Master thesis Business Analytics

Time series clustering

Pjotr Roelofsen

(2516177)

March 28, 2018

Supervisors:

prof. dr. S. Bhulai (VU supervisor)

dr. M. Hoogendoorn (VU co-reader)

MSc. M. Hoenderdos (PwC supervisor)

Vrije Universiteit Amsterdam
Faculty of Science
De Boelelaan 1081a
1081 HV Amsterdam

PricewaterhouseCoopers Advisory N.V.
Consulting - Data Analytics
Thomas R. Malthusstraat 5
1066 JR Amsterdam

Preface

This thesis is written for the final part of the Master's program in Business Analytics at the Vrije Universiteit in Amsterdam. This is a two-year multidisciplinary program, aimed at improving business processes by applying a combination of methods based on mathematics, computer science and business management. The Master's degree is concluded with a six-month individual internship at a company. My internship took place at the Data Analytics team of the consulting department of PwC. The research that I conducted at PwC on time series clustering is presented in this thesis.

Over the past six months I have had a great time at PwC and I would like to thank everyone in the Data Analytics team for letting me feel at home. I want to thank Maurice Hoenderdos, my supervisor at PwC, in particular for his guidance during my internship period. I found the meetings and constructive feedback very helpful for determining which steps to take during this research. Furthermore, I would like to thank Jop Daalmans for providing feedback on my thesis and helping me find the data set that I used during my practical research.

From the VU, I would like to thank Sandjai Bhulai, my first supervisor, for thinking along with my research and also for giving feedback on my thesis. I would also like to thank Mark Hoogendoorn for being the second reader of my thesis.

Pjotr Roelofsen - Amsterdam, March 2018

Executive summary

This thesis provides a comparative study between the different methods that are available for time series clustering. We first perform a literature study in which we explain the different methods in clustering and discuss their performance, time complexity and applicability. The different methods are then put into practice by applying them on an actual time series data set from one of PwC's clients.

Clustering is the practice of finding hidden patterns or similar groups in data. It is one of the most common methods for unsupervised learning, where a classification is given to every data entry without predefining the different classes. Cluster analysis can be divided into three different parts: determining a measure to quantify the similarity between observations, choosing which method to use for obtaining the clustering and selecting the desired number of clusters.

Many real-world applications nowadays generate and store time series data. Clustering can be applied on these time series to gain insight into the data. However, as it appears from previous research, general clustering methods, such as k-means, are not designed for time series data and therefore may not perform well. This is mainly caused by the fact that most general clustering methods are built around the Euclidean distance, which does not seem to be a good measure for time series data.

The literature study in this thesis resulted in a comparison between time series distance measures, a comparison of clustering algorithms and an evaluation on how to select the "optimal" number of clusters. After putting these methods in practice, the differences between the theoretical and practical results are discussed.

In the first part of our literature study (Section 2), we discuss and compare seven different distance measures. An overview of the methods that we discuss can be found in Table 1. The seven distance measures can be divided into three different categories: lock-step, elastic and feature-based distance measures. The two lock-step measures we discuss are often applied when clustering static (non time series) data, however they are less suitable for time series data. This is because they are limited in handling noise, time shifts and time warping. Elastic measures overcome these limitations, but at the cost of a higher time complexity. If the time complexity of elastic (and lock-step) measures becomes too high, feature-based measures can be applied. These measures decompose the time series into different features (sine waves, wavelets and strings in our case), after which features with a low significance are dropped. This causes a reduction in both dimensionality (and thus time complexity) and noise. At the end of Section 2, we present a flow-chart on the steps to take when determining which distance measure to use in time series clustering.

Table 1: Overview of the seven distance measures that are discussed in Section 2.

Lock-step	**Elastic**	**Feature-based**
Minkowski ($\forall p$)	Dynamic Time Warping (DTW)	Discrete Fourier Transform (DFT)
Pearson correlation	Longest Common SubSequence (LCSS)	Discrete Wavelet Transform (DWT)
		Symbolic Aggregate approXimation (SAX)

The second part of our literature study examines clustering methods (Section 3). While the distance measures we discuss in Section 2 are mainly focused on time series, the methods that are discussed in Section 3 are applicable for any type of clustering. We discuss two types of clustering methods: hierarchical clustering and partitional clustering.

In hierarchical clustering, a nested hierarchy of clusters is built, which can be organized as a tree. The advantage of hierarchical clustering is that it does not require the number of clusters to be predefined. A disadvantage is, however, that hierarchical clustering requires the distance matrix of all pairs of observations to be calculated, which can be a time-consuming operation for large data sets.

