhattan weighted by range), `sev` (squared Euclidean weighted by variance), `ser` (squared Euclidean weighted by range), `ser.2` (squared Euclidean weighted by squared range) and `se` (squared Euclidean). Their formulas are:

$$mrw_{ij} = \sum_{r=1}^{p_n} \frac{|x_{ir} - x_{jr}|}{R_r}$$

$$sev_{ij} = \sum_{r=1}^{p_n} \frac{(x_{ir} - x_{jr})^2}{s_r^2}$$

$$ser_{ij} = \sum_{r=1}^{p_n} \frac{(x_{ir} - x_{jr})^2}{R_r}$$

$$ser.2_{ij} = \sum_{r=1}^{p_n} \frac{(x_{ir} - x_{jr})^2}{R_r^2}$$

$$se_{ij} = \sum_{r=1}^{p_n} (x_{ir} - x_{jr})^2$$

where p_n is the number of numerical variables, R_r is the range of the *r*-th variables, s_r^2 is the variance of the *r*-th variables.

Value

Function returns a distance matrix with the number of rows equal to the number of objects in the `x` matrix (n_x) and the number of columns equals to the number of objects in the `y` matrix (n_y).

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

Examples

```
num <- as.matrix(iris[,1:4])
mrwdist <- distNumeric(num, num, method = "mrw")
mrwdist[1:6,1:6]
```

`fastkmed` *Simple and fast k-medoid algorithm*

Description

This function runs the simple and fast k-meoid algorithm proposed by Park and Jun (2009).

Usage

```
fastkmed(distdata, ncluster, iterate = 10, init = NULL)
```

Arguments

`distdata`	A distance matrix (*n x n*) or *dist* object.
`ncluster`	A number of clusters.
`iterate`	A number of iterations for the clustering algorithm.
`init`	A vector of initial objects as the cluster medoids (*see* **Details**).

Details

The simple and fast k-medoids, which sets a set of medoids as the cluster centers, adapts the k-means algorithm for medoid up-dating. The new medoids of each iteration are calculated in the within cluster only such that it gains speed.

`init = NULL` is required because the Park and Jun (2009) has a particular method to select the initial medoids. The initial medoids are selected by

$$v_j = \sum_{i=1}^{n} \frac{d_{ij}}{\sum_{l=1}^{n} d_{il}}, \quad j = 1, 2, 3, \ldots, n$$

where the first *k* of the v_j is selected if the number of clusters is *k*.

`init` can be provided with a vector of id objects. The length of the vector has to be equal to the number of clusters. However, assigning a vector in the `init` argument, the algorithm is no longer the simple and fast k-medoids algorithm. The `inckmed` function, for example, defines a different method to select the initial medoid though it applies the `fastkmed` function.

Value

Function returns a list of components:

`cluster` is the clustering memberships result.

`medoid` is the id medoids.

`minimum_distance` is the distance of all objects to their cluster medoid.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Park, H., Jun, C., 2009. A simple and fast algorithm for k-medoids clustering. Expert Systems with Applications 36, pp. 3336-3341.

Examples

```
num <- as.matrix(iris[,1:4])
mrwdist <- distNumeric(num, num, method = "mrw")
result <- fastkmed(mrwdist, ncluster = 3, iterate = 50)
table(result$cluster, iris[,5])
```

---

globalfood *Global food security index*

Description

A dataset containing four variables of 113 countries for their food security index based on panelists evaluation in 2017.

Usage

```
globalfood
```

Format

A data frame with 113 rows and 4 variables:

affordability Index of food affordability.

availability Index of food availability.

safety Index of food quality and safety.

resilience Index of natural resources and resilience.

Source

The original indicator variables consist of 27 variables. Then, they are summarized into four pillars of food security; they are affordability, availability, quality and safety, and natural resources and resilience. Food-security expertise panelists evaluate the score of each country from 0 to 100, where 0 is the least favorable towards food security.

http://foodsecurityindex.eiu.com

---

heart — *Heart Disease data set*

Description

A mixed variable dataset containing 14 variables of 297 patients for their heart disease diagnosis.

Usage

```
heart
```

Format

A data frame with 297 rows and 14 variables:

age Age in years (numerical).

sex Sex: 1 = male, 0 = female (logical).

cp Four chest pain types: (1) typical angina, (2) atypical angina (3)non-anginal pain, (4) asymptomatic (categorical).

trestbps Resting blood pressure (in mm Hg on admission to the hospital) (numerical).

chol Serum cholestoral in mg/dl (numerical).

fbs Fasting blood sugar more than 120 mg/dl (logical).

restecg Resting electrocardiographic results: (0) normal, (1) having ST-T wave abnormality, (2) showing probable or definite left ventricular hypertrophy by Estes' criteria (categorical).

thalach Maximum heart rate achieved (numerical).

exang Exercise induced angina (logical).

oldpeak ST depression induced by exercise relative to rest (numerical).

slope The slope of the peak exercise ST segment: (1) upsloping, (2) flat, (3) downsloping (categorical).

ca Number of major vessels (0-3) colored by flourosopy (numerical).

thal (3) normal, (6) fixed defect, (7) reversable defect (categorical).

class Diagonosis of heart disease (4 classes). It can be 2 classes by setting 0 for 0 values and 1 for non-0 values.

Source

The data set is taken from machine learning repository of UCI. The original data set consists of 303 patients with 6 NA's. Then, the missing values are omitted such that it reduces into 297 patients.

https://archive.ics.uci.edu/ml/datasets/Heart+Disease

References

Lichman, M. (2013). UCI machine learning repository.

`inckmed` *Increasing number of clusters in k-medoids algorithm*

Description

This function runs the increasing number of clusters in the k-medoids algorithm proposed by Yu et. al. (2018).

Usage

```
inckmed(distdata, ncluster, iterate = 10, alpha = 1)
```

Arguments

`distdata`	A distance matrix (*n x n*) or *dist* object.
`ncluster`	A number of clusters.
`iterate`	A number of iterations for the clustering algorithm.
`alpha`	A stretch factor to determine the range of initial medoid selection (*see* **Details**).

Details

This algorithm is claimed to manage with the weakness of the simple and fast-kmedoids (`fastkmed`). The origin of the algorithm is a centroid-based algorithm by applying the Euclidean distance. Then, Bbecause the function is a medoid-based algorithm, the object mean (centroid) and variance are redefined into medoid and deviation, respectively.

The `alpha` argument is a stretch factor, i.e. a constant defined by the user. It is applied to determine a set of medoid candidates. The medoid candidates are calculated by $O_c = \{X_i | \sigma_i \leq \alpha\sigma, i = 1, 2, \ldots, n\}$, where σ_i is the average deviation of object *i*, and σ is the average deviation of the data set. They are computed by

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} d(O_i, v_1)}$$

$$\sigma_i = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} d(O_i, O_j)}$$

where *n* is the number of objects, O_i is the object *i*, and v_1 is the most centrally located object.

Value

Function returns a list of components:

`cluster` is the clustering memberships result.

`medoid` is the id medoids.

`minimum_distance` is the distance of all objects to their cluster medoid.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Yu, D., Liu, G., Guo, M., Liu, X., 2018. An improved K-medoids algorithm based on step increasing and optimizing medoids. Expert Systems with Applications 92, pp. 464-473.

Examples

```
num <- as.matrix(iris[,1:4])
mrwdist <- distNumeric(num, num, method = "mrw")
result <- inckmed(mrwdist, ncluster = 3, iterate = 50, alpha = 1.5)
table(result$cluster, iris[,5])
```

`matching` *A pair distance for binary/ categorical variables*

Description

This function computes the simple matching distance from two data frames/ matrices.

Usage

```
matching(x, y)
```

Arguments

`x`	A first data frame or matrix (*see* **Details**).
`y`	A second data frame or matrix (*see* **Details**).

Details

The `x` and `y` arguments have to be data frames/ matrices with the same number of columns where the row indicates the object and the column is the variable. This function calculates all pairwise distance between rows in the `x` and `y` data frames/ matrices. If the `x` data frame/ matrix is equal to the `y` data frame/ matrix, it explicitly calculates all distances in the `x` data frame/ matrix.

The simple matching distance between objects *i* and *j* is calculated by

$$d_{ij} = \frac{\sum_{s=1}^{P}(x_{is} - x_{js})}{P}$$

where P is the number of variables, and $x_{is} - x_{js} \in \{0, 1\}$. $x_{is} - x_{js} = 0$, if $x_{is} = x_{js}$ and $x_{is} - x_{js} = 1$, when $x_{is} \neq x_{js}$.

As an example, the distance between objects 1 and 2 is presented.

object	x	y	z
1	1	2	2
2	1	2	1

The distance between objects 1 and 2 is

$$d_{12} = \frac{\sum_{s=1}^{3}(x_{is} - x_{js})}{3} = \frac{0+0+1}{3} = \frac{1}{3} = 0.33$$

Value

Function returns a distance matrix with the number of rows equal to the number of objects in the `x` data frame/ matrix (n_x) and the number of columns equals to the number of objects in the `y` data frame/ matrix (n_y).

