universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

Benchmarking in Cluster Analysis - Insights into Theory and Application

verfasst von / submitted by

Mag. Rainer Dangl

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr. techn.)

Wien 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	ao. Univ.-Prof. Mag. Dr. Marcus Hudec
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. techn. Friedrich Leisch

Acknowledgements

This thesis marks the end of a long period of study, beginning with my teacher training studies in English and Computer Science in the winter semester of 2004/05 and the subsequent doctoral degree in Computer Science. This has been an incredibly rewarding time and there are a number of people who I want to mention here.

First, I want to thank my thesis supervisors Marcus Hudec and Friedrich Leisch for the many years of support, guidance and patience. The valuable experience I was able to gain during my doctoral studies made the years of study absolutely worthwhile.

Another very important group of people are all my current and former colleagues and study companions. In particular I want to thank Christoph Grapa, who has become a great friend of mine and who has helped me in more than just one mathematical emergency.

Finally, I am most grateful to my family. Without their support, this endeavor would not have been possible. I want to thank my parents Elisabeth and Johann for their confidence throughout all these years, my wife Irene and my sisters Bernadette and Gabriela for their emotional support and particularly my grandmother, who never stopped believing in me.

Abstract

This doctoral thesis covers the topic of benchmarking in cluster analysis from two perspectives. After introductory chapters on clustering models and their validation, a benchmarking study on a specific question is first discussed, namely whether stability-oriented validation of clustering models offers a decisive advantage compared to simple validation and, if so, whether the way of drawing the resampling data sets from the overall data set plays a role. This issue is considered in the context of internal and external model validation. Results show that when clustering models are externally validated, a resampling-based approach tends to yield better results. Lessons learned from the conduct of the study prompted a discussion on general concepts for conducting benchmarking. This ranges from considerations such as systematic and transparent documentation of data sets, simulation parameters and design of the study to a concrete proposal of a grammar for benchmarking in clustering. Here, the essential building blocks of a study (data, methods, validation criteria) are assembled in the form of a benchmarking object and an implementation of the same in R is presented. This solution allows an easy combination of the building blocks into a clearly structured object, which contains all parameters and elements for the execution of the benchmark. Furthermore, the thesis deals with a concrete subarea of benchmarking, the generation of artificial data. For this purpose, an R package was developed that allows to generate artificial data using simulation setups. The latter describe the metadata of one or more data sets in a clearly specified form. From this metadata, the R package can subsequently generate the actual data sets. This should greatly facilitate transparency and reproducibility of studies. As a way to efficiently collect and access these simulation setups, a web application was also designed.

Zusammenfassung

Die Dissertation behandelt das Thema Benchmarking in der Clusteranalyse aus zwei Perspektiven. Nach einführenden Kapiteln zu Clusteringmodellen und deren Validierung wird zunächst eine Benchmarkingstudie zu einer konkreten Fragestellung behandelt, und zwar ob stabilitätsorientierte Validierung von Clusteringmodellen einen entscheidenden Vorteil im Vergleich zu einfacher Validierung bietet und wenn ja, ob die Art und Weise des Ziehens der Resampling-Datensätze aus dem Gesamtdatensatz eine Rolle spielt. Diese Fragestellung wird im Zusammenhang mit interner und externer Modellvalidierung betrachtet. Ergebnisse zeigen, dass bei externer Validierung von Clusteringmodellen ein Resamplingbasierter Zugang tendenziell bessere Ergebnisse liefert. Erkenntnisse aus der Durchführung der Studie zogen Überlegungen zu grundlegenden Konzepten zur Durchführung von Benchmarkingstudien nach sich. Dies reicht von generellen Überlegungen wie beispielsweise systematische und transparente Dokumentation von Datensätzen, Simulationsparametern und Design der Studie bis zum konkreten Vorschlag einer Grammatik für Benchmarking im Clustering. Dabei werden die grundlegenden Bausteine einer Studie (Daten, Methoden, Validierungskriterien) in Form eines Benchmarkingobjekts zusammengesetzt und eine Implementation desselben in R präsentiert. Dies ermöglicht ein einfaches Zusammensetzen der Bausteine zu einem klar strukturierten Objekt, das alle Parameter und Elemente für die Durchführung des Benchmarks enthält. Weiters beschäftigt sich die Dissertation mit einem konkreten Teilgebiet von Benchmarking, der Erzeugung von künstlichen Daten. Zu diesem Zweck wurde ein R Paket entwickelt, welches es ermöglicht künstliche Daten mit Hilfe von Simulationssetups zu erzeugen. Letztere beschreiben die Metadaten eines oder mehrerer Datensätze in einer klar vorgegebenen Form. Aus diesen Metadaten kann das R Paket in der Folge die tatsächlichen Datensätze erzeugen. Dies soll Transparenz und Reproduzierbarkeit von Studien deutlich erleichtern. Als Möglichkeit diese Simulationssetups effizient zu sammeln und zugänglich zu machen wurde zudem eine Webapplikation entworfen.

Contents

Introduction

This doctoral thesis was written to look at the topic of benchmarking in unsupervised learning, especially clustering, from several perspectives. First research initiatives focused on, as outlined in the thesis expose, the foundations of stability based cluster validation along with a benchmarking study addressing a concrete data analytical problem of resampling approaches in stability based validation. Results of this study were shared with project partners in Australia and provided an insight into the interrelations between validation measures, resampling strategies and validation methods. An introduction to data clustering, stability based validation and the benchmarking study are discussed in chapters 1, 2 and 3.

Conclusions and experiences from the benchmarking study inspired a wider discussion on the nature of benchmarking in cluster analysis. Particularly, difficulties in reproducibility and comparability of studies were a notable factor in investigating a general structural framework that should govern the setup of benchmarking. A review of rules, guidelines and recommendations is done in chapter 4, along with a prototypical implementation of how a structural frame of benchmarking in R may be achieved. The thesis expose highlights the importance of a *grammar for benchmarking* as a framework for implementing benchmarking studies in a transparent, reproducible and comparable manner. Chapter 4 addresses this by introducing elements that function as basic components of a grammar for benchmarking.

Furthermore, chapter 5 addresses another aspect that forms part of a benchmarking grammar, the generation of simulation data. As mentioned in the thesis expose, in cooperation with the IFCS task force on benchmarking, an R package was developed that aims to significantly improve the transparency and reproducibility of artificial data. This is done by establishing the notion of a metadata object in R that encapsulates all information necessary to generate actual data. This should serve as another piece in the puzzle that comprises the problem of how to address benchmarking in cluster analysis.

1. Data Clustering

1.1. Background

Data clustering is an unsupervised machine learning method that seeks to find significant patterns or features in a given data set, without the help of a teacher [110]. This can be done for various reasons, such as data reduction (computing cluster centers as representatives), hypothesis generation and testing and for predictive purposes (obtaining a classification based on a cluster model for new data) [85, 110, 178]. Furthermore, applications of clustering can be found in a wide range of disciplines, for example operations management [205], business processes [183, 205], bioinformatics [5], biomedical technology [5, 183, 204] and computer vision [167].

This chapter reviews general concepts of clustering methods, discusses relevant terminology and algorithms and thus establishes the foundation for the discussion in the chapters thereafter.

1.2. Definitions

This section discusses basic notions, definitions and terminology with regard to (un)supervised learning, clustering approaches and methods along with some illustrative examples.

1.2.1. Supervised vs. Unsupervised Classification Problems

To begin with, the goal of machine learning is to obtain a model that has been computed on the basis of some training data that can then predict values of new data points, which can either be qualitative or quantitative. If these predicted values are indeed quantitative in nature, one denotes this as a *regression* model, in the case of qualitative output values as a *classification* model [21]. In the following discussion and indeed chapters, we shall focus exclusively on the classification domain.

The training data that is required to compute a model can be defined as a set of predictor variables $X^T = (X_1, ..., X_p)$, where p is the number of variables, and the values that are to be predicted can be defined as a set of response variables

$Y = (Y_1, ..., Y_m)$ where m is the number of those variables (as there can be more than just one). The data points, or inputs are defined by $x_i^T = (x_{i1}, ..., x_{ip})$ where x_i is the ith data point in the training data set [77].

This basic terminology applies to machine learning in general. If the model is based on a training data set that includes response measurements y_i in the form of $(x_i, y_i), ..., (x_N, y_N)$ where N is the number of of data points, the model is based on a *supervised* learning method [77]. This is because the training data includes an already known response variable that shall be predicted for new data. Thus, the method can basically evaluate each x_i and the computed response variable $\hat{y}_i$ with regard to y_i that is provided in the training data. The error that needs to be minimized in order to fit the model as well as possible to the training data is generally characterized by some loss function $L(y, \hat{y})$, e.g. $L(y, \hat{y}) = (y - \hat{y})^2$ [77].

The other possibility is to conduct the computation of the classification model without the already known response variables in the training data, thus only the set of N observations $(x_1, ..., x_N)$ is provided. This means that the the learning process is *unsupervised*. As already mentioned, the purpose of qualitative classification is to obtain a model that can detect groups of observations that are in some shape or form similar. With regard to unsupervised learning, this task is called *clustering*. There are other unsupervised learning methods, however. *Principal Components Analysis (PCA)* and *Multidimensional Scaling (MDS)*, where high dimensional data are projected to a lower (e.g. two or three) dimension, are examples for unsupervised learning methods often applied for the purpose of data visualization and dimensionality reduction [21]. *Association rule analysis* is another method that tries to find rules that govern the occurrence of joint values of variables in a data set. For the case of binary data where $X_j \in 0, 1$ and $X = (X_1, X_2, ..., X_p)$, this is commonly referred to as *market basket analysis* [77]. For the subsequent sections and chapters, we again narrow the scope from qualitative classification to unsupervised classification, i.e. clustering.

As already mentioned, in a nutshell, the goal of clustering is to separate a finite, unlabeled training data set $\mathbf{X}$ into a finite and discrete set of *natural* groups [67, 86, 98, 112, 203]. In the following section, *clustering criteria* and their relevance for the notion of what precisely constitutes a cluster/group is discussed.

1.2.2. Clusters and Groups

Cluster analysis has a variety of applications, as already mentioned in section 1.1. As noted above, in cluster analysis objects are grouped together in clusters where objects within the same cluster are more similar to each other than to objects in a different cluster [77]. This naturally raises the question how to define similarity. Friedman et al. [77] and Kyan et al. [110] note that this cannot be answered by giving a general definition, but rather depends on the subject matter under

investigation, i.e. the *thematic aspect* that is subjected to optimization by means of the *clustering criterion*.

Animal Data Example The importance of the thematic aspect with regard to the outcome of the cluster analysis is illustrated in an example in Figure 1.1, taken from Theodoridis and Koutroumbas [178].

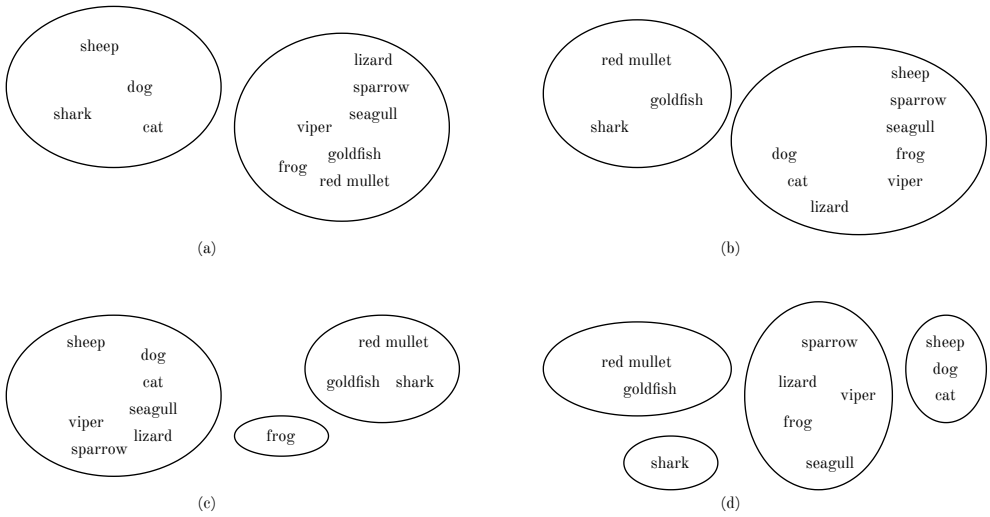

Figure 1.1.: Clusters according to various aspects

Several animals are organized into clusters according to different aspects. There are mammals (sheep, dog, cat), birds (sparrow, seagull), reptiles (viper, lizard), fish (goldfish, red mullet, blue shark), and amphibians (frog). Clustering (a) shows the animals grouped by their progeny as an underlying aspect. Alternatively, if the existence of lungs is used, clustering (b) is the result. Furthermore, if the living environment is used, clustering (c) is obtained, resulting in three clusters as the frog can live in water and on land. Also, it is of course possible to use combinations of aspects. Clustering (d) is obtained when the animals' progeny and existence of lungs is used as the underlying aspects. This shall illustrate that the exact same data cannot only result in different clusters with regard to the observations contained therein, but even different numbers of clusters.

If now a certain aspect is chosen that determines in which way observations in a data set are supposed to be similar to each other, the question arises how to measure similarity. This is discussed in the next section.

1.2.2.1. Proximity Measures

As Friedman et al. [77] claim, a fundamental part in determining the similarity respectively dissimilarity of objects in a data set is the choice of an adequate proximity measure: the results are generally expressed by an $N \times N$ dissimilarity matrix $\mathbf{D}$, where N is the number of observations in the data set and each element $d_{ii'}$ shows the distance between the ith and i'th observation. Friedman et al. [77] furthermore note that most algorithms presume such a symmetric matrix of dissimilarities with non-negative entries and zero diagonal elements (as there obviously is no dissimilarity/distance between the same object d_i) as their input. If then a value x_{ij} is given, where $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, p$ and p are variables (also called *attributes* or *features*), the pairwise dissimilarity $d_j(x_{ij}, x_{i'j})$ between observations i and i' is defined by some aggregate function ζ:

$$D(x_i, x_{i'}) = \zeta(d_j(x_{ij}, x_{ij})) \tag{1.1}$$

where ζ most often equates to the sum of the components

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} d_j(x_{ij}, x_{i'j}) \tag{1.2}$$

If $d_j(x_{ij}, x_{i'j})$ in equation 1.2 is then substituted with

$$d_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2 \tag{1.3}$$

squared Euclidean distance (ED2) is obtained, which as Kyan et al. [110] and Friedman et al. [77] note, is a very common distance measure:

$$D_{ED2}(x_i, x_{i'}) = \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \tag{1.4}$$

Furthermore, from equation 1.4 one can state Euclidean distance (ED) as follows:

$$D_{ED}(x_i, x_{i'}) = \sqrt{\sum_{j=1}^{p} (x_{ij} - x_{i'j})^2} \tag{1.5}$$

Additionally, a wide range of other possible distance metrics to measure dissimilarity exist, summarized in Table 1.1, such as for example a weighted (WED) or standardized (SED) form of Euclidean distance. WED can be used to put greater emphasis on one dimension over another, which means that the spherical cluster structure usually assumed by Euclidean distance can be transformed into an elliptical shape. This should help with discovering elongated rather than spherical

Distance metric	**Formulation**
Euclidean (ED)	$d_{ED}(x_i, x_{i'}) = \sqrt{\sum_{j=1}^{p}(x_{ij} - x_{i'j})^2}$
Euclidean Squared (ED2)	$d_{ED2}(x_i, x_{i'}) = \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2$
Weighted Euclidean (WED)	$d_{WED}(x_i, x_{i'}, w_i) = \sqrt{\sum_{j=1}^{p} w_j \times (x_{ij} - x_{i'j})^2}$ where w_i is a vector of weights w_j per dimension j
Standardized Euclidean (SED)	$d_{SED}(x_i, x_{i'}) = d_{WED}(x_i, x_{i'}, \frac{1}{s_i})$ where s_i is the sample variance per dimension j
Mahalanobis (MhD)	$d_{MhD}(x_i, x_{i'}) = \sqrt{(x_i - x_{i'})^T S^{-1}(x_i - x_{i'})}$ where S is any $p \times p$ positive definite covariance matrix If $S = I$, then this reduces to ED If $S = \sum$ is diagonal, this reduces to SED (using sample variances)
Manhattan/City block (MD)	$d_{MD}(x_i, x_{i'}) = \sum_{j=1}^{p} \lvert x_{ij} - x_{i'j}\rvert$
Cherbychev (CBD)	$d_{CBD}(x_i, x_{i'}) = max_{ij}\lvert x_{ij} - x_{i'j}\rvert$
Minkowski (MkD)	$d_{MD}(x_i, x_{i'}) = (\sum_{j=1}^{p} \lvert x_{ij} - x_{i'j}\rvert^m)^{\frac{1}{m}}$
Cosine (CD)	$d_{CD}(x_i, x_{i'}) = \frac{x_i \times x_{i'}}{\lvert\lvert x_i\rvert\rvert\lvert\lvert x_{i'}\rvert\rvert} = cos(\theta)$ where θ is the angle between vectors x_i and x_i'
Pearson correlation (PCD)	$d_{PCD}(x_i, x_{i'}) = 1 - r_{x_i x_{i'}}$ where $r_{x_i x_{i'}} = \frac{\sum_j^p (x_{ij} - x_i)(x_{i'j} - x_{i'})}{\sqrt{\sum_j^p (x_{ij} - x_i)}\sqrt{\sum_j^p (x_{i'j} - x_{i'})}}$ is the Pearson Correlation Coefficient between vectors x_i and $x_{i'}$

Table 1.1.: Commonly used distance metrics in clustering as shown in Kyan et al. [110]

clusters. This can also be achieved by using the standardized form of Euclidean distance (SED) where each dimension is normalized according to its respective variance. Consequently, high variance dimensions are shrunk in comparison to low variance dimensions [110]. A generalized form of this principle is Mahalanobis distance (MhD), where the correlation in the data is taken into account by using the inverse of the variance-covariance matrix of the data [49]. This again results in a weighting that can stretch or shrink the data space [110]. Manhattan distance on the other hand computes the sum of orthogonal distances between pairs of observations assuming that movement is restricted to one dimension at a time, which according to Kyan et al. [110] is much like traveling along paths defined by city blocks (hence the name). Cherbychev distance (MD), also called maximum value distance, reduces MD by considering only the maximum value of one dimension [49]. Minkowski distance, the generalized metric distance (MkD), generalizes between MD and ED [49, 110]. Cosine distance (CD) takes a completely different approach, by considering the angle between vectors x_i and $x_{i'}$. Finally, Pearson correlation distance (PCD) evaluates whether variations along an entire vector show a similar pattern as the variations along another vector [110].

1.3. Cluster Algorithms

As already mentioned, the main aim of cluster analysis is to group objects in clusters that are similar to each other. One way to measure this is to use pairwise dissimilarity of objects where dissimilarity in the same cluster is lower than of objects in different clusters [77]. In section 1.2.2.1, proximity measures were discussed that can be used to calculate pairwise dissimilarity; another major choice in cluster analysis naturally pertains to the cluster algorithm itself. In fact, using pairwise dissimilarity is only one of many options to cluster data. For example, Friedman et al. [77] mention three approaches: combinatorial algorithms, mixture models and mode-seeking. Whereas combinatorial and mode-seeking methods employ a non-parametric approach to model estimation, mixture modeling does the opposite: as the name suggests, this approach assumes that the data are composed of a mixture of component density functions, where each component density describes one of the clusters [77]. Xu and Wunsch [203] alternatively subsume mixture modeling and combinatorial algorithms such as k-means under the term *partitional* clustering and on the other hand *hierarchical* methods. For the purpose of the present discussion, the following categorization is used:

- partitional non-parametric methods such as the k-centers family of algorithms (k-means/k-medoids) and density based clustering
- partitional parametric methods such as Gaussian Mixture Models (GMM)

- hierarchical methods (single/complete linkage, etc.)

There are obviously considerably more algorithms in the unsupervised learning/-clustering domain such as neural network methods (e.g. Self Organizing Maps), probabilistic methods such as Hidden Markov Models or graph-theoretic approaches (e.g. spectral clustering) [110]. However, the present discussion seeks to outline the main branches of clustering and the enumeration above encompasses several quite commonly used methods and serves as a good basis for an overview of clustering methods. In the following, definitions and illustrative examples for the three categories are discussed.

1.3.1. Non-Parametric Partitional Methods

Exemplary for partitional methods, this section focuses on squared-error methods such as k-means/k-medoids and density based methods such as the DBSCAN algorithm by Ester et al. [65]. These methods are non-parametric algorithms, which means that they assign an observation to a cluster without regard to the probability model of the underlying data [77].

1.3.1.1. k-centers Methods

As mentioned above - a quite widely used form of determining clusters in a data set is by measuring dissimilarity between data points. Cichosz [42] claims that the family of k-centers algorithms are not only the simplest but also most popular algorithms that follow the dissimilarity approach. The basic algorithm common to all of the k-centers methods can be described as follows [42]:

1. the number of clusters K (with $K < N$) is predetermined
2. clusters are represented by single attribute value vectors (i.e. a *cluster center*)
3. the cluster formation and modeling process is done by iteratively assigning data points to their closest and therefore least dissimilar cluster center and afterward shifting the cluster centers to reflect the actual content of the particular cluster

Friedman et al. [77] point out that an assignment of an observation x_i of a data set $\mathbf{X}$ where $i \in 1, \ldots, N$ is done by a so called many-to-one mapping by an *encoder* $k = C(i)$ which assigns the ith observation to the kth cluster. The goal is to determine the encoder $C(i)$ that does this based on the dissimilarities $d(x_i, x_{i'})$ for every pair of observations in $\mathbf{X}$. In order to compute this, Friedman et al. [77] note

that a function that specifies the degree to which the clustering goal is not met needs to be minimized (essentially, a loss function), an example of this defined as

$$W(\mathcal{C}) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d(x_i, x_{i'}) \tag{1.6}$$

and commonly referred to as the within cluster point scatter of a clustering $\mathcal{C}$ that calculates how close observations in the same cluster are to each other. Considering the total point scatter

$$T = \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N} d_{ii'} = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\Big(\sum_{C(i')=k} d_{ii'} + \sum_{C(i')\neq k} d_{ii'}\Big) \tag{1.7}$$

which is constant for the data set $\mathbf{X}$ and

$$T = W(\mathcal{C}) + B(\mathcal{C}) \tag{1.8}$$

where $B(\mathcal{C})$ is the between cluster point scatter given by

$$B(\mathcal{C}) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')\neq k} d_{ii'} \tag{1.9}$$

and $W(\mathcal{C})$ can also be stated as

$$W(\mathcal{C}) = T - B(\mathcal{C}) \tag{1.10}$$

and shows that the goal of minimizing within cluster scatter results in maximizing $B(\mathcal{C})$. Importantly, Friedman et al. [77] note that doing cluster analysis in this way by combinatorial optimization seems simple in principle given equations 1.8 and 1.10. However, optimization by complete enumeration is computationally only feasible for small data sets due to the number of distinct assignments given by the Stirling number of the second kind [77, 99, 174, 193]

$$S(N, K) = \frac{1}{K!}\sum_{k=1}^{K}(-1)^{K-k}\binom{K}{k}k^N \tag{1.11}$$

which, as Steinley [174] notes, for $S(25, 4)$ results in approximately 4.69×10^{13} different partitions. Therefore, finding the global optimum is quite unlikely given that most clustering problems involve large data sets where only a small part of possible clusterings $\mathcal{C}$ can be examined. Due to this, strategies to find a suitable local optimum are based on iterative greedy descent where an initial partition is specified and cluster assignments are modified and thus improved stepwise un-

til convergence at a local optimum is reached where no iterative improvement is observed any more. As it is only possible to examine a small part of all possible assignments, the algorithm usually only converges to a local optimum, which can be sub-optimal when compared to the global optimum [59, 77]. Moreover, k-centers methods are obviously quite sensitive with regard to the initial configuration of cluster centers [110]. Still, as Meilă [125] notes, given well separated clusters the algorithm could indeed converge toward the global optimum. The algorithm for this kind of combinatorial optimization is thus given by algorithm 1.

Algorithm 1 General algorithm for combinatorial optimization in clustering [42]

Require: a data set $\mathbf{X}$

1: Select initial cluster centers $C_1, C_2, \ldots C_K$
2: **while** convergence not reached **do**
3: **for all** training observation $x_i, \ldots, x_N \in \mathbf{X}$ **do**
4: assign observation x_i to cluster $k = argmin_k d(x_i, C_k)$
5: **end for**
6: **for** $k = 1, 2, \ldots, K$ **do**
7: modify cluster center C_k based on cluster member set $\mathbf{X}_k$
8: **end for**
9: **end while**

k-means The most well known and widely used method that follows algorithm 1 is k-means, which uses vectors of attribute value means as cluster centers [42, 77, 98, 110]. Friedman et al. [77] claim that it is intended for situations where quantitative variables are used and squared Euclidean distance (equation 1.4) is chosen as a dissimilarity measure. The within point scatter of equation 1.6 can be defined as

$$W(\mathcal{C}) = \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - \bar{x}_k||^2 \tag{1.12}$$

where $\bar{x}$ is the mean vector of the kth cluster and N_k is the number of observations therein. Therefore, assigning observations to cluster centers where the average dissimilarity is minimized fulfills the convergence criterion of maximizing between-cluster and minimizing within-cluster scatter stated above in equations 1.7, 1.6 and 1.8. Thus, in order to obtain

$$\mathcal{C} = \min_{C} \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - \bar{x}_k||^2 \tag{1.13}$$

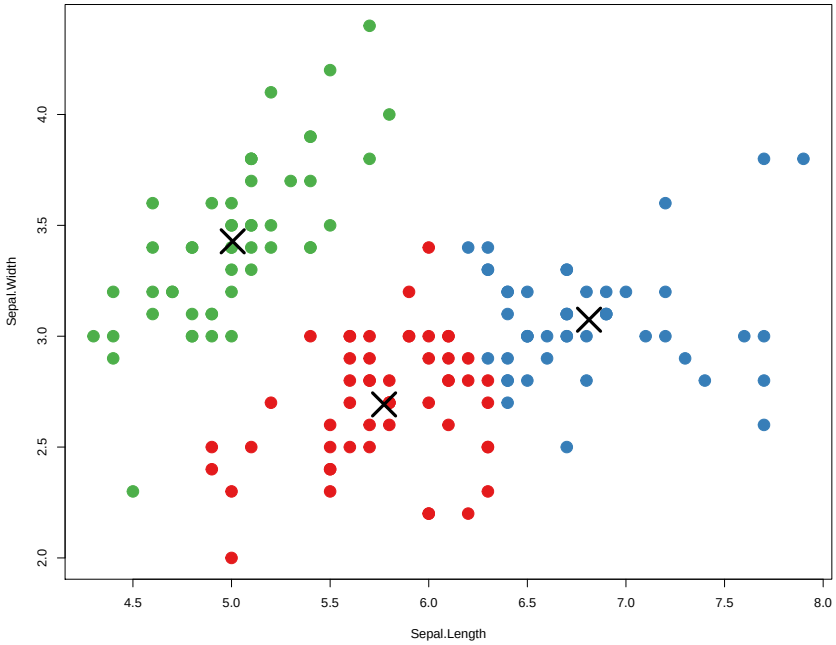

Figure 1.2.: iris data clustered with k-means (function kmeans())

given a set of observations S (where m is the mean of the currently assigned cluster)

$$\bar{x}_S = \arg\min_C \sum_{i \in S} ||x_i - m||^2 \tag{1.14}$$

the following optimization problem needs to be solved, which is done by an iterative process based on algorithm 1.

$$\min_{C, \{m_k\}} \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - m_k||^2 \tag{1.15}$$

A simple example of the k-means algorithm using the function `kmeans()` from package `stats` [146] is its application to the `iris` data set by Fisher [70] along with a scatterplot that illustrates the three clusters and their centers (Figure 1.2).

k-medoids The regular k-means algorithm has a notable drawback that pertains to the calculation of the cluster centers: using the mean to minimize within cluster scatter makes the method vulnerable to noise and outliers which might skew the mean and distort the cluster centers away from where they should be [119]. An effective, more robust method that counters this problem is k-medoids, i.e. the

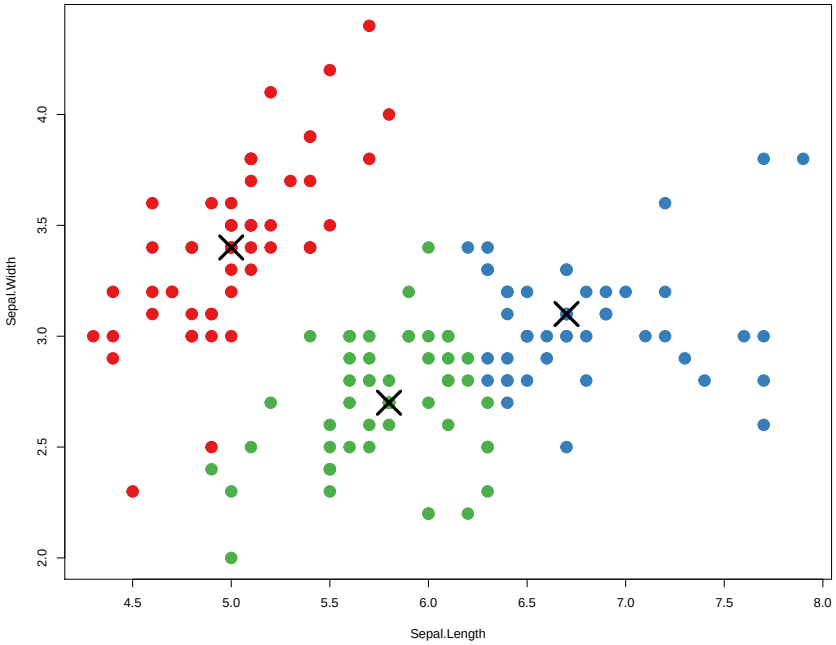

Figure 1.3.: iris data clustered with k-medoids (function pam())

partitioning around medoids method by Rousseeuw and Kaufman [154]. As the name suggests, it uses so called *medoids* - selected cluster members that are the least dissimilar to other members of the same cluster [42, 154]. It thus uses pairwise dissimilarity to determine these medoids which reduces the impact of outliers or noise. Furthermore, the cluster centers are not artificial observations, but actually present in the data. Naturally, as Cichosz [42] notes, calculating pairwise dissimilarities results in a considerably higher computational demand than k-means but can be beneficial in some applications. It certainly also depends on the data set, as can be seen in Figure 1.3, there are only slight variations in the clustering result. However, it is clearly visible that the cluster centers i.e. medoids are actual observations from the data set, not calculated means as in k-means. The model in this example was calculated using function `pam()` from package `cluster` [120] that is is based on the algorithm by Rousseeuw and Kaufman [154].

1.3.1.2. Density Based Clustering

Another non-parametric partitional method is density based clustering. However, instead of minimizing pairwise distances between cluster members and a cluster center as the k-center methods do, it tries to find groups by distinguishing high-density from low-density areas. Furthermore, density based methods no not require a pre-specified number of clusters as input and moreover do not make any assump-

tions with regard to the underlying probability density $p(x)$ and the variance in the data [106].

DBSCAN A well known method for density based clustering is the DBSCAN (Density Based Spatial Clustering of Applications with Noise) by Schubert et al. [161], an abstract description of the method is given in algorithm 2.

Algorithm 2 Abstract DBSCAN algorithm [161]

```
1: Compute neighbors for each point and identify core points
2: Join neighboring core points into clusters
3: for all non-core points do
4:     Add to a neighboring core point if possible
5:     Otherwise add to noise
6: end for
```

The DBSCAN method uses a simple minimum density level estimation, based on a minimum number of points (*minPts*) within a radius ε (based on some distance measure d) [161]. The ε neighborhood of an observation $x_i \in \mathbf{X}$ and other points $x_{i'}$ is defined as [65]

$$N_\epsilon(x_i) = \{x_{i'} \in \mathbf{X} | d(x_i, x_{i'} \leq \epsilon)\} \tag{1.16}$$

Data points with more than a certain minimum number of observations (*minPts*) within ε and thus $N_\varepsilon(x_i) \geq minPts$ are considered so called *core points*. If two points x_i and $x_{i'}$ satisfy

$$x_i \in N_\epsilon(x_{i'}) \tag{1.17}$$

and

$$N_\epsilon(x_i) \geq minPts \tag{1.18}$$

which is the core point criterion, these two points are defined as *direct density reachable*. If $x_{i'}$ does not fulfill the core point criterion in equation 1.18, it is defined as a *border point* instead of a core point. Furthermore, if two points x_i and $x_{i'}$ are not directly density reachable but there exists a chain of points $x_1 \dots x_N$ where $x_1 = x_i$ and $x_N = x_{i'}$ and all points in-between are directly density reachable, x_i and $x_{i'}$ are called *density reachable* [65]. Finally, points are defined as *density connected* if there is a third point $x_{i''}$ that is density reachable from both x_i and $x_{i'}$. A cluster is therefore defined as a set of density connected points. All other points are considered noise and do not belong to any cluster. Figure 1.4 illustrates the concept. Observation A is a core point, directly connected to

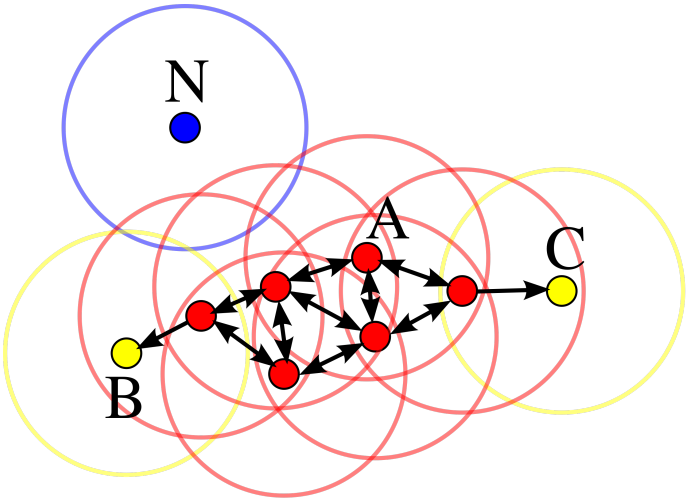

Figure 1.4.: Core points in DBSCAN [161]

three other observations in its neighborhood. Points B and C are not core points because there are fewer than *minPts* observations within their radius ε, but they are density connected via several core points, thus still belonging to the same cluster. They are furthermore labeled as border points. Point N is not in range of any core point, thus not density connected and therefore considered noise [161].

Density based methods differ notably from k-centers methods with regard to the limitations of the clustering. For example, as noted above, k-means can be more easily distorted by outliers and noise in the data and favors spherically shaped clusters [106]. In contrast, by separating high from low density regions, density based methods ignore noise and outliers in the data much more effectively and can model arbitrary cluster shapes such as elongated clusters. Due to this, the within-cluster scatter is not necessarily low [106]. An example for density based clustering is given in Figure 1.5. An artificial 2d data set containing 3 clusters was generated including some noise data, uniformly distributed across the data space. This is analyzed using the `dbscan()` function of package `dbscan` [84]. Using a k-nearest neighbor distance plot, the optimal value for ε is determined - 0.8, approximately the beginning of the elbow in the curve. The convex cluster hull plot then nicely shows that the method is able to separate the high-density clusters from the low density noise in the data. k-means on the other hand treats all data points the same and thus severely distorts the clusters by including all noise points.