Partitional clustering is a type of clustering where all observations in the data are partitioned into k different clusters. Here, the number k has to be specified beforehand. We discuss four different partitional clustering algorithms: k-means, k-medoids, CLARA and CLARANS.

Both k-means and k-medoids create clusterings based on all observations in the data, where the distance between cluster centers and the observations in clusters is minimized. The difference is that in k-means, cluster centers are taken to be the mean of all observations in the cluster, while in k-medoids the centers are actual observations in the data. This makes k-medoids more robust to outliers and noise.

CLARA (Clustering LARge Applications) and CLARANS (Clustering Large Applications based on RANdomized Search) are both variations of k-medoids that use sampling methods to handle large data sets. According to the literature, CLARANS generally outperforms CLARA, but at the cost of a higher time complexity.

The third part of our literature study, Section 4, discusses seven different indices that can be used for determining the "optimal" number of clusters. These indices are all, in some way, built around minimizing the within cluster distance and/or maximizing the between cluster distance. It is advised to apply multiple indices and use the majority rule to determine the "optimal" number of clusters in a cluster analysis. If there is no or barely any agreement between different indices, this might indicate that there is no significant cluster structure in the data.

In the section about our practical research, Section 5, we compare the different measures, methods and indices that we discuss in the literature study by applying them on an actual client time series data set. Quantifying the performance of distance measures and clustering methods is a difficult task, since this performance highly depends on the data that is clustered and the goal of the clustering. We therefore setup an experiment with a clear clustering goal: forecasting. In this setup we cluster 2,381 different time series based on two years of weekly sales data, after which we apply ARIMA forecasting on these clusters to forecast the sales of the next 1.5 years. This is all performed in R. The total difference between the forecasted sales of each cluster and the actual sales of the individual time series in each cluster is taken to be the benchmark to quantify clustering quality in our experiment.

In our practical research, there appeared to be a significant difference in time complexity when determining the distance matrices. While it took lock-step and feature-based methods only a few seconds to compute the whole distance matrix, both elastic measures took an hour or more to compute the corresponding distance matrix. Time complexity also played a role when determining the "optimal" number of clusters, as two out of the seven indices that we applied could not find the number of clusters due to their running time being too long. The remaining five indices did not tend to agree on an "optimal" number of clusters, indicating that there is no strong clustering structure in the data. To still decide on a reasonable number of clusters, the clustering outcomes for different numbers of clusters were visually compared.

The best clustering method, in our experiment, turns out to be the CLARANS algorithm in combination with the Euclidean distance. The clustering and forecasting that resulted from this method can be found in Figure 1. This result is surprising, since, according to the literature, it is expected that the Euclidean distance is outperformed by the elastic distance measures. However, no general conclusions can be drawn from this result, because we have only shown that this method performs best in our setup with this particular data set.

In conclusion, choosing a distance measure is the most important step in time series clustering. This decision is a trade-off between time complexity and performance and we propose to use the flow-chart that is presented at the end of Section 2 when making this decision. There is no 'best' clustering algorithm, however we suggest trying sampling methods like CLARA or CLARANS when clustering large data sets. At last, when determining the number of clusters, it is advised to apply multiple different indices and apply the majority rule. This may be a difficult task, due to time complexity and a possible lack of agreement between the indices.