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

Examples

```
set.seed(1)
a <- matrix(sample(1:2, 7*3, replace = TRUE), 7, 3)
matching(a, a)
```

---

pcabiplot *Biplot of a PCA object*

Description

This function creates a biplot from a pca object, which is generated by the `prcomp` function from the **stats** package.

Usage

```
pcabiplot(PC, x = "PC1", y = "PC2", var.line = TRUE, colobj = rep(1,
  nrow(PC$x)), o.size = 1)
```

Arguments

`PC`	A pca object generated by `prcomp` function.
`x`	X axis (*see* **Details**).
`y`	Y axis (*see* **Details**).
`var.line`	A logical input, if variable lines are plotted.
`colobj`	A vector to provide color in the objects (*see* **Details**).
`o.size`	A numeric number to set the object size.

Details

This is a function to plot a pca biplot from a pca object. The x and y axes can be supplied with any principle component. The length of the `colobj` vector has to be equal to the number of objects. This argument controls the color of the objects and is very convenient to explore the clustering result. The default value is that all object have the same color.

Value

Function returns a plot of pca.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

Examples

```
pcadat <- prcomp(iris[,1:4], scale. = TRUE)
pcabiplot(pcadat)
```

`rankkmed` *Rank k-medoid algorithm*

Description

This function runs the rank k-medoids algorithm proposed by Zadegan et. al. (2013).

Usage

```
rankkmed(distdata, ncluster, m = 3, iterate = 10, init = NULL)
```

Arguments

`distdata`	A distance matrix (*n x n*) or *dist* object.
`ncluster`	A number of clusters.
`m`	A number of objects to compute hostility (*see* **Details**).
`iterate`	A number of iterations for the clustering algorithm.
`init`	A vector of initial objects as the cluster medoids (*see* **Details**).

Details

This algorithm is claimed to cope with the local optima problem of the simple and fast-kmedoids algorithm (`fastkmed`). The `m` argument is defined by the user and has to be $1 < m \leq n$. The `m` is a hostility measure computed by

$$m_i = \sum_{X_j \in Y} r_{ij}$$

where x_j is the object *j*, *Y* is the set of objects as many as *m*, and r_{ij} is the rank distance, i.e. sorted distance, between object *i* and *j*.

`init` can be provided with a vector of id objects. The length of the vector has to be equal to the number of clusters. However, assigning a vector in the `init` argument, the algorithm is no longer the rank k-medoids algorithm.

Value

Function returns a list of components:

`cluster` is the clustering memberships result.

`medoid` is the id medoids.

`minimum_distance` is the distance of all objects to their cluster medoid.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

Zadegan, S.M.R, Mirzaie M, and Sadoughi, F. 2013. Ranked k-medoids: A fast and accurate rank-based partitioning algorithm for clustering large datasets. Knowledge-Based Systems 39, 133-143.

Examples

```
num <- as.matrix(iris[,1:4])
mrwdist <- distNumeric(num, num, method = "mrw")
result <- fastkmed(mrwdist, ncluster = 3, iterate = 50)
table(result$cluster, iris[,5])
```

---

shadow — *Centroid shadow value (CSV) index of each cluster based on medoid*

Description

This function is deprecated, use the `csv` function instead.

Usage

```
shadow(distdata, idmedoid, idcluster)
```

Arguments

`distdata`	A distance object/ a n x n distance matrix.
`idmedoid`	A vector of id medoids.
`idcluster`	A vector of cluster membership.

`sil` *Silhouette index and plot*

Description

This function creates silhouette indices and silhouette plots of each cluster. The plot presents also the mean of the silhouette indices per cluster.

Usage

```
sil(distdata, idmedoid, idcluster, title = "")
```

Arguments

`distdata`	A distance matrix (*n x n*) or *dist* object.
`idmedoid`	A vector of id medoids (*see* **Details**).
`idcluster`	A vector of cluster membership (*see* **Details**).
`title`	A title of the plot.

Details

The silhouette index of object *i* is calculated by

$$si(i) = \frac{b_i - a_i}{max(a_i, b_i)}$$

where a_i is the average distance of object *i* to all objects within the cluster, and b_i is the average distance of object *i* to all objects within the nearest cluster.

The `idmedoid` argument corresponds to the `idcluster` argument. If the length of `idmedoid` is 3, for example, the `idcluster` has to have 3 unique memberships, or it returns `Error` otherwise. The length of the `idcluster` has also to be equal to *n* (the number of objects).

Value

Function returns a list with following components:

`result` is a data frame of the silhouette indices for all objects

`plot` is the silhouette plots of each cluster.

Author(s)

Weksi Budiaji
Contact: <budiaji@untirta.ac.id>

References

P. J. Rousseeuw. 1987 Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics, vol. 20, pp. 53-65

Examples

```
distiris <- as.matrix(dist(iris[,1:4]))
res <- fastkmed(distiris, 3)
silhouette <- sil(distiris, res$medoid, res$cluster)
silhouette$result[c(1:3,70:75,101:103),]
silhouette$plot
```

---

silhoutte — *Silhoutte index of each cluster*

Description

This function is deprecated, use the `sil` function instead.

Usage

```
silhoutte(distdata, idmedoid, idcluster)
```

Arguments

`distdata`	A distance object/ a n x n distance matrix.
`idmedoid`	A vector of id medoids.
`idcluster`	A vector of cluster membership.

---

stepkmed — *Step k-medoid algorithm from Yu et al.*

Description

This function is deprecated, use the `inckmed` function instead.

Usage

```
stepkmed(distdata, ncluster, iterate = 10, alpha = 1)
```

Arguments

`distdata`	A matrix of distance objects (n x n) or a diss class.
`ncluster`	A number of clusters.
`iterate`	A number of iterations for the clustering algorithm.
`alpha`	A numeric number to determine the range of initial medoids selection.

Index

∗Topic **datasets**
clust4, 3
clust5, 4
globalfood, 17
heart, 18

barplotnum, 2

clust4, 3
clust5, 4
clustboot, *5*, 5, *8*
clustheatmap, 6
consensusmatrix, *6*, 7
cooccur, 9, *13*
csv, 11, *23*

distmix, 12
distNumeric, 14

fastkmed, *5*, *8*, *16*, 16, *19*, *22*

globalfood, 17

heart, 18

inckmed, *16*, 19, *25*

matching, *13*, 20

pcabiplot, 21

rankkmed, 22

shadow, 23
sil, 24, *25*
silhoutte, 25
stepkmed, 25

Appendix B: Vignette of kmed package

Vignette of kmed package cran R can be downloaded via
https://cran.r-project.org/web/packages/kmed/vignettes/kmed.html

kmed: Distance-Based K-Medoids

Weksi Budiaji

2019-06-14

Abstract

The **kmed** vignette consists of four sequantial parts of distance-based (k-medoids) cluster analysis. The first part is defining the distance. It has numerical, binary, categorical, and mixed distances. The next part is applying a clustering algorithm in the pre-defined distance. There are four k-medoids presented, namely the simple and fast k-medoids, k-medoids, ranked k-medoids, and increasing number of clusters in k-medoids. After the clustering result is obtained, a validation step is required. The cluster validation applies internal and relative criteria. The last part is visualizing the cluster result in a pca biplot or marked barplot.

1. Introduction

The **kmed** package is designed to analyse k-medoids based clustering. The features include:

- distance computation:
  - numerical variables:
    - Manhattan weighted by range
    - squared Euclidean weighted by range
    - squared Euclidean weighted by squared range
    - squared Euclidean weighted by variance
    - unweighted squared Euclidean
  - binary or categorical variables:
    - simple matching
    - co-occurrence
  - mixed variables:
    - Gower
    - Wishart
    - Podani
    - Huang
    - Harikumar and PV
    - Ahmad and Dey
- k-medoids algorithms:
  - Simple and fast k-medoids
  - K-medoids
  - Rank k-medoids
  - Increasing number of clusters k-medoids
- cluster validations:
  - internal criteria:
    - Silhouette

        - Centroid-based shadow value
    - relative criteria (bootstrap)
- Cluster visualizations:
    - pca biplot
    - marked barplot

2. Distance Computation

2.A. Numerical variables (`distNumeric`)

The `distNumeric` function can be applied to calculate numerical distances. There are four distance options, namely Manhattan weighted by range (`mrw`), squared Euclidean weighted by range (`ser`), squared Euclidean weighted by squared range (`ser.2`), squared Euclidean weighted by variance (`sev`), and unweighted squared Euclidean (`se`). The `distNumeric` function provides `method` in which the desired distance method can be selected. The default `method` is `mrw`.

The distance computation in a numerical variable data set is performed in the iris data set. An example of manual calculation of the numerical distances is applied for the first and second objects only to introduce what the `distNumeric` function does.

```
library(kmed)
```

```
## Warning: package 'kmed' was built under R version 3.5.3
```

```
iris[1:3,]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

2.A.1. Manhattan weighted by range (`method = "mrw"`)

By applying the `distNumeric` function with `method = "mrw"`, the distance among objects in the iris data set can be obtained.