HDBSCAN There are some noteworthy issues with the DBSCAN algorithm. In particular, the parameters ε and *minPts* are not always straightforward to select. Furthermore, a fixed value for ε and *minPts* results in the assumption that all clusters have similar density [37, 124]. A method that attempts to alleviate this

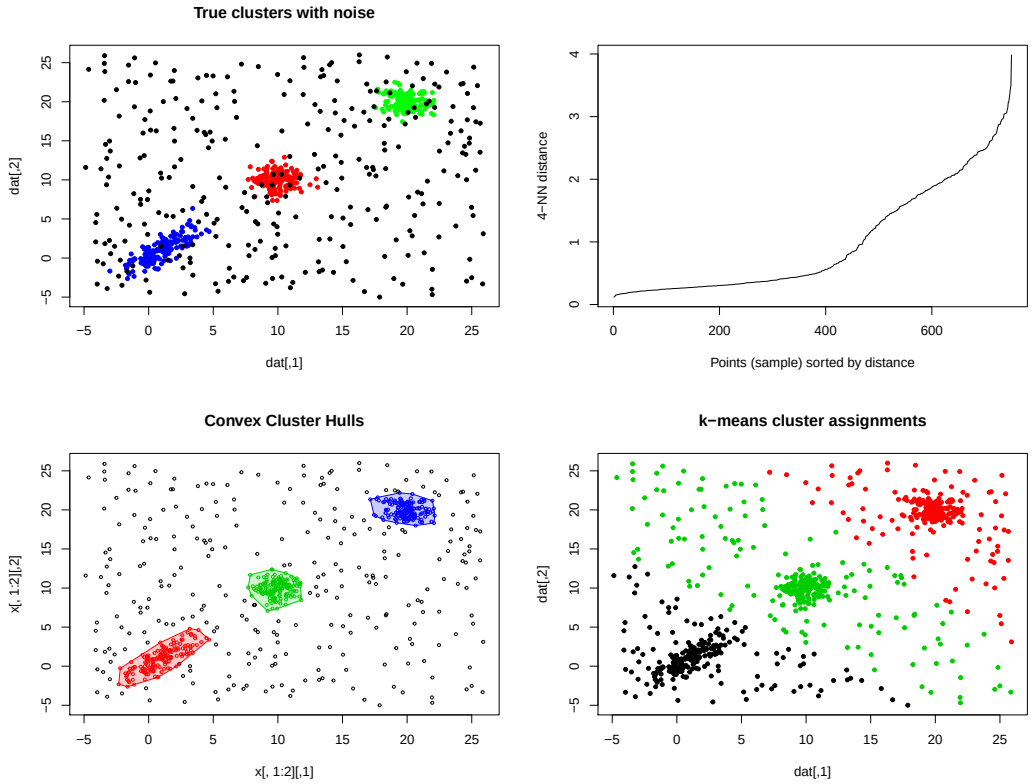

Figure 1.5.: Density based clustering compared to a k-means solution

issue is HDBSCAN by Campello et al. [37]. Apart from a data set $\mathbf{X}$ only a parameter m_{pts} is needed as input. This averts the problem of selecting ε and *minPts* and consequently a certain density assumption across all clusters. The following definitions are introduced by Campello et al. [37]:

1. The *core distance* $d_{core}(x_i)$ of an object $x_i \in \mathbf{X}$ with regard to m_{Pts} is defined as the distance from x_i to its m_{Pts}-nearest neighbor, including x_i

2. The ε *core* object is defined as an object $x_i \in \mathbf{X}$ for each value of ε that is greater or equal to the core distance of x_i with regard to m_{Pts}, thus $d_{core}(x_i) \leq \varepsilon$

3. The *mutual reachability distance* between two objects x_i and $x_{i'}$ is defined as $d_{mreach} = max\{d_{core}(x_i), d_{core}(x_{i'}), d(x_i, x_{i'})\}$

4. The *mutual reachability graph* is a complete graph $G_{m_{pts}}$, where all objects in $\mathbf{X}$ are vertices and the weight of each edge is defined by the mutual reachability distance between the respective pair of objects

Campello et al. [37] note that if a graph $G_{m_{pts},\varepsilon} \subseteq G_{m_{pts}}$ is obtained by removing all edges where the respective weight is greater than ε, the connected components of ε *core* objects correspond to the clusters obtained by applying DBSCAN with m_{Pts} and ε as parameters. It therefore follows that all DBSCAN partitions for $\varepsilon \in [0, \text{inf})$ can be computed in a nested, hierarchical way by iteratively removing edges in decreasing order of weight from $G_{m_{pts}}$. The exact procedure is stated in algorithm 3.
An example of how results between DBSCAN and HDBSCAN can differ is shown in Figure 1.6. The data (from McInnes et al. [124]) shows several clusters of varying shape and density and noise points. After estimating a value for ε, the DBSCAN clustering result is shown in the lower left. Quite clearly, 'bridges' between clusters cause the blue and red clusters to merge. Furthermore, several very small clusters are split from the blue cluster that should not be there. The HDBSCAN dendrogram nicely shows that there are six clusters that remain connected for quite some time while the value for ε decreases. Indeed, the cluster assignments in the lower right show that all clusters and noise from the original data are correctly identified.

Outlier Detection It should be noted that density based clustering methods are not only used for the purpose of obtaining actual clusters. Another important use case is the detection of outliers, i.e. points that do not belong to dense regions and thus clusters. Algorithms for this purpose have been proposed by Breunig et al. [35] (LOF - Local Outlying Factor) and Tang and He [177]. Outlier detection

Algorithm 3 HDBSCAN algorithm by Campello et al. [37]

Require: a data set $\mathbf{X}$ and m_{pts}

1: Compute the core distance with regard to m_{pts} for all data points in $\mathbf{X}$
2: Compute a Minimum Spanning Tree (MST) of $G_{m_{pts}}$, the mutual reachability graph
3: Extend the MST to obtain MST_{ext}, by adding for each vertex a *self edge* with the core distance of the corresponding object as weight
4: Extract the HDBSCAN hierarchy as a dendrogram from MST_{ext}:
5: For the root of the tree assign all data points to the same label
6: **for all** all edges in MST_{ext} in decreasing order of weights **do**
7: Set the dendrogram scale value of the current hierarchical level to the value of the edge with the highest weight
8: Remove edge with highest weight (or edges in case of equal weights)
9: Assign labels to the connected component(s) that contain(s) the end vertex (or vertices) of the removed edge (or edges) to obtain the next hierarchical level
10: **if** component contains at least one edge **then**
11: Assign a new cluster label
12: **else**
13: Assign a null label (i.e. noise)
14: **end if**
15: **end for**

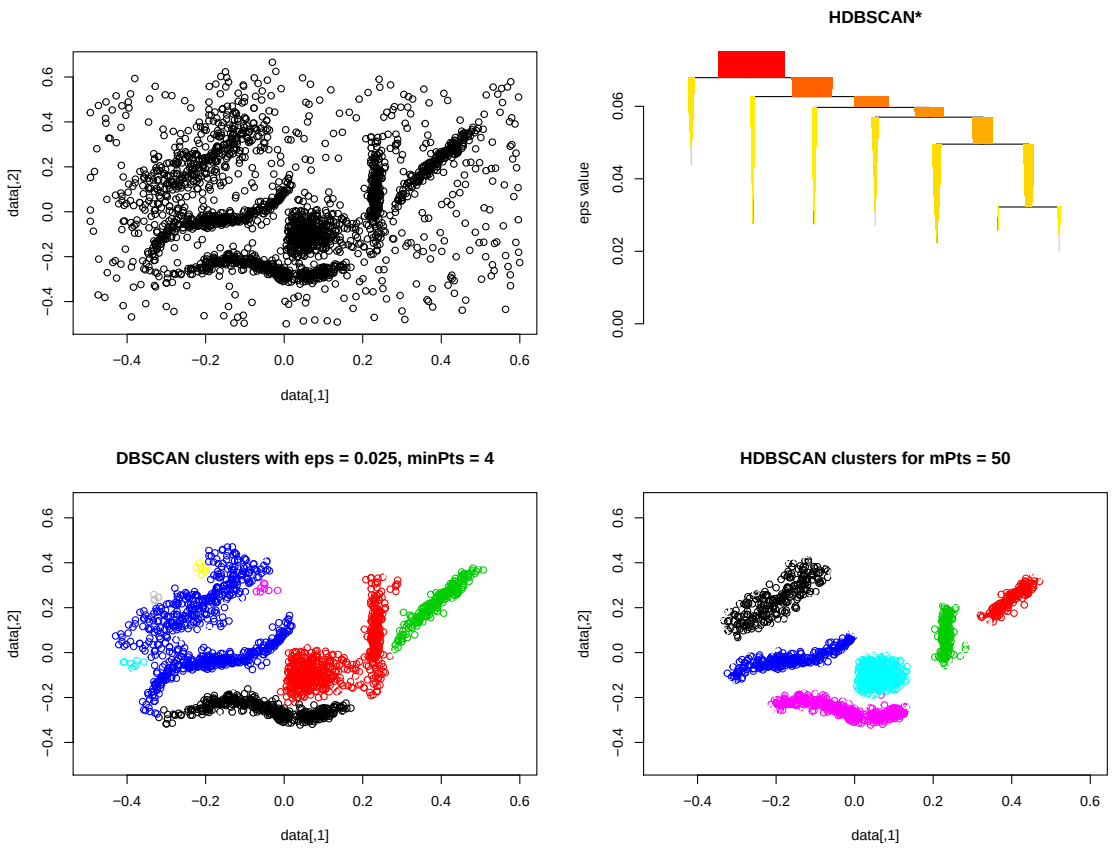

Figure 1.6.: Comparison of clusterings obtained by DBSCAN and HDBSCAN

methods are not the core focus of this thesis, therefore the respective methods are not discussed in further detail here. However, they shall be noted as an important application of density based data analysis that can be a notable factor in enhancing the quality of the data.

1.3.1.3. Implementation in R

Apart from the implementation of the k-means algorithm in R in form of function `kmeans()` in package `stats` [146] and function `pam()` in package `cluster` [120], there are plenty of other implementations, among them packages `flexclust` [115], `ClusterR` [132] and `kernlab` [101]. Implementations of density based clustering can be found in packages `dbscan` [84] and `fpc` [89].

1.3.2. Hierarchical Clustering

Hierarchical clustering is another non-parametric method that determines clusters based on a sequence of nested partitions [203]. This can either be done by an *agglomerative* (bottom-up) or *divisive* (top-down) approach. Agglomerative methods start at the bottom (i.e. the individual observations) and at each level recursively merge clusters until all observations are grouped into a single cluster [77]. Divisive methods work exactly the other way by splitting the data until each data point forms a cluster by itself.

Hierarchical clustering differs notably from k-means and k-medoids clustering as it does not require a pre-specified number of clusters or a starting assignment for the cluster centers. Instead, both strategies require a proximity matrix that contains the pairwise dissimilarities of observations which enables the algorithm to either find the least dissimilar clusters to merge (agglomerative) or the most dissimilar clusters to split (divisive) [203]. Due to the lack of required predefined parameters, hierarchical clustering lends itself to handling real-world data, as determining optimal values for the aforementioned parameters may be a challenge [25, 202]

A hierarchical clustering can be represented graphically by means of a cluster *dendrogram*, an example of which is shown in Figure 1.7, which also shows the two hierarchical cluster methods. Furthermore, in a cluster dendrogram the complete data set $\mathbf{X}$ is represented as the root node and the intermediate nodes describe the proximity of objects to each other. For example, observation x_1 and x_3 are quite similar to each other. Moreover, the height of the dendrogram signifies the distance between observations. Thus, although observations x_1 and x_3 are separated by the same number of intermediate nodes as x_5 and x_7, the distance between the latter is greater [203]. A cluster model is then obtained by cutting the dendrogram at an appropriate level, signified in Figure 1.7 by the dashed line. This model results in

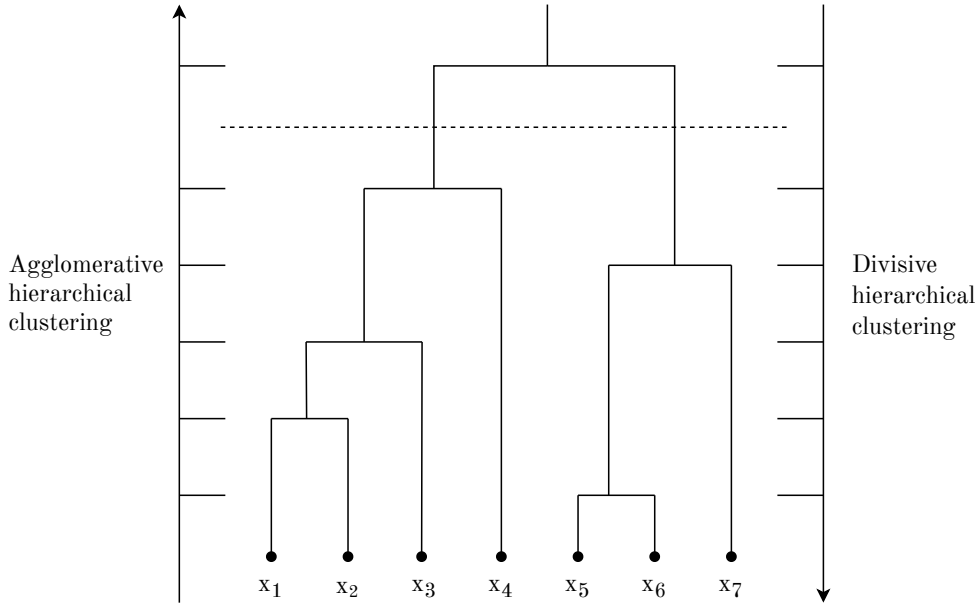

Figure 1.7.: The two options of hierarchical clustering as illustrated in Xu and Wunsch [203]. Clusters are obtained by cutting the dendrogram at an appropriate level, as indicated by the dashed line

two clusters with observations $x_1, \ldots, x_4$ in cluster 1, and x_5, x_6 and x_7 in cluster 2. The level of where to cut the dendrogram is entirely up to the researcher, and depends on the context of the analysis.

In general, divisive cluster methods are less frequently used in practice, as they are computationally intensive due to the fact that for a cluster with N observations, the algorithm needs to compute $2^{N-1} - 1$ possible two-subset divisions [1, 203]. Therefore, agglomerative methods are more commonly used. Even then, hierarchical methods have a computational complexity of at least $O(N^2)$, which poses a problem for large-scale data sets and several methods have been proposed to address this such as BIRCH (Balanced Iterative Reducing and Clustering Using Representatives) [207] and CURE (Clustering Using Representatives) [83].

1.3.2.1. Agglomerative Hierarchical Clustering

In agglomerative clustering, the algorithm starts (for a data set $\mathbf{X}$ with N number of observations) with N clusters, therefore each cluster includes one observation. These clusters are merged until one cluster is reached that contains all observations, as described in Figure 1.7. Algorithm 4 details the method.

Algorithm 4 Agglomerative hierarchical clustering [203]

1: Start with N singleton clusters
2: Calculate a proximity matrix M based on some distance function D
3: **while** more than one cluster available **do**
4: In M search the minimal distance $D(C_i, C_j) = \min_{\substack{1 \leq m,l \leq N \\ m \neq l}} D(C_m, C_l)$
5: Update M by computing distances between C_{ij} and the other clusters
6: **end while**

1.3.2.2. Cluster Linkage

Another important choice that affects the computation of the model is the distance function D, mentioned in step 3 of algorithm 4. There are several options for measuring linkage of clusters, the main methods are single, complete and average linkage, as well as Ward's method.

Single Linkage The linkage function $D(C_i, C_j)$ between clusters C_i and C_j is defined by (from Abu-Jamous et al. [1]):

$$D(C_i, C_j) = \min_{x_p \in C_i, x_q \in C_j} d(x_p, x_q) \tag{1.19}$$

where $d(x_p, X_j)$ denotes the distance between two observations x_p and x_j. This means that with single linkage, the minimum distance between two objects in different clusters is used to represent the distance between clusters C_i and C_j. Friedman et al. [77] note that single linkage clustering tends to combine, at a relatively low threshold, observations that are linked by a series of close intermediate observations.

Complete Linkage Complete linkage, also known as farthest neighbor clustering, computes the distance between two clusters by taking the maximum distance between a pair of observations in clusters C_i and C_j and is defined by (from Abu-Jamous et al. [1]):

$$D(C_i, C_j) = \max_{x_p \in C_i, x_q \in C_j} d(x_p, x_q) \tag{1.20}$$

Abu-Jamous et al. [1] note that complete linkage avoids the chaining problem of single linkage that can result in elongated clusters. On the contrary, complete linkage tends to find compact clusters of approximately equal diameters. However, it may be a less optimal choice when there is a certain amount of noise in the data.

Average Linkage Abu-Jamous et al. [1] mention two ways of defining average linkage. The first is the weighted pair group method with arithmetic mean (WPGMA) or McQuitty's method where the distance between cluster is calculated by a simple average. The distance between clusters C_i and C_j where C_j is composed of C_m and C_n is given by

$$D(C_i, C_j) = \frac{D(C_i, C_m) + D(C_i, C_n)}{2} \tag{1.21}$$

The second method is called unweighted pair group method with arithmetic mean (UPGMA) by Sokal et al. [169] and considered to be the superior method as it weighs the averages by the number of objects in each cluster, therefore ensuring that each observation in the data set is treated equally. The distance function is given by

$$D(C_i, C_j) = \frac{D(C_i, C_m)|C_m| + D(C_i, C_n)|C_n|}{|C_m| + |C_n|} \tag{1.22}$$

Centroid Method This method [1], assigns a centroid to each cluster, which is given by:

$$\bar{x_i} = \frac{1}{|C_i|} \sum_{x_p \in C_i} x_p \tag{1.23}$$

and which is used to compute the distances between the cluster center and all other cluster centroids which is defined by

$$D(C_i, C_j) = d(\bar{x_i}, \bar{x_j}) \tag{1.24}$$

Abu-Jamous et al. [1] furthermore note that when two clusters are merged, the new cluster centroid is computed from all newly combined data points (not as an average of the two centroids).

Ward's Method Proposed by Ward Jr [188], this method differs notably from the single/average/complete linkage approach. Ward's method uses the variance criterion that seeks to minimize the within cluster scatter $W(\mathcal{C})$ (equation 1.6) based on squared Euclidean distance (equation 1.4) to determine for each potential cluster merger the resulting increase in global heterogeneity, i.e. $W(\mathcal{C})$. This increase should be minimal, the criterion to be optimized is therefore given by (from Murtagh and Legendre [135]):

$$D(C_i, C_j) = \frac{|C_i| \times |C_j|}{|C_i| + |C_j|} ||\bar{x_i} - \bar{x_j}||^2 \tag{1.25}$$

where $\bar{x}$ is the centroid of the respective cluster. Naturally, when all data points form their own cluster, global heterogeneity is $W(\mathcal{C}) = 0$. Clusters that minimally increase this value are then merged [81]. Ward's method is shown in comparison to the other linkage methods in Figure 1.8. It should be noted that the determinant criterion by Friedman and Rubin [76]

$$\min_{\mathcal{C}} |W(\mathcal{C})| \tag{1.26}$$

can be regarded as a generalization of Ward's method [24]. While Ward assumes spherical clusters of equal volume, Friedman and Rubin [76] assume elliptical clusters due to non-independent variables, which are equal in volume, shape and orientation [24]. Furthermore, both Ward and the determinant criterion are non-parametric special cases of the general Gaussian mixture model (which is discussed in section 1.3.3.1).

Specifically, the GMM example in Figure 1.9 includes a plot that shows the Bayesian information criterion (BIC) of a number of GMMs where `EII` (spherical, equal volume) is equivalent to a clustering obtained by Ward's method and `EEE` (ellipsoidal, equal volume, shape and orientation) is equivalent to a clustering obtained by employing the determinant criterion. Obviously, neither are a good fit for the data in that particular example, because the clusters in the data are not spherical (which excludes Ward) and not equally oriented (thus excluding the determinant criterion).

1.3.2.3. Implementation in R

Hierarchical methods are implemented in functions `hclust()` of package `stats` [146] and `agnes()` (agglomerative) and `diana()` (divisive) of package `cluster` [120]. Other implementations are `isopam` [158], `genie` [78], `protoclust` [19], `fastcluster` [134] and `flashClust` [113].

1.3.3. Parametric Methods

Parametric methods assume that the groups in the data originate from a mixture of underlying probability distributions where the challenge is to estimate the respective parameters, which is typically done by maximum likelihood estimation using the EM algorithm [11, 110]. Gaussian mixture models are a common example for this approach and are used in the following section to illustrate parametric clustering. However, other probability distributions such as Weibull can be used as well, as demonstrated by Mair and Hudec [121].

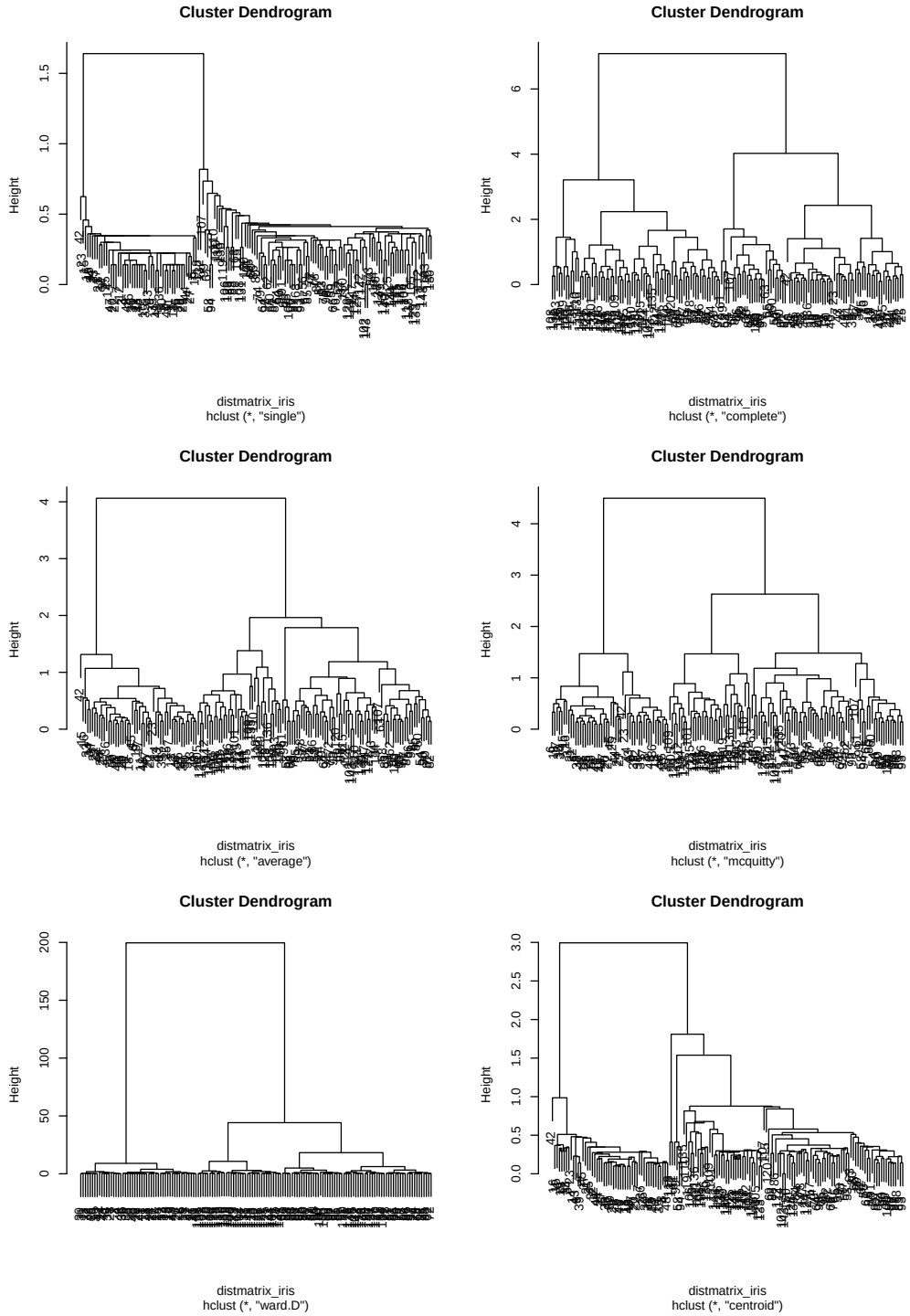

Figure 1.8.: Comparison of the major hierarchical clustering methods on iris data

1.3.3.1. Gaussian Mixture Models (GMM)

A Gaussian mixture model can be written as a linear superposition of several Gaussian components [21, 118]:

$$P(x|\Theta) = \sum_{k=1}^{K} \pi_k p(x|\theta_k) \tag{1.27}$$

where π_k can be regarded as positive weights with $\sum_{k=1}^{K} \pi_k = 1$ and Θ as $\Theta = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$ representing the parameters for all mixture components (i.e. clusters). Thus, each θ_k describes a Gaussian density function p_k where $p_k \sim \mathcal{N}(x|\mu_k, \Sigma_k)$ [118]. The optimal parameters Θ are determined by Maximum Likelihood (ML), which means that given a data set $\mathbf{X}$, ML estimation will try to find parameters Θ where $P(x|\Theta)$ is maximized. This is done by means of the log likelihood function

$$\mathcal{L}(\Theta) = logP(x|\Theta) = log \prod_{i=1}^{N} P(x_i|\Theta) = \sum_{i=1}^{N} log(\sum_{k=1}^{K} \pi_k p_k(x_i|\theta_k)) \tag{1.28}$$

Liu et al. [118] note that finding an optimal solution is difficult due to the log of the sum in the log likelihood function. They therefore suggest a latent variable $P(c|x)$ that represents the possibility of observation x contained in component c, which results in the complete log likelihood function [21, 118]

$$\sum_{i=1}^{N} \sum_{k=1}^{K} P(c_k|x_i)(log\pi_k + log\mathcal{N}(x_i|\mu_k, \Sigma_k)) \tag{1.29}$$

The complete log likelihood function in equation 1.29 is used to obtain estimates for Θ for fixed $P(c|x)$. This is done by means of the EM (Expectation-Maximization) algorithm proposed by Dempster et al. [50] and detailed in algorithm 5. First, initial values for parameters Θ are selected. Friedman et al. [77] suggest choosing μ_k randomly, setting Σ_k equal to the overall sample variance $\sum_{i=1}^{N}(x_i - \bar{x})^2/N$ and selecting 0.5 as an initial value for π. In the expectation step, the posterior probabilities $P(c_k|x_i)$ are computed, i.e. the probability γ_{ik} that x_i is generated by component (i.e. cluster) c_k. In the maximization step, the probabilities (or *responsibilities* in Bishop et al. [21]) γ_{ik} are used to update the estimates of Θ (the weighted means and variances). The two steps are repeated until convergence [77, 110, 118].

An example of a GMM model fit is shown in Figure 1.9, function `Mclust()` of package `mclust` [163] is used to obtain a model for a data set of a mixture of three

Algorithm 5 EM algorithm for Gaussian mixtures based on Bishop et al. [21]

1: Initialize parameters $\Theta = (\mu_1, \Sigma_1, \pi_1, \ldots, \mu_k, \Sigma_k, \pi_k)$ and evaluate initial log likelihood according to equation 1.29.

2: **E step:** evaluate the responsibilities using current parameter values

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_k, \Sigma_k)} \tag{1.30}$$

3: **M step:** Re-estimate the parameters using the current responsibilities:

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{ik} x_i \tag{1.31}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{ik} (x_i - \mu_k^{new})(x_i - \mu_k^{new})^T \tag{1.32}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{1.33}$$

4: Evaluate the log likelihood given in equation 1.29 and check for convergence. If convergence criteria not satisfied return to step 2.

```
gmm_model <- Mclust(testData)
summary(gmm_model)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EEV (ellipsoidal, equal volume and shape) model with
   3 components:

 log-likelihood   n df       BIC       ICL
      -1704.844 450 13 -3489.108 -3536.604

Clustering table:
  1   2   3
146 156 148
```

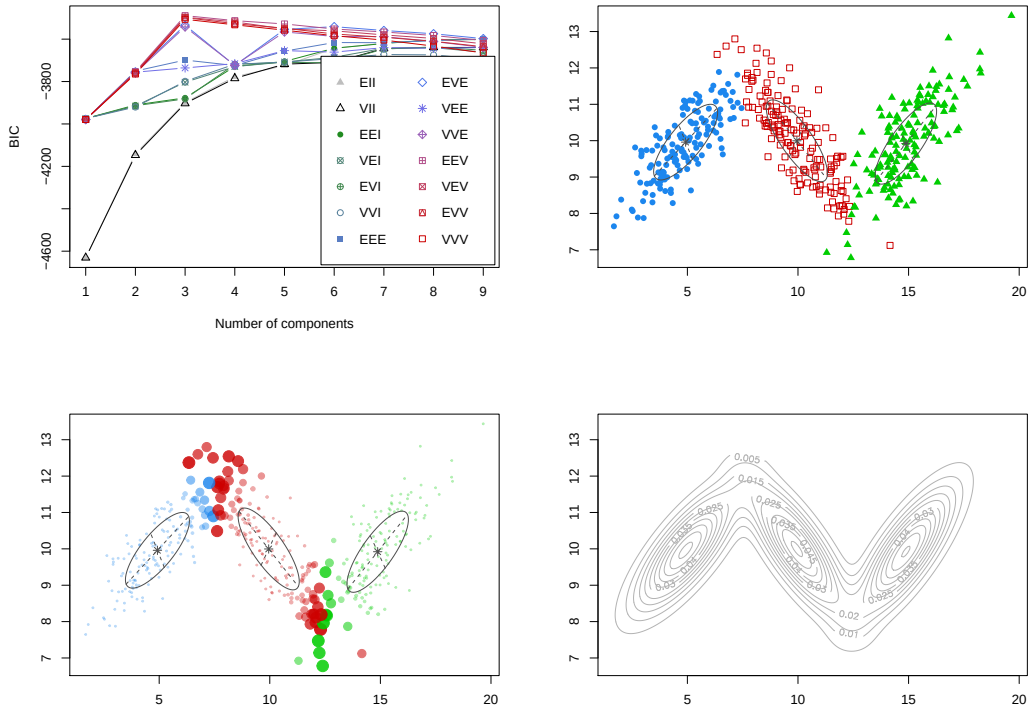

Figure 1.9.: GMM for artificial data (mixture of three Gaussians from [172])

Gaussians, containing 150 observations each. It is possible to specify the number of mixtures as an argument to function `Mclust()`, however, if left out, model selection is done by computing the BIC (Bayesian information criterion) [162] for a range of 1 to 9 Gaussians per default and the most suitable number according to the BIC is selected. In this case, the optimal value of 3 is indeed detected, as the elbow in the BIC curve in the upper left plot in Figure 1.9 shows. Due to the overlapping clusters there is some uncertainty in the cluster border regions as shown in the lower left plot. Overall, the classification plot (upper right) shows a quite accurate classification outcome along with the estimated densities for the three Gaussians (lower right).

Relationship with the k-means Algorithm Bishop et al. [21] note that the k-means algorithm shows a close similarity with the EM algorithm. While k-means does a hard assignment of observations to a particular cluster, the EM algorithm does a soft assignment based on the posterior probabilities. Therefore, fuzzy cluster assignments are possible, as illustrated in the example in Figure 1.9. Moreover, Bishop et al. [21] note that the k-means algorithm can be derived as a non-probabilistic limit of the EM algorithm for GMM [118].

1.3.3.2. Implementation in R

Model based clustering is implemented in a number of R packages, for example `mclust` [163], `EMCluster` [41], `funHDDC` [159],`funFEM` [33], `HDclassif` [16], `gmm` [40], `GMCM` [20] and `mixPHM` [122].

1.4. Cluster Analysis

Handl et al. [86] propose three major steps for cluster analysis, which are illustrated in Figure 1.10: pre-processing, the actual cluster analysis and model validation. Within each step, several choices need to be made.

Firstly, with regard to feature selection, the observations that should be clustered should obviously be represented by informative features [112]. This often may require pre-processing such as standardization/normalization. Furthermore, a crucial point is the selection of a suitable similarity measure/ distance function as discussed in section 1.2.2.1.

Secondly, the clustering itself poses the problem of selecting a particular algorithm. This is a difficult task, as a clustering algorithm encodes a model for the data and assumes a certain structural tendency (e.g. elongated or spherical clusters). [112]. A variety of possible approaches was discussed in section 1.2.2

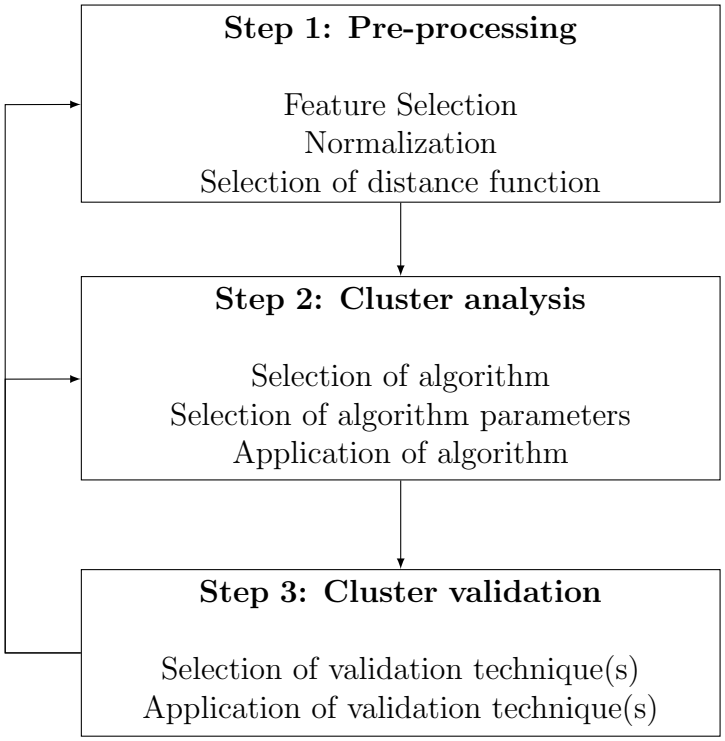

Figure 1.10.: The three main steps in cluster analysis in Handl et al. [86]

Thirdly, the obtained model has to be verified. A detailed review about the range of possible validation techniques is done in chapter 2. This step is likely the most important one, because a cluster algorithm will always produce a model for a given data set. The task during the validation step is to determine whether the model accurately reflects actually existing structure in the data, or whether the grouping obtained is random and does not have a foundation in the data. Furthermore, the validation step (as indicated in Figure 1.10), reflects back on the previous two steps and may lead to a different approach to the data that yields better results.

1.5. Summary

This chapter elaborated on essential terminology of unsupervised machine learning, specifically different approaches to clustering. Subsequently, different model validation techniques are discussed as an important step of the overall cluster analysis.

2. Cluster Validation

In this chapter, common validation methods for cluster models are introduced and discussed with regard to their advantages and drawbacks. Particular emphasis in this context is put on the notion of cluster stability. This technique is also used in the simulation study in chapter 3.

2.1. Background

The previous chapter discussed a variety of methods and approaches toward data clustering. However, as mentioned in section 1.4, selection of a specific method and model computation is only one part of the overall process. Particularly, as Xu and Wunsch [203] note, different methods and input parameters can result in different clusters and/or produce different cluster structures for the same data set (as was demonstrated in the example in Figure 1.5). It therefore follows that an objective and quantitative evaluation of the clusters and the derived cluster structure (i.e. cluster validation) is needed and especially important [60, 82, 85, 99]. If for example a data set contains no cluster structure at all, it is still possible to apply a cluster method but the output will be meaningless and by extension any further analysis [203]. Therefore, the problem of assessing *cluster validity* has two aspects: determining the correct number of clusters and the evaluation of the quality of a clustering solution [110]. This is particularly essential when dealing with data where the data space is beyond the possibilities of visual inspection [85, 110].

For the purpose of this thesis, the scope is now narrowed down to non-parametric partitional methods, specifically the k-centers family algorithms. This is done because particularly for these methods, the choice of the number of clusters is a pivotal parameter that presents a considerable challenge in cluster model validation and is mentioned by Dubes [60] as the "fundamental problem of cluster validity".

2.2. Assessing Cluster Validity

The two main possibilities to validate a partition is via external or internal measures. There is a fundamental difference between these approaches and they are applied in distinct experimental settings [86, 149].

2.2.1. External Measures

External measures use, as the name suggests, pre-specified information about the data set [85, 149, 203]. This is done by comparing the clustering $\mathcal{C}$ of a data set $\mathbf{X}$ with the known true group labels (a priori information) [203], therefore, essentially, the similarity of two sets of labels is calculated. This is done by pairwise comparisons of group labels Y_1 and Y_2. Consequently, four combinations of such a pairwise comparison are possible: two observations compared can be located in the same cluster in both partitions (a), in different clusters in both partitions (d) or in different clusters in Y_1 but in the same cluster in Y_2 (b) and vice versa (c). An illustration of these combinations is shown in Figure 2.1.

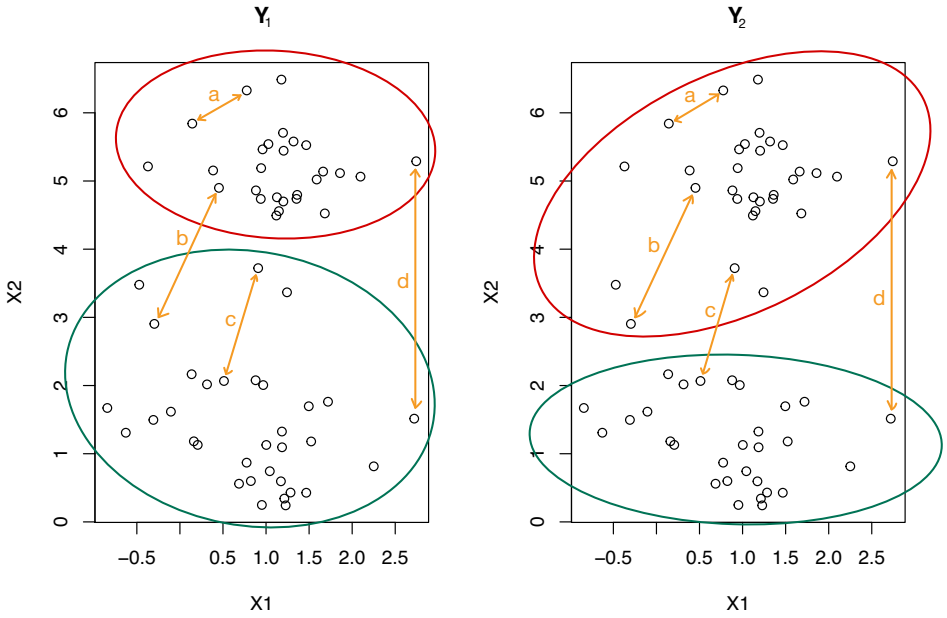

Figure 2.1.: Pairwise comparisons of group labels Y_1 and Y_2

2.2.1.1. Selected Examples of External Indices

In the following, an extensive (but not exhaustive) list of external validation indices is given. These indices are relevant for the subsequent experimental study in chapter 3. Generally, they range between 0 and 1, with some exceptions, as mentioned below.