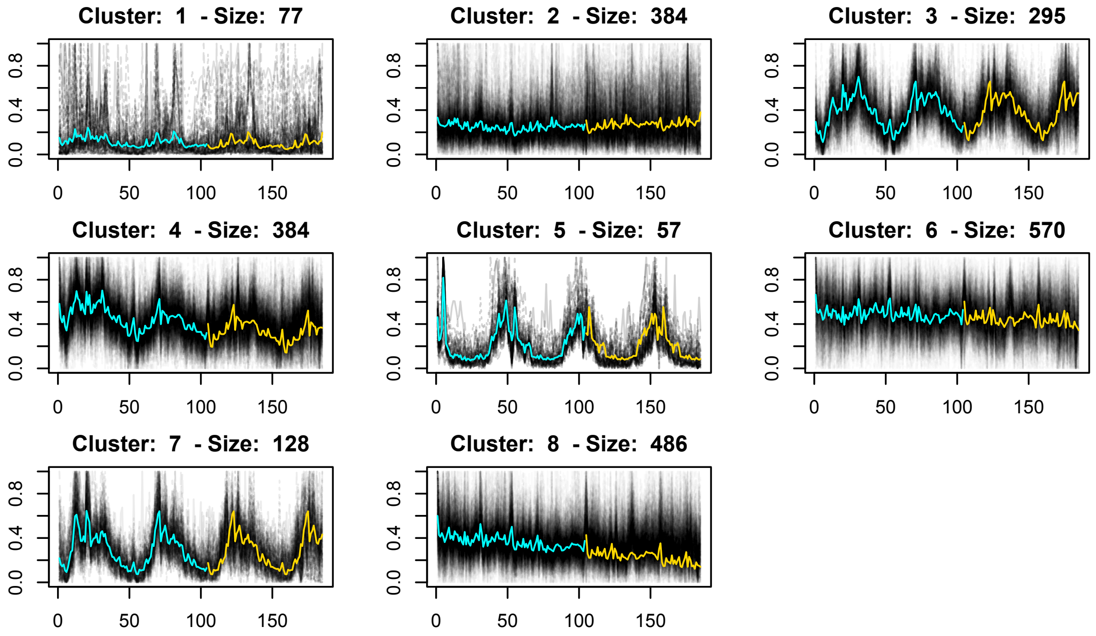


Figure 1: Visualization of the eight clusters that are obtained when applying the CLARANS clustering algorithm with Euclidean distance on our time series data set. This clustering assignment is the best in our experiments, according to the benchmark value we defined. The black lines in the background represent the time series within each cluster. The cyan line indicates the cluster mean of the first 104 points in time, which is used to obtain the ARIMA forecast. The ARIMA forecast is plotted by the yellow line. The size (number of time series) of each cluster is indicated above the plots.

Contents

1 Introduction

Clustering is the practice of finding hidden patterns or similar groups in data. It is one of the most common methods for unsupervised learning, where a classification is given to every data entry without predefining the different classes. There are many practical applications of clustering, for example in the fields of pattern recognition, bioinformatics and marketing.

Cluster analysis can be divided into three different parts: determining a measure to quantify the similarity between observations, choosing which method to use for obtaining the clustering and selecting the desired number of clusters. The order in which these parts are executed depends on the clustering method that is used. Some methods require the number of clusters as input, while other methods allow the user to decide on this number after the clustering is generated.

The majority of clustering analyses in previous research is performed on static data, which is data where the features do not change in time. However, with the increasing power of data storage during the last decade(s), nowadays many real-world applications store and keep data for a long time. As a consequence, time series data is generated in many different fields. Clustering can be applied to these time series to gain insight into the data. For example, a retailer could cluster the sales time series of its products to determine which products show similar sales behavior. However, as it appears from previous research, general clustering methods, such as k-means, are not designed for time series data and therefore may not perform well.

To overcome the shortcomings of general clustering approaches when clustering time series, several techniques are proposed in the literature. In this thesis, we present a comparative study between the different (time series) clustering techniques that are available. We compare these techniques based on time complexity, applicability and effectiveness. Besides conducting a literature study, we also apply the different techniques on an actual data set to test their performance. The main research question of this master thesis is formulated as follows:

"Which clustering methods should be used for time series clustering and how do they perform in practice?"

When comparing time series, there are two categories: whole sequence matching and subsequence matching. In whole sequence matching, whole (complete) time series of equal length are compared. In subsequence matching, one compares a given a subsequence with the subsequences of other (whole) sequences. The remainder of this thesis will only focus on whole sequence clustering.

In Section 2, we discuss seven different distance measures. Here we address the limitations of the Euclidean distance when comparing time series and present alternatives that overcome these limitations. We also discuss dimensionality reduction methods for handling large numbers of time series data.

Section 3 addresses the different methods that are available for clustering. We address two main approaches: hierarchical clustering and partitional clustering. For both approaches we discuss several algorithms for obtaining a clustering. These algorithms also include sampling methods, which are built for handling large data sets.

When performing clustering analysis, at some point the number of clusters has to be determined.

This can be done by using prior knowledge of the data and by estimation indices that calculate the "optimal" number of clusters based on a statistic. In Section 4, we discuss seven different indices for determining the optimal number of clusters and compare them with each other.

In Section 5, we discuss the practical research that is conducted and present the results. Here we apply several techniques from the literature on an actual sales time series data set. At last, a discussion on both the theoretical and practical findings will be presented in Section 6.