```
num <- as.matrix(iris[,1:4])
rownames(num) <- rownames(iris)
#calculate the Manhattan weighted by range distance of all iris objects
mrwdist <- distNumeric(num, num)
#show the distance among objects 1 to 3
mrwdist[1:3,1:3]
```

```
##           1         2         3
## 1 0.0000000 0.2638889 0.2530603
## 2 0.2638889 0.0000000 0.1558380
## 3 0.2530603 0.1558380 0.0000000
```

The Manhattan weighted by range distance between objects 1 and 2 is `0.2638889`. To calculate this distance, the range of each variable is computed.

```
#extract the range of each variable
apply(num, 2, function(x) max(x)-min(x))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          3.6          2.4          5.9          2.4
```

Then, the distance between objects 1 and 2 is

```
#the distance between objects 1 and 2
abs(5.1-4.9)/3.6 + abs(3.5 - 3.0)/2.4 + abs(1.4-1.4)/5.9 +
  abs(0.2-0.2)/2.4
```

```
## [1] 0.2638889
```

which is based on the data

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
```

(Back to Intoduction)

2.A.2. squared Euclidean weighted by range (`method = "ser"`)

```
#calculate the squared Euclidean weighthed by range distance of
#all iris objects
serdist <- distNumeric(num, num, method = "ser")
#show the distance among objects 1 to 3
serdist[1:3,1:3]
```

```
##            1          2          3
## 1 0.00000000 0.11527778 0.08363936
## 2 0.11527778 0.00000000 0.02947269
## 3 0.08363936 0.02947269 0.00000000
```

The squared Euclidean weighted by range distance between objects 1 and 2 is `0.11527778`. It is obtained by

```
#the distance between objects 1 and 2
(5.1-4.9)^2/3.6 + (3.5 - 3.0)^2/2.4 + (1.4-1.4)^2/5.9 + (0.2-0.2)^2/2.4
```

```
## [1] 0.1152778
```

(Back to Intoduction)

2.A.3. squared Euclidean weighted by squared range (`method = "ser.2"`)

```r
#calculate the squared Euclidean weighthed by squared range distance of
#all iris objects
ser.2dist <- distNumeric(num, num, method = "ser.2")
#show the distance among objects 1 to 3
ser.2dist[1:3,1:3]
```

```
##            1          2          3
## 1 0.00000000 0.04648920 0.02825795
## 2 0.04648920 0.00000000 0.01031814
## 3 0.02825795 0.01031814 0.00000000
```

The squared Euclidean weighted by squared range distance between objects 1 and 2 is `0.04648920` that is computed by

```r
(5.1-4.9)^2/3.6^2 + (3.5 - 3.0)^2/2.4^2 + (1.4-1.4)^2/5.9^2 +
  (0.2-0.2)^2/2.4^2
```

```
## [1] 0.0464892
```

where the data are

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
```

(Back to Intoduction)

2.A.4. squared Euclidean weighted by variance (`method = "sev"`)

```r
#calculate the squared Euclidean weighthed by variance distance of
#all iris objects
sevdist <- distNumeric(num, num, method = "sev")
#show the distance among objects 1 to 3
sevdist[1:3,1:3]
```

```
##           1         2         3
## 1 0.0000000 1.3742671 0.7102849
## 2 1.3742671 0.0000000 0.2720932
## 3 0.7102849 0.2720932 0.0000000
```

The squared Euclidean weighted by variance distance between objects 1 and 2 is `1.3742671`. To compute this distance, the variance of each variable is calculated.

```r
#calculate the range of each variable
apply(num[,1:4], 2, function(x) var(x))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.6856935    0.1899794    3.1162779    0.5810063
```

Then, the distance between objects 1 and 2 is

```r
(5.1-4.9)^2/0.6856935 + (3.5 - 3.0)^2/0.1899794 + (1.4-1.4)^2/3.1162779 +
  (0.2-0.2)^2/0.5810063
```

```
## [1] 1.374267
```

(Back to Intoduction)

2.A.5. squared Euclidean (`method = "se"`)

```r
#calculate the squared Euclidean distance of all iris objects
sedist <- distNumeric(num, num, method = "se")
#show the distance among objects 1 to 3
sedist[1:3,1:3]
```

```
##      1    2    3
## 1 0.00 0.29 0.26
## 2 0.29 0.00 0.09
## 3 0.26 0.09 0.00
```

The squared Euclidean distance between objects 1 and 2 is `0.29`. It is computed by

```r
(5.1-4.9)^2 + (3.5 - 3.0)^2 + (1.4-1.4)^2 + (0.2-0.2)^2
```

```
## [1] 0.29
```

(Back to Intoduction)

2.B. Binary or Categorical variables

There are two functions to calculate the binary and categorical variables. The first is `matching` to compute the simple matching distance and the second is `cooccur` to calculate the co-occurrence distance. To introduce what these functions do, the `bin` data set is generated.

```r
set.seed(1)
bin <- matrix(sample(1:2, 4*2, replace = TRUE), 4, 2)
rownames(bin) <- 1:nrow(bin)
colnames(bin) <- c("x", "y")
```

2.B.1. Simple matching (`matching`)

The `matching` function calculates the simple matching distance between two data sets. If the two data sets are identical, the functions calculates the distance among objects within the data set. The simple matching distance is equal to the proportion of the mis-match categories.

```r
bin
```

```
##   x y
## 1 1 1
## 2 1 2
## 3 2 2
## 4 2 2
```

```r
#calculate simple matching distance
matching(bin, bin)
```

```
##     1   2   3   4
## 1 0.0 0.5 1.0 1.0
## 2 0.5 0.0 0.5 0.5
## 3 1.0 0.5 0.0 0.0
## 4 1.0 0.5 0.0 0.0
```

As an example of the simple matching distance, the distance between objects 1 and 2 is calculated by

```r
((1 == 1) + (1 == 2))/ 2
```

```
## [1] 0.5
```

The distance between objects 1 and 2, which is `0.5`, is produced from *one mis-match* and *one match* categories from the two variables (`x` and `y`) in the `bin` data set. When `x1` is equal to `x2`, for instance, the score is 0. Meanwile, if `x1` is not equal to `x2`, the score is 1. These scores are also valid in the `y` variable. Hence, the distance between objects 1 and 2 is `(0+1)/2` that is equal to `1/2`.

(Back to Intoduction)

2.B.2. Co-occurrence distance (`cooccur`)

The co-ocurrence distance (Ahmad and Dey 2007; Harikumar and PV 2015) can be calculated via the `cooccur` function. To calculate the distance between objects, the distribution of the variables are taken into consideration. Compared to the simple matching distance, the co-occurrence distance redefines the score of **match** and **mis-match** categories such that they are *unnecessary* to be `0` and `1`, respectively. Due to relying on the distribution of all inclusion variables, the co-occurence distance of a data set with a single variable is **absent**.

The co-occurrence distance of the `bin` data set is

```r
#calculate co-occurrence distance
cooccur(bin)
```

```
##           1         2        3        4
## 1 0.0000000 0.6666667 1.166667 1.166667
## 2 0.6666667 0.0000000 0.500000 0.500000
## 3 1.1666667 0.5000000 0.000000 0.000000
## 4 1.1666667 0.5000000 0.000000 0.000000
```

To show how co-occurrence distance is calculated, the distance between objects 1 and 2 is presented.

```
bin
```

```
##   x y
## 1 1 1
## 2 1 2
## 3 2 2
## 4 2 2
```

Step 1 Creating cross tabulations

```
#cross tabulation to define score in the y variable
(tab.y <- table(bin[,'x'], bin[,'y']))
```

```
##
##     1 2
##   1 1 1
##   2 0 2
```

```
#cross tabulation to define score in the x variable
(tab.x <- table(bin[,'y'], bin[,'x']))
```

```
##
##     1 2
##   1 1 0
##   2 1 2
```

Step 2 Calculating the column proportions of each cross tabulation

```
#proportion in the y variable
(prop.y <- apply(tab.y, 2, function(x) x/sum(x)))
```

```
##
##     1         2
##   1 1 0.3333333
##   2 0 0.6666667
```

```
#proportion in the x variable
(prop.x <- apply(tab.x, 2, function(x) x/sum(x)))
```

```
##
##       1 2
##   1 0.5 0
##   2 0.5 1
```

Step 3 Finding the maximum values for each row of the proportion

```
#maximum proportion in the y variable
(max.y <- apply(prop.y, 2, function(x) max(x)))
```

```
##         1         2
## 1.0000000 0.6666667
```

```r
#maximum proportion in the x variable
(max.x <- apply(prop.x, 2, function(x) max(x)))
```

```
##   1   2
## 0.5 1.0
```

Step 4 Defining the scores of each variable

The score is obtained by a summation of the maximum value subtracted and divided by a constant. The constant has a value depending on the number of inclusion variables. For the `bin` data set, the constant is `1` because both `x` and `y` variables are only depended on *one* other variable, i.e. `x` depends on the distribution of `y` and `y` relies on the distribution of `x`.