Rand Index The Rand index by Rand [147] is defined by

$$RI = \frac{a+b}{a+b+c+d} \tag{2.1}$$

Adjusted Rand Index The ARI is a modified form of the Rand Index that corrects for matches that are due to pure chance - in contrast to the Rand Index the ARI can take on values between -1 and 1 [93].

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}] - [\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}]/\binom{n}{2}} \tag{2.2}$$

where n_{ij} is the number of observations that are common to cluster i in Y_1 and cluster j in Y_2 (i.e. a), and $ni.$ and $n_{.j}$ denote the number of observations in cluster i and j in Y and Y'.

Fowlkes-Mallows Index The Fowlkes-Mallows index [72] is given by equation 2.3

$$FM = \sqrt{\frac{a}{a+b}\frac{a}{a+c}} \tag{2.3}$$

Jaccard Similarity The Jaccard similarity by Jaccard [97] is given by

$$J = \frac{a}{a+b+c} \tag{2.4}$$

McNemar Index The McNemar index as described in [51], is defined by

$$NI = \frac{d-c}{\sqrt{d-c}} \tag{2.5}$$

Sokal-Sneath Index The Sokal-Sneath index [169] is given by

$$SSI = \frac{a}{a+2(b+c)} \tag{2.6}$$

Czekanowski-Dice Index The Czekanowski-Dice index as described in [51], is defined as

$$CDI = \frac{2a}{2a+b+c} \tag{2.7}$$

The index is the harmonic mean of precision and recall coefficients and thus identical to the F-measure.

Kulczynski Index The Kulczynski index is defined by (from Kulczyński [109]):

$$KI = \frac{1}{2}(\frac{a}{a+c} + \frac{a}{a+b}) \tag{2.8}$$

Hubert Gamma Index The Hubert $\hat{\Gamma}$ index, described in [85], is the correlation coefficient of two indicator variables Z_1 and Z_2 that are defined as binary variables that take on the value 1 if observations m_i and m_j $(i < j)$ are in the same cluster of the partition and 0 otherwise. The index is thus defined as

$$HUB = Corr(Z_1, Z_2) = \frac{\sum_{i<j}(Z_1(i,j) - \mu_{Z_1})(Z_2(i,j) - \mu_{Z_2})}{n\sigma_{Z_1}\sigma_{Z_2}} \tag{2.9}$$

Using the definition of the pairwise group membership definitions, the index can also be written as

$$HUB = \frac{n \times a - (a+b)(a+c)}{\sqrt{(a+b)(a+c)(d+b)(d+c)}}) \tag{2.10}$$

Unlike most other external indices, HUB takes on values between -1 and 1.

Rogers-Tanimoto Index The Rogers-Tanimoto index by Rogers and Tanimoto [151] is defined as follows:

$$RTI(k) = \frac{a+d}{a+d+2(b+c)} \tag{2.11}$$

2.2.2. Internal Measures

Contrary to external indices, internal measures only use information that is intrinsic to the data set [85], which means that there is no outside information with which to compare group labels. Handl et al. [86] note that internal indices put emphasis on either one or a combination of the clustering qualities such as cluster compactness/connectedness/spatial separation. This reflects the manner in which cluster methods can be categorized (k-centers methods prefer compact, density methods connected clusters), therefore indices that place emphasis on cluster compactness rely on the within-cluster scatter, whereas spatial separation suggests an emphasis on the between-cluster scatter and connectedness obviously focuses on density criteria.

Handl et al. [86] note that the literature provides several combinations of the three aforementioned approaches. Particularly popular in this respect are the indices that incorporate compactness and spatial separation, as they can show opposing trends: increasing the number of clusters tends to decrease the within-cluster

scatter, but at the same time also reduces the distance between clusters. Vice versa, with increasing between-cluster scatter, intra-cluster homogeneity tends to deteriorate. To alleviate this issue, many measures have been proposed that combine both criteria into a final score, e.g. the Dunn index, listed below in equation 2.23 [62], the Davies-Bouldin index in equation 2.20 [48] and the Silhouette index in equation 2.35 [153].

2.2.2.1. Selected Examples of Internal Indices

In the following, a number of commonly used internal indices is listed. The list is not exhaustive, however, it again provides an overview of the indices used with regard to the simulation study in chapter 3.

Hartigan Index The Hartigan index [87] is defined as

$$H(k) = (\frac{W_k}{W_{k+1}})(n-k-1) \tag{2.12}$$

Banfeld-Rafferty Index Banfield and Raftery [11] define an index which is calculated by the weighted sums of the logarithms of the mean within-cluster sum of squares of each cluster.

$$BFI(k) = \sum_{k=1}^{K} n_k \; log(\frac{W_k}{n_k}) \tag{2.13}$$

Calinski-Harabasz Index The CH index, introduced by Calinski and Harabasz [36] is defined as

$$CH(k) = \frac{n-k}{k-1}\frac{B_k}{W_k} \tag{2.14}$$

Krzanowski-Lai Index The KL index, proposed by Lai and T. [111] is given by equation 2.15:

$$KL(k) = |\frac{DIFF_k}{DIFF_{k+1}}| \tag{2.15}$$

where $DIFF$ is defined as

$$DIFF(k) = (k-1)^{2/p} trace(W_{k-1}) - q^{2/p} trace(W_k) \tag{2.16}$$

where p is the number of variables. The maximum of value of $KL(k)$ denotes the optimal number of clusters.

Log-SS-Ratio Index The Log-SS-Ratio index used in Dimitriadou et al. [56] is given by:

$$LSR(k) = log(\frac{B_k}{W_k}) \tag{2.17}$$

Trace-W Index The Trace-W index [52] in equation 2.18 is defined as the within-sum of squares of the partition:

$$TR(k) = W_k \tag{2.18}$$

Ball-Hall Index Ball and Hall [10] introduced an index that calculates the mean of the squared distances of all points with respect to their cluster centroid (from [52]):

$$BALL(k) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{i\in C_k}\|x_{ik} - \tilde{x}_k\|^2 \tag{2.19}$$

where $\tilde{x}$ is a cluster centroid. In the special case where all clusters have equal size, the equation is reduced to $\frac{1}{n_k}W_k$

Davies-Bouldin Index For the Davies-Bouldin index [48, 52], we first define δ_k the mean distance of the points that belong to cluster C_k to their cluster center $\tilde{x}_k$

$$\delta_k = \frac{1}{n_k}\sum_{i\in C_k}\|x_{ik} - \tilde{x}_k\| \tag{2.20}$$

We also define as $\delta_{k,k'}$ the distance between two cluster centers $\tilde{x}_k$ and $\tilde{x}_{k'}$

$$\Delta_{k,k'} = d(\tilde{x}_k, \tilde{x}_{k'}) = \|\tilde{x}_{k'} - \tilde{x}_k\| \tag{2.21}$$

For each cluster k, the maximum M_k of $\frac{\delta_k+\delta_{k'}}{\delta_{k,k'}}$ is computed for all $k \neq k'$. The Davies-Bouldin index is the mean of the values M_k:

$$DB(k) = \frac{1}{K}\sum_{k=1}^{K}M_k = \frac{1}{K}\sum_{k=1}^{K}\max_{k\neq k'}\frac{\delta_k + \delta_{k'}}{\Delta_{k,k'}} \tag{2.22}$$

Dunn Index The Dunn index [52, 62] measures the distance between two clusters (C_k) by calculating the distance between their closest points:

$$d_{k,k'} = \min_{i \in C_k j \in C'_k} \|m_i^k - m_j^{k'}\| \tag{2.23}$$

and taking d_{min} as the minimum of the vector of distances $d_{k,k'}$:

$$d_{min} = \min_{k \neq k'} d_{k,k'} \tag{2.24}$$

Furthermore, we denote with D_k the largest distance between two points within a cluster

$$D_k = \max_{i,j \in C_k i \neq j} \|x_{ik} - x_{jk'}\| \tag{2.25}$$

and define d_{max} as the largest distance of the cluster diameters D_k

$$d_{max} = \max_{1 \leq k \leq K} D_k \tag{2.26}$$

Finally, the Dunn index is defined as:

$$DUNN(k) = \frac{d_{min}}{d_{max}} \tag{2.27}$$

PBM Index The PBM index, developed by Pakhira et al. [139], is calculated as follows [52]:

$$PBM(k) = (\frac{1}{k} \times \frac{E_T}{E_W} \times D_B)^2 \tag{2.28}$$

where E_W is the sum of the distances of the points in each cluster to their centroid and E_T the same to the data set centroid (i.e. the one-cluster solution). Obviously, E_T does not depend on the partition or the number of clusters, but is a constant value. D_B denotes the largest distance between two cluster centroids $(\tilde{x}_k, \tilde{x}_{k'})$:

$$D_B = \max d(\tilde{x}_k, \tilde{x}_{k'}) \tag{2.29}$$

Silhouette Index The silhouette index, introduced by Rousseeuw [153] is computed as follows (from Desgraupes [52]). First, the within-cluster mean distance of a point x_i to all other points of the same cluster is calculated:

$$a(i) = \frac{1}{n_k - 1} \sum_{\substack{i' \in C_k \\ i \neq i'}} d(x_i, x_{i'}) \tag{2.30}$$

Second, the mean distance of x_i to points of the other clusters is given by

$$D(x_i, C_{k'}) = \frac{1}{n_{k'}} \sum_{i' \in C_{k'}} d(x_i, x_{i'}) \tag{2.31}$$

where $b(i)$ is defined as the minimum of these distances

$$b(i) = \min_{k' \neq k} D(x_i, C_{k'}) \tag{2.32}$$

for each point x_i the silhouette width is then given by

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} \tag{2.33}$$

which, as Desgraupes [52] notes, results in a value between 1 and -1, where 1 means that point x_i is assigned to the correct cluster, and -1 means that x_i should be assigned to a different cluster. The mean silhouette width for a cluster C_k is then given by

$$S_k = \frac{1}{n_k} \sum_{i \in C_k} s(i) \tag{2.34}$$

which finally results in the silhouette index that is the mean of all cluster mean silhouettes

$$SIL(k) = \frac{1}{K} \sum_{k=1}^{K} S_k \tag{2.35}$$

Xu Index Proposed by Xu [201], the Xu index is given by:

$$XU(k) = p \log(\sqrt{\frac{W_k}{dn^2}}) + \log(k) \tag{2.36}$$

where p is the number of variables in the data set.

Gap Statistic The Gap statistic, proposed by Tibshirani et al. [180] is computed as follows:

$$GAP(k) = \frac{1}{B} \sum_{b=1}^{B} \log W_{kb} - \log W_k \tag{2.37}$$

where B is the number of reference data sets generated from a uniform distribution within the bounding rectangle of the original data. W_{kb} denotes the within cluster

sum of squares of the reference data sets. The optimal number of clusters is indicated by the largest gap in the index values.

2.2.3. Issues with Internal and External Validation

Unfortunately, as Handl et al. [86] point out, there are issues for both validation approaches. External measures suffer from biases with regard to the number of clusters and the distribution of cluster and class sizes in a partitioning [85]. For example, when looking at completeness of clusters, a one-cluster partition will of course score perfectly and decrease as more clusters are added. Internal measures show a similar tendency, as many of them are based on intra-cluster scatter which is highest for the one-cluster solution and decreases with a growing number of clusters. Also, internal measures can show biases with regard to the shape of the underlying data and structure of the partitioning [86], because as mentioned above, these measures rely on different clustering qualities.

2.3. Cluster Stability

As mentioned above, some validation measures may show biases toward a particular clustering quality that could affect the meaningfulness of the validation process. Therefore, if one seeks to establish a general procedure for cluster validation, it should be applicable to any clustering algorithm and not be by definition restricted to a particular group of methods [112]. This avoids the aforementioned bias by not relying on assumptions about group structure or cluster quality in the data. Lange et al. [112] therefore suggest a *notion of the stability of clustering solutions*. This means that in principle, for two data sets drawn from the same probabilistic source, the clustering should deliver similar results. In essence, it is assessed whether solutions of clustering are replicable [112, 187].

This approach is illustrated in Figure 2.2. Three data sets are drawn from the same probabilistic source (the true number of groups in this simulated data set is 4) and clustered repeatedly for several values of K. In the case of $K = 2$, where K is smaller than the actual value, the true clusters are merged randomly. Given the cluster structure in this example, the observations could reasonably be grouped in two ways, and indeed these two versions occur during the three iterations. This number of groups is thus not stable. The situation is even worse for $K = 3$ and $K = 5$. One cluster is either randomly merged or split and the resulting clustering is clearly not stable over the three iterations. Only the correct value $K = 4$ consistently produces the same cluster structure each time, even though the data sets differ slightly for each iteration. They are however drawn from the same source and therefore the correct number of clusters produces a stable result. According

to von Luxburg [187], the stability notion also elegantly avoids the question of defining what a good clustering is - it only requires the results to be consistent.

A suggestion of how a general stability based validation algorithm can be formulated is provided by von Luxburg [187]. To begin with, a distance measure is needed to determine similarity between two clusterings $\mathcal{C}(\mathbf{X}_N)$ and $\mathcal{C}(\mathbf{X'}_N)$ for two data sets $\mathbf{X}$ and $\mathbf{X'}$ and sample size N. The *instability* of a clustering algorithm $\mathcal{A}$ for a fixed number of clusters K and sample size N is then given by the expected distance between the two clusterings on the different data sets [187]:

$$Instab(K, n) := E(d(\mathcal{C}_K(\mathbf{X}_n), \mathcal{C}_K(\mathbf{X}'_n))) \tag{2.38}$$

The instability in equation 2.38 should be minimized for a stable clustering solution. Algorithm 6 has been proposed by von Luxburg [187] as a general concept that a variety of stability based validation methods follow. Essentially, for each number of clusters where cluster stability should be evaluated, a certain number of perturbed data sets are drawn. These are clustered and pairwise distances calculated. Finally the instability is computed and the number of clusters K is chosen where a minimum is found.

The algorithm requires three important choices: firstly, the manner of the data modification to obtain perturbed versions. Secondly, which method should be used to compare the two clusterings. Thirdly, a choice of a distance measure to compute the pairwise distances of clusterings.

With regard to data set perturbation, von Luxburg [187] mentions several strategies that have been used in practice, such as *drawing a random subsample without replacement* [15, 74, 112, 116], *adding random noise* [22, 127], *random projections in low-dimensional spaces* (in the case of high-dimensional data) [168], *sample data from a model* (for model-based clustering) [102] or *drawing a random sample with replacement* (bootstrapping) [181]. It is essential to strike a careful balance when employing one of the aforementioned strategies. If for example too much noise is added to the data or the subsample is too small, the cluster structure might be distorted and/or destroyed and thus made undetectable for the cluster algorithm. If on the other hand too little modification is applied, the algorithm will always obtain the same results and therefore trivial stability [187].

The second choice, how to compare clusterings in line 7 of algorithm 6, can be addressed in three ways [187]: either the clustering obtained on the original data can be compared with the clusterings of the subsamples [116], or clusterings of overlapping subsamples [15], or clusterings of disjoint subsamples (which requires an extension operator to extend each clustering to the domain of the second) [74, 112].

The third choice pertains to the measure used to calculate the distance between clusterings. If the two clusterings are defined on the same data, any external index

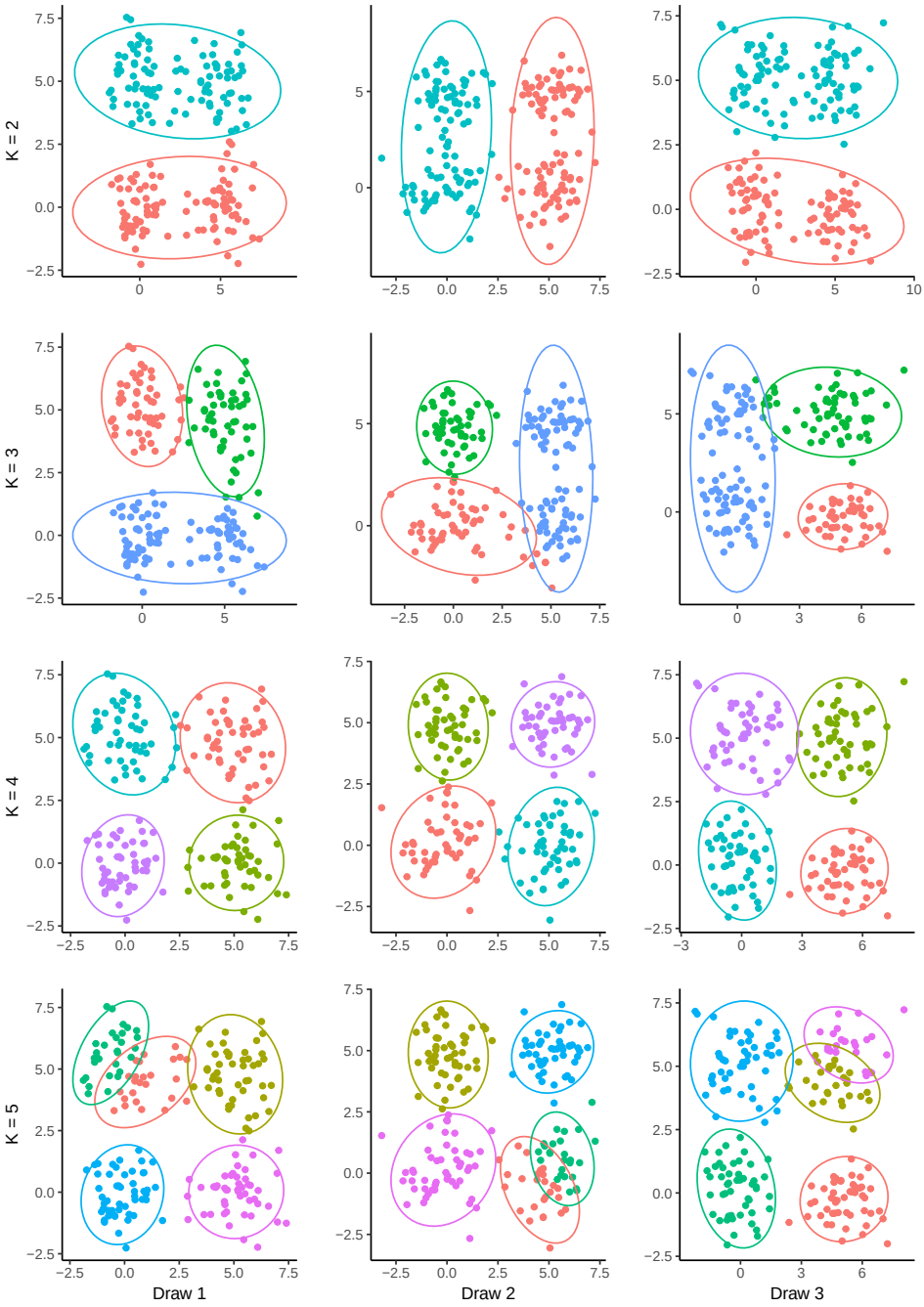

Figure 2.2.: Clustering results on three data sets drawn from the same source for several values of K. Only for the correct number of clusters $K = 4$ all three runs produce the same and thus stable result

Algorithm 6 Stability-based validation procedure [187]

Require: a set $\mathbf{X}$ of data points
Require: a clustering algorithm $\mathcal{A}$
Require: the number of clusters K
Require: the number of modified data sets b

1: **for** $k = 2, \ldots, K$ **do**
2: Generate perturbed versions $\mathbf{X}_b (b = 1, \ldots, b_{max})$ of the original data set
3: **for** $b = 1, \ldots, b_{max}$ **do**
4: Cluster the data set $\mathbf{X}_b$ with $\mathcal{A}$ into k clusters to obtain clustering $\mathcal{C}_b$
5: **end for**
6: **for** $b, b' = 1, \ldots, b_{max}$ **do**
7: Compute pairwise distances $d(\mathcal{C}_b, \mathcal{C}_{b'})$ between these clusterings
8: **end for**
9: **end for**
10: Compute instability as the mean distance between clusterings $\mathcal{C}_b$:

$$\widehat{Instab}(k, n) = \frac{1}{b_{max}^2} \sum_{b,b'=1}^{b_{max}} d(\mathcal{C}_b, \mathcal{C}_{b'}) \tag{2.39}$$

11: Choose the parameter k that gives the best stability, in the simplest case as follows:

$$K := \arg\min_k \widehat{Instab}(k, n) \tag{2.40}$$

such as those listed in section 2.2.1.1 can be used. If the clusterings are defined on different data sets, one can either use a *restriction operator* to restrict the clusterings to the data points that are common to both data sets (in the case of overlapping subsamples) or use an *extension operator* to extend the domain of the first clustering into the domain of the second. This can be done by assigning new data to the cluster centers of each clustering [187].

Finally, the calculated distances are summarized - in the case of line 10 of algorithm 6 by using the mean. However, this is the simplest summary statistic, and more elaborate statistics may be used [15, 18, 187]. The decision for a concrete number of clusters is then done by selecting K for which minimum instability is observed [116, 187]. von Luxburg [187] notes that $\widehat{Instab}(k, n)$ trivially scales with increasing values of k, regardless of the underlying data structure. It therefore may be necessary to normalize the values of $\widehat{Instab}$, for example by using a null reference distribution [18, 74] or random cluster label permutation [112]. The first method is used in the CLEST [61] method, where a reference distribution is defined on the same domain as the data set, but without any cluster structure. Using a random uniform distribution is a simple means of doing so [187]. The stability score of the reference data is then used to normalize the observed score, thus $\widehat{Instab}_{norm} := \frac{\widehat{Instab}}{\widehat{Instab}_{null}}$ [187]. The second method, random label permutation, is suggested by Lange et al. [112][187] and consists of first clustering the data set $\mathbf{X}$ to obtain clustering $\mathcal{C}$. The cluster labels are then randomly permuted, resulting in $\mathcal{C}_{perm}$. Instability is then calculated for both, and normalized instability is then again given by $\widehat{Instab}_{norm} := \frac{\widehat{Instab}}{\widehat{Instab}_{perm}}$. The number of clusters is then chosen according to the smallest normalized instability given by

$$K = \underset{k=2,\ldots,k_{max}}{\arg\min}\ \widehat{Instab}_{norm}(k, n) \tag{2.41}$$

2.3.1. Stability Based Cluster Validity Methods

In the following, several methods are discussed that incorporate in some form the general notion of cluster stability discussed in the previous section in algorithm 6. This is certainly not an exhaustive list; however, it provides an overview of concrete implementations of the stability criterion in cluster validation.

2.3.1.1. Model Explorer

An early approach of stability based validation is the *Model Explorer* method, introduced by Ben-Hur et al. [15] and shown in algorithm 7. The basic premise of the algorithm is to draw two subsamples of the original data, cluster both and measure according to some similarity criterion S whether the two cluster

models produce a similar result on the points common to both subsamples. This is repeated for a specified number of subsampling runs and some range for k. An important parameter is the selection of the percentage of the original data that is used for a subsample, f. If chosen too small, the subsample might be not sufficient to accurately reflect the clusters in the data. Some could be distorted or disappear entirely. Ben-Hur et al. [15] recommend a value of 0.8.

Algorithm 7 Model Explorer algorithm [15]

Require: a data set $\mathbf{X}$
Require: a maximum number of clusters k_{max}
Require: a number of subsamples n_{sub}
Require: a clustering algorithm $\mathcal{A}(\mathbf{X}, k)$
Require: a similarity measure between labels $s(Y_1, Y_2)$
Set a size of the subsamples: $f = 0.8$
for $k = 2, \ldots, k = k_{max}$ **do**
    for $i = 1, \ldots, n_{sub}$ **do**
        $X_1' = \text{subsamp}(\mathbf{X}, f)$
        $X_2' = \text{subsamp}(\mathbf{X}, f)$
        $Y_1 = \mathcal{A}(X_1', k)$
        $Y_2 = \mathcal{A}(X_2', k)$
        $Int = X_1' \cap X_2'$
        $S(i,k) = s(Y_1(Int), Y_2(Int))$, which computes the similarity on the points common to both subsamples
    end for
end for
Select k where similarity is close to 1

2.3.1.2. Figure of Merit

Another noteworthy early approach to resampling based validation is the *Figure of Merit* by Levine and Domany [116]. The basic premise is the same as for the model explorer algorithm, there is a parameter f (called *dilution factor*) that is used to generate a number of subsamples of the original data. The difference is that not pairs of subsamples are compared, but rather the cluster-connectivity matrices of the subsamples which are given by

$$\mathcal{T}_{ij} = \begin{cases} 1, & \text{if points } i \text{ and } j \text{ belong to the same cluster} \\ 0, & \text{otherwise} \end{cases} \tag{2.42}$$

Levine and Domany [116] then define the *figure of merit* $\mathcal{M}(V)$ as the percentage

of agreement between the connectivity matrices obtained from the resampled data sets $\mathcal{T}^{(\mu)}$ where $\mu = 1 \dots m$ and m is the number of resamples drawn and the original data connectivity matrix $\mathcal{T}$

$$\mathcal{M}(V) = \langle \langle \delta_{\mathcal{T}_{ij}, \mathcal{T}_{ij}^{(\mu)}} \rangle \rangle m \tag{2.43}$$

Thus for each pair of observations in the original data an average is calculated of how many are present in one result of a subsample cluster solution. This is in a second step averaged over all m resampling iterations. Consequently, $0 \leq \mathcal{M}(V) \leq 1$, where a score of 1 would indicate a perfect solution. The procedure is summarized in algorithm 8.

Algorithm 8 Figure of merit method Levine and Domany [116]

Require: a data set $\mathbf{X}$
Require: a cluster algorithm $\mathcal{A}$
1: Choose parameters V of the cluster algorithm $\mathcal{A}$
2: Perform cluster analysis on the full data set $\mathbf{X}$
3: Construct m subsets of the original data set, by randomly selecting a subset of size f
4: **for** $m = 1, \dots, m_{max}$ **do**
5: Perform cluster analysis for each subset
6: Calculate $\mathcal{M}(V)$
7: **end for**
8: Vary parameters V in order to receive stable clusters where a local maximum of $\mathcal{M}$ is reached

2.3.1.3. Stability Based Model Order Selection

Lange et al. [112] propose a different approach to resampling based validation by noting that the algorithms by Levine and Domany [116] and Ben-Hur et al. [15] could be biased as they measure similarity of clusterings on two non-disjoint data sets. Therefore the overlap of the two subsets could already determine the group structure, which in turn can lead to an artificially induced stability that is of course not desired.

Thus Lange et al. [112] suggest to split the original data $\mathbf{X}$ (over r resampling runs) into two disjoint data sets X_1' and X_2' of equal size and apply a clustering algorithm $\mathcal{A}$ to both. Consequently, the results are not comparable because they are computed from different data sets. This again is precisely the difference to the methods proposed by Levine and Domany [116] and Ben-Hur et al. [15] whose approaches to measuring similarity use subset overlap. In this case a mechanism is

needed to make the two clusterings comparable, as mentioned above, an extension operator is necessary to extend the domain of one clustering into the other. Lange et al. [112] suggest using $\mathbf{X}$ and its clustering result to train a classifier. The classifier in turn is used to predict the group labels for X'. These can then be compared to the result of the cluster algorithm for X'.

Thereafter, s random labelings of the clusters are generated and distances computed. The average dissimilarity of the random labeling algorithm $\mathcal{R}$ is calculated as $\hat{S}(\mathcal{R}_k)$ and used to normalize the average dissimilarity of the original data set clustering $\hat{S}(\mathcal{A}_k)$ as given in equation 2.44. Finally, k is chosen according to the lowest obtained value of $\bar{S}$. Algorithm 9 details the method.

Algorithm 9 Stability based model order selection by Lange et al. [112]

Require: a data set $\mathbf{X}$
Require: a number of splits r
1: **for** $k_{min}, \ldots, k_{max}$ **do**
2: **for** $r = 1, \ldots, r_{max}$ **do**
3: Split data into two halves X'_1 and X'_2 and apply $\mathcal{A}$ to both
4: Use $(X'_1, \mathcal{A}(X'_1))$ to train a classifier ϕ and compute $\phi(X'_2)$
5: Compute distance between solutions $\phi(X'_2)$ and $\mathcal{A}(X'_2)$ for X'_2
6: **end for**
7: Sample s random k-labelings and compute $\hat{S}(\mathcal{R}_k)$
8: Normalize each $\hat{S}(\mathcal{A}_k)$ with $\hat{S}(\mathcal{R}_k)$ to get an estimate for $\bar{S}(\mathcal{A}_k)$:

$$\bar{S}(\mathcal{A}_k) := \frac{\hat{S}(\mathcal{A}_k)}{\hat{S}(\mathcal{R}_k)} \tag{2.44}$$

9: **end for**
10: Return $\hat{k} = argmin_k \bar{S}(\mathcal{A}_k)$ as the estimated number of clusters

2.3.1.4. Consensus Clustering

Consensus clustering, shown in algorithm 10 and proposed by Monti et al. [128], implements the stability notion by calculating connectivity matrices which again show the pairwise cluster co-memberships. These are computed for a certain number of resampling iterations r according to some resampling scheme R (the exact method of generating perturbed versions of the data is not strictly prescribed). The definition of these connectivity matrices is the same as in 2.42, defined here by $M^{(r)}(i, j)$ [128]. In addition, an indicator matrix $I^{(r)}(i, j)$ is needed, which analogously encodes whether observations i and j of the original data are both present in the resampled data set.

Algorithm 10 Consensus clustering by Monti et al. [128]

Require: a data set $\mathbf{X}$
Require: a clustering algorithm $\mathcal{A}$
Require: a resampling scheme R
Require: a number of resampling iterations r
for $k = 2, \ldots, k_{max}$ **do**
    $M \leftarrow 0$ (empty set of connectivity matrices)
    for $r = 1, \ldots, r_{max}$ **do**
        $D^{(r)} \leftarrow R(D)$
        $M^{(r)} \leftarrow \mathcal{A}(D^{(r)}, k)$
        $M \leftarrow M \cup M^{(r)}$
    end for
    $\mathcal{M}^{(k)} \leftarrow$ compute consensus matrix from $M = \{M^{(1)} \ldots M^{(r)}\}$
end for
Choose k based on the consensus distribution of $\mathcal{M}^{(k)}$

Once a set of connectivity matrices has been obtained, the consensus matrix $\mathcal{M}$ can be calculated as follows:

$$\mathcal{M}_{(i,j)} = \frac{\sum_r M^{(r)}(i,j)}{\sum_r I^{(r)}(i,j)} \tag{2.45}$$

Therefore each entry of $\mathcal{M}$ can take on a value between 0 and 1, where 1 denotes perfect consensus. If the correct value for k was chosen, there should be a clear block structure when the consensus matrix is plotted as a color coded heat map. An example of this is shown in 2.3, where gene expression data (included as example data set in the R package implementation `ConsensusClusterPlus` [198]) is clustered using k-means and Euclidean distance for $k = 2$ to $k = 4$ along with the cumulative distribution functions. $k = 2$ shows the clearest block structure.

Utro et al. [184] note that the consensus clustering method may be of little use for large data sets due to its significant computational demand, claiming that computation time may take weeks on a state of the art PC.

2.3.1.5. CLEST

The CLEST algorithm (11) by Dudoit and Fridlyand [61] works similarly to the method proposed by Lange et al. [112]. For a range of number of clusters $k = 2, \ldots, K$, disjoint subsamples X_1' and X_2' are clustered for partitions Y_1 and Y_2 and a classifier is built with X_1' and the cluster labels obtained from Y_1. The classifier as well as the cluster algorithm are applied to X_2'. The two sets of labels for X_2' are then compared by means of an external index I. This is done for B subsampling

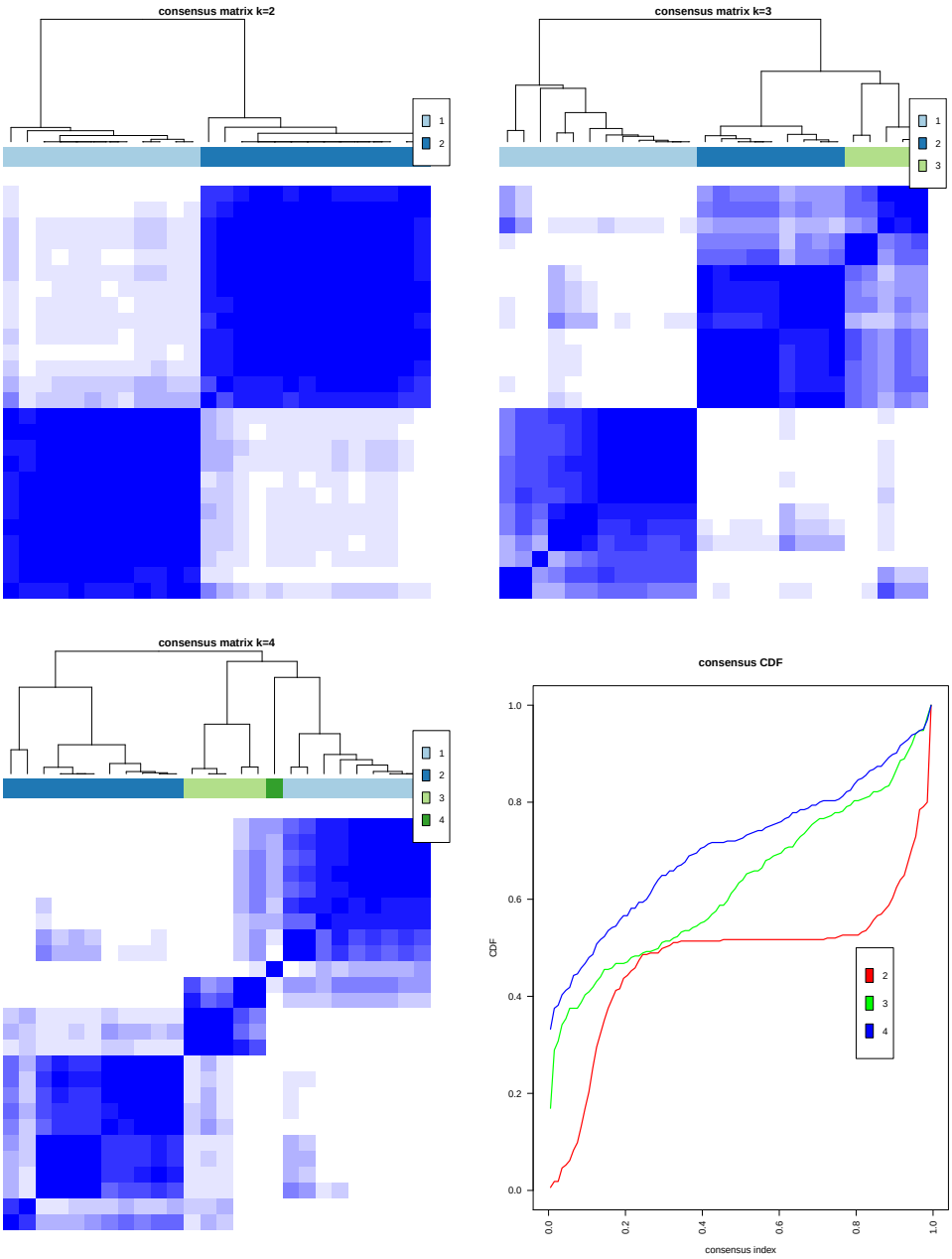

Figure 2.3.: Consensus matrices and CDF plot for k = 2 to k = 4

runs, and a median value of I, defined as t_k, is calculated. The same procedure is then done for artificial data generated under a null hypothesis $H_0 : K = 1$, i.e. no cluster structure in the data, resulting in a value $t_k^0 = [\frac{1}{B_0}] \sum_{b=1}^{B_0} t_{k,b}$ where p_k is the proportion of $t_{k,b}$ that are at least as large as the observed value t_k, thus a p-value for t_k. $d_k = t_k - t_k^0$ is then defined as the difference between the observed and expected value under the null hypothesis of $K = 1$. After repeating for the complete range of possible number of clusters, the set K^- is defined as $K^- = \{2 \leq k \leq K : p_k \leq p_{max}, d_k \geq d_{min}\}$ where p_{max} and d_{min} are preset thresholds. If K^- is empty, the null hypothesis (no cluster structure) holds. Otherwise the number of clusters is estimated as the number that corresponds to the largest significant difference statistic d_k.

Crucial parameters for the CLEST algorithm are the choice of the classifier and the external index for comparison. Dudoit and Fridlyand [61] recommend as classifier linear discriminant analysis with diagonal covariance matrix (DLDA) and as external index the Fowlkes-Mallows similarity measure (FM). Furthermore, Dudoit and Fridlyand [61] recommend PAM (partitioning around medoids) as clustering procedure, p_{max} and d_{min} are both recommended to be set to 0.05, and the reference data sets are proposed to be sampled from the uniform distribution.