2 Distance measures

Clustering is the practice of finding hidden patterns or similar groups in data. To determine whether observations (i.e. time series) in the data are similar, however, one must first decide on a measure to quantify this similarity. For this purpose, many different similarity (distance) measures are proposed in the literature. This section will give an overview of the most commonly used and most promising distance measures. First we introduce the notation we use and some general background on distance measures, such as the different categories of distance measures and the time complexity of applying them. In Sections 2.1, 2.2 and 2.3, we elaborate on a number of distance measures from different categories. Ensemble schemes that combine different distance measures are then examined in Section 2.4. At last, a comparison between the different distance measures is given in Section 2.5.

Notation

We use the notation $d(x, y)$ to represent the distance between time series (observations) $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_m)$. Here, time series x and y can be seen as numerical vectors with dimensions n and m, such that $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. Note that m and n can thus differ in value. The value of $d(x, y)$ for every pair of x and y is a single non-negative number that depends on the distance measure that is used. The lower the value of $d(x, y)$, the closer the two observations are according to the chosen distance measure. The total number of time series will be indicated by N.

Time complexity

Most clustering methods require computing the distance or dissimilarity matrix, which contains the distance or (dis)similarity between each pair of observations. When determining the distance matrix for N different observations, a total of $\frac{N(N-1)}{2}$ distances have to be calculated. The required calculation time for determining a distance matrix thus grows with $O(N^2)$ and this is without taking into account the time complexity of the individual distance measures. These individual time complexities will be discussed in the next subsections. Due to the quadratic time complexity, determining the distance matrix for large sets of observations ($\gg 1000$) can be a time consuming operation.

Different categories of distance measures

Esling and Agon (2012) divided the different distance measures for time series into four categories:

1. Shape-based distances
    (a) Lock-step measures
    (b) Elastic measures
2. Feature-based distances
3. Edit-based distances
4. Structure-based distances

The first type of distance measures, shape-based distances, compares the overall shape of time series based on the actual (scaled) values of the time series. Two subcategories can be identified: lock-step measures and elastic measures. Lock-step measures require both time series to be of equal length ($n = m$) and compare time point i of time series x with time point i of time series y, whereas elastic measures are more flexible and allow one-to-many or one-to-none point matchings (Wang et al. (2010)). The second type of distance measures, feature-based distances, first extracts features from the time series and then measures the distance between these features.

Feature-based distances are often applied to obtain a reduction in both dimensionality and noise. The third category, edit-based distances, expresses the dissimilarity of two time series based on the minimum number of operations that are required to transform one series into the other series. In the last category, structure-based distances, the dissimilarity between time series is obtained by comparing higher level structures that are obtained by modelling or compressing the time series.

In this thesis, we only focus on shape-based and feature-based distance measures. This is because these measures occur the most in time series clustering literature and therefore we expect these methods to be most promising. Also, we conducted tests on small time series sets ($N = 500$) to get an idea of the performance of edit-based and structure-based distances. These tests indicate that both the running time and the obtained clusters for edit-based and structure-based distances tend to be similar or worse compared to shape-based and feature-based distance measures.

When clustering time series, it is demonstrated that lock-step distance measures, such as the ones we discuss in Subsection 2.1, are often outperformed by special time series distance measures (Aach and Church (2001), Bar-Joseph et al. (2002), Grabusts and Borisov (2009), Yi et al. (1998), Xi et al. (2006)). This is mainly caused by the fact that lock-step distance measures are sensitive to noise, scale and time shifts. Special elastic distance measures have been developed to overcome these problems. To give an example: where the Euclidean distance (lock-step) only compares time point x_i with time point y_i for every i, Dynamic Time Warping (elastic) also takes into account the surrounding points in time to allow for shifts in time.

In the next subsections, we discuss seven different distance measures. An overview of the different distance measures that we consider in this section can be found in Table 2. This set of distance measures is chosen to represent both a variety in distance measure categories as well as the most commonly used (and effective) distance measures according to the literature (Wang et al. (2010); Liao (2005); Mori et al. (2016a)). The distance measures are sorted by category and we will also discuss them in this order. In Subsection 2.5, we compare the seven distance measures based on time complexity, performance and other properties.

Table 2: Overview of the seven distance measures that we consider in this thesis.