```r
#score mis-match in the y variable
(sum(max.y) - 1)/1
```

```
## [1] 0.6666667
```

```r
#score mis-match in the x variable
(sum(max.x) - 1)/1
```

```
## [1] 0.5
```

It can be implied that the score for mis-match categories are `0.5` and `0.67` in the `x` and `y` variables, respectively. Note that the score for **match** categories is **alwalys** `0`. Thus, the distance between objects 1 and 2 is `0+0.6666667 = 0.6666667` and between objects 1 and 3 is `0.5+0.6666667 = 1.1666667`

(Back to Intoduction)

2.C. Mixed variables (`distmix`)

There are six available distance methods for a mixed variable data set. The `distmix` function calculates mixed variable distance in which it requires *column id* of each class of variables. The `mixdata` data set is generated to describe each method in the `distmix` function.

```r
cat <- matrix(c(1, 3, 2, 1, 3, 1, 2, 2), 4, 2)
mixdata <- cbind(iris[c(1:2, 51:52),3:4], bin, cat)
rownames(mixdata) <- 1:nrow(mixdata)
colnames(mixdata) <- c(paste(c("num"), 1:2, sep = ""),
                       paste(c("bin"), 1:2, sep = ""),
                       paste(c("cat"), 1:2, sep = ""))
```

2.C.1 Gower (`method = "gower"`)

The `method = "gower"` in the `distmix` function calculates the Gower (1971) distance. The original Gower distance allows missing values, while it is not allowed in the `distmix` function.

```
mixdata
```

```
##   num1 num2 bin1 bin2 cat1 cat2
## 1  1.4  0.2    1    1    1    3
## 2  1.4  0.2    1    2    3    1
## 3  4.7  1.4    2    2    2    2
## 4  4.5  1.5    2    2    1    2
```

The Gower distance of the `mixdata` data set is

```
#calculate the Gower distance
distmix(mixdata, method = "gower", idnum = 1:2,
        idbin = 3:4, idcat = 5:6)
```

```
##           1         2         3         4
## 1 0.0000000 0.5000000 0.9871795 0.8232323
## 2 0.5000000 0.0000000 0.8205128 0.8232323
## 3 0.9871795 0.8205128 0.0000000 0.1895882
## 4 0.8232323 0.8232323 0.1895882 0.0000000
```

As an example, the distance between objects 3 and 4 is presented. The range of each numerical variables is necessary.

```
#extract the range of each numerical variable
apply(mixdata[,1:2], 2, function(x) max(x)-min(x))
```

```
## num1 num2
##  3.3  1.3
```

The Gower distance calculates the Gower similarity first. In the Gower similarity, the **mis-match** categories in the binary/ categorical variables are scored **0** and the **match** categories are **1**. Meanwhile, in the numerical variables, 1 is subtracted by a ratio between the absolute difference and its range. Then, the Gower similarity can be weighted by the number of variables. Thus, the Gower similarity between objects 3 and 4 is

```
#the Gower similarity
(gowsim <- ((1-abs(4.7-4.5)/3.3) + (1-abs(1.4-1.5)/1.3) +
              1 + 1 + 0 + 1)/ 6 )
```

```
## [1] 0.8104118
```

The Gower distance is obtained by subtracting 1 with the Gower similarity. The distance between objects 3 and 4 is then

```
#the Gower distance
1 - gowsim
```

```
## [1] 0.1895882
```

(Back to Intoduction)

2.C.2 Wishart (`method = "wishart"`)

The Wishart (2003) distance can be calculated via `method = "wishart"`. Although it allows missing values, it is again illegitimate in the `distmix` function. The Wishart distance for the `mixdata` is

```
#calculate the Wishart distance
distmix(mixdata, method = "wishart", idnum = 1:2,
        idbin = 3:4, idcat = 5:6)
```

```
##           1         2         3         4
## 1 0.0000000 0.7071068 1.2871280 1.2277616
## 2 0.7071068 0.0000000 1.2206686 1.2277616
## 3 1.2871280 1.2206686 0.0000000 0.4144946
## 4 1.2277616 1.2277616 0.4144946 0.0000000
```

To calculate the Wishart distance, the variance of each numerical variable is required. It weighs the squared difference of a numerical variable.

```
#extract the variance of each numerical variable
apply(mixdata[,1:2], 2, function(x) var(x))
```

```
##   num1   num2
## 3.4200 0.5225
```

Meanwhile, the **mis-match** categories in the binary/ categorical variables are scored **1** and the **match** categories are **0**. Then, all score of the variables is added and squared rooted. Thus, the distance between objects 3 and 4 is

```
wish <- (((4.7-4.5)^2/3.42) + ((1.4-1.5)^2/0.5225) + 0 + 0 + 1 + 0)/ 6
#the Wishart distance
sqrt(wish)
```

```
## [1] 0.4144946
```

(Back to Intoduction)

2.C.3 Podani (`method = "podani"`)

The `method = "podani"` in the `distmix` function calculates the Podani (1999) distance. Similar to The Gower and Wishart distances, it allows missing values, yet it is not allowed in the `distmix` function. The Podani distance for the `mixdata` is

```
#calculate Podani distance
distmix(mixdata, method = "podani", idnum = 1:2, idbin = 3:4, idcat = 5:6)
```

```
##          1        2        3        4
## 1 0.000000 1.732051 2.419105 2.209629
```

```
## 2 1.732051 0.000000 2.202742 2.209629
## 3 2.419105 2.202742 0.000000 1.004784
## 4 2.209629 2.209629 1.004784 0.000000
```

The Podani and Wishart distances are similar. They are different in the denumerator for the numerical variables. Instead of a variance, the Podani distance applies the squared range for a numerical variable. Unlike the Gower and Podani distances, the number of variables as a weight is absent in the Podani distance. Hence, the distance between objects 3 and 4 is

```r
poda <- ((4.7-4.5)^2/3.3^2) + ((1.4-1.5)^2/1.3^2) + 0 + 0 + 1 + 0
#the Podani distance
sqrt(poda)
```

```
## [1] 1.004784
```

which is based on data

```
##   num1 num2 bin1 bin2 cat1 cat2
## 3  4.7  1.4    2    2    2    2
## 4  4.5  1.5    2    2    1    2
```

(Back to Intoduction)

2.C.4 Huang (`method = "huang"`)

The `method = "huang"` in the `distmix` function calculates the Huang (1997) distance. The Huang distance of the `mixdata` data set is

```r
#calculate the Huang distance
distmix(mixdata, method = "huang", idnum = 1:2,
        idbin = 3:4, idcat = 5:6)
```

```
##           1         2         3         4
## 1  0.000000  3.858249 17.474332 15.158249
## 2  3.858249  0.000000 16.188249 15.158249
## 3 17.474332 16.188249  0.000000  1.336083
## 4 15.158249 15.158249  1.336083  0.000000
```

The average standard deviation of the numerical variables is required to calculate the Huang distance. This measure weighs the binary/ categorical variables.

```r
#find the average standard deviation of the numerical variables
mean(apply(mixdata[,1:2], 2, function(x) sd(x)))
```

```
## [1] 1.286083
```

While the squared difference of the numerical variables is calculated, the **mis-match** categories are scored **1** and the **match** categories are **0** in the binary/ categorical variables. Thus, the distance between objects 3 and 4 is

```r
(4.7-4.5)^2 + (1.4-1.5)^2 + 1.286083*(0 + 0) + 1.286083*(1 + 0)
```

```
## [1] 1.336083
```

(Back to Intoduction)

2.C.5 Harikumar and PV (`method = "harikumar"`)

The Harikumar and PV (2015) distance can be calculated via `method = "harikumar"`. The Harikumar and PV distance for the `mixdata` is

```r
#calculate Harikumar-PV distance
distmix(mixdata, method = "harikumar", idnum = 1:2,
        idbin = 3:4, idcat = 5:6)
```

```
##     1   2   3   4
## 1 0.0 3.0 7.5 6.9
## 2 3.0 0.0 7.5 7.4
## 3 7.5 7.5 0.0 0.8
## 4 6.9 7.4 0.8 0.0
```

The Harikumar and PV distance requires an absolute difference in the numerical variables and unweighted simple matching, i.e. Hamming distance, in the binary variables. For the categorical variables, it applies co-occurrence distance. The co-occurence distance in the categorical variables is (for manual calculation see co-occurrence subsection)

```r
cooccur(mixdata[,5:6])
```

```
##     1 2   3   4
## 1 0.0 2 1.0 0.5
## 2 2.0 0 2.0 2.0
## 3 1.0 2 0.0 0.5
## 4 0.5 2 0.5 0.0
```

Hence, the distance between objects 1 and 3 is

```r
abs(4.7-4.5) + abs(1.4-1.5) + (0 + 0) + (0.5)
```

```
## [1] 0.8
```

where the data are

```
##   num1 num2 bin1 bin2 cat1 cat2
## 3  4.7  1.4    2    2    2    2
## 4  4.5  1.5    2    2    1    2
```

(Back to Intoduction)

2.C.6 Ahmad and Dey (`method = "ahmad"`)

The `method = "ahmad"` in the `distmix` function calculates the Ahmad and Dey (2007) distance. The Ahmad and Dey distance of the `mixdata` data set is