2.3.1.6. Prediction Strength

Prediction Strength, proposed by Tibshirani and Walther [179], modifies the concept of the external index somewhat in the sense that it measures agreement between partitions not in general, but cluster-wise. Algorithm 12 details the method. It is not a typical stability based method along the lines of algorithm 6, but rather an extended external index where cluster-wise agreement is measured on two subsamples (splits) of the original data.

After the data is split into a training and a test set (X_1' and X_2'), the distance between the two clusterings is calculated. This is done by using the training set cluster centroids to predict the test set cluster memberships. The labels obtained are compared to the labels from the test set clustering. Distance here is defined by the co-memberships of observations - i.e. whether the same pairs of observations end up in the same cluster in the test set clustering and when training set cluster centroids are used for assignment (algorithm line 4, where $D[P(X_{tr}, k), X_{te}]_{ii'}$ is a $N \times N$ matrix with the ii'th element $= 1$ if observations i and i' from the test data fall into the same cluster if the cluster centers from the training set clustering ($P(X_{tr}, k)$) were used for assignment). Afterward, the prediction strength is computed using equation 2.46, where the minimum of co-membership agreements and thus the least stable cluster is defined as the prediction strength. After repeating for each possible k, the clustering $\mathcal{C}_k$ is selected where the prediction strength is at its maximum.

Algorithm 11 CLEST by Dudoit and Fridlyand [61]

Require: a data set $\mathbf{X}$
Require: a clustering method $\mathcal{A}$
Require: a classifier Cl
Require: an external index I
Require: a number of subsampling runs B
Require: a number or reference data sets B_0
Require: a maximum number of clusters K
Require: thresholds for p_k and d_k

for $k = 2, \ldots, k = K$ **do**
  for $b = 1, \ldots, b = B$ **do**
    Randomly split $\mathbf{X}$ into two disjoint sets, X_1' and X_2'
    Apply $\mathcal{A}$ to the learning set X_1' to obtain a partition Y_1
    Build Cl using the learning set and the cluster labels from Y_1
    Apply Cl to the test set X_2' to obtain labels Y
    Apply $\mathcal{A}$ to the test set to obtain a partition Y_2
    Use I to compare Y and Y_2
  end for
  Calculate the median t_k of the similarity statistics obtained for each k
  Generate B_0 data sets under a suitable null hypothesis.
  for all reference data sets B_0 **do**
    repeat steps 2 to 10 to obtain B_0 similarity statistics $t_{k,1}, \ldots, t_{k,B_0}$
  end for
  Define $t_k^0 = [\frac{1}{B_0}] \sum_{b=1}^{B_0} t_{k,b}$ and let p_k be the proportion of the $t_{k,b}$, $1 \leq b \leq B$ that are at least as large as the observed value t_k
  Define $d_k = t_k - t_k^0$
end for
Define the set K^- as

$$K^- = \{2 \leq k \leq K : p_k \leq p_{max}, d_k \geq d_{min}\}$$

if set K^- is empty **then**
  $\bar{k} = 1$
else
  $\bar{k} = \arg\max_{k \in K^-} d_k$
end if

Algorithm 12 Prediction strength by Tibshirani and Walther [179]

Require: a data set $\mathbf{X}$

1: **for** $k = 2, \ldots, k_{max}$ **do**
2: Split $\mathbf{X}$ in a training and test set (X'_1 and X'_2 respectively)
3: Cluster X'_1 and X'_2 into k clusters
4: Calculate $D[P(X_{tr}, k), X_{te}]_{ii'}$
5: Calculate Prediction Strength (PS):

$$PS(k) = \min_{1 \leq j \leq k} \frac{1}{n_{kj}(n_{kj} - 1)} \sum_{i \neq i' \in A_{kj}} D[P(X_{tr}, k), X_{te}]_{ii'} \tag{2.46}$$

6: **end for**
7: Select the clustering $\mathcal{C}_k$ with the maximum Prediction Strength

2.4. Summary

This chapter has introduced the basic concepts of cluster validation, specifically the notion of validation by cluster stability. Several methods have been discussed that use this procedure. In chapter 3, this matter is explored further by investigating the resampling component of some of the methods discussed and how it might affect the results of validation indices. For this purpose, a benchmarking study was done that incorporates aspects of the methods that have been discussed above. For example, CLEST by Dudoit and Fridlyand [61] and Prediction strength by Tibshirani and Walther [179] are included as validation indices, while Model explorer by Ben-Hur et al. [15] and Stability based model order selection by Lange et al. [112] provided the algorithmic framework for the experimental design of the study. Consensus clustering by Monti et al. [128] and the Figure of Merit by Levine and Domany [116] demonstrate another approach to cluster stability via connectivity matrices. Due to considerations with regard to study design and computational demands, the latter two methods are not included in the study of chapter 3 (although it should be noted that Prediction Strength does include the aspect of cluster-wise co-memberships).

3. An Evaluation of Resampling Based Cluster Validation

This chapter was published in Dangl and Leisch [47].

3.1. Background and Related Work

Generally, a clustering algorithm finds groups of observations in a data set that are similar to each other. While there is a wide variety of clustering methods and algorithms (hierarchical clustering, partitioning algorithms, density based methods, etc.), the basic process of how to achieve a meaningful partition of the data is common to all of them and illustrated in Figure 1.10. A quite important point to note is that the third part of the process - model validation - is perhaps the most important part as it reflects back on the first to steps. The choice of the algorithm, distance measure, variables etc. is obviously put to the test by validation. Thus, thorough validation of a clustering result is crucial to the successful analysis and interpretation of an obtained model. In principle, one commonly distinguishes two main categories of validation measures: internal and external indices. They are calculated for a specific range of number of clusters and the final decision is made according to some optimal value - for some indices this is the maximum value, for others the minimum and for some the biggest drop or increase in index levels (often called the 'elbow' method).

Internal indices use information for the calculation that is intrinsic to the clustering model - e.g. the within-cluster sum-of-squares or the squared distance between cluster centroids. They therefore measure to a varying extent either the degree of cluster separation, compactness and/or connectedness [86]. An overview of internal indices that are used in this study is given in Table 3.1, with a detailed description given in section 2.2.2.1. One remark with regard to the Silhouette index: $a(i)$ and D from equations 2.30 and 2.31 were slightly modified in this study. Usually, these values indicate the average dissimilarity of the ith object to all other objects of the same and nearest cluster respectively. In this study this was replaced with the average dissimilarity to the cluster centroids. This greatly reduces the computational burden and is a justified trade-off for the possibly slightly reduced accuracy. Therefore, the Silhouette index is abbreviated QSIL (Quasi-Silhouette).

Index Name	optimal value
Gap Statistic (GAP)	elbow
Calinski-Harabasz Index (CH)	maximum
Krzanowski-Lai Index (KL)	maximum
Davies-Bouldin Index (DB)	minimum
PBM Index (PBM)	elbow
Hartigan Index (HART)	elbow
Ball Index (BALL)	elbow
Silhouette Index (QSIL)	maximum
Cluster Sum of Squares (TR_W)	elbow
Log-Sum of Squares Ratio (LSR)	elbow
Banfield-Rafferty Index (BFI)	elbow
Xu Index (XU)	elbow
Dunn Index (DUNN)	maximum

Table 3.1.: Internal validation indices

External indices follow a quite different approach: they disregard the internal structure of the data according to the clustering completely and only compare the group labels to some known gold standard. As this means that external indices essentially measure the similarity between sets, their values typically range between 0 and 1, where obviously a higher value indicates higher similarity (there are exceptions; for example, the Adjusted Rand index and the Hubert $\hat{\Gamma}$ similarity range between -1 and 1). A list of indices selected for this study is given in Table 3.2 and detailed in section 2.2.1.1.

3.2. Resampling Methods

In the following, we discuss the method of resampling based cluster validation and in conjunction the notion of cluster stability. Increased computational power has greatly facilitated the application of resampling-based methods in recent years, and numerous studies have been carried out that indicate that such an approach helps to detect cluster structure present in the data more reliably [112, 133, 152]. The concept of cluster stability is closely connected to resampling-based validation methods: if clusters exist in the data, a clustering algorithm should, given the appropriate number of clusters is selected, detect these clusters reliably over repeated runs (as discussed in section 2.3. This should also be the case if the data is slightly perturbed, as is the case with resampling [86].

As discussed previously, several validation methods that incorporate the notion of stability and resampling have been introduced. For example Monti et al.

Index Name	decision value
Rand Index (RI)	maximum
Adjusted Rand Index (ARI)	maximum
Fowlkes-Mallows Index (FM)	maximum
Jaccard Similarity (J)	maximum
Prediction Strength (PS)	maximum
CLEST Algorithm (CLEST)	maximum
Hubert Similarity Statistic (HUB)	maximum
Czernandowski-Dice Index (CDI)	maximum
Sokal-Sneath Index (SSI)	maximum
Rogers-Tanimoto Index (RTI)	maximum
McNemar Index (NI)	maximum
Kulczinski Index (KI)	maximum

Table 3.2.: External validation indices

[128] propose the method of consensus clustering where a consensus over multiple clustering runs on resampled data is reached. Tibshirani and Walther [179] have developed an external index (PS) that requires splitting of the original data to determine the degree of stability that a certain number of clusters used provides. As indicated above in Table 3.2, the index is also used in this benchmarking study. Another index that is used in this study, the CLEST algorithm by Dudoit and Fridlyand [61], also incorporates repeated resampling in its method. Other measures that base their approach on the notion of stability have been introduced by Volkovich et al. [186], Tseng and Wong [182], Khan and N. [103], and Levine and Domany [116]. As all of these studies show that the stability based approach might be a promising one, the attempt of the study at hand is to investigate the effects of resampling and cluster stability considerations in a systematic way. Therefore we compare the the 'simple' approach, where all validation is done on the original data and its partition, with the resampling approach. Generally, each partition is chosen by applying the k-means algorithm with three random restarts for each value of the number of clusters k. With regard to the resampling approach, there are several possible ways to resample data. We denote the following three methods as the most widely used and representative options:

- Bootstrapping: drawing a subsample with replacement from the original dataset with the same length. Bootstrap samples thus can contain multiple entries of the same observation while others are not present in the subsample at all.
- Splitting: the original data set is split into two non-overlapping halves.

- Subsetting: a certain percentage of the original data set is randomly selected (in this study 75%). Consequently, overlap of varying degree is possible in the two subsamples.

We define the original data set as $\mathbf{X}$, two resampled data sets (e.g. bootstrap samples) as X_1 and X_2, and the respective partitions as Y_1 and Y_2. Furthermore, all indices are calculated for a specific range of clusters $k = 2$ to $k = K$ (the case of $k = 1$ is obviously disregarded, as it would always result in perfectly stable values).

Similarly as Roth et al. [152], who define stability as the variability of two clustering solutions drawn from the same source, we proceed in the case of external indices - the cluster labels from Y_1 and Y_2 are compared to each other repeatedly to measure stability (in contrast to the usual way of using external indices - by comparing the cluster labels to a known truth). In order to calculate similarity, all external indices in this study use co-memberships of observations. This is done by calculating whether a pair of observations is either located in the same cluster in both Y_1 and Y_2 or in different clusters in both Y_1 and Y_2 (agreement of the two partitions) or in the same cluster in Y_1 but in different clusters in Y_2 and vice versa (disagreement between partitions) - see Figure 2.1. There are two exceptions that modify this general approach to some degree: Prediction Strength and the CLEST method (see the previous chapter for details). As mentioned above, for the 'simple' approach, all validation is done on the original data, therefore the original data is used twice for the external validation measures, including the CLEST and Prediction Strength algorithms (which, as mentioned above, are actually designed to be resampling-based methods). However, for the sake of investigating the primary hypothesis, we use the original data as pretend-resampled data.

For internal indices, we use a slightly different approach - as these indices do not need a second clustering to measure agreement, we only draw one subsample. This means that the splitting and subsetting scheme only differ with regard to the percentage of observations selected. For the simple approach, the original data is used. In both cases we assume that a correct number of clusters is found when over repeated resampling runs, index values show a stable trend toward a particular k.

3.3. Data

In order to accurately evaluate index performance and differences between resampling strategies, artificial data sets are used. These are drawn from different finite mixtures of multivariate Gaussian distributions, with features that are selected to reflect a range of difficulty with regard to number of variables, cluster separation, number of observations and number of clusters. A full-factorial design is used,

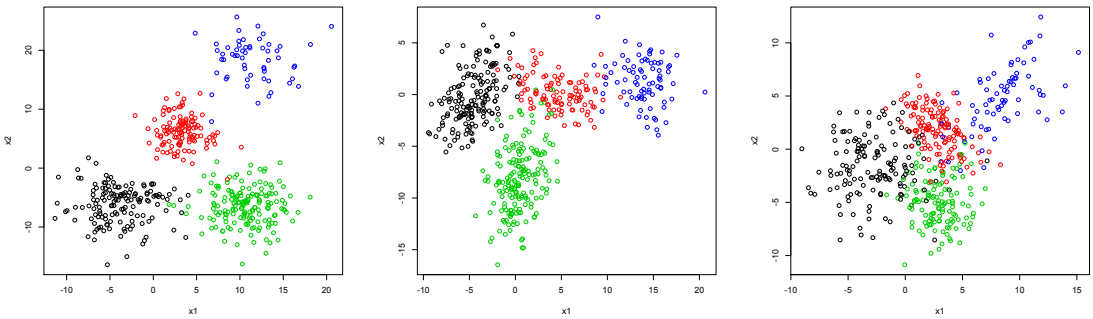

Figure 3.1.: From left to right: good, close and overlapping clusters

resulting in the case at hand in 54 data sets (or rather, simulation settings) of varying difficulty. The values for the four data set features are illustrated in Table 3.3, and a table with detailed parameters for each setting is shown in section 3.4.2.

The values are taken from Dolnicar et al. [57], but the setup in this study is much smaller. We use less possibilities with regard to the number of observations - 30 and 60 times the number of variables - as for the purpose of this study only a smaller and a bigger option for data set size is needed. We also do not include noise variables in the data sets.

No. of variables	10, 16, 22
No. of clusters	4, 6, 8
No. of observations	30/60 times the no. of variables
Cluster separation	0.1, 0, -0.1

Table 3.3.: Summary of data set features

The values for cluster separation refer to the separation index as described by Qiu and Joe [144]. 1 indicates maximum possible cluster separation, -1 means complete overlap. Thus the three values above indicate slight separation, close clusters and beginning overlap. For a two-dimensional data set, the cluster separation values are illustrated in Figure 3.1.

3.4. Benchmarking Setup

3.4.1. Experimental Design

The study is set up as follows: for each of the 54 simulation settings, 50 instances are generated. These 50 instances of one data set are clustered with the k-means algorithm and the aforementioned indices for a range of 2 to 10 clusters are calculated. The supposed number of clusters is derived from these index values and finally a percentage indicates how accurate the index is over the 50 instances of a particular data set. This is the approach for the simple calculation, i.e. without resampling. Additionally, each of the 50 instances of one data set is resampled 100 times for each resampling strategy mentioned above. The median index value of the 100 subsamples is selected as the index value for each of the 50 instances. The rest of the procedure is analogous to the simple calculation. Figure 3.2 illustrates the process.

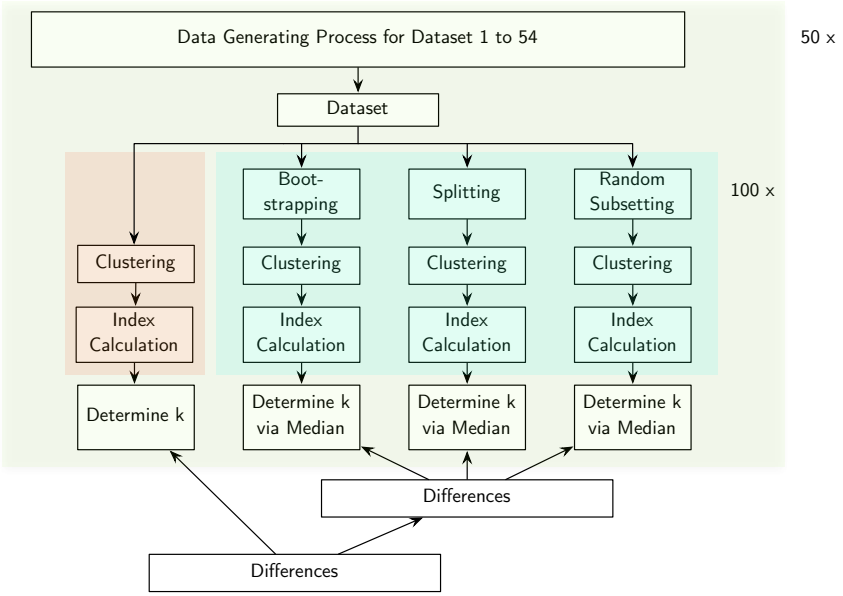

Figure 3.2.: Benchmarking setup

This form of the setup (50 instances for each data set, 100 subsamples) has been inspired by Tibshirani and Walther [179] and Ben-Hur et al. [15]. According to Ben-Hur et al. [15], the huge number of subsamples that are drawn, clustered and put through index calculation is used to make sure that the resampled median index values are based on a sufficiently large number of calculations and thus satisfy the needs of stability based validation. Compared to the simple calculations, we

can then reliably test the significance of whether resampling is actually worthwhile given the considerably higher computational effort. Furthermore, as Figure 3.2 indicates, we also test whether there is a significant difference between the three main possibilities of doing resampling.

3.4.2. Simulation Settings

Table 3.4 lists the parameters for each of the simulation settings. These values are used as input to the function `genRandomClust()` of package `clusterGeneration`. The size of the clusters is determined by selecting a value $\frac{Observations \times Variables}{k}$, and multiplying by 0.97 and 1.03 for an lower and upper boundary. Within this range, a value for each cluster is randomly selected. By this, the clusters have roughly equal, but not exactly the same size. All other inputs of `genRandomClust()` were left at their default values.

3.4.3. Hardware and Software

With regard to the hardware and software used, calculations were run on the Vienna Scientific Cluster (VSC2), which provides 1314 nodes with 16 processor cores each. The statistical programming environment R [146] was used to carry out the analysis and in particular, package `clusterGeneration` [145] was used to create the benchmark data sets and the package `flexclust` [115] was used for the clustering. The internal and external indices and benchmarking functions were written from scratch for this analysis and implemented to work with `flexclust`.

3.5. Results

The data produced by the simulation study is analyzed as follows: for each of the 54 simulation settings, a result table was calculated that shows the index accuracy percentage out of the 50 draws generated with respect to that particular setting. Table 3.5 provides an example. As already mentioned and illustrated in Figure 3.2, for the simple slot, these percentages are only calculated from the 50 index values obtained from the original data clustering instances (obviously, as there is no subsample). For the resampling slots, the 50 index values are in turn based on the median value of 100 subsample clusterings. The median was used due to its higher robustness than the mean in order to ease the influence of bad resamples and the resulting low index value.

Furthermore, each of the result tables have been reduced to an average value over all indices per scheme. For the values in Table 3.5, this is shown in Table 3.6. This allows for a direct comparison of the four schemes; Figure 3.3 shows

Setting	Clusters	No. of observations (times no. of variables)	No. of vars.	Degree of Sep.
1	4.00	30.00	10.00	-0.10
2	4.00	30.00	10.00	0.00
3	4.00	30.00	10.00	0.10
4	4.00	30.00	16.00	-0.10
5	4.00	30.00	16.00	0.00
6	4.00	30.00	16.00	0.10
7	4.00	30.00	22.00	-0.10
8	4.00	30.00	22.00	0.00
9	4.00	30.00	22.00	0.10
10	4.00	60.00	10.00	-0.10
11	4.00	60.00	10.00	0.00
12	4.00	60.00	10.00	0.10
13	4.00	60.00	16.00	-0.10
14	4.00	60.00	16.00	0.00
15	4.00	60.00	16.00	0.10
16	4.00	60.00	22.00	-0.10
17	4.00	60.00	22.00	0.00
18	4.00	60.00	22.00	0.10
19	6.00	30.00	10.00	-0.10
20	6.00	30.00	10.00	0.00
21	6.00	30.00	10.00	0.10
22	6.00	30.00	16.00	-0.10
23	6.00	30.00	16.00	0.00
24	6.00	30.00	16.00	0.10
25	6.00	30.00	22.00	-0.10
26	6.00	30.00	22.00	0.00
27	6.00	30.00	22.00	0.10
28	6.00	60.00	10.00	-0.10
29	6.00	60.00	10.00	0.00
30	6.00	60.00	10.00	0.10
31	6.00	60.00	16.00	-0.10
32	6.00	60.00	16.00	0.00
33	6.00	60.00	16.00	0.10
34	6.00	60.00	22.00	-0.10
35	6.00	60.00	22.00	0.00
36	6.00	60.00	22.00	0.10
37	8.00	30.00	10.00	-0.10
38	8.00	30.00	10.00	0.00
39	8.00	30.00	10.00	0.10
40	8.00	30.00	16.00	-0.10
41	8.00	30.00	16.00	0.00
42	8.00	30.00	16.00	0.10
43	8.00	30.00	22.00	-0.10
44	8.00	30.00	22.00	0.00
45	8.00	30.00	22.00	0.10
46	8.00	60.00	10.00	-0.10
47	8.00	60.00	10.00	0.00
48	8.00	60.00	10.00	0.10
49	8.00	60.00	16.00	-0.10
50	8.00	60.00	16.00	0.00
51	8.00	60.00	16.00	0.10
52	8.00	60.00	22.00	-0.10
53	8.00	60.00	22.00	0.00
54	8.00	60.00	22.00	0.10

Table 3.4.: Summary of simulation parameters

	Bootstrapping	Splitting	Subsetting	Simple
ARI	1.00	1.00	0.98	0.80
RI	1.00	1.00	0.98	0.82
J	1.00	1.00	0.98	0.80
FM	1.00	1.00	0.98	0.80
PS	1.00	1.00	1.00	0.90
GAP	1.00	1.00	1.00	1.00
CH	1.00	1.00	1.00	1.00
KL	0.98	0.96	0.98	0.74
DB	0.96	0.98	0.96	0.98
PBM	0.48	0.52	0.42	0.14
HART	0.80	0.86	0.76	0.64
BALL	0.94	0.92	0.96	0.70
CLEST	0.90	0.94	0.94	0.76
QSIL	0.98	0.98	0.98	0.98
TR_W	1.00	1.00	1.00	1.00
LSR	0.94	0.94	0.94	0.96
HUB	1.00	1.00	0.98	0.80
CDI	1.00	1.00	0.98	0.80
RTI	1.00	1.00	0.98	0.82
SSI	1.00	1.00	0.98	0.80
NI	0.96	0.96	0.96	0.58
KI	1.00	1.00	0.98	0.80
BFI	1.00	1.00	1.00	1.00
XU	1.00	1.00	0.98	1.00
DUNN	0.70	0.60	0.70	0.34

Table 3.5.: Benchmarking result for data set 3 (10 variables, 300 Observations, 4 clusters, good separation)

Bootstrapping	Splitting	Subsetting	Simple
0.9456	0.9464	0.9360	0.7984

Table 3.6.: Scheme means for Dataset 3

these average accuracies per scheme for all data sets. The values are sorted by increasing accuracy, with the simple scheme as the reference category. The same sorting is obviously used for the resampling slots, because of this the increase is not monotonic.

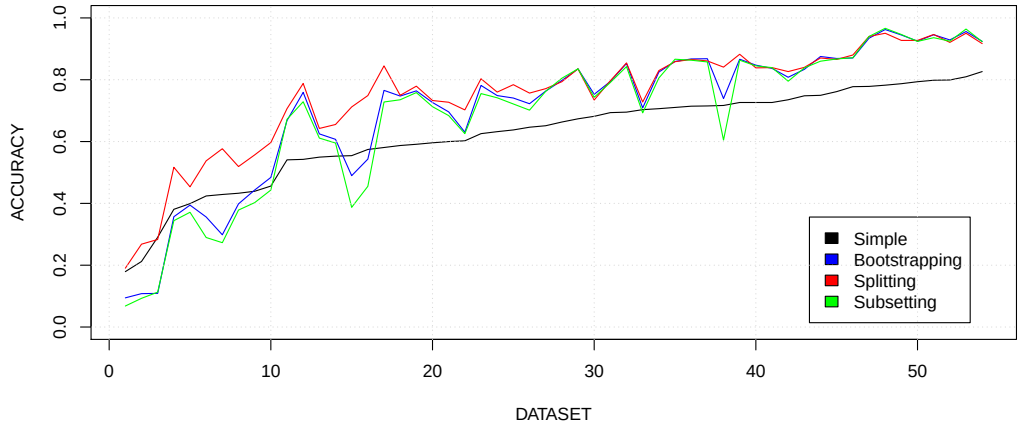

Figure 3.3.: Accuracy summary per scheme

The first observation that can be made in this analysis is that the three resampling accuracies are notably distinct from the simple method. In order to test whether this distinction is significant, a repeated measures ANOVA was performed and p-values of 0.0127 (splitting), 0.03755 (bootstrapping) and 0.09722 (subsetting) were obtained. This, for $\alpha = 0.05$, indeed suggests a significant advantage of two resampling methods over the simple validation approach, the reference category in the ANOVA (Table 3.12). Secondly, one can clearly observe that the three resampling methods show a relatively similar pattern with regard to accuracy. It is only in the lower accuracy range (the lower fifth of the data sets), where the resampling schemes drift apart and the splitting method seems to have a notable advantage. It is furthermore the only scheme that at virtually all times performs better than the simple approach and in almost all cases better than the other resampling schemes. We therefore used another ANOVA to test for possi-

ble significant differences between the three resampling categories. This time the splitting method is used as reference category, and with p-values of 0.136 and 0.273 for subsetting and bootstrapping respectively, the null hypothesis of no significant difference when it comes to the resampling method as such is retained. However, the splitting scheme appears to have a slight, yet non-significant edge over the other strategies.

We continue to deepen the analysis by focusing on each scheme individually. Therefore we first take the simple scheme from Figure 3.3 and do not only look at the average accuracy across the data sets, but on the performance of the individual indices. This is shown in Figure 3.4. Below the graph, a levelplot indicates the complexity of the respective dataset; red for *difficult* (few observations, many variables, cluster overlap, higher number of clusters), yellow for *mid-range*, blue for *easy* (more observations, less variables, good separation, few clusters). The first aspect to note is that several indices do not seem to perform well at all for the benchmark data. Therefore the worst performing indices are shown in lighter colors for better readability. The better performing indices obviously show a trend of increasing accuracy from the more complex to the simpler data sets. However, the spread even between the top indices is considerable - about 50 percentage points, as for example in the mid-range data sets, accuracy values range between about 0.5 and 1.

In contrast, Figures 3.5, 3.6 and 3.7 show the same individual index accuracies for the resampling schemes. Two striking differences can be observed: for once, the overall accuracy rate is notably higher (as was already concluded from Figure 3.3). This again is particularly true for the top performing indices. Those that failed completely are as before depicted in lighter colors. Secondly, the top performing index accuracies spread across a much narrower range of about 20 percentage points especially for the mid to easy data sets.

The level plot below each graph indicates which factor impacts accuracy to which degree. The most apparent observation is here that *separation* values for close and overlapping clusters are almost exclusively found in the lower half of the accuracy range. To formally determine the significance of factor influence we test again by means of an ANOVA. The obtained p-values are highly significant for factors *separation*, *clusters* and *observations*. The mid to hard difficulty levels of these factors are overwhelmingly found in the lower half of the accuracy range, while for factor *variables*, this is not true to the same extent - the difficulty levels of this factor are spread much more evenly across the accuracy range and hence the non-significant test result. A summary of the p-values for all factors, among with their model estimates and confidence intervals per scheme is given in Tables 3.7, 3.8, 3.9, 3.10. The reference category is always the 'easiest' factor - 4 clusters, many observations (60 times the number of variables), separation value 0.1 and 10

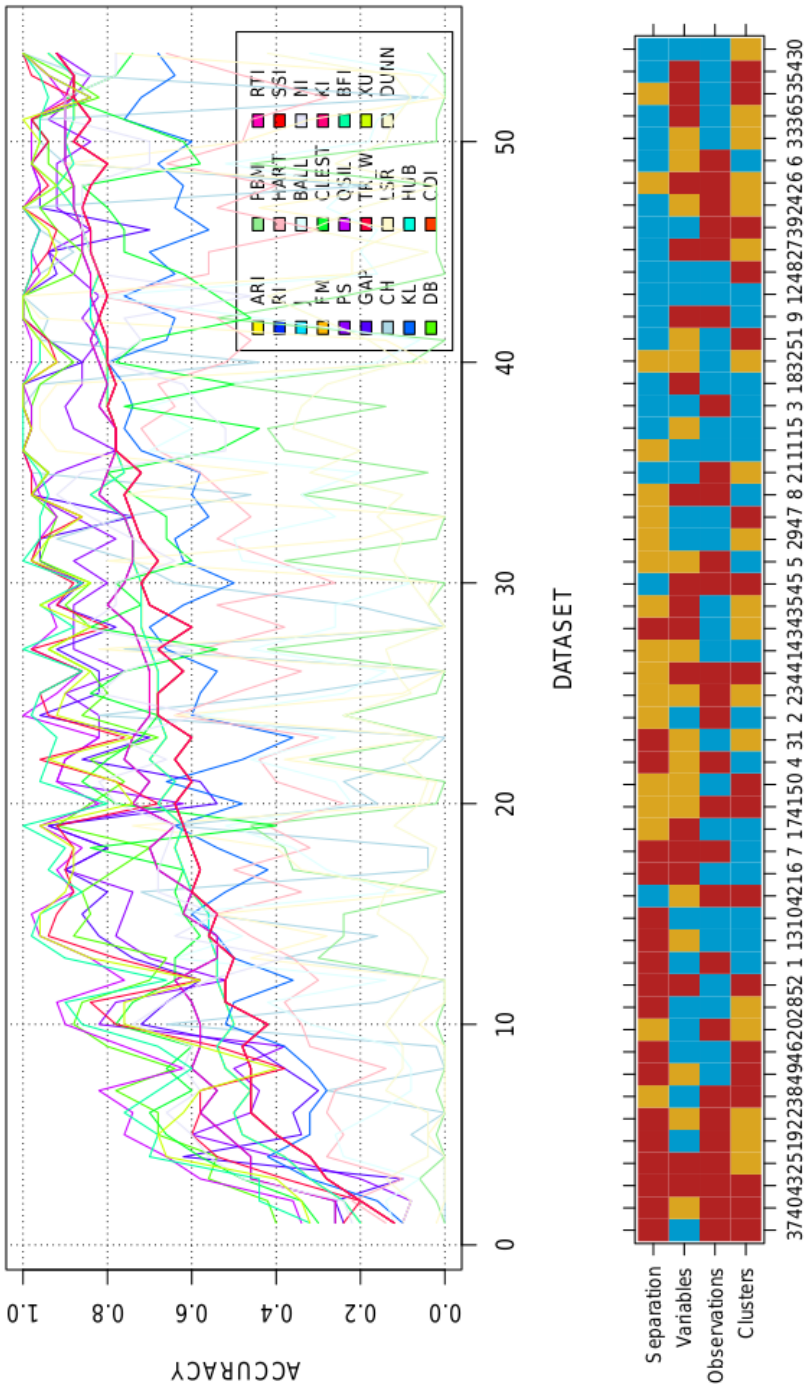

Figure 3.4.: Accuracy plots for simple

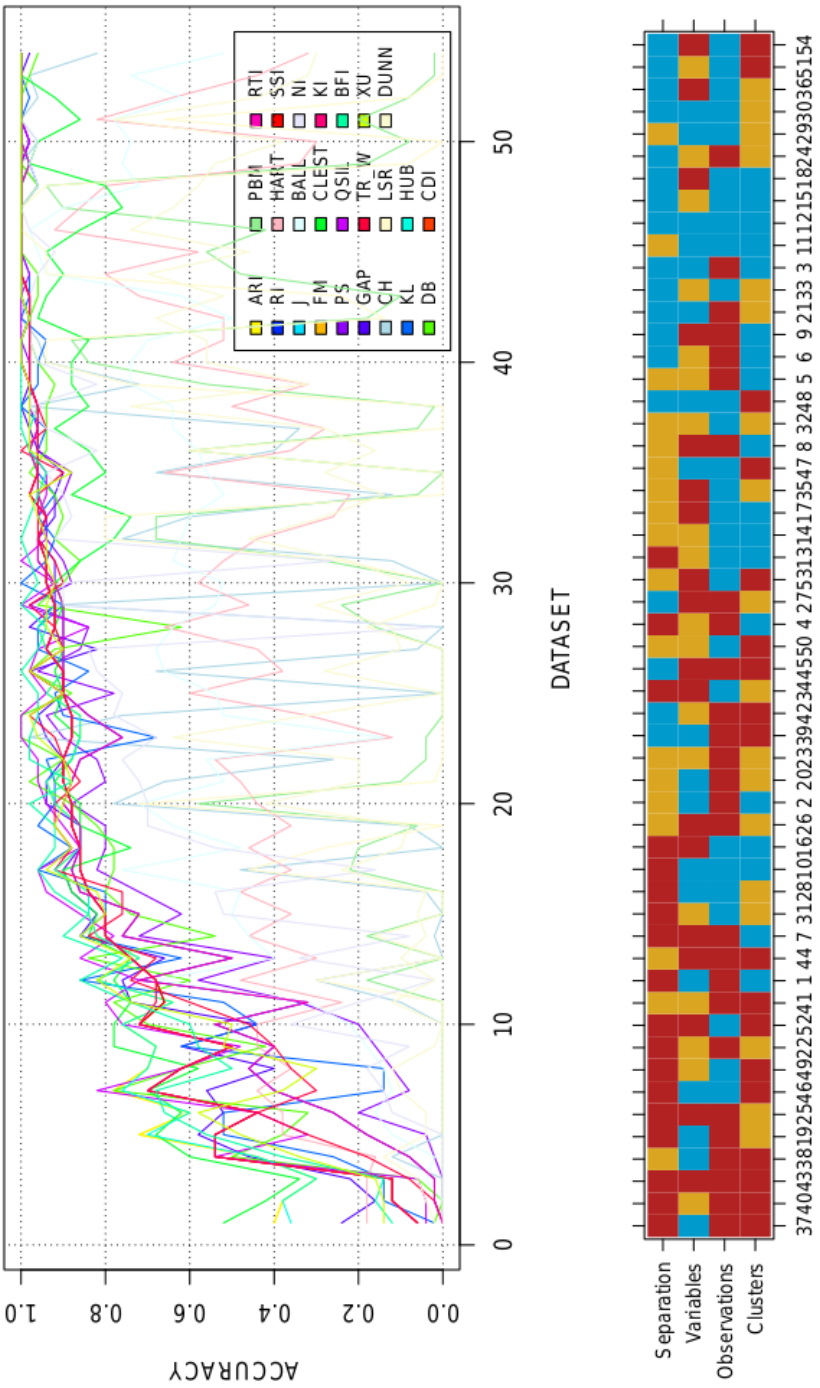

Figure 3.5.: Accuracy plot for bootstrapping

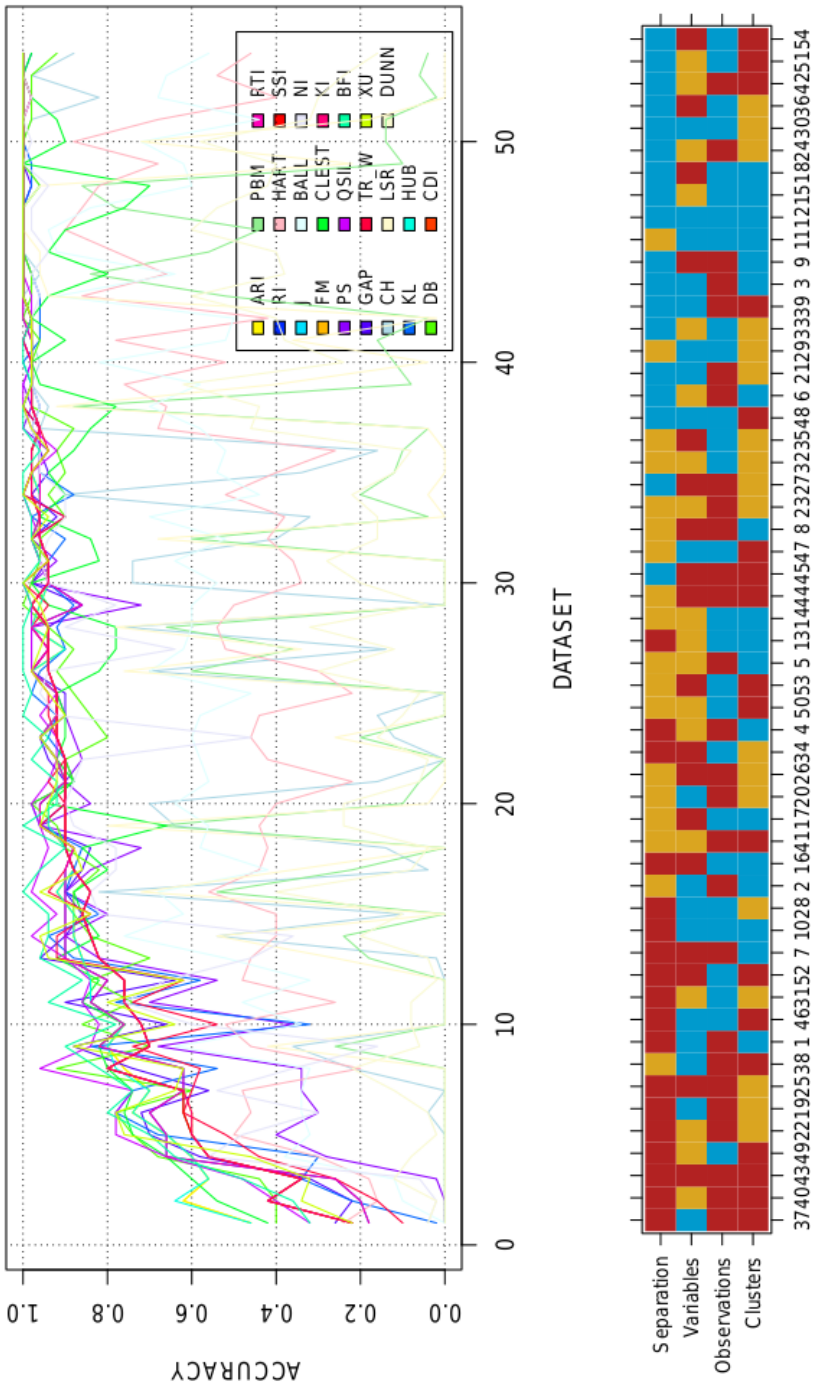

Figure 3.6.: Accuracy plot for splitting

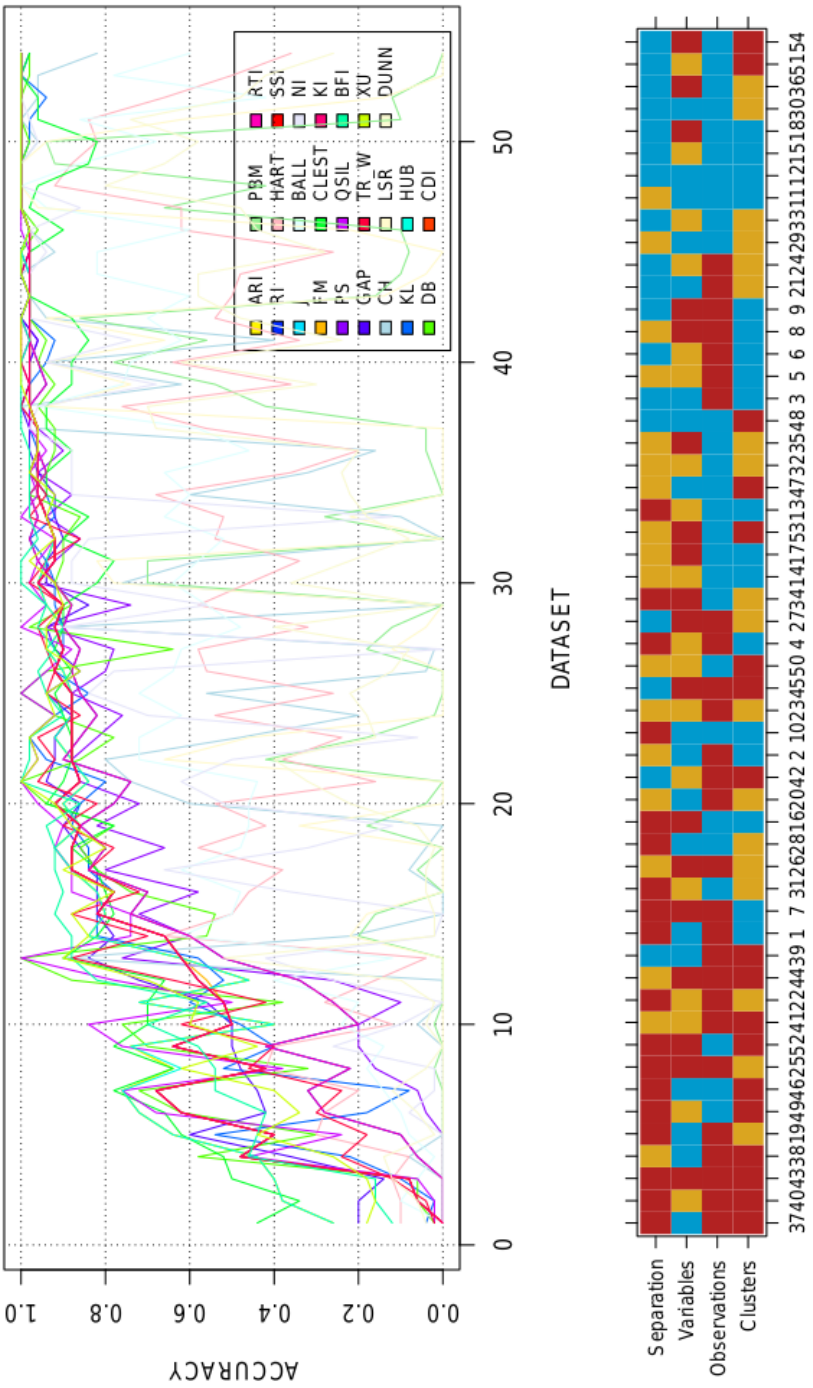

Figure 3.7.: Accuracy plot for subsetting

variables).