Shape-based distances
Lock-step measures
Minkowski ($\forall p$)
Pearson correlation
Elastic measures
Dynamic Time Warping
Longest Common Subsequence
Feature-based distances
Discrete Fourier Transform
Discrete Wavelet Transform
SAX representation

2.1 Lock-step measures

In this subsection we examine two lock-step distance measures: the Minkowski distance and the Pearson correlation distance. As with all lock-step measures, these distance measures require both time series to be of equal length ($n = m$) and compare time point i of time series x with the same time point i of time series y. Note that we examine the lock-step distance measures from a time series perspective, but that these measures can also be used for non-time series clustering assignments. The only requirement is that all observations are numerical vectors of equal length.

2.1.1 Minkowski distance

The Minkowski distance is the L_p-norm of the difference between two vectors of equal length ($n = m$). It is defined as follows:

$$d_{min}(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}. \tag{1}$$

It is the generalization of the commonly used Euclidean distance ($p = 2$), Manhattan distance ($p = 1$) and Chebyshev distance ($p = \infty$). For clustering, usually only the Euclidean distance and Manhattan distance are considered. The formulas for those distance measures can be found in Equations (2) and (3). The time complexity of the Minkowski distance for all p is $O(n)$ and thus determining the distance matrix with this measure takes $O(nN^2)$ time.

$$\textbf{Euclidean distance:} \quad d_{euc}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{2}$$

$$\textbf{Manhattan distance:} \quad d_{man}(x, y) = \sum_{i=1}^{n} |(x_i - y_i)|. \tag{3}$$

2.1.2 Pearson correlation distance

The Pearson correlation distance takes into account the linear association between two vectors of variables. It does so by using the Pearson correlation coefficient, which is defined as follows:

$$\rho(x, y) = \frac{\text{Cov}(x, y)}{\sigma_x \sigma_y} = \frac{\mathbb{E}\left[(x - \mu_x)(y - \mu_y)\right]}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}}, \tag{4}$$

where μ_x and μ_y are the means of x and y and σ_x and σ_y are the standard deviations of x and y, respectively. Note that the coefficient requires $n = m$ and that it is invariant for scaling. The values of ρ lie within the range $[-1, 1]$, where $\rho = 1$ indicates a perfect positive relationship between x and y, $\rho = -1$ indicates a perfect negative relationship between x and y and $\rho = 0$ indicates no relationship between the two variables. For the other values that lie within this range, the strength of the relationship varies based on the value of ρ, where values closer to -1 and 1 indicate a stronger negative or positive relationship, respectively.

For a distance measure that takes into account the correlation between two vectors, it is desired to generate low distance values for positively correlated vectors, since these vectors are most similar. The Pearson correlation distance is therefore defined as:

$$d_{cor}(x, y) = 1 - \rho(x, y). \tag{5}$$

Note that this distance measure can take values in the range $[0, 2]$. The time complexity of the Pearson correlation distance is $O(n)$ and thus determining the distance matrix with this measure takes $O(nN^2)$ time.

Alternative correlation measures include Spearman's Rank and Kendall's Tau correlation coefficients. These coefficients indicate correlation based on rank, whereas the Pearson correlation coefficient is based on a linear relationship between two vectors. This makes Spearman's Rank and Kendall's Tau less sensitive to noise and outliers compared to the Pearson correlation coefficient (Fujita et al. (2009)). However, this comes with an increase in time complexity: $O(n \log n)$ for Spearman's Rank and $O(n^2)$ for Kendall's Tau (Xu et al. (2013)). Another alternative is a distance based on autocorrelation, which accounts for lags in time and has a time complexity of $O(n \log n)$.
We will not consider the alternative correlation measures mentioned above, due to the increase in time complexity and other lock-step (or elastic) distance measures generally outperforming correlation distance measures in time series clustering (Iglesias and Kastner (2013)).

2.2 Elastic measures

In this subsection we discuss two elastic distance measures: Dynamic Time Warping and Longest Common Subsequence. In contrast to lock-step distance measures, elastic distance measures allow one-to-many or one-to-none point matching. This makes it possible for elastic distance measures to warp in time and be more robust when it comes to, for example, handling outliers. A disadvantage, however, is that elastic distance measures generally come with an increase in time complexity.

2.2.1 Dynamic Time Warping (DTW)

Dynamic Time Warping is a time series distance measure that is introduced in the field of data mining to overcome some of the disadvantages of the Euclidean distance (Keogh and Ratanamahatana (2004)). It warps two sequences x and y non-linearly in time in order to cope with time deformations and varying speeds in time dependent data.