```
#calculate Ahmad-Dey distance
distmix(mixdata, method = "ahmad", idnum = 1:2,
        idbin = 3:4, idcat = 5:6)
```

```
##          1        2        3        4
## 1  0.00000  4.45679 20.04605 16.48827
## 2  4.45679  0.00000 16.33000 15.30000
## 3 20.04605 16.33000  0.00000  0.30000
## 4 16.48827 15.30000  0.30000  0.00000
```

The Ahmad and dey distance requires a squared difference in the numerical variables and co-occurrence distance for both the binary and categorical variables. The co-occurrence distance in the `mixdata` data set is

```
cooccur(mixdata[,3:6])
```

```
##          1        2        3        4
## 1 0.000000 2.111111 2.777778 2.277778
## 2 2.111111 0.000000 2.000000 2.000000
## 3 2.777778 2.000000 0.000000 0.500000
## 4 2.277778 2.000000 0.500000 0.000000
```

Thus, the distance between objects 2 and 3 is

```
(1.4-4.7)^2 + (0.2-1.4)^2 + (2)^2
```

```
## [1] 16.33
```

which is based on the data

```
##   num1 num2 bin1 bin2 cat1 cat2
## 2  1.4  0.2    1    2    3    1
## 3  4.7  1.4    2    2    2    2
```

(Back to Intoduction)

3. K-medoids algorithms

There are some k-medoids algorithms available in this package. They are the simple and fast k-medoids (`fastkmed`), k-medoids, ranked k-medoids (`rankkmed`), and increasing number of clusters k-medoids (`inckmed`). All algorithms have a list of results, namely the cluster membership, id medoids, and distance of all objects to their medoid.

In this section, the algorithms are applied in the `iris` data set by applying the `mrw` distance (see Manhattan weighted by range). The number of clusters in this data set is 3.

3.A. Simple and fast k-medoids algorithm (`fastkmed`)

The simple and fast k-medoid (SFKM) algorithm has been proposed by Park and Jun (2009). The `fastkmed` function runs this algorithm to cluster the objects. The compulsory inputs are a distance matrix or distance object and a number of clusters. Hence, the SFKM algorithm for the `iris` data set is

```
#run the sfkm algorihtm on iris data set with mrw distance
(sfkm <- fastkmed(mrwdist, ncluster = 3, iterate = 50))
```

```
## $cluster
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   3   3   3   2
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##   3   2   3   2   2   2   2   2   2   2   2   3   2   2   2   2   3   2
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   2   2   2   3   3   3   2   2   2   2   2   2   2   2   3   2   2   2
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   2   2   2   2   2   2   2   2   2   2   3   3   3   3   3   3   2   3
## 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
##   3   3   3   3   3   3   3   3   3   3   3   2   3   3   3   3   3   3
## 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
##   3   3   3   3   3   3   3   3   2   3   3   3   3   3   3   3   3   3
## 145 146 147 148 149 150
##   3   3   3   3   3   3
##
## $medoid
## [1]   8  95 148
##
## $minimum_distance
## [1] 48.76718
```

Then, a classification table can be obtained.

```
(sfkmtable <- table(sfkm$cluster, iris[,5]))
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         39         3
##   3      0         11        47
```

Applying the SFKM algorithm in `iris` data set with the Manhattan weighted by range, the misclassification rate is

```r
(3+11)/sum(sfkmtable)
```

```
## [1] 0.09333333
```

(Back to Intoduction)

3.B. K-medoids algorithm

Reynolds et al. (2006) has been proposed a k-medoids (KM) algorithm. It is similar to the SFKM such that the `fastkmed` can be applied. The difference is in the initial medoid selection where the KM selects the initial medoid randomly. Thus, the KM algorithm for the `iris` data set by setting the `init` is

```r
#set the initial medoids
set.seed(1)
(kminit <- sample(1:nrow(iris), 3))
```

```
## [1] 40 56 85
```

```r
#run the km algorihtm on iris data set with mrw distance
(km <- fastkmed(mrwdist, ncluster = 3, iterate = 50, init = kminit))
```

```
## $cluster
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   3   3   3   2
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##   2   2   3   2   2   2   2   2   2   2   2   3   2   2   2   2   3   2
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   2   2   2   3   2   3   2   2   2   2   2   2   2   2   3   2   2   2
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   2   2   2   2   2   2   2   2   2   2   3   3   3   3   3   3   2   3
## 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
##   3   3   3   3   3   3   3   3   3   3   3   2   3   3   3   3   3   3
## 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
##   3   3   3   3   3   3   3   3   2   3   3   3   3   3   3   3   3   3
## 145 146 147 148 149 150
##   3   3   3   3   3   3
##
## $medoid
## [1]   8 100 148
##
## $minimum_distance
## [1] 48.8411
```

The classification table of the KM algorithm is

```
(kmtable <- table(km$cluster, iris[,5]))
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         41         3
##   3      0          9        47
```

with the misclassification rate

```
(3+9)/sum(kmtable)
```

```
## [1] 0.08
```

Compared to the SFKM algorithm, which has `9.33`% misclassification, the misclassification of the KM algorithm is slightly better (8%).

(Back to Intoduction)

3.C. Rank k-medoids algorithm (`rankkmed`)

A rank k-medoids (RKM) has been proposed by Zadegan, Mirzaie, and Sadoughi (2013). The `rankkmed` function runs the RKM algorithm. The `m` argument is introduced to calculate a hostility score. The `m` indicates how many closest objects is selected. The selected objects as initial medoids in the RKM is randomly assigned. The RKM algorithm for the `iris` data set by setting `m = 10` is then

```
#run the rkm algorihtm on iris data set with mrw distance and m = 10
(rkm <- rankkmed(mrwdist, ncluster = 3, m = 10, iterate = 50))
```

```
## $cluster
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2   2   2   2
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   3   2
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   2   2   2   2   2   3   2   2   2   2   2   3   2   2   2   2   2   2
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   2   2   2   2   2   2   2   2   2   2   3   3   3   3   3   3   3   3
## 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
##   3   3   3   3   3   3   3   3   3   3   3   2   3   3   3   3   3   3
## 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
##   3   3   3   3   3   3   3   2   2   3   3   3   3   3   3   3   3   3
```

```
## 145 146 147 148 149 150
##   3   3   3   3   3   3
##
## $medoid
## [1] "50"  "92"  "128"
##
## $minimum_distance
## [1] 56.71822
```

Then, a classification table is attained by

```
(rkmtable <- table(rkm$cluster, iris[,5]))
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         47         3
##   3      0          3        47
```

The misclassification proportion is

```
(3+3)/sum(rkmtable)
```

```
## [1] 0.04
```

With 4% misclassification rate, the RKM algorithm is the best among the three previous algorithms.

(Back to Intoduction)

3.D. Increasing number of clusters k-medoids algorithm (`inckmed`)

Yu et al. (2018) has been proposed an increasing number of clusters k-medoids (INCKM) algorithm. This algorithm is implemented in the `inckmed` function. The `alpha` argument indicates a stretch factor to select the initial medoids. The SFKM, KM and INCKM are similar algorithm with a different way to select the initial medoids. The INCKM algorithm of the `iris` data set with `alpha = 1.1` is

```
#run the inckm algorihtm on iris data set with mrw distance
#and alpha = 1.2
(inckm <- inckmed(mrwdist, ncluster = 3, alpha = 1.1, iterate = 50))
```

```
## $cluster
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   2   2   2   1
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
```

```
##   1   1   2   1   1   1   1   1   1   1   1   2   1   1   1   1   2   1
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   1   1   1   2   1   2   1   1   1   1   1   1   1   1   2   1   1   1
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   1   1   1   1   1   1   1   1   1   1   2   2   2   2   2   2   1   2
## 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
##   2   2   2   2   2   2   2   2   2   2   2   1   2   2   2   2   2   2
## 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
##   2   2   2   2   2   2   2   2   1   2   2   2   2   2   2   2   2   2
## 145 146 147 148 149 150
##   2   2   2   2   2   2
##
## $medoid
## [1] 100 148   8
##
## $minimum_distance
## [1] 48.8411
```

Then, the classification table can be attained.

```r
(inckmtable <- table(inckm$cluster, iris[,5]))
```

```
##
##     setosa versicolor virginica
##   1      0         41         3
##   2      0          9        47
##   3     50          0         0
```

The misclassification rate is

```r
(9+3)/sum(inckmtable)
```

```
## [1] 0.08
```

The algorithm has 8% misclassification rate such that the RKM algorithm performs the best among the four algorithms in the `iris` data set with the `mrw` distance.

(Back to Intoduction)

4. Cluster validation

The clustering algorithm result has to be validated. There are two types of validation implemented in the **kmed** package. They are internal and relative criteria validations.

4.A. Internal criteria

4.A.1. Silhouette (`sil`)

Rousseeuw (1987) has proposed a silhouette index as an internal measure of validation. It is based on the average distance of objects within a cluster and between the nearest cluster. The `sil` function calculates the silhouette index of clustering result. The arguments are a distance matrix or distance object, id medoids, and cluster membership. It produce a list of silhouette indices and sihouette plots.

The silhouette index and plot of the best clustering result of `iris` data set via RKM is presented.