Bootstrapping				
	p-value	Estimate	2.5 %	97.5%
Clusters_6	0.00086	-0.11289	-0.17663	-0.04914
Clusters_8	3.89731E-11	-0.27240	-0.33614	-0.20866
Observations_30x	1.26438E-06	-0.14406	-0.19611	-0.09201
Variables_16	0.83746	0.00653	-0.05721	0.07028
Variables_22	0.91773	-0.00329	-0.06703	0.06046
Separation_0	0.00021	-0.12742	-0.19117	-0.06368
Separation_-0.1	1.28333E-15	-0.37627	-0.44001	-0.31252

Table 3.7.: p-values for the bootstrapping scheme with model coefficients and confidence intervals

Splitting				
	p-value	Estimate	2.5 %	97.5%
Clusters_6	0.00149	-0.08640	-0.13788	-0.03492
Clusters_8	1.27165E-09	-0.19364	-0.24512	-0.14217
Observations_30x	0.00042	-0.07938	-0.12141	-0.03735
Variables_16	0.91191	-0.00284	-0.05432	0.04863
Variables_22	0.73232	-0.00880	-0.06028	0.04268
Separation_0	0.00066	-0.09333	-0.14481	-0.04186
Separation_-0.1	1.02811E-15	-0.30578	-0.35725	-0.25430

Table 3.8.: p-values for the splitting scheme with model coefficients and confidence intervals

We now extend the discussion beyond a mere comparison of resampling vs. non-resampling based validation and the factors that could affect index accuracy. The next step is to analyze the individual indices themselves with regard to their general approach - external vs. internal validation. Furthermore, we draw a link to the respective resampling schemes and the performance in relation to the index used. Table 3.11 shows the average index accuracy over all data sets for each scheme. The top performing external indices are highlighted in yellow, while the top performing internal indices are highlighted in green. It is again noteworthy that the highest accuracies are found in the splitting scheme, which is consistent with the results discussed above. Moreover, all external indices achieve their best result in the splitting scheme and all of them show very high accuracy rates. Only index NI is slightly below 0.8 with a value of 0.78, but all other indices have values above 0.8, ARI and HUB even 0.9. This strongly suggests that external indices

Subsetting				
	p-value	Estimate	2.5 %	97.5%
Clusters_6	0.00042	-0.12116	-0.18532	-0.05699
Clusters_8	3.01720E-12	-0.29893	-0.36310	-0.23477
Observations_30x	1.05620E-07	-0.16376	-0.21615	-0.11137
Variables_16	0.74571	0.01040	-0.05377	0.07457
Variables_22	0.89301	0.00431	-0.05986	0.06848
Separation_0	0.00012	-0.13347	-0.19763	-0.06930
Separation_-0.1	9.56361E-16	-0.38196	-0.44612	-0.31779

Table 3.9.: p-values for the subsetting scheme with model coefficients and confidence intervals

Simple				
	p-value	Estimate	2.5 %	97.5%
Clusters_6	0.00108	-0.07262	-0.11452	-0.03072
Clusters_8	5.76925E-11	-0.17662	-0.21852	-0.13472
Observations_30x	0.00011	-0.07176	-0.10598	-0.03755
Variables_16	0.88855	-0.00293	-0.04484	0.03897
Variables_22	0.59461	0.01116	-0.03075	0.05306
Separation_0	4.65925E-05	-0.09360	-0.13550	-0.05170
Separation_-0.1	3.92373E-17	-0.27271	-0.31461	-0.23081

Table 3.10.: p-values for the simple scheme with model coefficients and confidence intervals

benefit uniformly from a resampling based strategy, especially splitting, because accuracy rates in the simple scheme are much less uniform and considerably lower. Therefore these observation suggests that external indices clearly benefit from a resampling based approach. In terms of recommendations, ARI and HUB are, as already mentioned, the two top performing indices. However, the rest does not lag behind dramatically except NI and PS, which performed slightly worse with around 0.8 accuracy. Yet still, based on these simulations, a resampling based external validation approach should yield good results without depending too much on a particular index.

As for internal indices, the matter is quite different. Several indices did not perform well at all and are apparently not appropriate for the benchmarking data at hand, for example DUNN, PBM and LSR produce average accuracy rates below approximately 0.3. This makes selecting an index difficult, as of course for real world data there is not such an easy means of checking accuracy. Yet still, some indices performed exceptionally well, and can be regarded as recommendations for internal validation strategies: QSIL and BFI produced the best results closely followed by TR_W , XU and DB. Furthermore, there is almost no difference between simple and resampling based validation for these indices. While when using the splitting scheme, QSIL and BFI produced a correct cluster estimate > 0.9, the only very slight difference to the simple slot is almost negligible, particularly given the considerable savings in computational effort. The same is essentially true for TR_W, XU, and DB. Although GAP increases its accuracy slightly from 0.78 to 0.87, the only index to strongly increase its performance when used in a resampling based validation is KL, as it jumps from 0.56 to 0.85. Still, on the whole we conclude that internal indices generally benefit little from a resampling based approach, especially if one takes the huge computational cost of some of the indices (particularly QSIL and GAP) into account. It is also important that all comparisons so far have been between the simple scheme and the best performing resampling scheme, splitting. Subsetting and bootstrapping yield quite similar results, generally a few percentage points lower in accuracy than the splitting scheme. However, several internal indices perform even slightly worse in the bootstrapping and subsetting scheme than in the simple approach. This underscores our observation that given a well selected index, the simple scheme appears to be sufficient for an internal validation strategy.

The usefulness of resampling for external vs. the underwhelming effect on internal indices can be visually demonstrated. We return to Figure 3.3, which shows the mean accuracies for each data set and scheme. If we now use for the same visualization only either external or internal indices, we obtain Figure 3.8. The gap between the resampling schemes and the simple method widens on the external-only graph, whereas on the internal only graph the gap is much narrower (even

more so if we were to use only the top performing indices). Furthermore, once we conduct another run of an ANOVA comparing the schemes, this time with the selected subset of the two index groups the result is quite different to the full index ensemble result at the beginning of this section. For the internal-only ANOVA, only splitting comes close to statistical significance (0.0546), bootstrapping and subsetting show much higher p-values (0.2843 and 0.4007). In contrast, for the external-only ANOVA, all resampling methods show significant improvements over the simple method (splitting 1.35e-0.5, bootstrapping 0.00279, subsetting 0.01634). The p-values for the full ensemble of indices and for internal and external-only ANOVAs are listed for comparison in Tables 3.12, 3.13 and 3.14. It thus appears that the significant advantage of the resampling methods, in particular the splitting method, largely stems from the improved performance of external validation indices. Internal measures contribute rather little to the statistical significance obtained at the beginning.

3.6. Summary

Finally, in order to sum up the analysis, Table 3.15 shows at a glance a quick summary of the benchmarking results. On the whole the study shows that a combination of external and resampling based validation seems to promise consistently good results. Should the computational power not be available, internal validation is preferable, yet the selection of the index should be done with care. The other two cases, external/non-resampling and internal/resampling are less optimal, as for the former accuracy rates are lower and not as consistent as with resampling, for the latter the resampling approach would not be an efficient use of computational power that does not generally result in greater accuracy.

3.7. Conclusion

On the whole, this study has found that a there is a certain merit to using a resampling based approach when validating a clustering model. This is particularly the case when external criteria are used and the necessary computational power is available, as this seems to produce reliable and accurate results. While using resampling based validation can therefore help to boost model selection accuracy in comparison to simple validation, the differences between the resampling methods are not as clear-cut. The splitting scheme achieved slightly higher accuracy rates than bootstrapping and subsetting, but not in a significant manner.

However, it is also important to stress that this benchmarking study certainly cannot cover the entire range of clustering problems and thus the conclusions that

	Bootstrapping	Splitting	Subsetting	Simple
ARI	0.86	0.90	0.85	0.68
RI	0.78	0.88	0.74	0.71
J	0.83	0.87	0.81	0.65
FM	0.83	0.87	0.81	0.65
PS	0.73	0.81	0.71	0.76
GAP	0.84	0.87	0.82	0.78
CH	0.47	0.49	0.46	0.50
KL	0.81	0.85	0.79	0.56
DB	0.81	0.87	0.81	0.84
PBM	0.20	0.21	0.19	0.13
HART	0.45	0.49	0.44	0.44
BALL	0.53	0.55	0.53	0.30
CLEST	0.82	0.85	0.83	0.67
QSIL	0.85	0.91	0.85	0.89
TR_W	0.80	0.86	0.80	0.84
LSR	0.24	0.27	0.24	0.31
HUB	0.86	0.90	0.85	0.68
CDI	0.83	0.87	0.81	0.65
RTI	0.78	0.88	0.74	0.71
SSI	0.83	0.87	0.81	0.65
NI	0.63	0.78	0.56	0.68
KI	0.83	0.87	0.81	0.65
BFI	0.86	0.92	0.85	0.88
XU	0.83	0.88	0.83	0.84
DUNN	0.28	0.22	0.27	0.14

Table 3.11.: Average accuracies over all data sets

All indices				
	p-value	Estimate	2.5%	97.5%
Splitting	0.00127	0.12546	0.04975	0.20118
Bootstrapping	0.03755	0.08038	0.00467	0.15609
Subsetting	0.09722	0.06398	-0.01172	0.13969

Table 3.12.: ANOVA summary for all indices (reference category: simple method)

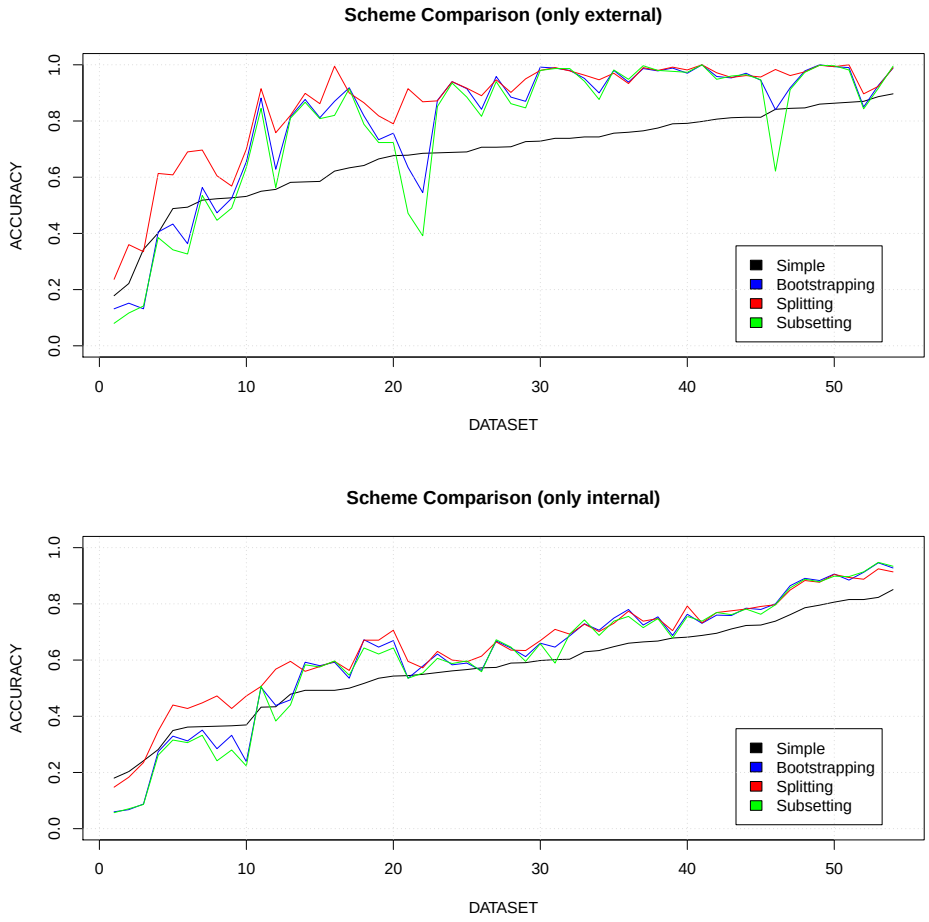

Figure 3.8.: Accuracy summary per scheme internal and external indices separately)

Only internal indices				
	p-value	Estimate	2.5%	97.5%
Splitting	0.0546	0.07447	-0.00148	0.15043
Bootstrapping	0.2843	0.04137	-0.03459	0.11732
Subsetting	0.4007	0.03245	-0.04350	0.10840

Table 3.13.: ANOVA summary for internal indices (reference category: simple method)

Only external indices				
	p-value	Estimate	2.5%	97.5%
Splitting	1.35e-05	0.18071	0.10078	0.26063
Bootstrapping	0.00279	0.12265	0.04272	0.20258
Subsetting	0.01634	0.09815	0.01822	0.17807

Table 3.14.: ANOVA summary for external indices (reference category: simple method)

	External validation (Top: ARI, FM, HUB, J)	**Internal validation** (Top: QSIL, TR_W, BFI, XU, GAP)
Resampling based (recommended: splitting)	very reliable strategy, mostly index-independent	little to no gain in accuracy does not justify the computational effort
Non-resampling based	accuracy significantly lower, internal validation preferable	performance strongly dependent on index, great variability in accuracy

Table 3.15.: Analysis summary

have been drawn are of course limited to the scope of the benchmarking data.

The conclusions from this study shall also serve as a basis for an evaluation on real world benchmarking data to further deepen the understanding of the stability notion and its implications for practical model selection in clustering.

4. Elements of Benchmarking in Cluster Analysis

The previous chapters discussed model validation issues in clustering and a concrete implementation of a benchmarking problem. In the course of conducting the study in chapter 3, a number of aspects were identified that caused difficulties in designing the study. Therefore, this chapter aims to discuss benchmarking in clustering from a general perspective, ranging from current issues to proposed solutions and a prototypical implementation in R.

4.1. Background and Related Work

Clustering is a complex data analytic process that requires choices with every step as discussed in section 1.4 (pre-processing of data, the analysis itself and the validation of the output). Particularly, not only is a wide selection of options available for these steps, there are also constantly new methods introduced and developed. Therefore, comparative evaluation, i.e. benchmarking, is critically important to not only the developer of a new method (as a means of justification) but also the researcher who seeks to make a decision about the best method for a given problem [185]. Benchmarking studies are thus carried out to measure performance of different methods by using reference data sets and performance criteria for evaluation purposes. This can be done either, as mentioned above, by a developer to demonstrate the merit of a new method, or by an independent investigator without a vested interest in one of the methods under scrutiny [191]. Weber et al. [191] mention that a number of reviews have investigated this matter already from various angles, such as benchmarking design in general [8, 26, 30, 123, 137, 142, 208], design practices in existing benchmarking studies [123], neutrality vs. bias in benchmarking [27], real-world data design principles [28, 29], simulation study design [129], incorporation of meta-analysis techniques [66, 79, 80, 92], organization and role of community challenges [32, 75] and benchmarking design for specific types of methods [7, 185]. A more general discussion of benchmarking as a form of meta-research is found in Ioannidis [95].

It is thus evident that a wide range of considerations are necessary in order to organize a transparent, reproducible and neutral comparison study. For example,

a potential bias in benchmarking studies was found by Boulesteix et al. [27], in which a range of studies were investigated where either a new method was introduced and compared to existing solutions, or a 'pure' comparison of methods was done. The survey makes three main observations: firstly, benchmarking studies where a new method is presented quite often present the new method as the best solution, where pure comparison studies not always identify a winner. Secondly, computational journals tend to publish less pure benchmarking studies than those comparing a new method to existing ones. Finally, papers and studies presenting new methods are never negative with respect to the new method. This suggests that if a benchmarking study is conducted with a vested interest in one of the methods under investigation, a potential bias might be the result. This naturally is not desired in benchmarking, therefore Weber et al. [191] suggest the following principles:

1. Define purpose and scope of the benchmarking
2. Include all relevant methods
3. Select (or design) representative data sets
4. Choose appropriate parameter values and software versions
5. Evaluate methods according to key quantitative performance metrics
6. Evaluate secondary measures including computational requirements, user-friendliness, installation procedures, and documentation quality
7. Interpret results and provide recommendations from both user and method developer perspectives
8. Publish results in an accessible format
9. Design the benchmark to enable future extensions
10. Follow reproducible research best practices, by making code and data publicly available

With regard to (1), the purpose and scope of the benchmarking study should be clearly defined at the beginning, as this will affect its design and implementation [191]. In principle, Weber et al. [191] identify three main purposes of a benchmarking study: demonstrating the merit of a new method (e.g. [114, 117, 138, 190, 209]), a neutral comparison of methods (e.g. [13, 54, 63, 73, 100, 105, 148, 156, 157, 170, 171, 189]) and studies in the form of community challenges, such as the DREAM [43, 68, 90, 108, 192], FlowCAP [3, 4], CASP [130, 131], CAMI [164], Assemblation

[34, 64], MAQC/SEQC [165, 166, 176] and GA4GH consortia [107]. The examples and consortia mentioned are from the field of molecular biology as mentioned in Weber et al. [191]. In determining the scope of a benchmarking study, the question of available resources is certainly a notable trade-off. If chosen too broad, the scope may exceed the available resources, if chosen too narrow the study might not be representative and produce misleading results [191]. In order to strike a balance, the challenge for neutral studies is to select methods where investigating researchers are as equally familiar with as possible, and in the case of presenting the merits of a new method, the developer should be careful to select state-of-the-art methods and not disadvantage competing methods intentionally (e.g. by tuning parameters or data sets) [185, 191]. This aspect is closely related to point (2), where Van Mechelen et al. [185] note that a suitable choice of methods means that within the scope of the study, there should be a broad choice of methods that includes strong competitors [28].

Another crucial aspect is the choice of (real-world) or design of (artificial) data sets (3), as it allows the methods under investigation to be evaluated under a range of different conditions and settings [191]. In the case of real world data, the choice should be representative of possible applications in practice [191] i.e. the scope of generalization for the data sets should be well defined [185]. An attempt to provide a set of well documented data sets is for example the benchmarking data set repository of the IFCS task force on benchmarking [71, 185]. Using simulated data provides the advantage that the exact structure of the data set is known and therefore measurement of method performance is more straightforward. However, the design of artificial data itself is less so, particularly the degree of complexity, as the data scenario should neither be too simplistic or too complex [191]. If a range of methods either fail or achieve perfect performance on a number of data sets, evaluating actual performance becomes difficult. Therefore, if a set of different parameters for artifical data is used, a full factorial design should be employed to enable a thorough analysis on several levels of data complexity [185].

As already mentioned above, parameter settings for methods can also impact the objectiveness of a benchmarking study. In point (4), Weber et al. [191] note that 'appropriate parameter values' should be chosen. This essentially means that methods should not be disadvantaged in comparison to others when for the former default values are used and for the latter extensive parameter tuning is done. Weber et al. [191] suggest that in its simplest form all parameters for all methods could be left at their default values, as for example in Saelens et al. [157], Couronné et al. [44] and Schneider et al. [160].

The choice of performance evaluation measures, mentioned in point (5) is a pivotal aspect of a study design. Of particular importance is the decision on the nature of the criterion, as different criteria may imply different clustering solutions

[88, 185] which in turn could hinder a proper direct comparison of methods [191]. Moreover, subjectivity with regard to the choice of evaluation measures may lead to results that do not reflect performance in real-life applications and/or give an over-optimistic impression of the method under investigation [191]. These considerations need to be taken into account when deciding upon a range of evaluation metrics, particularly because the computational effort also is a factor; some metrics in the benchmarking study in chapter 3 such as the Gap statistic or the CLEST method demand a significant amount of computational power. Thus depending on the methods and data, a suitable selection has to be made. For clustering, a range of such measures is listed in sections 2.2.1.1 and 2.2.2.1, depending on whether external or internal measures are used.

Weber et al. [191] also note in point (6) that not only pure performance evaluation in terms of accuracy of clustering solutions is important, but also secondary performance aspects, such as algorithm scalability, runtime, computational requirements, code quality, user friendliness and documentation. Weber et al. [191] further note that in Weber and Robinson [189] runtime of different methods varied greatly on the same data sets, which along with other computational requirements may be a prohibitive demand for some users in practice.

Van Mechelen et al. [185] claim that for interpreting and summarizing study results in point (7), unconditional statements should be used with some caution, particularly if they are based on some form of averaging as distortions of a mean value can happen if some methods perform significantly worse than others. Furthermore, Weber et al. [191] note that not only methods might not be directly comparable, different users might be interested in different performance aspects of the methods under investigation, therefore the target audience and their requirements should be kept in mind when giving recommendations, interpretations and guidelines. Importantly, it needs to be emphasized that every study is of course somewhat limited by the scope of the methods and data sets, which should be made clear to the reader [185].

After study completion, the question arises in which form the results and resources pertaining to the study can be made available (8) and whether possible future extensions (9) are possible. A much more accessible means of not only transporting, but also visualizing results and exploring them in an interactive manner is for example by means of a web app using the shiny package [39] (examples include Saelens et al. [157], Soneson and Robinson [171]), but a drawback of this of course is that these resources need to be built and maintained, which cannot always be guaranteed in the long run [191]. With regard to future extensions, Weber et al. [191] note that a as new methods emerge, benchmark studies can quickly become outdated. Therefore, if code and data is available in some form or fashion, comparisons against newer methods, other data set scenarios or parameter settings

are considerably easier. This could be combined with an interactive web app as mentioned above (e.g. in Barton [12]), however, a practical implementation which achieves that may require a substantial additional effort from study authors [191].

Finally, scientific findings and particularly benchmarking results should adhere to principles of reproducible research (10), which is a concern in numerous study areas [94, 191]. Donoho [58] notes that putting emphasis on reproducibility has a couple of advantages: apart from changing oneself's work habits, it facilitates efficient teamwork. Also, by the fact that other researchers can access code and data more easily, the impact of the original work is increased through more frequent citations and thus a greater impact on the scientific community. This practice should also be actively encouraged by journals [91], as a mere encouragement to do so is usually not sufficient [31, 191]. Such an approach is facilitated by software infrastructure such as reproducible workflow platforms, for example Galaxy [2, 191] or KNIME [17]. With regard to analyses done with R, Weber et al. [191] suggest making parameter values such as software version, random number seeds and the operating system available, but also using packages that are specifically dedicated towards managing benchmarking workflows and data sets. These include for example R package `SummarizedBenchmark` [104, 150], a package that offers classes and methods for managing benchmarking experiments and that originates from comparison studies in computational biology. Other examples include package `DataPackagerR` [69] for data management, Dynamic Statistical Comparisons [38], a tool to easily generate extendable benchmarks (written in Python, but implementing additional tools in R) and R package `workflowr` [23] for analysis organization, collaboration and result sharing, which works with `knitr` [200] and `rmarkdown` [6] and version control software (git) to create a website containing time-stamped, versioned and documented results.

The guidelines discussed in this section should provide an overview of steps to produce a transparent, reproducible and neutral benchmarking study. In the next section, a more detailed look is taken at how to implement particularly transparency and reproducibility considerations in R code when conducting a study specifically in the context of clustering.

4.2. Grammar as a Data Structure

When attempting to formulate a structural framework for executing benchmarking in R, the term *grammar* lends itself for this purpose. The Cambridge Dictionary defines it as follows: "(the study or use of) the rules about how words change their form and combine with other words to make sentences" [53]. In other words, this definition can be rephrased and applied to data analysis as "(the study or use of) the rules about how data and methods can be combined to form a data

analytic process". Such a process could be a clustering model or a benchmarking study. Therefore, even though the term itself is taken from the study of language, its abstract definition as a set of rules that govern the formation of a process can also be applied to data analysis. This is not without precedence; Wilkinson [199] applied the notion to the field of graphics, and implementations are found in package `ggplot2` [194, 196] and `ggbio` [206]. Moreover, Wickham et al. [197] have also applied the notion to data manipulation in package `dplyr`.

For example, in package `dplyr` the data set (which functions as the noun) is modified by a number of functions such as `mutate()`, `filter()`, `summarise()`, `select()` and `arrange()` (which function as verbs). An implementation such as this makes the code considerably more readable, by narrowing the gap between syntax and semantics. A similar approach shall thus be taken toward the issue of implementing benchmarking problems in clustering. As already mentioned, existing packages such as Kimes and Reyes [104], Finak et al. [69] or Carbonetto et al. [38] already offer some functionality with regard to this matter, however, the following prototypical implementation of a benchmarking grammar shall not only address the issue of data structure definitions, but also emphasize easy readability and user friendly setup of benchmarking.

4.3. Building Blocks of a Benchmarking Grammar

In order to translate the aforementioned definition of language grammar to its corresponding structure in the context of clustering, the main components (the equivalents to nouns in language) need to be identified. As mentioned in section 4.1, a benchmarking study consists of three main components that in this case lend themselves as grammatical elements:

- **methods**: methods or algorithms that are defined by the scope and purpose of the benchmarking study
- **data sets**: artificial and real-world data
- **criteria**: performance evaluation of investigated methods

A data-grammatical structure is then needed to arrange these fundamental building blocks into a unified structure that shall be defined as a *benchmarking object*, basically encapsulating all information that is needed in one object. Subsequently, the benchmarking object is processed by a benchmarking function, here defined as `runBenchmark()` that uses the information to compute the cluster models and the evaluation criteria. An illustration of this concept is given in Figure 4.1. The missing link between these components is some form of concatenation (in language

grammar terms - a conjunction) that takes on the task of connecting methods, data sets and criteria in a way that it can be efficiently processed by the benchmarking function, in Figure 4.1 shown as %+%. A suggestion of how to achieve this is demonstrated in the following section by means of a prototypical implementation of the concept in Figure 4.1.

4.4. Prototypical Implementation in R

We start by defining the benchmarking object and its components (R code of this section can be found in appendix A). The implementation follows an object-oriented approach using the S4 class framework in R. S4 classes have a strict formal definition (contrary to the S3 framework which is used throughout base R) and lends itself more readily for OOP [195].

Therefore, as shown in the code below, a class *benchmarkObject* is defined. Furthermore, S4 objects require so called *slots* that represent the state of the object. As shown below, the object consists of six such slots, where `methods`, `data` and `criteria` are obligatory as input slots. If external criteria are used for model validation, the true group labels are also required. Thus an additional fourth slot `trueLabels` can be used to store that information. If only internal criteria are used, this slot can be left empty. Finally, `models` and `validation` are slots where the results of the calculation done by method `runBenchmark()` are stored. Thereby, after the benchmark has been executed, all information is available clearly structured and encapsulated in one object.

Figure 4.1 shows the input and output slots of the Benchmarking object. The symbol is used to access such a slot in actual R code - `foo@data` would therefore show the data sets stored in the benchmark object `foo`.

```
setClass("benchmarkObject",
    representation(
        methods = "list",
        data = "list",
        criteria = "list",
        trueLabels = "list",
        models = "list",
        validation = "list"))
```

The question now arises how to efficiently assign and store the items in the respective slots. Particularly methods and their parameters are a notable issue. As mentioned above, when applying sets of parameters, a full factorial setup combined with the data sets is preferable. This is where the concept of a conjunction

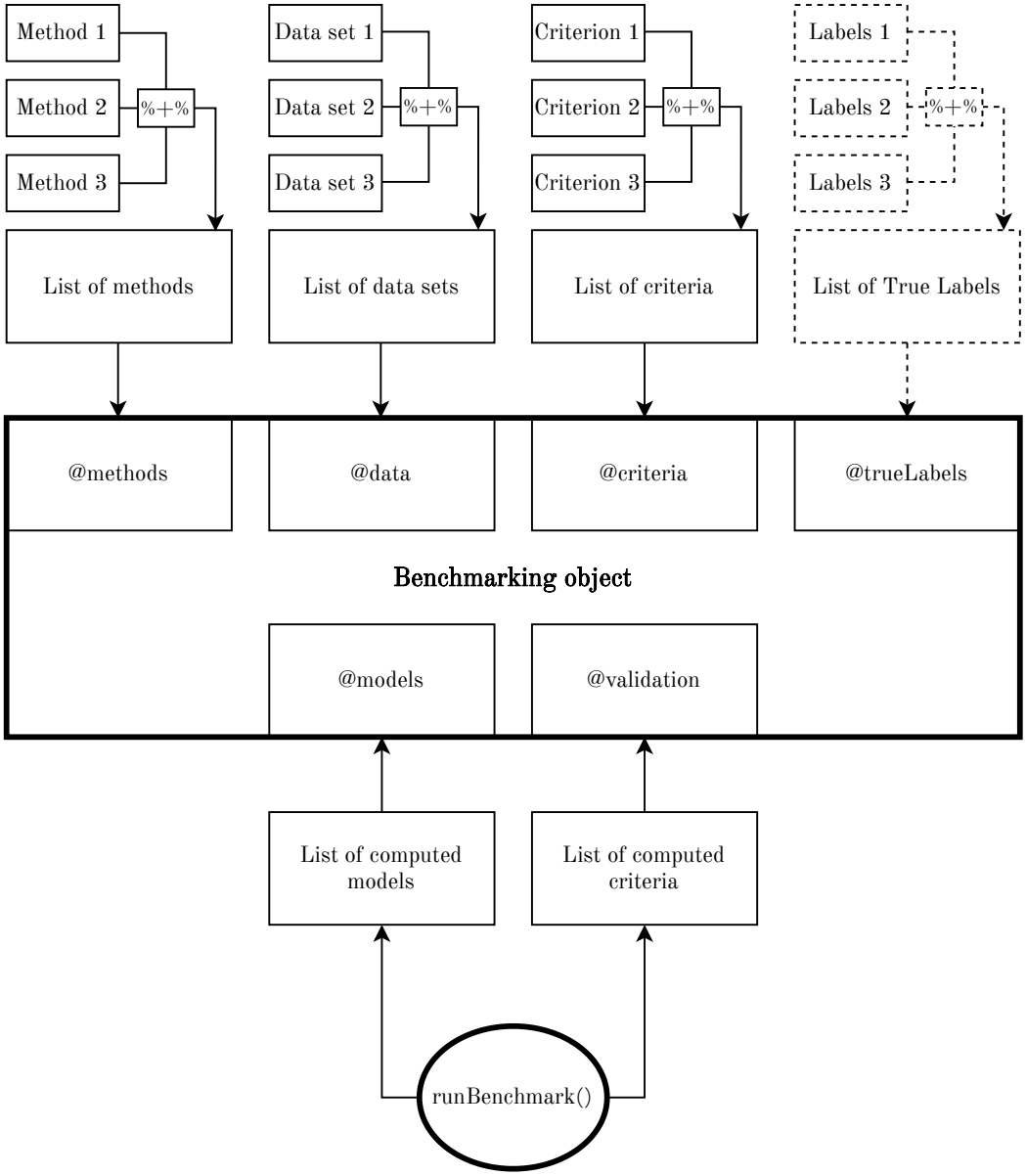

Figure 4.1.: Components of a benchmarking grammar

operator comes in. This is not new to R code; the pipe operator %>% of package `magrittr` [9] is similarly used to link function executions and thus reduces the need for complicated nested function calls. However, in this case, not sequences of functions (i.e. verbs) are concatenated, objects such as data set or function calls are to be combined in lists. It is therefore suggested to do this by means of a second operator, %+% (based on %>% of package `magrittr` [9]). In its simplest form, it plainly concatenates objects into a list, such as data sets. In the following code, the `iris` data [70] and the artificial data set from Figure 1.9 are combined:

```
data_1 <- iris[,-5]

dim1 <- mvrnorm(150, c(5,10), matrix(c(2,1.2,2,1.2),2,2))
dim2 <- mvrnorm(150, c(10,10),matrix(c(2,-1.2,2,1.2),2,2))
dim3 <- mvrnorm(150, c(15,10),matrix(c(2,1.2,2,1.2),2,2))
data_2 <- rbind(dim1, dim2, dim3) %>% as.data.frame

data_list <- data_1 %+% data_2

> str(data_list, vec.len = 1)
List of 2
 $ data_1:'data.frame': 150 obs. of  4 variables:
  ..$ Sepal.Length: num [1:150] 5.1 4.9 ...
  ..$ Sepal.Width : num [1:150] 3.5 3 ...
  ..$ Petal.Length: num [1:150] 1.4 1.4 ...
  ..$ Petal.Width : num [1:150] 0.2 0.2 ...
 $ data_2:'data.frame': 450 obs. of  2 variables:
  ..$ V1: num [1:450] 5.28 ...
  ..$ V2: num [1:450] 11.8 ...
```