```
#calculate silhouette of the RKM result of iris data set
siliris <- sil(mrwdist, rkm$medoid, rkm$cluster,
                      title = "Silhouette plot of Iris data set via RKM")
```

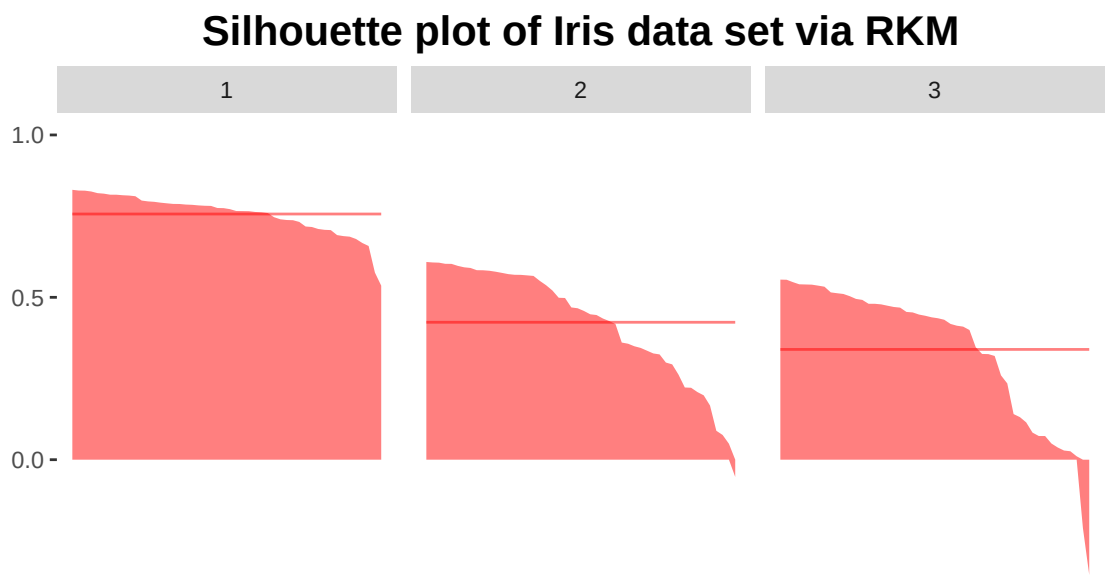
The silhouette index of each object can be obtained by

```
#silhouette indices of objects 49 to 52
siliris$result[c(49:52),]
```

```
##    silhouette cluster
## 49 0.78952089       1
## 50 0.82084673       1
## 51 0.07607567       2
## 52 0.22234719       2
```

Then, the plot is presented by

```
siliris$plot
```



(Back to Intoduction)

4.A.2. Centroid-based shadow value (csv)

An other way to measure internal validation with its corresponding plot is by presenting the centroid-based shadow value (Leisch 2010). The `csv` function calculates and plots the shadow value of each object, which is based on the first and second closest medoids. The centroid of the original version of the csv is replaced by medoids in the `csv` function to adapt the k-medoids algorithm.

The required arguments in the `csv` function is identical to the silhouette (`sil`) function. Thus, the shadow value and plot of the best clustering result of `iris` data set via RKM can be obtained by

```
#calculate shadow value of the RKM result of iris data set
csviris <- csv(mrwdist, rkm$medoid, rkm$cluster,
                      title = "Shadow value plot of Iris data set via RKM")
```

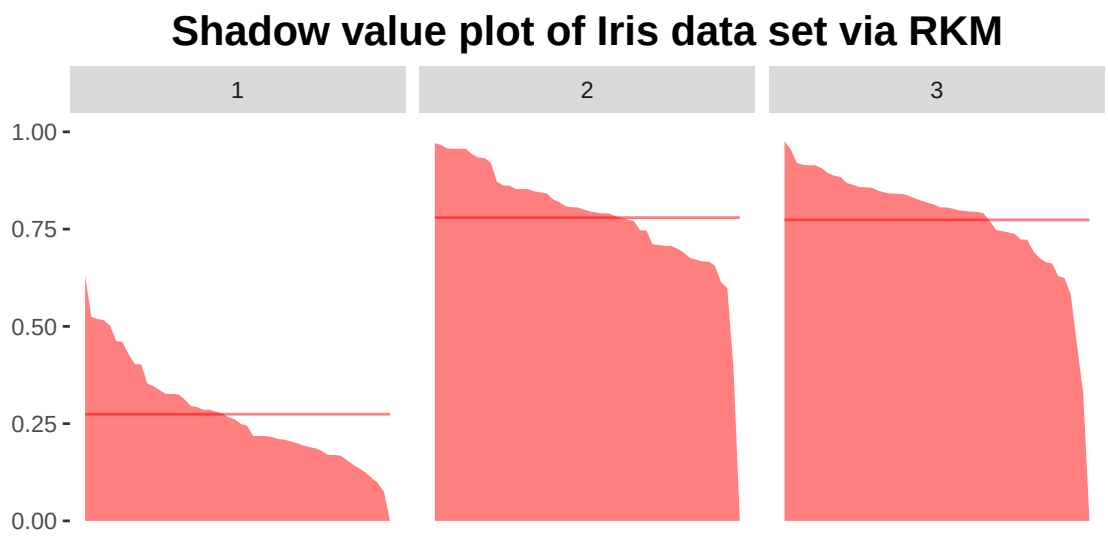
The shadow values of objects 49 to 52, for instance, are presented by

```
#shadow values of objects 49 to 52
csviris$result[c(49:52),]
```

```
##      shadval cluster
## 49 0.2955819       1
## 50 0.0000000       1
## 51 0.7923323       2
## 52 0.7705314       2
```

The shadow value plot is also produced.

```
csviris$plot
```



(Back to Intoduction)

4.B. Relative criteria

The relative criteria evaluate a clustering algorithm result by applying re-sampling strategy. Thus, a bootstrap strategy can be applied. It is expected that the result of the cluster bootstraping is robust over all replications (Dolnicar and Leisch 2010). There are three steps to validate the cluster result via the boostraping strategy.

Step 1 Creating a matrix of bootstrap replicates

To create a matrix of bootstrap replicates, the `clustboot` function can be applied. There are five arguments in the `clustboot` function with the `algorithm` argument being the most important. The `algorithm` argument is the argument for a clustering algorithm that a user wants to evaluate. It has to be a *function*. When the RKM of `iris` data set is validated, for instance, the RKM function, which is required as an input in the `algorithm` argument, is

```
#The RKM function for an argument input
rkmfunc <- function(x, nclust) {
  res <- rankkmed(x, nclust, m = 10, iterate = 50)
  return(res$cluster)
}
```

When a function is created, it has to have two input arguments. They are `x` (a distance matrix) and `nclust` (a number of clusters). The output, on the other hand, is *a vector* of cluster membership (`res$cluster`). Thus, the matrix of bootstrap replicates can be produced by

```
#The RKM algorthim evaluation by inputing the rkmfunc function
#in the algorithm argument
rkmbootstrap <- clustboot(mrwdist, nclust=3, nboot=50,
                          algorithm = rkmfunc)
```

with the objects 1 to 4 on the first to fifth replications being

```
rkmbootstrap[1:4,1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    1    1
## [2,]    2    0    1    1    1
## [3,]    2    0    0    0    0
## [4,]    1    0    0    1    0
```

The `rkmbootstrap` is a matrix of bootrstrap replicates with a dimension of *150 x 50*, i.e. n x b, where n is the number of objects and b is the number of bootstrap replicates. **Note** that the default evaluated algorithm is the SFKM algorithm such that if a user ignores the

algorithm argument, the matrix of bootstrap replicates can still be produced. However, it misleads because it does not evaluate the user's own algorithm.

Step 2 Transforming the bootstrap matrix into a consensus matrix

The matrix of bootstrap replicates produced by the clustboot in the step 1 can be transformed into a consensus matrix with a dimension of n x n via the consensusmatrix function. An element of the consensus matrix in row i dan column j is an agreement value between objects i and j to be in the same cluster when they are taken as a sample at the same time (Monti et al. 2003).

However, it requires an algorithm to order the objects in such a way that objects in the same cluster are close to each other. The consensusmatrix function has the reorder argument to comply this task. It is similar to the algorithm argument in the clustboot function in the step 1 where the reorder has to be a function that has two arguments and a vector of output.

Transforming the rkmbootstrap into a consensus matrix via the ward linkage algorithm to oder the objects, for example, can obtained by

```
#The ward function to order the objects in the consensus matrix
wardorder <- function(x, nclust) {
  res <- hclust(as.dist(x), method = "ward.D2")
  member <- cutree(res, nclust)
  return(member)
}
consensusrkm <- consensusmatrix(rkmbootstrap, nclust = 3, wardorder)
```

The first to fourth rows and columns can be displayed as

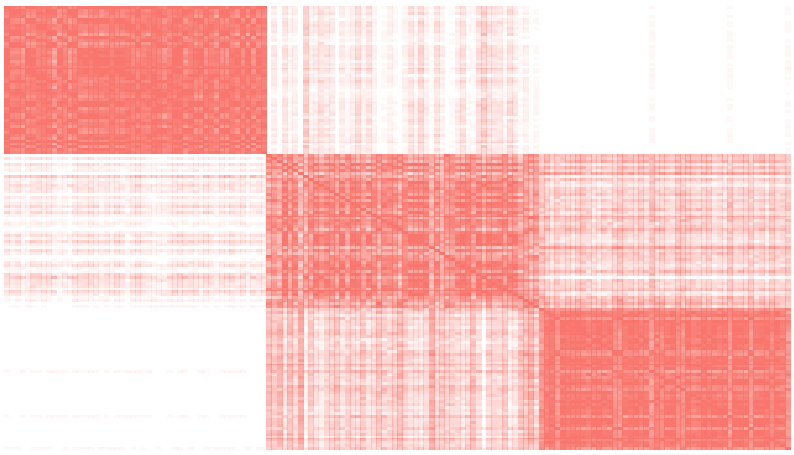
```
consensusrkm[c(1:4),c(1:4)]
```

```
##           1         1         1         1
## 1 1.0000000 0.9583333 0.9130435 0.8421053
## 1 0.9583333 1.0000000 1.0000000 0.9565217
## 1 0.9130435 1.0000000 1.0000000 0.9500000
## 1 0.8421053 0.9565217 0.9500000 1.0000000
```

Step 3 Visualizing the consensus matrix in a heatmap

The ordered consensus matrix in the step 2 can be visualized in a heatmap applying the clustheatmap function. The agreement indices in the consensus matrix can be transformed via a non-linear transformation (Hahsler and Hornik 2011). Thus, the consensusrkm can visualize into

```
clustheatmap(consensusrkm,
             "Iris data evaluated by the RKM, ordered by Ward linkage")
```



Iris data evaluated by the RKM, ordered by Ward linka

(Back to Intoduction)

5. Cluster visualization

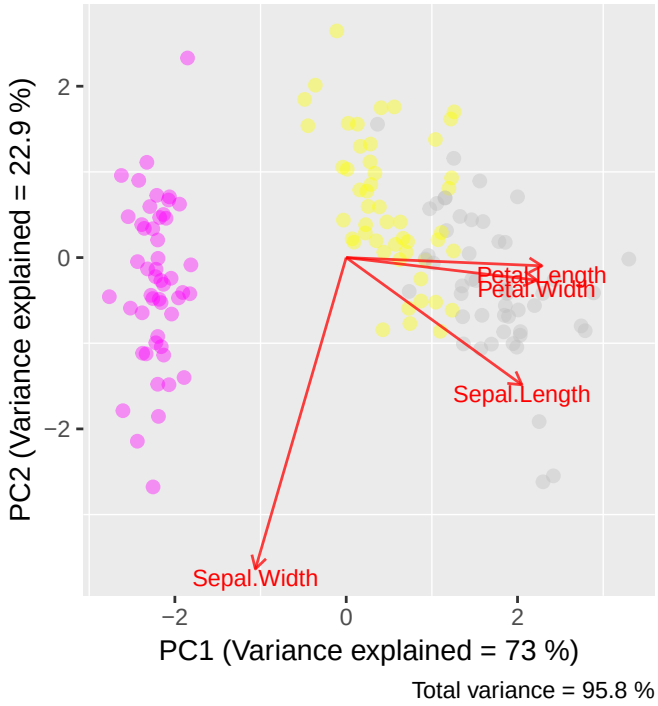
A cluster visualization of the clustering result can enhance the data structure understanding. The biplot and marked barplot are presented to visualize the clustering result.

A. Biplot

The `pcabiplot` function can be applied to plot a clustering result from a numerical data set. The numerical data set has to be converted into a principle component object via the `prcomp` function. The `x` and `y` axes in the plot can be replaced by any component of the principle components. The colour of the objects can be adjusted based on the cluster membership by supplying a vector of membership in the `colobj` argument.

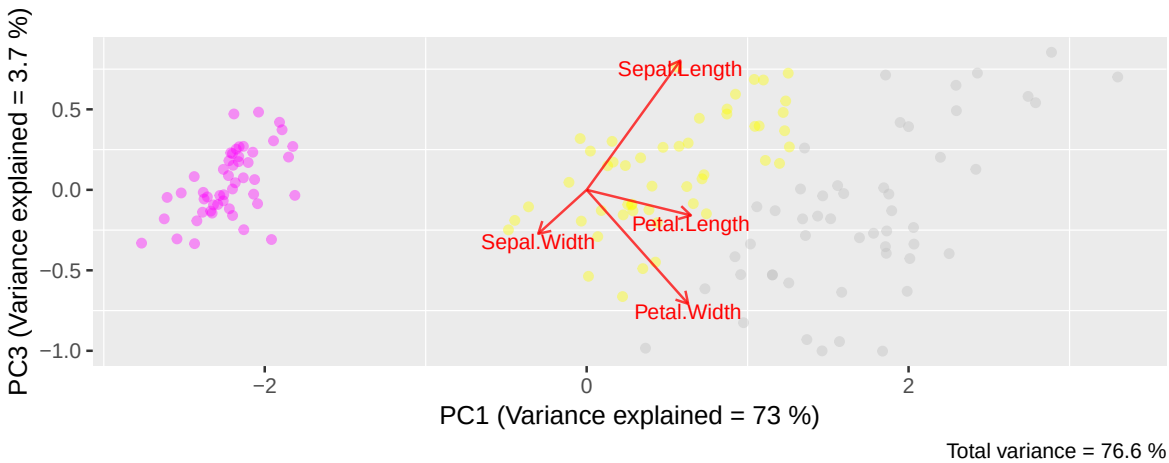
The `iris` data set can be plotted in a pca biplot with the colour objects based on the RKM algorithm result.

```
#convert the data set into principle component object
pcadat <- prcomp(iris[,1:4], scale. = TRUE)
#plot the pca with the corresponding RKM clustering result
pcabiplot(pcadat, colobj = rkm$cluster+5, o.size = 2)
```



The second principle component can be replaced by the third principle component.