The more challenging problem arises when the methods under investigation should be added to the slot `methods`. An option could be to just store the parameters required by the methods in list form. A very simple example could be:

```
method_list <- kmeans %+% pam

> str(method_list)
List of 2
 $ kmeans:Dotted pair list of 6
  ..$ x        : symbol
  ..$ centers  : symbol
  ..$ iter.max : int 10
  ..$ nstart   : int 1
  ..$ algorithm: language c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen")
  ..$ trace    : logi FALSE
 $ pam   :Dotted pair list of 12
  ..$ x           : symbol
  ..$ k           : symbol
  ..$ diss        : language inherits(x, "dist")
  ..$ metric      : language c("euclidean", "manhattan")
  ..$ medoids     : NULL
  ..$ stand       : logi FALSE
  ..$ cluster.only: logi FALSE
  ..$ do.swap     : logi TRUE
  ..$ keep.diss   : language !diss && !cluster.only && n < 100
  ..$ keep.data   : language !diss && !cluster.only
```

```
  ..$ pamonce     : logi FALSE
  ..$ trace.lev   : num 0
```

The question however remains how to pass parameters and at the same time efficiently realize (if so desired) a full factorial design. Therefore, the operator is also able to handle parameter assignments:

```
method_list <- kmeans(x=data_1, centers=3) %+% pam(x=data_1, k=3)

> str(method_list, vec.len = 1)
List of 2
 $ :List of 2
  ..$ method: chr "kmeans"
  ..$ params:List of 1
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ centers: num 3
 $ :List of 2
  ..$ method: chr "pam"
  ..$ params:List of 1
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ k: num 3
```

By this implementation, the methods are organized in lists with the parameters passed in the function call. It should be noted that the parameters do not necessarily have to have the same name, a correct list is created nonetheless. Finally, it would be desirable to specify not only a simple function call, but specify a range of data sets or a range of clusters centers or a range of other parameters that should be subject to a full factorial design. This is also possible in this case. In the next example, two data sets are created and passed as arguments to the parameter `x`, which normally would only accept a single data set as argument:

```
data_1 <- iris[,-5]

dim1 <- mvrnorm(150, c(5,10), matrix(c(2,1.2,2,1.2),2,2))
dim2 <- mvrnorm(150, c(10,10),matrix(c(2,-1.2,2,1.2),2,2))
dim3 <- mvrnorm(150, c(15,10),matrix(c(2,1.2,2,1.2),2,2))
data_2 <- rbind(dim1, dim2, dim3) %>% as.data.frame

data_list <- data_1 %+% data_2

method_list <- kmeans(x=data_list, centers=3) %+% pam(x=data_list, k=3)

```

```
> str(method_list, vec.len = 1)
List of 2
 $ :List of 2
  ..$ method: chr "kmeans"
  ..$ params:List of 2
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ centers: num 3
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 3.67 ...
  .. .. .. ..$ data.V2: num [1:450] 8.42 ...
  .. .. ..$ centers: num 3
 $ :List of 2
  ..$ method: chr "pam"
  ..$ params:List of 2
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ k: num 3
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 3.67 ...
  .. .. .. ..$ data.V2: num [1:450] 8.42 ...
  .. .. ..$ k: num 3
```

In this example, not only are there two list entries for the two clustering methods, there are also two sub-lists each resulting in four possible method/parameter combinations. The exact same approach applies to other parameters, such as for example the number of clusters. In the following example there are 2 data sets and a range of four number of clusters, resulting in 8 different parameter sets for each method:

```
data_1 <- iris[,-5]

dim1 <- mvrnorm(150, c(5,10), matrix(c(2,1.2,2,1.2),2,2))
dim2 <- mvrnorm(150, c(10,10),matrix(c(2,-1.2,2,1.2),2,2))
dim3 <- mvrnorm(150, c(15,10),matrix(c(2,1.2,2,1.2),2,2))
data_2 <- rbind(dim1, dim2, dim3) %>% as.data.frame

data_list <- data_1 %+% data_2

k = 2:5

method_list <- kmeans(x=data_list, centers=k) %+% pam(x=data_list, k=k)

> str(method_list, vec.len = 1)
List of 2
 $ :List of 2
  ..$ method: chr "kmeans"
```

```
  ..$ params:List of 8
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ centers: int 2
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ centers: int 2
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ centers: int 3
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ centers: int 3
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ centers: int 4
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ centers: int 4
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ centers: int 5
  .. ..$ :List of 2
  .. .. ..$ x      :'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ centers: int 5
 $ :List of 2
  ..$ method: chr "pam"
  ..$ params:List of 8
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ k: int 2
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
```

```
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ k: int 2
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ k: int 3
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ k: int 3
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ k: int 4
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ k: int 4
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 150 obs. of  4 variables:
  .. .. .. ..$ data.Sepal.Length: num [1:150] 5.1 4.9 ...
  .. .. .. ..$ data.Sepal.Width : num [1:150] 3.5 3 ...
  .. .. .. ..$ data.Petal.Length: num [1:150] 1.4 1.4 ...
  .. .. .. ..$ data.Petal.Width : num [1:150] 0.2 0.2 ...
  .. .. ..$ k: int 5
  .. ..$ :List of 2
  .. .. ..$ x:'data.frame': 450 obs. of  2 variables:
  .. .. .. ..$ data.V1: num [1:450] 4.85 ...
  .. .. .. ..$ data.V2: num [1:450] 9.3 ...
  .. .. ..$ k: int 5
```

The same procedure is done for the validation criteria. In this example, package `clusterCrit` [52] is used due to its straightforward implementation of internal and external indices, although it is possible to pass any other (user-defined) function as well. For this simple example, two internal (the within-cluster scatter *Trace_W* and the *Dunn index* [62]) and two external indices (Rand [147] and Jaccard [97]) are used.

```
library(clusterCrit)
int <- c("Dunn", "Trace_W")
ext <- c("Jaccard", "Rand")
ind <- intCriteria(crit=int) %+% extCriteria(crit=ext)

> str(ind)
List of 2
 $ :List of 2
  ..$ method: chr "intCriteria"
  ..$ params:List of 2
  .. ..$ :List of 1
```

```
  .. .. ..$ crit: chr "Dunn"
  .. ..$ :List of 1
  .. .. ..$ crit: chr "Trace_W"
 $ :List of 2
  ..$ method: chr "extCriteria"
  ..$ params:List of 2
  .. ..$ :List of 1
  .. .. ..$ crit: chr "Jaccard"
  .. ..$ :List of 1
  .. .. ..$ crit: chr "Rand"
```

Finally, having defined all components of the benchmarking, the benchmarking object is assembled and passed to function `runBenchmark()` that performs the model and validation computations and stores the results in slots `models` and `validation`. In addition, the true cluster labels are also stored in order to compute the external validation indices. The complete setup therefore is done as follows:

```
data_1 <- iris[,-5]

dim1 <- mvrnorm(150, c(5,10), matrix(c(2,1.2,2,1.2),2,2))
dim2 <- mvrnorm(150, c(10,10),matrix(c(2,-1.2,2,1.2),2,2))
dim3 <- mvrnorm(150, c(15,10),matrix(c(2,1.2,2,1.2),2,2))
data_2 <- rbind(dim1, dim2, dim3) %>% as.data.frame

data_list <- data_1 %+% data_2

groups_d1 <- iris[,5]
groups_d2 <- as.integer(c(rep(1,100), rep(2,100)))

k <- 2:5

int <- c("Dunn", "Trace_W")
ext <- c("Jaccard", "Rand")
index_list <- intCriteria(crit=int) %+% extCriteria(crit=ext)

bench <- new("benchmarkObject")
bench@data <- data_list
bench@methods <- kmeans(x=data_list, centers=k) %+% pam(x=data_list, k=k)
bench@criteria <- index_list
bench@trueLabels <- list(groups_d1, groups_d2)

bench <- runBenchmark(bench)
```

The structure of the benchmark object `bench` is quite extensive and not practical to be shown here. However, an important part is obviously the `validation` slot, the structure of which is given in the following example. It should be noted that not only the calculated values are included, but also the full factorial setup of the parameters to allow for easy filtering per method.

```
> str(bench@validation, vec.len = 1)
List of 2
 $ kmeans:'data.frame': 8 obs. of  6 variables:
  ..$ x       : chr [1:8] "data_1" ...
```

```
  ..$ centers: chr [1:8] "2" ...
  ..$ Dunn   : chr [1:8] "0.0765063348396643" ...
  ..$ Trace_W: chr [1:8] "152.347951760358" ...
  ..$ Jaccard: chr [1:8] "0.572307705879211" ...
  ..$ Rand   : chr [1:8] "0.763668894767761" ...
 $ pam   :'data.frame': 8 obs. of  6 variables:
  ..$ x      : chr [1:8] "data_1" ...
  ..$ k      : chr [1:8] "2" ...
  ..$ Dunn   : chr [1:8] "0.0811107105653812" ...
  ..$ Trace_W: chr [1:8] "153.325715983363" ...
  ..$ Jaccard: chr [1:8] "0.587206482887268" ...
  ..$ Rand   : chr [1:8] "0.771901547908783" ...
```

Therefore, the results for data set 1 (the `iris` data) of method `pam()` can be displayed by:

```
> bench@validation$pam %>% filter(x=="data_1")
       x k                Dunn          Trace_W           Jaccard              Rand
1 data_1 2 0.0811107105653812 153.325715983363 0.587206482887268 0.771901547908783
2 data_1 3  0.098807393328081  78.851441426146 0.695858776569366  0.87973153591156
3 data_1 4  0.100843896817922 57.8779660876758 0.602400779724121 0.854765117168427
4 data_1 5  0.123508045438893 47.1354907407407 0.551614463329315 0.835973143577576
```

These values can then be analyzed further as desired. The emphasis of this approach on simple syntax enables easy extension of the setup. For example, if additional methods should be included this can simply be done by concatenating as many methods as needed (in the following case `cclust()` [55] and a parametric model of package `mclust` [163]. This can of course also be done for validation criteria or data sets.

```
bench@methods <- kmeans(x=data_list, centers=k) %+% pam(x=data_list, k=k) %+%
                 cclust(x=data_list, k=k) %+% Mclust(data=data_list, G=k)

bench <- runBenchmark(bench)

> str(bench@validation, vec.len = 1)
List of 4
 $ kmeans:'data.frame': 8 obs. of  6 variables:
  ..$ x      : chr [1:8] "data_1" ...
  ..$ centers: chr [1:8] "2" ...
  ..$ Dunn   : chr [1:8] "0.0765063348396643" ...
  ..$ Trace_W: chr [1:8] "152.347951760358" ...
  ..$ Jaccard: chr [1:8] "0.572307705879211" ...
  ..$ Rand   : chr [1:8] "0.763668894767761" ...
 $ pam   :'data.frame': 8 obs. of  6 variables:
  ..$ x      : chr [1:8] "data_1" ...
  ..$ k      : chr [1:8] "2" ...
  ..$ Dunn   : chr [1:8] "0.0811107105653812" ...
  ..$ Trace_W: chr [1:8] "153.325715983363" ...
  ..$ Jaccard: chr [1:8] "0.587206482887268" ...
  ..$ Rand   : chr [1:8] "0.771901547908783" ...
 $ Mclust:'data.frame': 8 obs. of  6 variables:
  ..$ data   : chr [1:8] "data_1" ...
  ..$ G      : chr [1:8] "2" ...
```

```
  ..$ Dunn    : chr [1:8] "0.338908682082323" ...
  ..$ Trace_W: chr [1:8] "154.947" ...
  ..$ Jaccard: chr [1:8] "0.595141708850861" ...
  ..$ Rand    : chr [1:8] "0.776286363601685" ...
 $ cclust:'data.frame': 8 obs. of  6 variables:
  ..$ x       : chr [1:8] "data_1" ...
  ..$ k       : chr [1:8] "2" ...
  ..$ Dunn    : chr [1:8] "0.0765063348396643" ...
  ..$ Trace_W: chr [1:8] "152.347951760358" ...
  ..$ Jaccard: chr [1:8] "0.572307705879211" ...
  ..$ Rand    : chr [1:8] "0.763668894767761" ...
```

The major advantage of this approach, beside the flexible setup of methods and data sets, is that all information on methods, data and their parameters along with the computation results are stored clearly structured in one object. This furthermore means that code that analyzes the benchmarking results can easily be re-used, as the structure of the benchmarking object is of course always the same.

4.5. Summary

This chapter has reviewed the guidelines and theoretical considerations on how benchmarking as such should be done. Furthermore, the concept of regarding the concrete implementation of benchmarking in programming code as being governed by structural rules that can be defined as a grammar was introduced and demonstrated by an exemplary implementation. This provides for a transparent and reproducible benchmarking process.

The next chapter focuses on one particular aspect of the benchmarking process: artificial data. Rather than prescribing a set of rules of how simulation data should be designed, the aim is to (much in the same way as was done in this chapter with the benchmarking as a whole) establish a transparent and reproducible way to create and share artificial data.

5. A Framework for Transparent and Reproducible Generation of Artificial Data

This chapter was published in Dangl and Leisch [46].

5.1. Background

When setting up a proper benchmarking study, one of the most important factors that comes to mind intuitively is objectiveness. It is certainly inherent to the definition of benchmarking that methods should be compared in a most objective and neutral way in order to determine an unbiased winner. As noted in chapter 4, Boulesteix et al. [27] have conducted a survey that investigates the outcome of comparison studies with respect to objectiveness. They found that benchmarking studies which are conducted as part of a paper that presents a new method very often identify said new method as the winner, while studies that exclusively compare methods do not always identify a clear winner. As already mentioned, this discrepancy may certainly be rooted in many aspects - for example overall benchmarking study design, selection of methods to compare to, or selection of benchmarking data. This chapter intends to address the latter of the three as one aspect that can be improved upon. Selection of benchmarking data sets is certainly one of the most crucial choices to make, and especially artificial data and its design is a major factor to a successful comparison study.

However, it is not the focus of this chapter to introduce guidelines on how artificial data should look like; the proposed framework shall rather be regarded as a contribution to an effort in the computational science community that has gained some momentum in recent years: reproducible research [126]. In order to substantiate scientific claims, this keyword refers to the necessity that researchers should not only need to describe the results of experiments and studies, but also provide a clear protocol which allows replication of those results by the reader [126]. Unfortunately, computations and the resulting conclusions are quite often taken at face value [58], which is especially problematic as studies find that frequently either a number of details essential for successful reproduction are missing

[136], or replication is at least difficult [96]. These are points that are obviously quite problematic in the context of benchmarking. Furthermore, insufficient reproducibility also leads to an increase in retracted papers [173] and has quite severe implications in practice, such as failed clinical trials [14, 143], which is certainly also not intended by any researcher. Stodden [175] proposes several points that should help to improve the situation, like providing links to source code and data, keeping track of the computing environment and versions of software used and publishing data and code in non-proprietary formats and under open licenses. With regard to the topic in this chapter - artificial data generation - the contribution to reproducible research is an infrastructure making the data set that was used for the computation easily available, a requirement that is also emphasized in Peng et al. [141]. One of the primary problems is that quite often a major hindrance to replicating results is that code that generates data is no longer available [140].

This issue shall be addressed by proposing a development framework for artificial data that allows to easily create, exchange, and generate data sets. The framework makes the development and generation of artificial data more transparent, and more importantly, reproducible. Basically, the framework consists of a web application that is used to store information on data sets and an R package that is used to generate the data sets. The main advantage of introducing a common framework for artificial data is that it is then very easy to obtain data from previous studies, should the author choose to make the code available to others (which of course has to be cited properly - another incentive); furthermore, if suitable data is already available from previous studies, there is no real argument to develop new data from scratch. This in turn again reduces the tendency to develop data selectively. Also, as noted above, quite often artificial data is insufficiently described in publications, or the code is not available (or only available for another software package), which makes re-coding of already used data quite cumbersome. This is also greatly reduced by an artificial data repository.

5.2. Framework Design and Terminology

The framework was designed in order to achieve a maximum of platform independence and ease of use, also for users who are not accustomed to programming. Furthermore, an open source approach also greatly increases availability and willingness to actually use this new way of managing artificial data.

The foundation of the whole concept is the R programming language due to its widespread use in the statistical community. While some R experience is necessary to create new benchmarking data sets with this framework, it is quite simple to download and generate data from existing setups.
There are two main parts to the process: an R package and a web repository. The

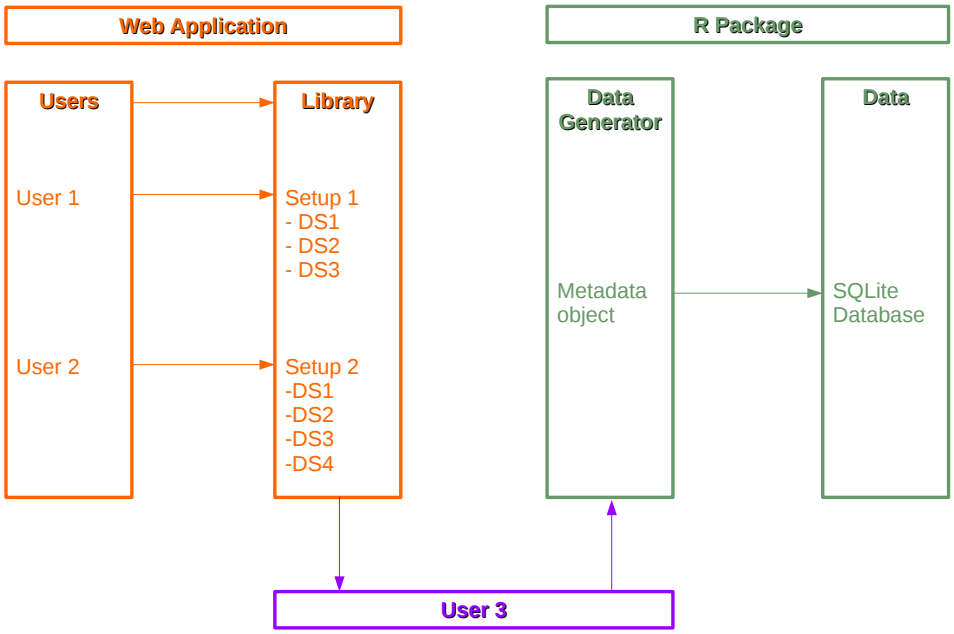

Figure 5.1.: Framework layout

R package implements all functionalities that are needed to generate artificial data sets from R script files that contain the metadata information. It also offers other tools that are described in more detail below. The script files are the essential part; basically, they have to be written from scratch by the user. They have to conform to specific rules in order to work with the package. The web repository as the second important part serves as a means to conveniently collect and exchange script files. As illustrated in Figure 5.1, the repository hosts data sets that are summarized in so called *benchmarking setups*. One user uploads a script file that contains a benchmarking setup which consists of several data sets. Another user can download the script file and generate the metadata information and in turn the actual data with the R package on the local computer.

5.3. Metadata

The framework is based around the notion of using metadata as a means to most efficiently store artificial data. The actual random numbers are not of great relevance; if all parameters of how the data set was generated (data generating process, random number seeds, etc.) are thoroughly documented, the data set can easily

be reproduced, eliminating the need to store the actual numbers. Furthermore, this greatly increases transparency, because all essential parameters about the data that could be of need in further analysis of test results are available. This also highlights why the title of this chapter proposes a 'metadata framework' - everything is centered around metadata information. Therefore the script file contains no code that generates actual numbers, it rather produces for each data set included in the setup a metadata object (an S4 object, in R programming terminology) that complies with the specifications in the R package and that is processed by it to produce actual data. The structure of the metadata object varies to some degree according to which data type is used (metric, binary, ordinal, etc.). Yet the basic structure is approximately the same and is illustrated in Figure 5.2. The parameters are assembled cluster-wise by the R package. All information necessary has to be included in the script file, for example with regard to metric data this includes at least the cluster centers, the variance-covariance matrix for each cluster, and the number of observations in each cluster. Furthermore, also a number generating function is needed. This can be just the name of the function if it already exists in R or in some other package, but also a custom function can be provided. Finally, also information on the random number seed is needed (which one to use, which random number seed, etc.). This way, the metadata object can be processed.

This structure of metadata has several convenient advantages. There is no need to awkwardly sift through programming code to extract information about artificial data that somebody else has implemented. All information needed is encapsulated in a clearly structured way in the metadata object. Moreover, the design of the object is very flexible, it allows a very broad scope of possible data sets. There is no prescribed random number generating function, no prescribed list of obligatory parameters for each cluster. It merely prescribes a certain structure that the metadata has to comply with. This provides a reliable structure that is the same across all data sets of this type and this greatly simplifies understanding and transparency.

5.3.1. Data Types

There are various data types for which metadata objects can be created. The most commonly used type is certainly metric data, but there are several more: in order to deal with categories there are binary and ordinal objects available; the functional data type allows implementation of time series data and other scenarios needed for functional clustering; there is also an implementation to generate random string data.

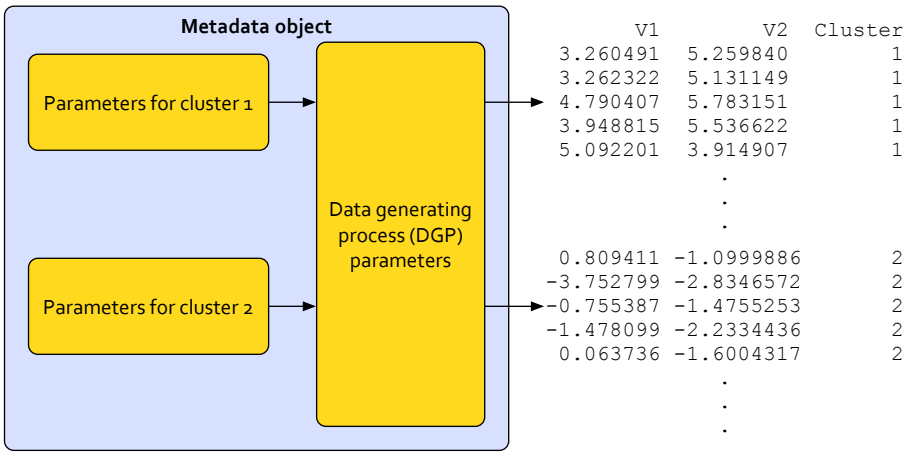

Figure 5.2.: Metadata to actual data

Metric Data

The metadata object for metric data primarily consists of two very important slots: the `distribution` slot that contains the function which generates random numbers (default is `mvrnorm()` from package `MASS`), and the `clusters` slot that contains the parameters that the function in slot `distribution` requires. Consequently, the arguments of the generating function and the list items in the clusters slot have to correspond. The data set is then assembled cluster by cluster by the respective function in the R package of the framework. The complete metadata object has an additional third slot: `seedinfo` contains information on the random number generator. Firstly, which one to use (the default is *Mersenne-Twister*, as it is the default in R), which version of R and thus the generator (default is the current R version) and the random number seed (default is 100). Figure 5.3 illustrates a simple metadata object for metric data.

Ordinal and Binary Data

For binary and ordinal data, the basic structure is essentially the same as for metric data, only the random number generating function is obviously different (the default here is `ordsample()` from package `GenOrd` and `generate.binary()` from package `MultiOrd` for ordinal and binary data respectively).

Functional Data

Functional data is the only type of metadata whose object differs notably from the other types in terms of overall structure. The cluster centers here are functions,

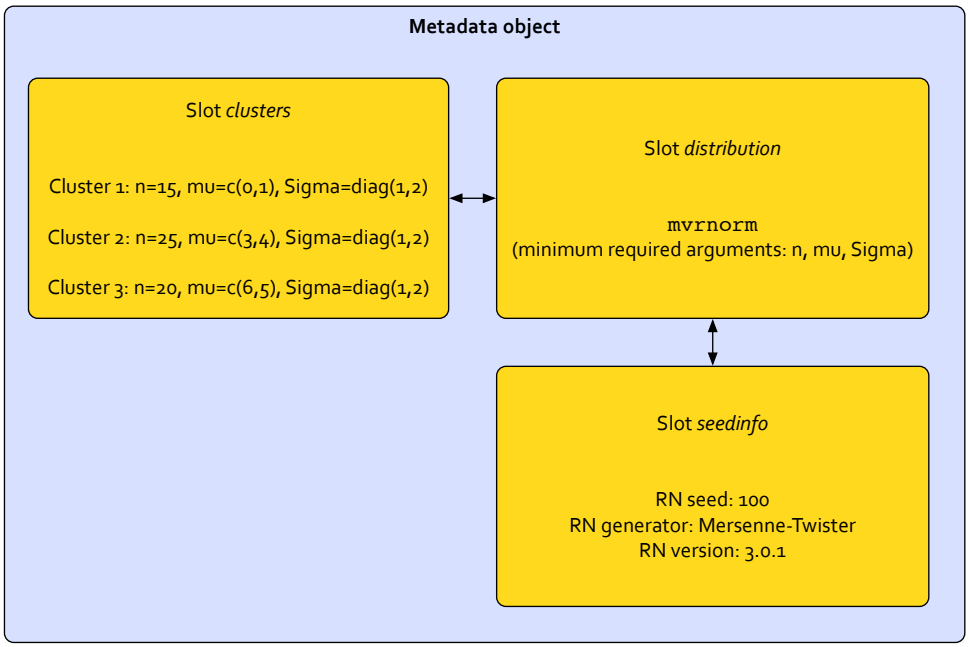

Figure 5.3.: Metric metadata object

which are stored as a list in a slot of the same name. The other slots provide all other parameters that are needed to evaluate the functions; the primary element being the `gridMatrix`, which encodes for each instance of a cluster center the number of time points and location of evaluation. In slot `interval`, the upper and lower boundaries are given; argument `granularity` determines the steps in-between. Figure 5.4 for example shows that instance 1 of a particular cluster center function is evaluated on position 3,4,6,7,8 and 10 of the interval. The evaluations are irregular, otherwise each instance would be evaluated at the same time points. Slots `sd` and `sd_distribution` determine the distribution and deviation of the instances around the cluster center function.

String Data

String data is implemented to support benchmarking of string distance measures. The structure of the metadata object is again quite similar to metric data in Figure 5.3, except in slot `distribution` there is a function that generates random strings based on a certain reference string and given a permissible maximum string distance. Those parameters (reference string, which type of distance and the maximum allowed distance) are again stored as a list for each cluster in slot `clusters`.

```
> sampleGrid(total_n=10, minT=4, maxT=7, granularity=10)

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0    0    1    1    0    1    1    1    0     1
 [2,]    1    0    1    0    0    0    0    1    0     1
 [3,]    1    1    0    0    0    1    1    0    0     0
 [4,]    0    1    0    1    1    1    1    0    1     0
 [5,]    1    0    1    1    0    1    1    1    1     0
 [6,]    1    1    0    0    1    0    0    0    1     1
 [7,]    0    1    0    0    1    1    1    1    1     1
 [8,]    1    1    0    0    1    1    0    1    1     1
 [9,]    1    1    1    1    0    1    1    0    0     1
[10,]    1    1    0    0    0    1    1    1    1     1
```

Figure 5.4.: gridMatrix for 10 instances of a cluster center

5.4. The R Package bdlp

The R package `bdlp` [45] is the primary tool to develop and generate data. It processes so called benchmarking setup files, which basically are R script files that produce the metadata objects in the form described above. The package implements the metadata object classes and provides functions that create templates for script files, check benchmarking setup files before submission and generate data from metadata objects.

Files containing a benchmarking setup (which in turn contains definitions for metadata objects) is in its most basic form an R script file that contains only one function. This function can return two things: either information on the data sets, or the metadata object for a specific data set. The former is merely intended as information for the user, in order to get an overview and choose a data set to generate; the latter is the basis to generate actual data, as explained above. An very minimal example that can generate metadata for two data sets and the accompanying info table is shown in Figure 5.5. For simplicity reasons, the arguments setting the random number seed parameters and the corresponding code have been omitted.

The obligatory function in the script file has to have a specific name: the author of the setup and the year it was created and/or published. The arguments of the function are also fixed. The `info` argument is used to toggle between info table or metadata output, for the latter the respective `setnr` is obviously also necessary. The `seedinfo` and `metaseedinfo` arguments are used to set the random number generator parameters for metadata and actual data generation. As it is of course possible to also have random effects already when generating metadata

```
require(MASS)

dangl2014 <- function(setnr = NULL,
                      seedinfo = [...],
                      info = FALSE,
                      metaseedinfo = [...]){


  inf <- data.frame(n = c(50, 40), k = c(2,2),
                    shape = c("spherical", "spherical"))
  ref <- "Dangl R. (2014) A small simulation study.
          Journal of Simple Datasets 10(2), 1-10"
  if(info == T) return(list(summary = inf, reference = ref))

  [...]

  if(setnr == 1) {
    return(new("metadata.metric",
               clusters = list(c1 = list(n = 25, mu = c(4,5),
                                         Sigma=diag(1,2)),
                               c2 = list(n = 25, mu = c(-1,-2),
                                         Sigma=diag(1,2))),
               dist = "mvrnorm", seedinfo = seedinfo))
  }

  if(setnr == 2){
    return(new("metadata.metric",
               clusters = list(c1 = list(n = 20, mu = c(0,2),
                                         Sigma=diag(1,2)),
                               c2 = list(n = 20, mu = c(-1,-2),
                                         Sigma=diag(1,2))),
               dist = "mvrnorm", seedinfo = seedinfo))
  }
}
```

Figure 5.5.: Minimal example for an experimental setup file (simplified)

```
> source("dangl2014.R")
> dangl2014(info = T)
$summary
n k     shape
1 50 2 spherical
2 40 2 spherical

$reference
[1] "Dangl R. (2014) A small simulation study. Journal of
   Simple Datasets 10(2), 1-10"
```

Figure 5.6.: Info output for an experimental setup file

```
> meta <- dangl2014(setnr = 1)
> str(meta)
Formal class 'metadata.metric' [package "bdlp"] with 4 slots
..@ standardization: chr "NONE"
..@ clusters       :List of 2
.. ..$ c1:List of 3
.. .. ..$ n    : num 25
.. .. ..$ mu   : num [1:2] 4 5
.. .. ..$ Sigma: num [1:2, 1:2] 1 0 0 1
.. ..$ c2:List of 3
.. .. ..$ n    : num 25
.. .. ..$ mu   : num [1:2] -1 -2
.. .. ..$ Sigma: num [1:2, 1:2] 1 0 0 1
..@ genfunc        :function (n = 1, mu, Sigma, tol = 1e-06,
    empirical = FALSE, EISPACK = FALSE)
..@ seedinfo       :List of 3
.. ..$ : num 100
.. ..$ : chr "4.0.3"
.. ..$ : chr [1:3] "Mersenne-Twister" "Inversion" "Rejection
   "
```

Figure 5.7.: Metadata object

```
> head(generateData(meta))
         V1        V2
1 3.022824 6.146590
2 4.127758 6.030671
3 2.393572 5.210740
4 3.453221 4.794338
5 4.225362 6.570782
6 4.239723 4.162089
```

Figure 5.8.: Generation of data from the metadata object

```
> createDataset(name="dangl2014", setnr=1, draws=10)
|============================================================| 100\%

10 version(s) of set no. 1 of setup dangl2014 generated.
Base seed 100 was used and is included in file name.
```

Figure 5.9.: Generation of data in form of an SQLite database

(e.g. location of cluster centers), two distinct sets of parameters can be defined.

Contrary to the strictly prescribed structure of the function, its content is highly flexible though - apart from the requirement that either the info table or a valid metadata object has to be returned, no specific restrictions are put on the code that can be executed in the function body. It is certainly possible to load required packages, even custom functions that are written from scratch by the developer can be included. In this case these functions are added below and simply sourced at runtime when the function is executed.

The benchmarking setup file can then be sourced and used to generate data. Figure 5.6 shows how information on the setup can be displayed and Figure 5.7 shows the generation and structure of the metadata object for one of the sample data sets of the example file in Figure 5.5. The metadata object can then be passed as a parameter to the function `generateData()`, which generates the actual data set (Figure 5.8). Once the script file is complete and all metadata objects work as desired, multiple draws of data sets from one metadata object can be generated (Figure 5.9). This is done by calling function `createDataset()`, where the name of the benchmarking setup, the desired data set number therein, and the number of draws have to be specified (and optionally of course the random number seed parameters, which overrides the default ones specified in the script file). An SQLite

```
createFileskeleton("dangl2014", "rainer.dangl@boku.ac.at",
                   "BOKU Vienna", "Rainer Dangl", "metric",
                   data.frame(n = c(50, 40), k = c(2,2),
                   shape = c("spherical", "spherical")),
                   "Dangl R. (2014) A small simulation
                      study.
                   Journal of Simple Datasets 10(2), 1-10")
```

Figure 5.10.: Creating a new template

database file is produced that contains the data sets. At this point, the data sets can then be further processed in other software environments that support this file type, not necessarily in R.

In principle, there are three ways to arrive at a benchmarking setup file that the package can process. Firstly, one can write the script file completely from scratch, according to the formatting guidelines. Secondly, one can generate a template for the script file using `createFileskeleton()` (Figure 5.10). It takes as arguments some author information, the reference and the info table (basically a stripped-down version of function `saveSetup()` described below). The resulting .R file looks like Figure 5.5, just without the two metadata objects. Then, the file can be modified further in a text editor. Thirdly, one can generate the script file completely automatically as shown in Figure 5.11. Each metadata object is initialized using the function `initializeObject()`. The arguments for this function are the data type, number of clusters and the random number generating function. Furthermore, one can also add information on the random number seed. The objects are then modified by the user, i.e. filled with the cluster parameters. Once the metadata objects are complete, the complete script file can be generated using `saveSetup()`.

If the user wishes to contribute the benchmarking setup to the web repository, the user should run function `checkSetup()` (shown in Figure 5.12) before uploading. This saves time because the same check is run on submitted files, and would result in instant rejection if unsuccessful. After a successful check and upload, the repository maintainers will have a final review of the new setup before it is made public in the library.

```
obj1 <- initializeObject(type="metric", k=2, distfunc="mvrnorm")
obj2 <- initializeObject(type="metric", k=3, distfunc="mvrnorm")

[...]

objlist <- list(obj1, obj2)

saveSetup(name="dangl2014", inst="BOKU",
          table=data.frame(n = c(50, 40), k = c(2,2),
                           shape = c("spherical", "spherical")),
          author="Rainer Dangl",
          cit="Dangl R. (2014) A small simulation study.
               Journal of Simple Datasets 10(2), 1-10",
          mail="rainer.dangl@boku.ac.at",
          objects=objlist)
```

Figure 5.11.: Creating a script file automatically

```
> checkSetup("dangl2014.R")
Sourcing input file ...
Done.
Checking consistency of function names ...
Done.
Checking reference ...
Done.
Checking whether a summary is produced ...
Done.
Checking whether metadata and datasets can be generated ...
Done.
Check complete! You can upload your benchmarking setup!
```

Figure 5.12.: Checking a setup file

5.5. The Web Repository

The website has been built using the R package `shiny` [39], which allows the development of R based web applications. There is an open source server environment available by RStudio [155] that runs a web app developed with `shiny`. `shiny` allows to implement reactive programming paradigms in R which means that function outputs are immediately updated once one of the input arguments changes. This means that calculations such as plotting are available in real time on the website and can react to changing inputs. For example, it is possible to select a benchmarking setup and a particular data set therein and look at a plot, switching to another data set triggers rendering of an updated plot. For this purpose, one instance of the data set is generated from the metadata information and a plot is drawn in real time and displayed on the website. Furthermore, it is possible to select a benchmarking setup and to have a look at the source code before actually downloading. These features also make using the framework easier for users that have little or no programming experience or who do not usually use R in their work.

The web repository (Figure 5.14) serves as a complement to the R package. It is not strictly necessary in order to generate data - this can be done entirely locally with the R package - but it is a convenient means to collect benchmarking setups and making them available to the public in an easily manageable manner. Benchmarking setups can be contributed by everybody and after an approval process by the maintainer they are put in the library. Figure 5.13 shows the structure of the website.

The download section includes a search page that features a live search form that uses the obligatory reference part of the info output of the script file to filter available setups based on authors, publication year and keywords in the title (Figure 5.15). The download page itself features a form to filter by data type and available setups and the included data sets thereof (Figure 5.16). Data sets can be plotted (Figure 5.17) and their parameters modified directly in the source code (Figure 5.18).