```
pcabiplot(pcadat, y = "PC3",colobj = rkm$cluster+5, o.size = 1.5)
```



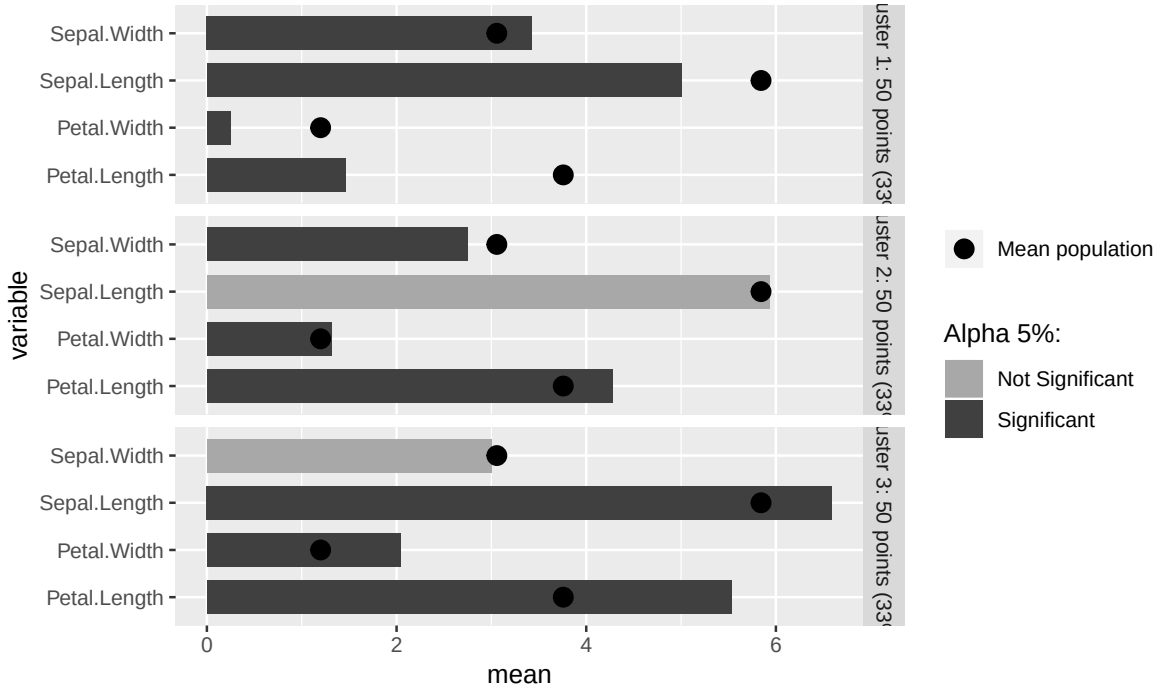
(Back to Intoduction)

B. Marked barplot

A marked barplot has been proposed by Dolnicar and Leisch (2014); Leisch (2008) where the mark indicates a significant difference between the cluster's mean and population's mean in each variable. The `barplot` function creates a barplot of each cluster with a particular significant level. The layout of the barplot is set in the `nc` argument.

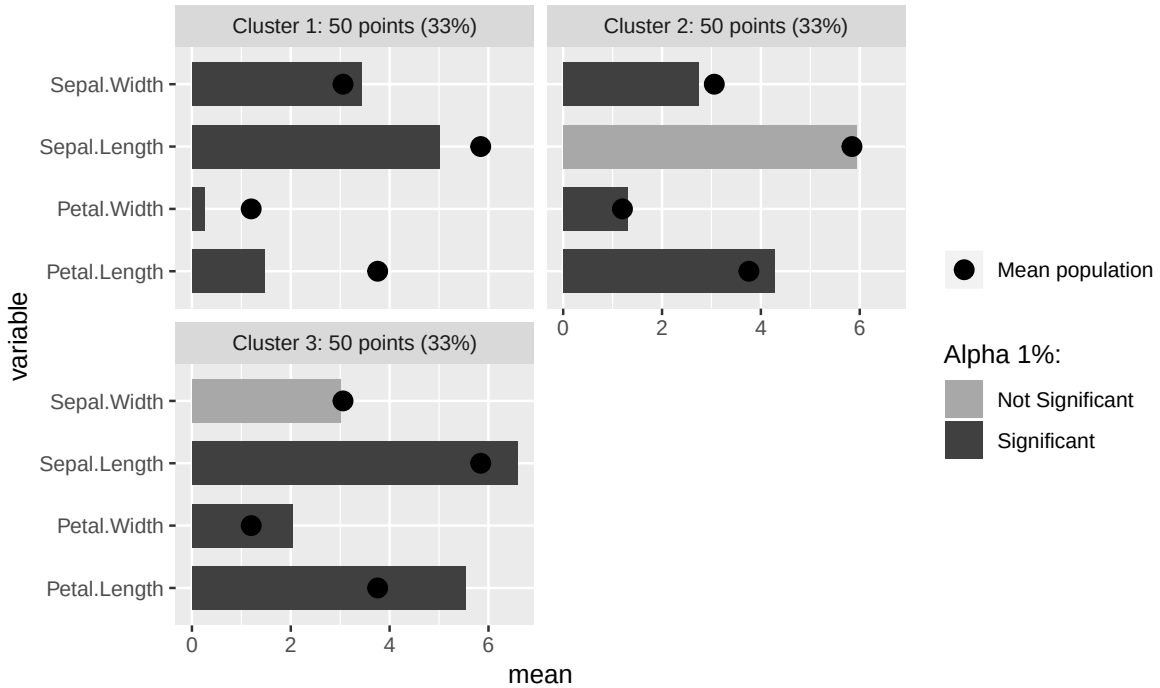
The barplot of `iris` data set partitioned by the RKM algorithm is

```
barplotnum(iris[,1:4], rkm$cluster, alpha = 0.05)
```



while the layout is changed into 2 columns and the alpha is set into 1%, it becomes

```
barplotnum(iris[,1:4], rkm$cluster, nc = 2, alpha = 0.01)
```



(Back to Intoduction)

References

Ahmad, A., and L. Dey. 2007. "A K-Mean Clustering Algorithm for Mixed Numeric and Categorical Data." *Data and Knowledge Engineering* 63 (November). Elsevier:503–27. https://doi.org/10.1016/j.datak.2007.03.016.

Dolnicar, S., and F. Leisch. 2010. "Evaluation of Structure and Reproducibility of Cluster Solutions Using the Bootstrap." *Marketing Letters* 21 (March). Springer:83–101. https://doi.org/10.1007/s11002-009-9083-4.

———. 2014. "Using Graphical Statistics to Better Understand Market Segmentation Solutions." *International Journal of Market Research* 56 (March). Sage:207–30. https://doi.org/10.2501/IJMR-2013-073.

Gower, J. 1971. "A General Coefficient of Similarity and Some of Its Properties." *Biometrics* 27 (December). International Biometric Society:857–71.

Hahsler, M., and K. Hornik. 2011. "Consensus Clustering; Dissimilarity Plots; A Visual Exploration Tool for Partitional Clustering." *Journal of Computational and Graphical Statistics* 20 (August):335–54. https://doi.org/10.1198/jcgs.2010.09139.

Harikumar, S., and S. PV. 2015. "K-Medoid Clustering for Heterogeneous Data Sets." *JProcedia Computer Science* 70. Elsevier:226–37. https://doi.org/10.1016/j.procs.2015.10.077.

Huang, Z. 1997. "Clustering Large Data Sets with Mixed Numeric and Categorical Values." In *The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 21–34.

Leisch, F. 2008. "Visualizing Cluster Analysis and Finite Mixture Models." In *Handbook of Data Visualization*, 561–87. Springer Verlag. https://doi.org/10.1007/978-3-540-33037-0_22.

———. 2010. "Neighborhood Graphs, Stripes and Shadow Plots for Cluster Visualization." *Statistics and Computing* 20 (October). Springer:457–69. https://doi.org/10.1007/s11222-009-9137-8.

Monti, S., P. Tamayo, J. Mesirov, and T. Golub. 2003. "Consensus Clustering; A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data." *Machine Learning* 52 (July). Springer:91–118. https://doi.org/10.1023/A:1023949509487.

Park, H., and C. Jun. 2009. "A Simple and Fast Algorithm for K-Medoids Clustering." *Expert Systems with Applications* 36 (2). Elsevier:3336–41. https://doi.org/10.1016/j.eswa.2008.01.039.

Podani, J. 1999. “Extending Gower’s General Coefficient of Similarity to Ordinal Characters.” *Taxon* 48 (May). Wiley:331–40.

Reynolds, A. P., G. Richards, B. De La Iglesia, and V. J. Rayward-Smith. 2006. “Clustering Rules; A Comparison of Partitioning and Hierarchical Clustering Algorithms.” *Journal of Mathematical Modelling and Algorithms* 5 (March). Springer:475–504. https://doi.org/10.1007/s10852-005-9022-1.

Rousseeuw, P. J. 1987. “A Graphical Aid to the Interpretation and Validation of Cluster Analysis.” *Journal of Computational and Applied Mathematics* 20 (November). Elsevier:53–65. https://doi.org/10.1016/0377-0427(87)90125-7.

Wishart, D. 2003. “K-Means Clustering with Outlier Detection, Mixed Variables and Missing Values.” In *Exploratory Data Analysis in Empirical Research; Proceedings of the 25th Annual Conference of the Gesellschaft Fur Klassifikation E.v., University of Munich, March 14-16, 2001*, 27:216–26. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-55721-7_23.

Yu, D., G. Liu, M. Guo, and X. Liu. 2018. “An Improved K-Medoids Algorithm Based on Step Increasing and Optimizing Medoids.” *Expert Systems with Applications* 92 (February). Elsevier:464–73. https://doi.org/10.1016/j.eswa.2017.09.052.

Zadegan, S.M.R, M. Mirzaie, and F. Sadoughi. 2013. “Ranked K-Medoids A Fast and Accurate Rank-Based Partitioning Algorithm for Clustering Large Datasets.” *Knowledge-Based Systems* 39 (February). Elsevier:133–43. https://doi.org/10.1016/j.knosys.2012.10.012.

Table of abbreviations

GDF	Generalized distance function
GSDF	Generalized spatial distance function
INCKM	Increasing number of cluster k-medoids
KM	K-medoids
PAM	Partitioning around medoids
RKM	Rank k-medoids
SFKM	Simple and fast k-medoids
SKM	Simple k-medoids

Curriculum Vitae

Personal Data

Name	Weksi Budiaji
Date of Birth	March, 08 1983
Place of Birth	Klaten, Central Java
Country	Indonesia
Nationality	Indonesia
Martial status	Married
University	University of Natural Resources and Life Sciences

Education

1998-2001	High School (*Natural and Life Sciences*) SMA N 1 Klaten, Central Java, Indonesia
2001-2005	Bachelor of Science (*Statistics*) Bogor Agricultural University, Bogor, Indonesia
2010-2012	Master of Science *Statistical Science for the Life and Behavioural Sciences* Leiden University, Leiden, The Netherlands
since 2015	Doctor rerum naturalium technicarum *Agriculture Program* Institute of Statistics University of Natural Resources and Life Sciences, Vienna, Austria

Professional Experience

02/2005-04/2005	Statistics and Database Assistant, The Ministry of Forestry Semarang, Central Java, Indonesia
01/2006-04/2006	Actuaries, Takaful Insurance Indonesia Jakarta, Indonesia
since 04/2006	Statistics Lecturer, Agribusiness Department Sultan Ageng Tirtayasa University, Serang, Indonesia
10/2013-10/2015	Vice-Head of Agribusiness Department Sultan Ageng Tirtayasa University, Serang, Indonesia

Vienna June, 2019