The contribute section is quite simple; there is a guide that links to the vignette (Figure 5.19) of package `bdlp` that contains a description of how to write a setup file. The upload page itself only contains a submit form. After the upload is complete, the website maintainer will review the submission. The R package includes the aforementioned function `checkSetup()` that allows the user to check the formal correctness of the script file already before submitting. If the reviewer determines that the submission is not only correct with regard to syntax but also content (especially whether there is a proper/correct citation), the setup is put in the library. The submitter will be notified in any case, be it acceptance or rejection. Furthermore, to simplify the generation of a new script file, a wizard

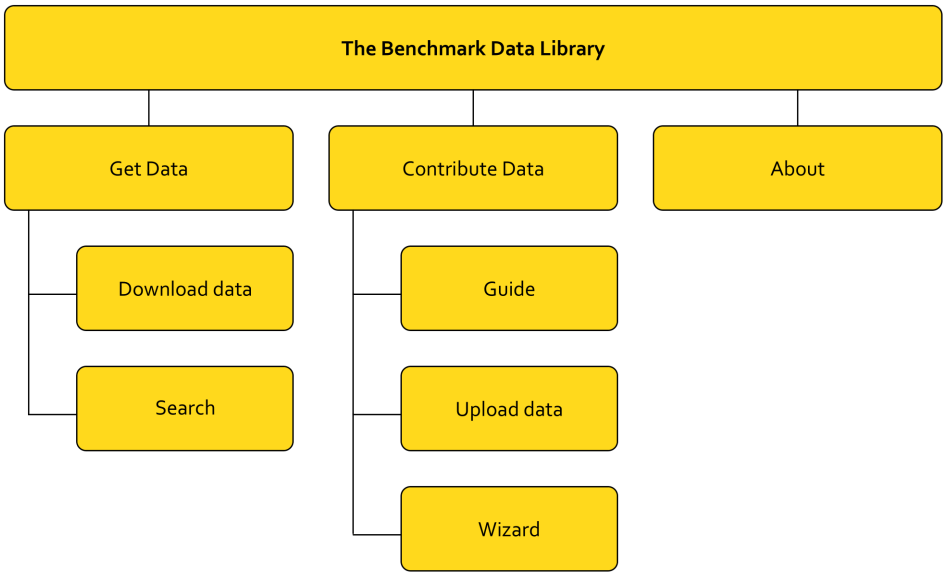

Figure 5.13.: Repository site map

can create the file skeleton (Figure 5.20), which is essentially the web version of the `createFileskeleton()` function of package `bdlp`.

5.6. Summary

The Benchmark Data Library and the accompanying R package `bdlp` are not only an attempt to standardize the format with which artificial data is stored and generated, it is an initiative to promote increasing objectiveness in benchmarking studies. The package should not just be a tool to create data, it should help researchers to collaborate and share their work. This certainly cannot be established overnight, but as the project is embedded in the efforts of the IFCS task force on benchmarking, the project will receive attention and does have a promising starting point.

Application in practice will also show further necessary modifications to the platform and the package (e.g. adapting the structure of metadata objects, add new data types, etc.), in order to support as many benchmarking applications as possible.

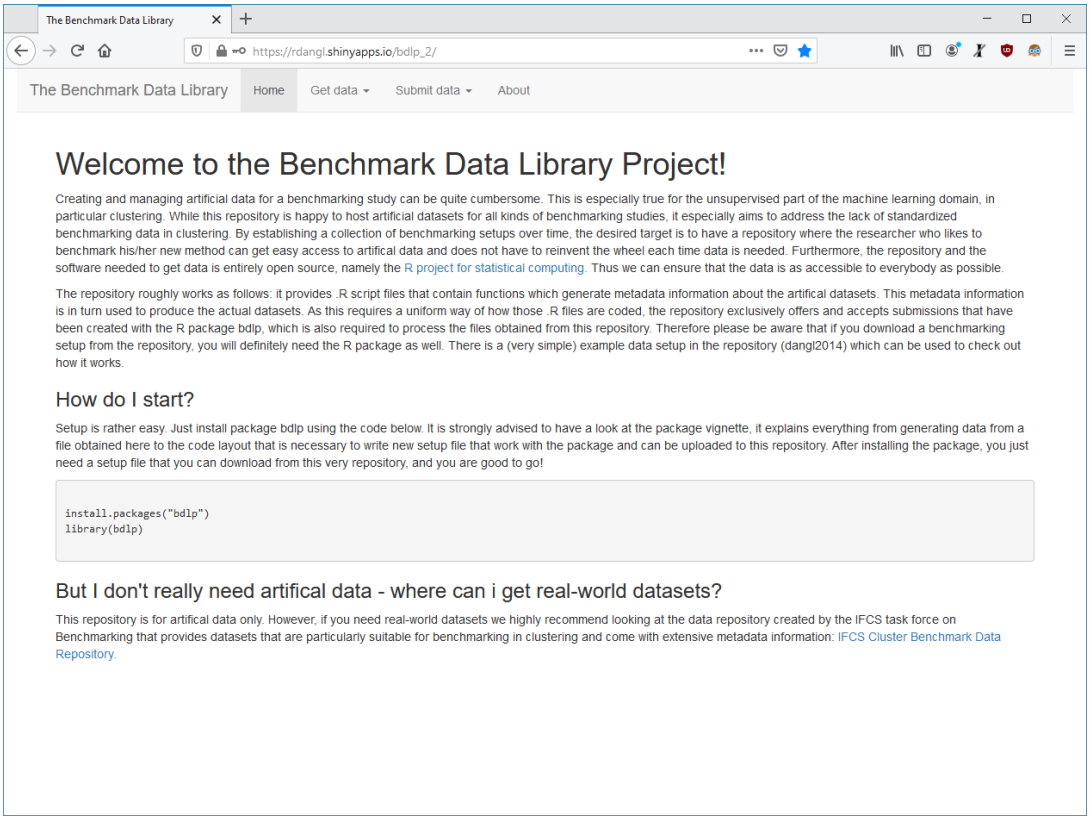

Figure 5.14.: Home page

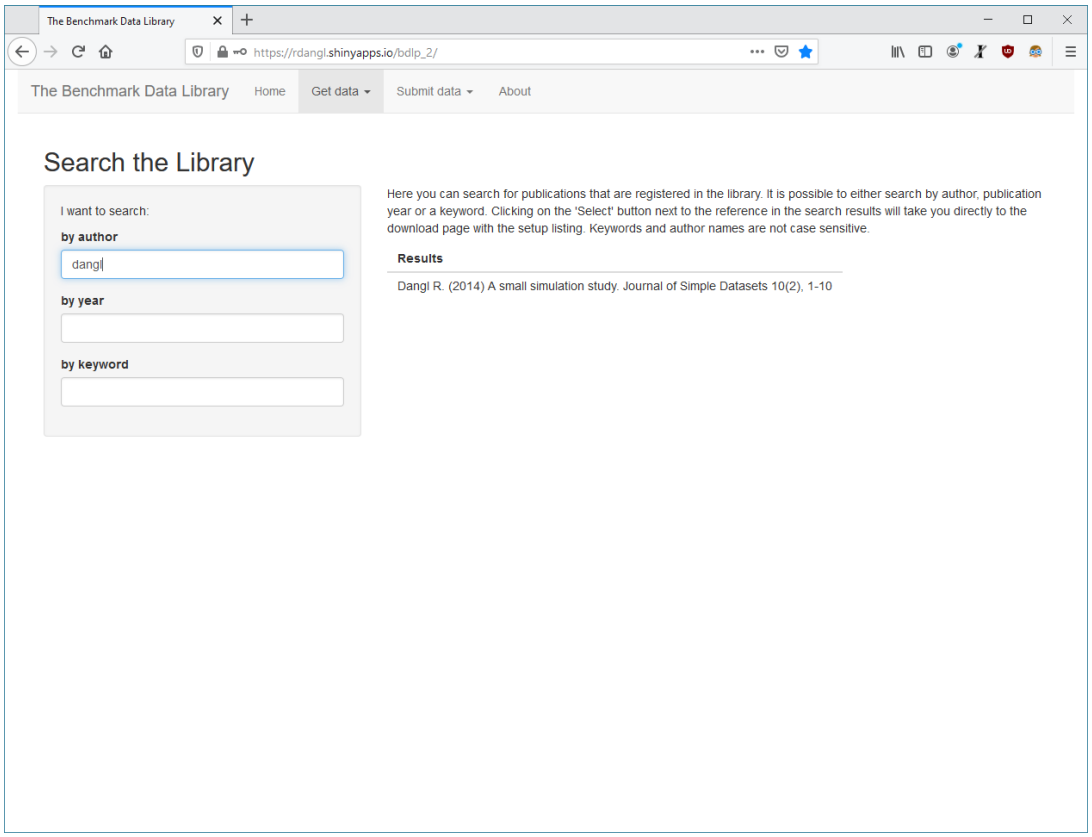

Figure 5.15.: Search page

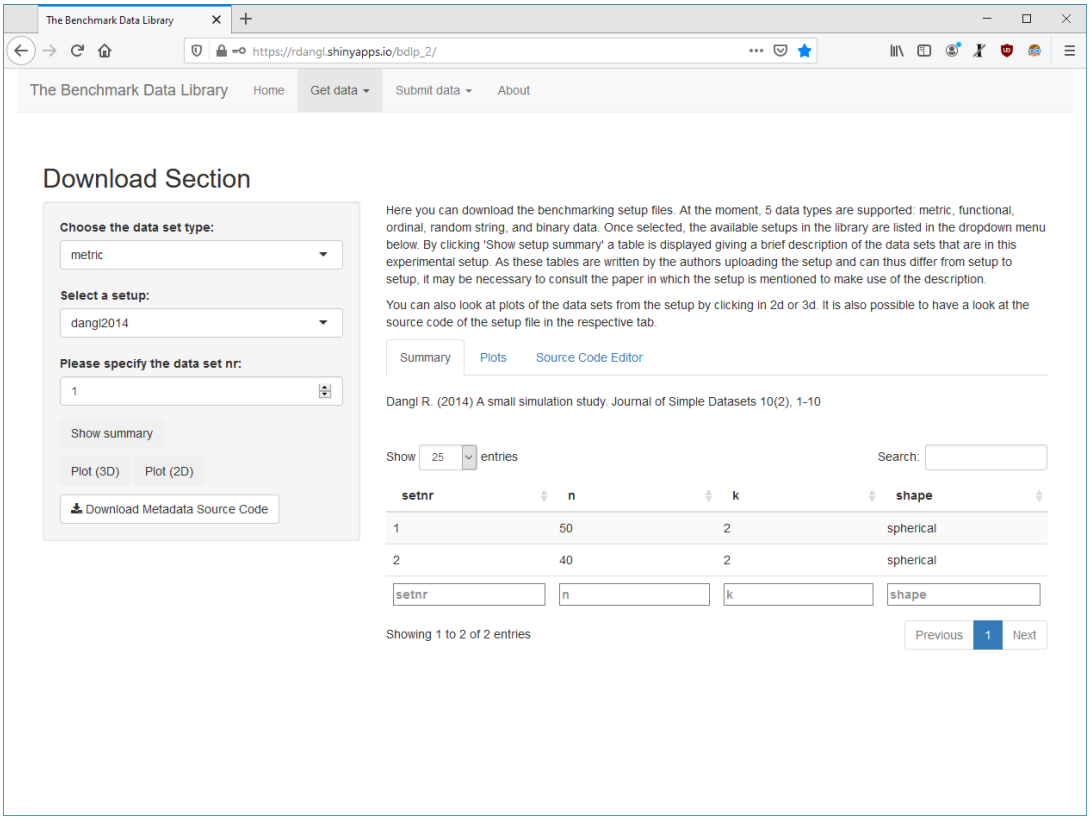

Figure 5.16.: Show setup summary

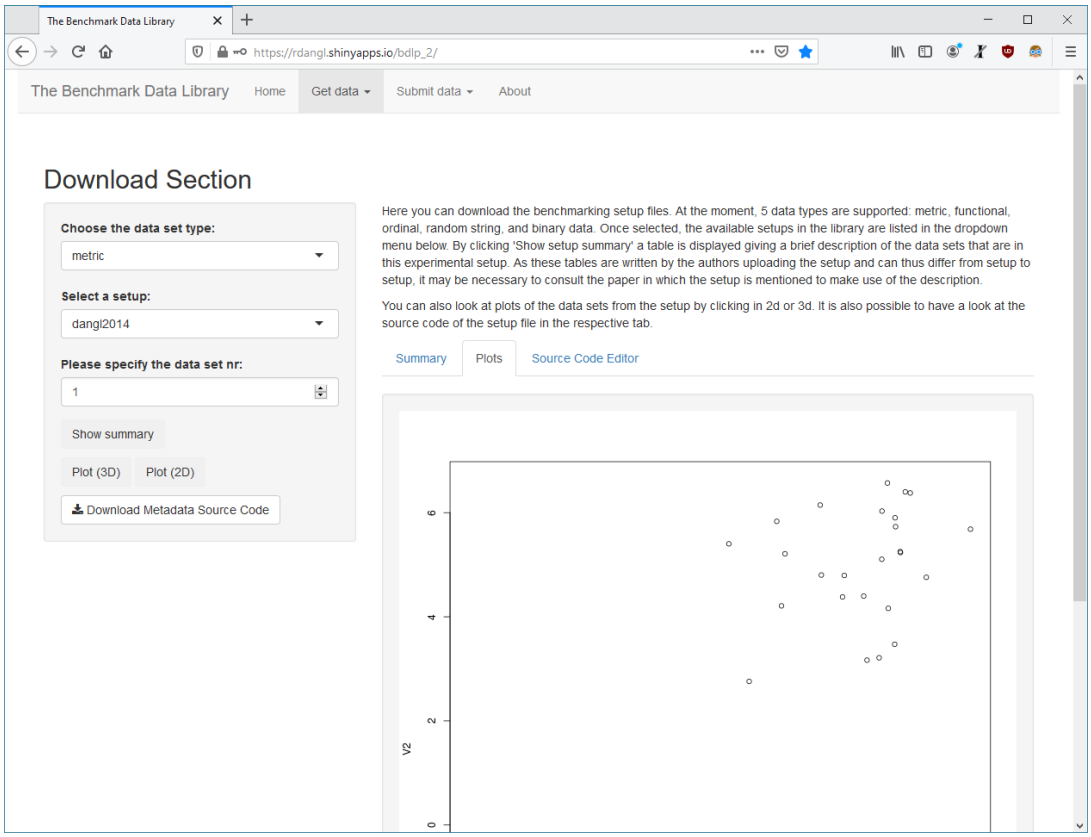

Figure 5.17.: Plot view

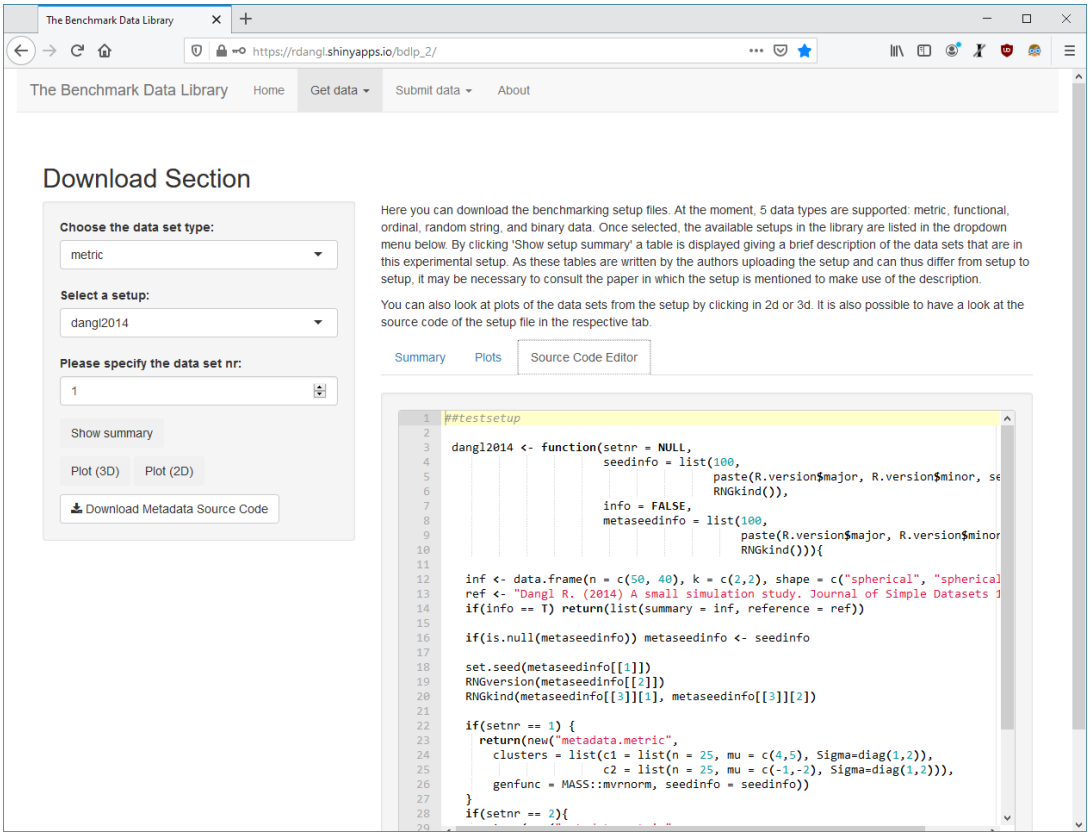

Figure 5.18.: Editor view

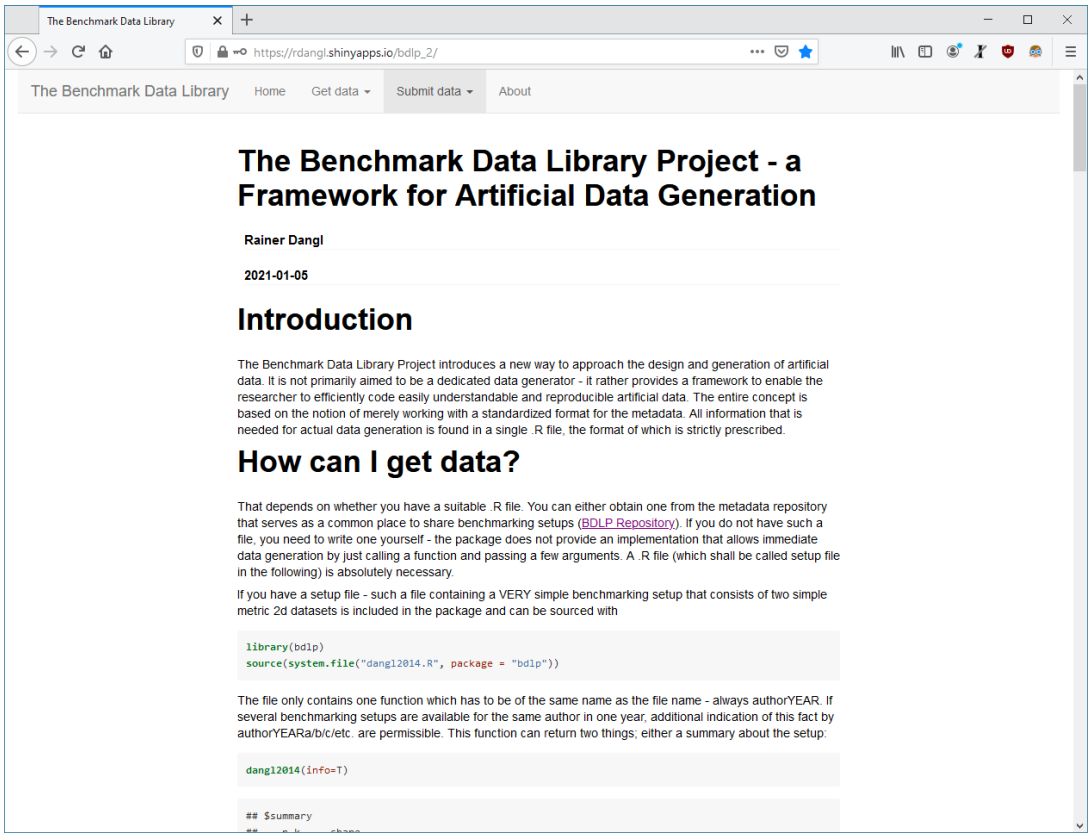

Figure 5.19.: bdlp package vignette

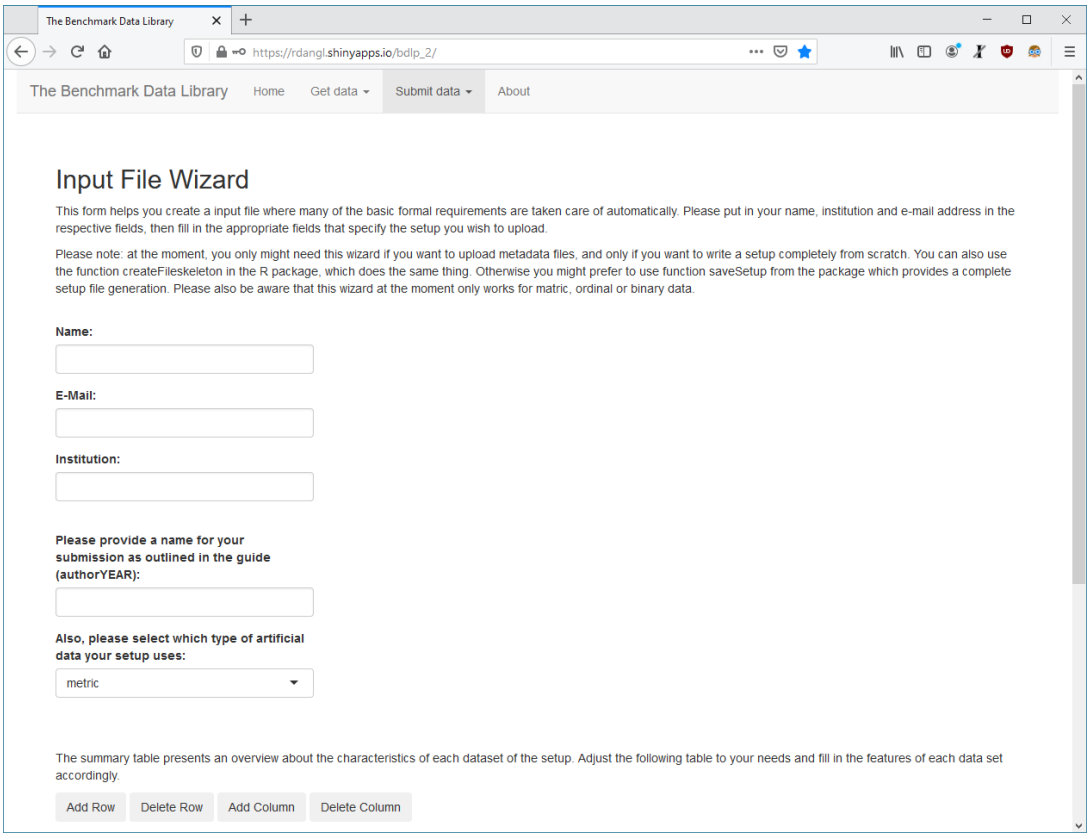

Figure 5.20.: File creator wizard

Conclusion

In the course of this dissertation, the topic of benchmarking in unsupervised learning was addressed from several angles. Chapters 1 and 2 presented a general overview of common algorithms in the field of clustering, with particular focus on algorithms that employ stability based considerations. This led to a comprehensive benchmarking study in chapter 3, where the interrelations between internal and external validation measures, stability based methods and the method of implementing the stability aspect were investigated. This resulted in findings and recommendations regarding possible combinations of validation indices and strategies for the application of stability-based methods. The results of this study were shared with project partners and subsequently published in Dangl and Leisch [47].

Secondly, lessons learned from setting up this benchmarking study sparked a discussion on how to approach benchmarking as such. Difficulties in replicating simulation setups and comparability issues with other studies resulted in considerations regarding some form of structural framework for benchmarking which follows similar regularities as language grammar. The aim of chapter 4 therefore was to discuss the foundations for a grammar of benchmarking. The prototypical implementation presented in the thesis emphasizes simple study setup and expandability. The concept of a benchmarking object that contains all method, data, and parameter information allows for a clearly structured and transparent benchmarking setup, where the information can be easily combined by a conjunction operator $\%+\%$. The implementation was done in R using the S4 class object oriented programming framework.

Thirdly, the specific aspect of artificial data generation in a benchmarking study was investigated in chapter 5 and published in Dangl and Leisch [46]. Work in this area was done in coordination with the IFCS task force on benchmarking that resulted in Van Mechelen et al. [185]. While the IFCS task force is working on a repository for real-world benchmarking data, chapter 5 of this thesis forms the counterpart for artificial data. Specifically, the framework introduced in chapter 5 and implemented in R package `bdlp` (published on CRAN [45]) provides infrastructure to easily and transparently generate artificial data in a reproducible manner. This intends to not only help streamline the benchmarking process as discussed in chapter 4, but in particular the aspect of artificial data generation. Furthermore, a web application based on package `shiny` [39] was developed to serve as a means to efficiently store and share data setups generated with `bdlp`.

On the whole, the contribution of this thesis to the field of unsupervised learning, specifically cluster analysis, is twofold. With regard to the application of benchmarking in cluster analysis, this thesis offers new insights into stability based model validation and how to apply it in practice. Furthermore, R package `bdlp` serves as a means to improve and facilitate the generation of artificial data. With regard to benchmarking theory, a discussion on how to conduct objective, transparent and reproducible studies was done, along with a prototypical implementation that offers a concrete way to do benchmarking that fulfills exactly these requirements.

Bibliography

[1] Basel Abu-Jamous, Rui Fa, and Asoke K Nandi. *Integrative cluster analysis in bioinformatics*. John Wiley & Sons, 2015.

[2] Enis Afgan, Dannon Baker, Marius Van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, et al. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research*, 44(W1): W3–W10, 2016.

[3] Nima Aghaeepour, Greg Finak, Holger Hoos, Tim R Mosmann, Ryan Brinkman, Raphael Gottardo, Richard H Scheuermann, FlowCAP Consortium, Dream Consortium, et al. Critical assessment of automated flow cytometry data analysis techniques. *Nature methods*, 10(3):228, 2013.

[4] Nima Aghaeepour, Pratip Chattopadhyay, Maria Chikina, Tom Dhaene, Sofie Van Gassen, Miron Kursa, Bart N Lambrecht, Mehrnoush Malek, GJ McLachlan, Yu Qian, et al. A benchmark for evaluation of algorithms for identification of cellular correlates of clinical outcomes. *Cytometry Part A*, 89(1):16–21, 2016.

[5] Levent L Albayrak, Kamil Khanipov, Mark Rojas, George Golovko, Maria Pimenova, Michael Kosoy, and Yuriy Fofanov. Exploration of natural alignment scoring rules and clustering thresholds for bacterial core/pan genome analysis. In *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 249–254. IEEE, 2017.

[6] JJ Allaire, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. *rmarkdown: Dynamic Documents for R*, 2020. URL `https://github.com/rstudio/rmarkdown`. R package version 2.1.

[7] Alexandre Angers-Loustau, Mauro Petrillo, Johan Bengtsson-Palme, Thomas Berendonk, Burton Blais, Kok-Gan Chan, Teresa M Coque, Paul Hammer, Stefanie Heß, Dafni M Kagkli, et al. The challenges of designing a benchmark strategy for bioinformatics pipelines in the identification

of antimicrobial resistance determinants using next generation sequencing technologies. *F1000Research*, 7, 2018.

[8] Mohamed Radhouene Aniba, Olivier Poch, and Julie D Thompson. Issues in bioinformatics benchmarking: the case study of multiple sequence alignment. *Nucleic Acids Research*, 38(21):7353–7363, 2010.

[9] Stefan Milton Bache and Hadley Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL `https://CRAN.R-project.org/package=magrittr`. R package version 1.5.

[10] G. H. Ball and D. J. Hall. ISODATA. A novel method of data analysis and pattern classification. Technical report, Menlo Park: Stanford Research Institute, 1965.

[11] Jeffrey D. Banfield and Adrian E. Raftery. Model-Based Gaussian and Non-Gaussian Clustering. *Biometrics*, 49(3):803–821, 1993.

[12] Michael Barton. nucleotid.es: an assembler catalogue. `http://nucleotid.es`, (accessed April 17, 2020).

[13] Giacomo Baruzzo, Katharina E Hayer, Eun Ji Kim, Barbara Di Camillo, Garret A FitzGerald, and Gregory R Grant. Simulation-based comprehensive benchmarking of rna-seq aligners. *Nature methods*, 14(2):135, 2017.

[14] C Glenn Begley and Lee M Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, 2012.

[15] Asa Ben-Hur, Andre Elisseeff, and Isabelle Guyon. A stability based method for discovering structure in clustered data. In *Pacific symposium on biocomputing*, volume 7, pages 6–17, 2001.

[16] Laurent Bergé, Charles Bouveyron, and Stéphane Girard. HDclassif: An R package for model-based clustering and discriminant analysis of high-dimensional data. *Journal of Statistical Software*, 46(6):1–29, 2012. URL `http://www.jstatsoft.org/v46/i06/`.

[17] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. Knime - the konstanz information miner: Version 2.0 and beyond. *SIGKDD Explor. Newsl.*, 11(1):26–31, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656280. URL `http://doi.acm.org/10.1145/1656274.1656280`.

[18] Alberto Bertoni and Giorgio Valentini. Discovering multi–level structures in bio-molecular data through the bernstein inequality. *BMC bioinformatics*, 9 (S2):S4, 2008.

[19] Jacob Bien and Rob Tibshirani. *protoclust: Hierarchical Clustering with Prototypes*, 2019. URL `https://CRAN.R-project.org/package=protoclust`. R package version 1.6.3.

[20] Anders Ellern Bilgrau, Poul Svante Eriksen, Jakob Gulddahl Rasmussen, Hans Erik Johnsen, Karen Dybkaer, and Martin Boegsted. GMCM: Unsupervised clustering and meta-analysis using gaussian mixture copula models. *Journal of Statistical Software*, 70(2):1–23, 2016. doi: 10.18637/jss.v070.i02.

[21] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[22] Meltzer Bittner, P Meltzer, Yidong Chen, Y Jiang, E Seftor, M Hendrix, M Radmacher, Rm Simon, Z Yakhini, A Ben-Dor, et al. Molecular classification of cutaneous malignant melanoma by gene expression profiling. *Nature*, 406(6795):536–540, 2000.

[23] John D Blischak, Peter Carbonetto, and Matthew Stephens. Creating and sharing reproducible research code the workflowr way [version 1; peer review: 3 approved]. *F1000Research*, 8(1749), 2019. doi: 10.12688/f1000research.20843.1. URL `https://doi.org/10.12688/f1000research.20843.1`.

[24] Hans Hermann Bock. *Automatische Klassifikation: Theoret. u. prakt. Methoden z. Gruppierung u. Strukturierung von Daten (Cluster-Analyse)*, volume 24. Vandenhoeck & Ruprecht, 1974.

[25] Athman Bouguettaya, Qi Yu, Xumin Liu, Xiangmin Zhou, and Andy Song. Efficient agglomerative hierarchical clustering. *Expert Systems with Applications*, 42(5):2785–2797, 2015.

[26] Anne-Laure Boulesteix. Ten simple rules for reducing overoptimistic reporting in methodological computational research. *PLoS computational biology*, 11(4), 2015.

[27] Anne-Laure Boulesteix, Lauer Sabine, and Eugster Manuel J. A. A Plea for Neutral Comparison Studies in Computational Sciences. *PLoS ONE*, 8(4): e61562, 04 2013. doi: 10.1371/journal.pone.0061562. URL `http://dx.doi.org/10.1371%2Fjournal.pone.0061562`.

[28] Anne-Laure Boulesteix, Robert Hable, Sabine Lauer, and Manuel J. A. Eugster. A statistical framework for hypothesis testing in real data comparison studies. *The American Statistician*, 69(3):201–212, 2015. doi: 10.1080/00031305.2015.1005128. URL `http://dx.doi.org/10.1080/00031305.2015.1005128`.

[29] Anne-Laure Boulesteix, Rory Wilson, and Alexander Hapfelmeier. Towards evidence-based computational statistics: lessons from clinical research on the role and design of real-data benchmark studies. *BMC medical research methodology*, 17(1):138, 2017.

[30] Anne-Laure Boulesteix, Harald Binder, Michal Abrahamowicz, Willi Sauerbrei, and Simulation Panel of the STRATOS Initiative. On the necessity and design of studies comparing statistical methods. *Biometrical Journal*, 60(1):216–218, 2018.

[31] Anne-Laure Boulesteix, Silke Janitza, Roman Hornung, Philipp Probst, Hannah Busen, and Alexander Hapfelmeier. Making complex prediction rules applicable for readers: Current practice in random forest literature and recommendations. *Biometrical Journal*, 61(5):1314–1328, 2019.

[32] Paul C Boutros, Adam A Margolin, Joshua M Stuart, Andrea Califano, and Gustavo Stolovitzky. Toward better benchmarking: challenge-based methods assessment in cancer genomics. *Genome biology*, 15(9):462, 2014.

[33] Charles Bouveyron. *funFEM: Clustering in the Discriminative Functional Subspace*, 2015. URL `https://CRAN.R-project.org/package=funFEM`. R package version 1.1.

[34] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, Inanç Birol, Sébastien Boisvert, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):2047–217X, 2013.

[35] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[36] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics - Theory and Methods*, 3(1):1–27, 1974.

[37] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-37456-2.

[38] Peter Carbonetto, Matthew Stephens, and Gao Wang. Dsc: Dynamic statistical comparisons. `https://stephenslab.github.io/dsc-wiki/overview.html`, (accessed April 17, 2020).

[39] Winston Chang, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. *shiny: Web Application Framework for R*, 2020. URL `https://CRAN.R-project.org/package=shiny`. R package version 1.5.0.

[40] Pierre Chaussé. Computing generalized method of moments and generalized empirical likelihood with R. *Journal of Statistical Software*, 34(11):1–35, 2010. URL `http://www.jstatsoft.org/v34/i11/`.

[41] Wei-Chen Chen and Ranjan Maitra. EMCluster: EM algorithm for model-based clustering of finite mixture gaussian distribution, 2015. R Package, URL http://cran.r-project.org/package=EMCluster.

[42] Paweł Cichosz. Hierarchical clustering. In *Data Mining Algorithms*, pages 349–372. John Wiley & Sons, Ltd, Chichester, UK, 2015. ISBN 9781118332580.

[43] James C Costello, Laura M Heiser, Elisabeth Georgii, Mehmet Gönen, Michael P Menden, Nicholas J Wang, Mukesh Bansal, Petteri Hintsanen, Suleiman A Khan, John-Patrick Mpindi, et al. A community effort to assess and improve drug sensitivity prediction algorithms. *Nature biotechnology*, 32 (12):1202, 2014.

[44] Raphael Couronné, Philipp Probst, and Anne-Laure Boulesteix. Random forest versus logistic regression: a large-scale benchmark experiment. *BMC bioinformatics*, 19(1):270, 2018.

[45] Rainer Dangl. *bdlp: Transparent and Reproducible Artificial Data Generation*, 2017. URL `https://CRAN.R-project.org/package=bdlp`. R package version 0.9-2.

[46] Rainer Dangl and Friedrich Leisch. On a comprehensive metadata framework for artificial data in unsupervised learning. *Archives of Data Science, Series A (Online First)*, 2(1):63–78, 2017. doi: 10.5445/KSP/1000058749/22.

[47] Rainer Dangl and Friedrich Leisch. Effects of resampling in determining the number of clusters in a data set. *Journal of Classification*, Jul 2019. ISSN 1432-1343. doi: 10.1007/s00357-019-09328-2. URL `https://doi.org/10.1007/s00357-019-09328-2`.

[48] David L. Davies and Donald W. Bouldin. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1 (2):224–227, April 1979.

[49] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50 (1):1–18, 2000.

[50] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[51] Bernard Desgraupes. Clustering indices. *University Paris Ouest Lab Modal'X*, 2013.

[52] Bernard Desgraupes. *clusterCrit: Clustering Indices*, 2018. URL `https://CRAN.R-project.org/package=clusterCrit`. R package version 1.2.8.

[53] Cambridge English Dictionary, 2020. URL `https://dictionary.cambridge.org/dictionary/english/grammar`.

[54] Marie-Agnès Dillies, Andrea Rau, Julie Aubert, Christelle Hennequet-Antier, Marine Jeanmougin, Nicolas Servant, Céline Keime, Guillemette Marot, David Castel, Jordi Estelle, et al. A comprehensive evaluation of normalization methods for illumina high-throughput rna sequencing data analysis. *Briefings in bioinformatics*, 14(6):671–683, 2013.

[55] Evgenia Dimitriadou. *cclust: Convex Clustering Methods and Clustering Indexes*, 2017. URL `https://CRAN.R-project.org/package=cclust`. R package version 0.6-21.

[56] Evgenia Dimitriadou, Sara Dolničar, and Andreas Weingessel. An examination of indexes for determining the number of clusters in binary data sets. *Psychometrika*, 67(1):137–159, March 2002.

[57] Sara Dolnicar, Bettina Grün, Friedrich Leisch, and Kathrin Schmidt. Required Sample Sizes for Data-Driven Market Segmentation Analyses in Tourism. *Journal of Travel Research*, page 0047287513496475, 2013.

[58] David L Donoho. An invitation to reproducible computational research. *Biostatistics*, 11(3):385–388, 2010.

[59] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V Vinay. Clustering in large graphs and matrices. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 291–299. Society for Industrial and Applied Mathematics, 1999.

[60] Richard C Dubes. Cluster analysis and related issues. In *Handbook of pattern recognition and computer vision*, pages 3–32. World Scientific, 1999.

[61] Sandrine Dudoit and Jane Fridlyand. A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biology*, 3(7), 2002.

[62] J. C. Dunn. Well-Separated Clusters and Optimal Fuzzy Partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[63] Angelo Duò, Mark D Robinson, and Charlotte Soneson. A systematic performance evaluation of clustering methods for single-cell rna-seq data. *F1000Research*, 7, 2018.

[64] Dent Earl, Keith Bradnam, John St John, Aaron Darling, Dawei Lin, Joseph Fass, Hung On Ken Yu, Vince Buffalo, Daniel R Zerbino, Mark Diekhans, et al. Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome research*, 21(12):2224–2241, 2011.

[65] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[66] Evangelos Evangelou and John PA Ioannidis. Meta-analysis methods for genome-wide association studies and beyond. *Nature Reviews Genetics*, 14 (6):379–389, 2013.

[67] Brian S Everitt. *Cluster Analysis*. Edward Arnold, 1993.

[68] Adam D Ewing, Kathleen E Houlahan, Yin Hu, Kyle Ellrott, Cristian Caloian, Takafumi N Yamaguchi, J Christopher Bare, Christine P'ng, Daryl Waggott, Veronica Y Sabelnykova, et al. Combining tumor genome simulation with crowdsourcing to benchmark somatic single-nucleotide-variant detection. *Nature methods*, 12(7):623–630, 2015.

[69] Greg Finak, Bryan Mayer, William Fulp, Paul Obrecht, Alicia Sato, Eva Chung, Drienna Holman, and Raphael Gottardo. Datapackager: Reproducible data preprocessing, standardization and sharing using r/bioconductor for collaborative data analysis. *Gates open research*, 2, 2018.

[70] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of human genetics*, 7(2):179–188, 1936.

[71] IFCS Benchmarking Task Force. Cluster benchmark data repository, 2021. URL `https://ifcs.boku.ac.at/repository/index.html`.

[72] Edward B Fowlkes and Colin L Mallows. A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78 (383):553–569, 1983.

[73] Saskia Freytag, Luyi Tian, Ingrid Lönnstedt, Milica Ng, and Melanie Bahlo. Comparison of clustering tools in r for medium-sized 10x genomics single-cell rna-sequencing data. *F1000Research*, 7, 2018.

[74] Jane Fridlyand and Sandrine Dudoit. Applications of resampling methods to estimate the number of clusters and to improve the accuracy of a clustering method. Technical report, Technical Report 600, Department of Statistics, UC Berkeley, 2001.

[75] Iddo Friedberg, Mark N Wass, Sean D Mooney, and Predrag Radivojac. Ten simple rules for a community computational challenge. *PLoS computational biology*, 11(4), 2015.

[76] Herman P Friedman and Jerrold Rubin. On some invariant criteria for grouping data. *Journal of the American Statistical Association*, 62(320): 1159–1178, 1967.

[77] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001.

[78] Marek Gagolewski, Maciej Bartoszuk, and Anna Cena. Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm. *Information Sciences*, 363:8–23, 2016. doi: 10.1016/j.ins.2016.05.003.

[79] Paul P Gardner, James M Paterson, Fatemeh Ashari Ghomi, Sinan Uğur U Umu, Stephanie McGimpsey, and Aleksandra Pawlik. A meta-analysis of bioinformatics software benchmarks reveals that publication-bias unduly influences software accuracy. *BioRxiv*, page 092205, 2017.

[80] Paul P Gardner, Renee J Watson, Xochitl C Morgan, Jenny L Draper, Robert D Finn, Sergio E Morales, and Matthew B Stott. Identifying accurate metagenome and amplicon software via a meta-analysis of sequence to taxonomy benchmarking studies. *PeerJ*, 7:e6160, 2019.

[81] James E Gentle. *Computational statistics*, volume 308. Springer, 2009.

[82] Allan D Gordon. Cluster validation. In *Data Science, Classification, and Related Methods*, pages 22–39. Springer, 1998.

[83] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. *ACM Sigmod record*, 27(2):73–84, 1998.

[84] Michael Hahsler, Matthew Piekenbrock, and Derek Doran. dbscan: Fast density-based clustering with R. *Journal of Statistical Software*, 91(1):1–30, 2019. doi: 10.18637/jss.v091.i01.

[85] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On Clustering Validation Techniques. *J. Intell. Inf. Syst.*, 17(2-3):107–145, 2001.

[86] Julia Handl, Joshua D. Knowles, and Douglas B. Kell. Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21(15):3201–3212, 2005.

[87] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 99th edition, 1975. ISBN 047135645X.

[88] Christian Hennig. What are the true clusters? *Pattern Recognition Letters*, 64:53 – 62, 2015. ISSN 0167-8655. doi: http://dx.doi.org/10.1016/j.patrec.2015.04.009. URL `http://www.sciencedirect.com/science/article/pii/S0167865515001269`. Philosophical Aspects of Pattern Recognition.

[89] Christian Hennig. *fpc: Flexible Procedures for Clustering*, 2020. URL `https://CRAN.R-project.org/package=fpc`. R package version 2.2-5.

[90] Steven M Hill, Laura M Heiser, Thomas Cokelaer, Michael Unger, Nicole K Nesser, Daniel E Carlin, Yang Zhang, Artem Sokolov, Evan O Paull, Chris K Wong, et al. Inferring causal molecular networks: empirical assessment through a community-based effort. *Nature methods*, 13(4):310–318, 2016.

[91] Benjamin Hofner, Matthias Schmid, and Lutz Edler. Reproducible research in statistics: A review and guidelines for the biometrical journal. *Biometrical journal*, 58(2):416–427, 2016.

[92] Fangxin Hong and Rainer Breitling. A comparison of meta-analysis methods for detecting differentially expressed genes in microarray experiments. *Bioinformatics*, 24(3):374–382, 2008.

[93] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. ISSN 0176-4268. doi: 10.1007/BF01908075. URL `http://dx.doi.org/10.1007/BF01908075`.

[94] John PA Ioannidis. Why most published research findings are false. *PLos med*, 2(8):e124, 2005.

[95] John PA Ioannidis. Meta-research: Why research on research matters. *PLoS biology*, 16(3):e2005468, 2018.

[96] John PA Ioannidis, David B Allison, Catherine A Ball, Issa Coulibaly, Xiangqin Cui, Aedín C Culhane, Mario Falchi, Cesare Furlanello, Laurence Game, Giuseppe Jurman, et al. Repeatability of published microarray gene expression analyses. *Nature genetics*, 41(2):149–155, 2009.

[97] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[98] Anil K Jain. Data clustering: 50 years beyond k-means. *ECML/PKDD (1)*, 5211:3–4, 2008.

[99] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[100] Alexander Kanitz, Foivos Gypas, Andreas J Gruber, Andreas R Gruber, Georges Martin, and Mihaela Zavolan. Comparative assessment of methods for the computational inference of transcript isoform abundance from rna-seq data. *Genome biology*, 16(1):150, 2015.

[101] Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL `http://www.jstatsoft.org/v11/i09/`.

[102] M Kathleen Kerr and Gary A Churchill. Bootstrapping cluster analysis: assessing the reliability of conclusions from microarray experiments. *Proceedings of the national academy of sciences*, 98(16):8961–8965, 2001.

[103] G. J. McLachlan Khan and N. On a resampling approach for tests on the number of clusters with mixture model-based clustering of tissue samples. *Journal of Multivariate Analysis*, 90:90–105, 2004.

[104] Patrick K Kimes and Alejandro Reyes. Reproducible and replicable comparisons using SummarizedBenchmark. *Bioinformatics*, 35(1):137–139, 07 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty627. URL https://doi.org/10.1093/bioinformatics/bty627.

[105] Keegan Korthauer, Patrick K Kimes, Claire Duvallet, Alejandro Reyes, Ayshwarya Subramanian, Mingxiang Teng, Chinmay Shukla, Eric J Alm, and Stephanie C Hicks. A practical guide to methods controlling false discoveries in computational biology. *Genome biology*, 20(1):118, 2019.

[106] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *WIREs Data Mining and Knowledge Discovery*, 1(3):231–240, 2011. doi: 10.1002/widm.30. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.30.

[107] Peter Krusche, Len Trigg, Paul C Boutros, Christopher E Mason, M Francisco, Benjamin L Moore, Mar Gonzalez-Porta, Michael A Eberle, Zivana Tezak, Samir Lababidi, et al. Best practices for benchmarking germline small-variant calls in human genomes. *Nature biotechnology*, 37(5):555–560, 2019.

[108] Robert Küffner, Neta Zach, Raquel Norel, Johann Hawe, David Schoenfeld, Liuxia Wang, Guang Li, Lilly Fang, Lester Mackey, Orla Hardiman, et al. Crowdsourced analysis of clinical trial data to predict amyotrophic lateral sclerosis progression. *Nature biotechnology*, 33(1):51, 2015.

[109] Stanisław Kulczyński. *Die Pflanzenassoziationen der Pieninen*. Imprimerie de l'Université, 1928.

[110] Matthew Kyan, Paisarn Muneesawang, Kambiz Jarrah, and Ling Guan. *Unsupervised Learning: A Dynamic Approach*. John Wiley & Sons, Ltd, 2014. ISBN 9781118875568. doi: 10.1002/9781118875568.ch2. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118875568.ch2.

[111] W. J. Lai and Krzanowski Y. T. A Criterion for Determining the Number of Groups in a Data Set Using Sum-of-Squares Clustering. *Biometrics*, 44: 23–34, March 1988.

[112] Tilman Lange, Volker Roth, Mikio L. Braun, and Joachim M. Buhmann. Stability-Based Validation of Clustering Solutions. *Neural Computation*, 16 (6):1299–1323, 2004.

[113] Peter Langfelder and Steve Horvath. Fast R functions for robust correlations and hierarchical clustering. *Journal of Statistical Software*, 46(11): 1–17, 2012. URL `http://www.jstatsoft.org/v46/i11/`.

[114] Charity W Law, Yunshun Chen, Wei Shi, and Gordon K Smyth. voom: Precision weights unlock linear model analysis tools for rna-seq read counts. *Genome biology*, 15(2):R29, 2014.

[115] Friedrich Leisch. A Toolbox for K-Centroids Cluster Analysis. *Computational Statistics and Data Analysis*, 51(2):526–544, 2006.

[116] Erel Levine and Eytan Domany. Resampling Method For Unsupervised Estimation Of Cluster Validity. *Neural Computation*, 13:2573–2593, 2001.

[117] Jacob H Levine, Erin F Simonds, Sean C Bendall, Kara L Davis, D Amir El-ad, Michelle D Tadmor, Oren Litvin, Harris G Fienberg, Astraea Jager, Eli R Zunder, et al. Data-driven phenotypic dissection of aml reveals progenitor-like cells that correlate with prognosis. *Cell*, 162(1):184–197, 2015.

[118] Jialu Liu, Deng Cai, and Xiaofei He. Gaussian mixture model with local consistency. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[119] Tagaram Soni Madhulatha. Comparison between k-means and k-medoids clustering algorithms. In David C. Wyld, Michal Wozniak, Nabendu Chaki, Natarajan Meghanathan, and Dhinaharan Nagamalai, editors, *Advances in Computing and Information Technology*, pages 472–481, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-22555-0.

[120] Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2019. R package version 2.1.0 — For new features, see the 'Changelog' file (in the package source).

[121] Patrick Mair and Marcus Hudec. Multivariate weibull mixtures with proportional hazard restrictions for dwell-time-based session clustering with incomplete data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 58(5):619–639, 2009. doi: 10.1111/j.1467-9876.2009.00665.x. URL `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9876.2009.00665.x`.

[122] Patrick Mair and Marcus Hudec. *mixPHM: Mixtures of Proportional Hazard Models*, 2015. URL `https://CRAN.R-project.org/package=mixPHM`. R package version 0.7-2.

[123] Serghei Mangul, Lana S Martin, Brian L Hill, Angela Ka-Mei Lam, Margaret G Distler, Alex Zelikovsky, Eleazar Eskin, and Jonathan Flint. Systematic benchmarking of omics computational tools. *Nature communications*, 10(1):1–11, 2019.

[124] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11):205, 2017.

[125] Marina Meilă. The uniqueness of a good optimum for k-means. In *Proceedings of the 23rd international conference on Machine learning*, pages 625–632. ACM, 2006.

[126] Jill P Mesirov. Computer science. accessible reproducible research. *Science (New York, NY)*, 327(5964), 2010.

[127] Ulrich Moller and Dörte Radke. A cluster validity approach based on nearest-neighbor resampling. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 1, pages 892–895. IEEE, 2006.

[128] Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. Consensus clustering – A resampling-based method for class discovery and visualization of gene expression microarray data. In *Machine Learning, Functional Genomics Special Issue*, pages 91–118, 2003.

[129] Tim P Morris, Ian R White, and Michael J Crowther. Using simulation studies to evaluate statistical methods. *Statistics in medicine*, 38(11):2074–2102, 2019.

[130] John Moult, Krzysztof Fidelis, Andriy Kryshtafovych, Torsten Schwede, and Anna Tramontano. Critical assessment of methods of protein structure prediction: Progress and new directions in round xi. *Proteins: Structure, Function, and Bioinformatics*, 84:4–14, 2016.

[131] John Moult, Krzysztof Fidelis, Andriy Kryshtafovych, Torsten Schwede, and Anna Tramontano. Critical assessment of methods of protein structure prediction (casp)—round xii. *Proteins: Structure, Function, and Bioinformatics*, 86:7–15, 2018.

[132] Lampros Mouselimis. *ClusterR: Gaussian Mixture Models, K-Means, Mini-Batch-Kmeans, K-Medoids and Affinity Propagation Clustering*, 2019. URL `https://CRAN.R-project.org/package=ClusterR`. R package version 1.2.1.

[133] G Bel Mufti, P Bertrand, and EL Moubarki. Determining the number of groups from measures of cluster stability. In *Proceedings of International Symposium on Applied Stochastic Models and Data Analysis*, pages 17–20, 2005.

[134] Daniel Müllner. fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python. *Journal of Statistical Software*, 53(9):1–18, 2013. URL `http://www.jstatsoft.org/v53/i09/`.

[135] Fionn Murtagh and Pierre Legendre. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of classification*, 31(3):274–295, 2014.

[136] Anton Nekrutenko and James Taylor. Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. *Nature Reviews Genetics*, 13(9):667–672, 2012.

[137] Raquel Norel, John Jeremy Rice, and Gustavo Stolovitzky. The self-assessment trap: can we all be better than average? *Molecular systems biology*, 7(1), 2011.

[138] Malgorzata Nowicka and Mark D Robinson. Drimseq: a dirichlet-multinomial framework for multivariate count outcomes in genomics. *F1000Research*, 5, 2016.

[139] Malay K Pakhira, Sanghamitra Bandyopadhyay, and Ujjwal Maulik. Validity index for crisp and fuzzy clusters. *Pattern recognition*, 37(3):487–501, 2004.

[140] Roger D Peng. Reproducible research in computational science. *Science (New York, Ny)*, 334(6060):1226, 2011.

[141] Roger D Peng, Francesca Dominici, and Scott L Zeger. Reproducible epidemiologic research. *American journal of epidemiology*, 163(9):783–789, 2006.

[142] Bjoern Peters, Steven E Brenner, Edwin Wang, Donna Slonim, and Maricel G Kann. Putting benchmarks in their rightful place: the heart of computational biology. *PLoS computational biology*, 14(11), 2018.

[143] Florian Prinz, Thomas Schlange, and Khusru Asadullah. Believe it or not: how much can we rely on published data on potential drug targets? *Nature reviews Drug discovery*, 10(9):712–712, 2011.

[144] Weiliang Qiu and Harry Joe. Generation of Random Clusters with Specified Degree of Separation. *J. Classification*, 23(2):315–334, 2006.

[145] Weiliang Qiu and Harry Joe. *clusterGeneration: random cluster generation (with specified degree of separation)*, 2013. URL `http://CRAN.R-project.org/package=clusterGeneration`. R package version 1.3.1.

[146] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL `http://www.R-project.org/`.

[147] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[148] Franck Rapaport, Raya Khanin, Yupu Liang, Mono Pirun, Azra Krek, Paul Zumbo, Christopher E Mason, Nicholas D Socci, and Doron Betel. Comprehensive evaluation of differential gene expression analysis methods for rna-seq data. *Genome biology*, 14(9):3158, 2013.

[149] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M Quiroz. Internal versus external cluster validation indexes. *International Journal of computers and communications*, 5(1):27–34, 2011.

[150] Alejandro Reyes and Patrick Kimes. *SummarizedBenchmark: Classes and methods for performing benchmark comparisons*, 2019. R package version 2.0.1.

[151] David J. Rogers and Taffee T. Tanimoto. A computer program for classifying plants. *Science*, 132(3434):1115–1118, 1960. doi: 10.1126/science.132.3434.1115. URL `http://www.sciencemag.org/content/132/3434/1115.short`.

[152] Volker Roth, Tilman Lange, Mikio Braun, and Joachim Buhmann. A resampling approach to cluster validation. In *In Intl. Conf. on Computational Statistics*, pages 123–128, 2002.

[153] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(0):53–65, 1987.

[154] Peter J Rousseeuw and L Kaufman. Finding groups in data. *Hoboken: Wiley Online Library*, 1, 1990.

[155] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, PBC, Boston, MA, 2020. URL `http://www.rstudio.com/`.

[156] Wouter Saelens, Robrecht Cannoodt, and Yvan Saeys. A comprehensive evaluation of module detection methods for gene expression data. *Nature communications*, 9(1):1–12, 2018.

[157] Wouter Saelens, Robrecht Cannoodt, Helena Todorov, and Yvan Saeys. A comparison of single-cell trajectory inference methods. *Nature biotechnology*, 37(5):547–554, 2019.

[158] Sebastian Schmidtlein, Lubomir Tichy, Feilhauer Hannes, and Faude Ulrike. A brute force approach to vegetation classification. *Journal of Vegetation Science*, 21(6):1162–1171, 2010. doi: 10.1111/j.1654-1103.2010.01221.x.

[159] A Schmutz and J. Jacques & C. Bouveyron. *funHDDC: Univariate and Multivariate Model-Based Clustering in Group-Specific Functional Subspaces*, 2019. URL `https://CRAN.R-project.org/package=funHDDC`. R package version 2.3.0.

[160] Jochen Schneider, Alexander Hapfelmeier, Sieglinde Thöres, Andreas Obermeier, Christoph Schulz, Dominik Pförringer, Simon Nennstiel, Christoph Spinner, Roland M Schmid, Hana Algül, et al. Mortality risk for acute cholangitis (mac): a risk prediction model for in-hospital mortality in patients with acute cholangitis. *BMC gastroenterology*, 16(1):15, 2016.

[161] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[162] Gideon Schwarz. Estimating the dimension of a model. *Ann. Statist.*, 6(2): 461–464, 03 1978. doi: 10.1214/aos/1176344136. URL `https://doi.org/10.1214/aos/1176344136`.

[163] Luca Scrucca, Michael Fop, Thomas Brendan Murphy, and Adrian E. Raftery. mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *The R Journal*, 8(1):205–233, 2016. URL `https://journal.r-project.org/archive/2016-1/scrucca-fop-murphy-etal.pdf`.

[164] Alexander Sczyrba, Peter Hofmann, Peter Belmann, David Koslicki, Stefan Janssen, Johannes Dröge, Ivan Gregor, Stephan Majda, Jessika Fiedler, Eik Dahms, et al. Critical assessment of metagenome interpretation—a benchmark of metagenomics software. *Nature methods*, 14(11):1063–1071, 2017.

[165] Leming Shi, Laura H Reid, Wendell D Jones, Richard Shippy, Janet A Warrington, Shawn C Baker, Patrick J Collins, Francoise De Longueville, Ernest S Kawasaki, Kathleen Y Lee, et al. The microarray quality control (maqc) project shows inter-and intraplatform reproducibility of gene expression measurements. *Nature biotechnology*, 24(9):1151, 2006.

[166] Leming Shi, Gregory Campbell, Wendell D Jones, Fabien Campagne, Zhining Wen, Stephen J Walker, Zhenqiang Su, Tzu-Ming Chu, Federico M Goodsaid, Lajos Pusztai, et al. The microarray quality control (maqc)-ii study of common practices for the development and validation of microarray-based predictive models. *Nature biotechnology*, 28(8):827, 2010.

[167] Fernanda B Silva, Rafael de O Werneck, Siome Goldenstein, Salvatore Tabbone, and Ricardo da S Torres. Graph-based bag-of-words for classification. *Pattern Recognition*, 74:266–285, 2018.

[168] Mark Smolkin and Debashis Ghosh. Cluster stability scores for microarray data in cancer studies. *BMC bioinformatics*, 4(1):36, 2003.

[169] Robert R Sokal, Peter HA Sneath, et al. Principles of numerical taxonomy. *Principles of numerical taxonomy.*, 1973.

[170] Charlotte Soneson and Mauro Delorenzi. A comparison of methods for differential expression analysis of rna-seq data. *BMC bioinformatics*, 14(1):91, 2013.

[171] Charlotte Soneson and Mark D Robinson. Bias, robustness and scalability in single-cell differential expression analysis. *Nature methods*, 15(4):255, 2018.

[172] Azad Soni. Clustering with gaussian mixture model, 2021. URL `https://medium.com/clustering-with-gaussian-mixture-model/clustering-with-gaussian-mixture-model-c695b6cd60da`.

[173] R Grant Steen. Retractions in the scientific literature: is the incidence of research fraud increasing? *Journal of medical ethics*, pages jme–2010, 2010.

[174] Douglas Steinley. K-means clustering: a half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1):1–34, 2006.

[175] Victoria C Stodden. Reproducible research: Addressing the need for data and code sharing in computational science. *Computing in Science & Engineering*, 12(5):8–12, 2010.

[176] Zhenqiang Su, Paweł P Łabaj, Sheng Li, Jean Thierry-Mieg, Danielle Thierry-Mieg, Wei Shi, Charles Wang, Gary P Schroth, Robert A Setterquist, John F Thompson, et al. A comprehensive assessment of rna-seq accuracy, reproducibility and information content by the sequencing quality control consortium. *Nature biotechnology*, 32(9):903, 2014.

[177] Bo Tang and Haibo He. A local density-based approach for outlier detection. *Neurocomputing*, 241:171 – 180, 2017. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2017.02.039. URL http://www.sciencedirect.com/science/article/pii/S0925231217303302.

[178] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition.*, volume 4th ed. Academic Press, 2009. ISBN 9781597492720.

[179] Robert Tibshirani and Guenther Walther. Cluster Validation by Prediction Strength. *Journal of Computational and Graphical Statistics*, 14(3):511–528, 2005.

[180] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[181] Robert J Tibshirani and Bradley Efron. An introduction to the bootstrap. *Monographs on statistics and applied probability*, 57:1–436, 1993.

[182] George C. Tseng and Wing H. Wong. Tight Clustering: A Resampling-Based Approach for Identifying Stable and Tight Patterns in Data. *Biometrics*, 61: 10–16, March 2006.

[183] Marcia Urban, Martin Klemm, Kay Olaf Ploetner, and Mirko Hornung. Airline categorisation by applying the business model canvas and clustering algorithms. *Journal of Air Transport Management*, 71:175–192, 2018.

[184] Raffaele Giancarlo Utro, Davide Scaturro, and Filippo. Computational cluster calidation for microarray data analysis: experimental assessment of Clest, Consensus Clustering, Figure of Merit, Gap Statistics and Model Explorer. *BMC Bioinformatics*, 9(462), 2008.

[185] Iven Van Mechelen, Anne-Laure Boulesteix, Rainer Dangl, Nema Dean, Isabelle Guyon, Christian Hennig, Friedrich Leisch, and Douglas Steinley. Benchmarking in cluster analysis: A white paper. *arXiv preprint arXiv:1809.10496*, 2018.

[186] Zeev Volkovich, Zeev Barzily, and L Morozensky. A statistical model of cluster stability. *Pattern Recognition*, 41(7):2174–2188, 2008.

[187] Ulrike von Luxburg. Clustering stability: An overview. *Foundations and Trends® in Machine Learning*, 2(3):235–274, 2009. ISSN 1935-8237. doi: 10.1561/2200000008. URL http://dx.doi.org/10.1561/2200000008.

[188] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[189] Lukas M Weber and Mark D Robinson. Comparison of clustering methods for high-dimensional single-cell flow and mass cytometry data. *Cytometry Part A*, 89(12):1084–1096, 2016.

[190] Lukas M Weber, Malgorzata Nowicka, Charlotte Soneson, and Mark D Robinson. diffcyt: Differential discovery in high-dimensional cytometry via high-resolution clustering. *Communications biology*, 2(1):1–11, 2019.

[191] Lukas M Weber, Wouter Saelens, Robrecht Cannoodt, Charlotte Soneson, Alexander Hapfelmeier, Paul P Gardner, Anne-Laure Boulesteix, Yvan Saeys, and Mark D Robinson. Essential guidelines for computational method benchmarking. *Genome biology*, 20(1):125, 2019.

[192] Matthew T Weirauch, Atina Cote, Raquel Norel, Matti Annala, Yue Zhao, Todd R Riley, Julio Saez-Rodriguez, Thomas Cokelaer, Anastasia Vedenko, Shaheynoor Talukder, et al. Evaluation of methods for modeling transcription factor sequence specificity. *Nature biotechnology*, 31(2):126, 2013.

[193] Eric W Weisstein. *CRC concise encyclopedia of mathematics*. CRC press, 2002.

[194] Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

[195] Hadley Wickham. *Advanced r*. CRC press, 2019.

[196] Hadley Wickham, Winston Chang, et al. ggplot2: An implementation of the grammar of graphics. *R package version 0.7, URL: http://CRAN. R-project. org/package= ggplot2*, 3, 2008.

[197] Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. *dplyr: A Grammar of Data Manipulation*, 2018. URL `https://CRAN.R-project.org/package=dplyr`. R package version 0.7.6.

[198] Wilkerson, Matthew D., Hayes, and D. Neil. Consensusclusterplus: a class discovery tool with confidence assessments and item tracking. *Bioinformatics*, 26(12):1572–1573, 2010. URL `http://bioinformatics.oxfordjournals.org/content/26/12/1572.abstract`.

[199] Leland Wilkinson. The grammar of graphics. In *Handbook of Computational Statistics*, pages 375–414. Springer, 2012.

[200] Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL `https://yihui.org/knitr/`. ISBN 978-1498716963.

[201] Lei Xu. Bayesian ying–yang machine, clustering and number of clusters. *Pattern Recognition Letters*, 18(11):1167–1178, 1997.

[202] Qin Xu, Qiang Zhang, Jinpei Liu, and Bin Luo. Efficient synthetical clustering validity indexes for hierarchical clustering. *Expert Systems with Applications*, page 113367, 2020.

[203] Rui Xu and Don Wunsch. *Clustering*, volume 10. John Wiley & Sons, 2008.

[204] Rui Xu and Donald C Wunsch. Clustering algorithms in biomedical research: a review. *IEEE reviews in biomedical engineering*, 3:120–154, 2010.

[205] Menno Yap, Ding Luo, Oded Cats, Niels van Oort, and Serge Hoogendoorn. Where shall we sync? clustering passenger flows to identify urban public transport hubs and their key synchronization priorities. *Transportation Research Part C: Emerging Technologies*, 98:433–448, 2019.

[206] Tengfei Yin, Dianne Cook, and Michael Lawrence. ggbio: an r package for extending the grammar of graphics for genomic data. *Genome biology*, 13 (8):R77, 2012.

[207] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM Sigmod Record*, 25 (2):103–114, 1996.

[208] Siyuan Zheng. Benchmarking: contexts and details matter. *Genome biology*, 18(1):129, 2017.

[209] Xiaobei Zhou, Helen Lindsay, and Mark D Robinson. Robustly detecting differential expression in rna sequencing data using observation weights. *Nucleic acids research*, 42(11):e91–e91, 2014.

List of Figures

List of Tables

A. Prototypical Implementation of a Benchmarking Framework in R

```
setClass("benchmarkObject",
  representation(
    methods = "list",
    data = "list",
    criteria = "list",
    trueLabels = "list",
    models = "list",
    validation = "list")
)

findFunc <- function(fun) {
  objects <- ls(envir = environment(fun))
  for (i in objects) {
    if (identical(fun, get(i, envir = environment(fun)))) {
      return(i)
    }
  }
}

combineThings <- function(items){

  require(magrittr)
  items <- str_trim(items)
  objlist <- list()

  for(i in 1:length(items)){

    if(str_detect(items[i], "\\(") == T){
      objlist[[i]] <-  getSetup(items[i])
    }
    else if(class(eval(parse(text=items[i]))) == "list" && class(
        eval(parse(text=items[i+1]))) == "data.frame"){
      dat <- eval(parse(text=items[i]))
      d <- as.data.frame(eval(parse(text=items[2])))
      dat <- list.append(dat, d)
      names(dat)[length(dat)] <- items[2]
      objlist <- dat
      break
```

```
38      }
39      else if(class(eval(parse(text=items[i]))) == "list" && str_
            detect(items[i+1], "\\(") == T){
40        funcs <- eval(parse(text=items[i]))
41        f <- getSetup(items[2])
42        funcs <- list.append(funcs, f)
43        objlist <- funcs
44        break
45      }
46      else if(is.function(items[i] %>% parse(text=.) %>% eval) == T)
            {
47      f <- items[i] %>% parse(text=.) %>% eval
48      objlist[[i]] <- formals(f)
49      names(objlist)[i] <- findFunc(f)
50      }
51      else if(items[i] == "NULL"){
52        break
53      }
54      else {
55        objlist[[i]] <- eval(parse(text=items[i]))
56        names(objlist)[i] <- items[i]
57      }
58    }
59    objlist
60  }
61
62  pipe <- function(){
63    function(lhs, rhs){
64      parent <- parent.frame()
65      env <- new.env(parent = parent)
66      chain_parts <- splitExpr(match.call(), env)
67      combineThings(chain_parts)
68    }
69  }
70
71  splitExpr <- function(expr, env){
72    str <- as.character(expr)
73    c(unlist(strsplit(str[2], split = "%+%", fixed = T)), str[3])
74  }
75
76  `%+%` <- pipe()
77
78  getSetup <- function(a){
79
80    require(tidyverse)
81    require(rlist)
82    require(stringr)
83
84    p <- str_sub(a, as.integer(str_locate(a, "\\(")[1,1])+1, -2)
```

```
  method <- str_split(a, "\\(")[[1]][1]
  params <- str_split(p, ",[:blank:]")
  paramlist <- str_split(params[[1]], "=")
  paraml <- list()
  gridlist <- list()

  for(i in 1:length(paramlist)){
    paramlist[[i]] <- str_trim(paramlist[[i]])
    paraml <- list.append(paraml, paramlist[[i]][2])
    names(paraml)[i] <- paramlist[[i]][1]

    paraml[[i]] <- eval(parse(text=paraml[[i]]))
    if(class(paraml[[i]]) == "data.frame" || class(paraml[[i]]) ==
        "matrix")
      paraml[[i]] <- nest(.data=paraml[[i]], data=everything())
    if(class(paraml[[i]]) == "list" && class(paraml[[i]][[1]]) ==
       "data.frame"){
      for(j in 1:length(paraml[[i]])){
        paraml[[i]][[j]] <- nest(.data=paraml[[i]][[j]], data=
           everything())
      }
    }
    gridlist <- list.append(gridlist, paraml[[i]])
  }

  fullgrid <- cross(gridlist)
  parameter_formalsgrid <- list()
  for(i in 1:length(fullgrid)){
    names(fullgrid[[i]]) <- names(paraml)
    parameter_formalsgrid <- list.append(parameter_formalsgrid,
       formals(method))
    if(any(names(parameter_formalsgrid[[i]]) == "..."))
      parameter_formalsgrid[[i]]$... <- NULL
    pm <- fullgrid[[i]]

    for(j in 1:length(pm)){
      if(any(class(pm[[j]]) == "data.frame"))
      pm[[j]] <- as.data.frame(unlist(pm[[j]], recursive = F))
      if(any(class(pm[[j]]) == "list"))
      pm[[j]] <- as.data.frame(unlist(pm[[j]], recursive = F))
    }
    fullgrid[[i]] <- pm
  }
  return(list(method = method, params = fullgrid))
}

runBenchmark <- function(obj){
  f <- obj@methods
```

```
129   indexnames <- unlist(obj@criteria)[names(unlist(bench@criteria))
          == "params.crit"]
130   datanames <- names(obj@data)
131   methodnames <- vector()
132
133   for(i in 1:length(f)){
134     obj@validation[[i]] <- as.data.frame(matrix(ncol = length(f[[i
          ]]$params[[1]]) + length(indexnames), nrow=length(f[[i]]$
          params)))
135     names(obj@validation[[i]]) <- c(names(f[[i]]$params[[1]]),
          indexnames)
136     methodnames <- append(methodnames, f[[i]]$method)
137
138     for(j in 1:length(f[[i]]$params)){
139       m <- do.call(f[[i]]$method, f[[i]]$params[[j]])
140       obj@models <- list.append(obj@models, m)
141
142       for(h in 1:length(obj@data)){
143         for(s in 1:length(f[[i]]$params[[j]])){
144           x <- suppressWarnings(try(obj@data[[h]] == f[[i]]$params
                [[j]][[s]], silent=T))
145           if(class(x) == "try-error" || all(x) == FALSE) {
146             next
147           }
148           else {
149             x <- all(x)
150             paramname_dat <- names(f[[i]]$params[[j]])[s]
151             break
152           }
153         }
154       if(all(x))
155         break
156       }
157
158       if(x){
159         setname <- datanames[h]
160         set <- obj@data[[h]]
161       }
162
163       s4flag = FALSE
164
165       if(isS4(m)){
166         s4flag = TRUE
167         n <- slotNames(m)
168       }
169
170       for(b in 1:length(f[[i]]$params[[j]])){
171         for(w in 1:length(m)){
172           if(s4flag==TRUE){
```

```
173             gr <- suppressWarnings(try(f[[i]]$params[[j]][[b]] ==
                    length(unique(as.integer(eval(parse(text=str_c("m@"
                    , n[w])))))) && is.vector(eval(parse(text=str_c("m@
                    ", n[w])))) == TRUE && length(eval(parse(text=str_c
                    ("m@", n[w])))) == nrow(set), silent=T))
174           } else {
175             gr <- suppressWarnings(try(f[[i]]$params[[j]][[b]] ==
                      length(unique(as.integer(m[[w]]))) && is.vector(m
                    [[w]]) == TRUE && length(m[[w]]) == nrow(set),
                    silent=T))
176           }
177           if(class(gr)=="try-error" || gr == FALSE)
178             next
179           else if(gr==TRUE){
180             if(s4flag==TRUE)
181             clust <- eval(parse(text=str_c("m@", n[w])))
182             else
183             clust <- m[[w]]
184             break
185           }
186         }
187         if(class(gr)=="try-error" || gr == FALSE){
188           next
189         }
190         else if(gr==TRUE)
191           break
192       }
193
194       if(s4flag){
195         for(k in 1:length(n)){
196           y=suppressWarnings(try(length(eval(parse(text=str_c("m@"
                , n[k])))) == nrow(set)  && length(unique(eval(parse(
                text=str_c("m@", n[k]))))) == length(unique(clust)),
                silent=T))
197           if(y == TRUE)
198             labels <- eval(parse(text=str_c("m@", n[k])))
199           else if(class(y) == "try-error")
200             next
201         }
202       } else{
203         for(k in 1:length(m)){
204           y <- suppressWarnings(try(length(m[[k]]) == nrow(set) &&
                  length(unique(m[[k]])) == length(unique(clust)),
                silent=T))
205           if(y == TRUE){
206             labels <- m[[k]]
207           }
208         }
209       }
```

```
210
211       datindex <- which(names(f[[i]]$params[[j]])==paramname_dat)
212       values <- unlist(f[[i]]$params[[j]][-datindex])
213       obj@validation[[i]][j,1] <- setname
214       obj@validation[[i]][j,2:(2+length(values)-1)] <- values
215
216       require(clusterCrit)
217       for(p in 1:length(obj@criteria)){
218         if(obj@criteria[[p]]$method =="intCriteria") {
219           freeRowIndex <- which(is.na(obj@validation[[i]][j,]))
220           intCalculatedIndices <- unlist(do.call(obj@criteria[[p]]
                $method, list(as.matrix(set), as.integer(labels),
                unlist(obj@criteria[[p]]$params))))
221           obj@validation[[i]][j,freeRowIndex[1:length(
                intCalculatedIndices)]] <- intCalculatedIndices
222         }
223         else if(obj@criteria[[p]]$method =="extCriteria"){
224           freeRowIndex <- which(is.na(obj@validation[[i]][j,]))
225           extCalculatedIndices <- unlist(do.call(obj@criteria[[p]]
                $method, list(as.integer(labels), as.integer(
                obj@trueLabels[[match(setname, datanames)]]), unlist(
                obj@criteria[[p]]$params))))
226           obj@validation[[i]][j,freeRowIndex[1:length(
                extCalculatedIndices)]] <- extCalculatedIndices
227         }
228       }
229     }
230   }
231   names(obj@validation) <- methodnames
232   return(obj)
233 }
```