When determining the DTW distance between two time series, first an $(n \times m)$ local cost matrix (LCM) is calculated, where element (i, j) contains the distance between x_i and y_j. This distance is usually defined as the quadratic difference: $d(x_i, y_j) = (x_i - y_j)^2$. Next, a warping path $W = w_1, w_2, ..., w_K$ is determined, where $\max(n, m) \leq K \leq m+n-1$. This path traverses the LCM under three constraints:

1. **Boundary condition:** The path must start and end in the diagonal corners of the LCM: $w_1 = (1, 1)$ and $w_K = (n, m)$.
2. **Continuity:** Only adjacent elements in the matrix are allowed for steps in the path. This includes diagonal adjacent elements. So, if $w_q = (i, j)$, then w_{q+1} is either element $(i + 1, j)$, $(i, j+1)$ or $(i+1, j+1)$ for $q = 1, ..., K-1$ and $i = 1, ..., n-1$ and $j = 1, ..., m-1$.
3. **Monotonicity:** Subsequent steps in the path must be monotonically spaced in time. In the example at constraint 2 this can be observed by the fact that indices i and j must be non-decreasing in subsequent steps.

The total distance for path W is obtained by summing the individual elements (distances) of the LCM that the path traverses. To obtain the DTW distance, the path with minimum total distance is required. This path can be obtained by an $O(nm)$ algorithm that is based on dynamic programming (DP). The following DP recurrence can be used to find the path with minimum cumulative distance:

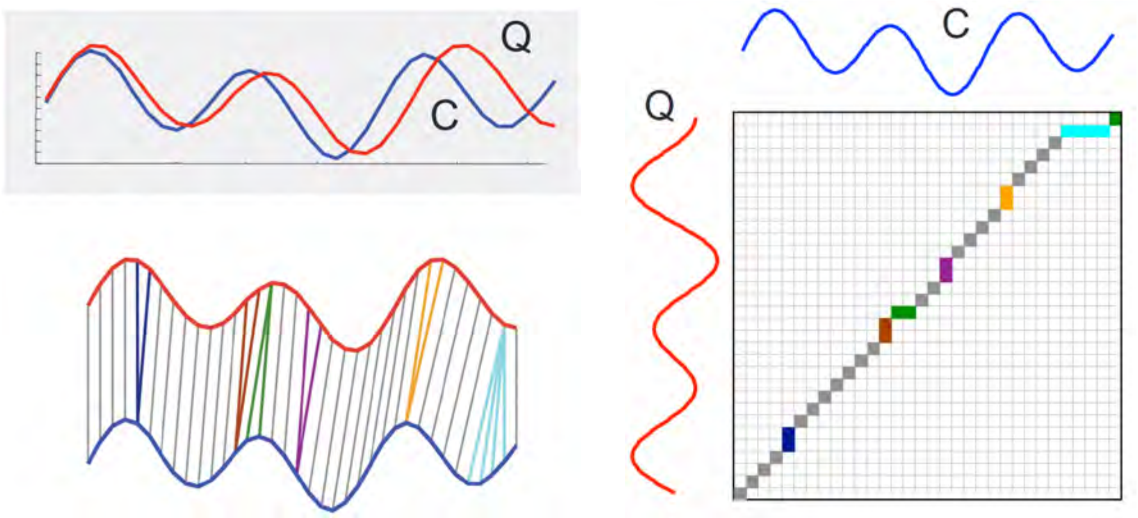
$$d_{cum}(i, j) = d(x_i, y_j) + \min\left\{d_{cum}(i-1, j-1), d_{cum}(i-1, j), d_{cum}(i, j-1)\right\}. \tag{6}$$

We now obtain the DTW distance by summing the elements of the path with minimum cumulative distance. In the literature, different scales of this sum are taken to be the DTW distance. We will use the definition from Gorecki (2017) and thus take the root of the sum:

$$d_{DTW}(x, y) = \min \sqrt{\sum_{k=1}^{K} w_k}, \tag{7}$$

where w_k is the distance that corresponds to the kth element of warping path W. Note that this distance is equal to the Euclidean distance for the case where $n = m$ and only the diagonal of the LCM is traversed. Furthermore, the DTW distance does not satisfy the triangle inequality, even when the local distance measure is a metric (Mico and Oncina (1998)).

An example of DTW can be found in Figure 2, where for two time series the minimum warping path through the LCM is shown together with the linked time points of the series.



(a) Time series C and Q and the linked points in time by DTW.

(b) Minimum warping path W through the LCM of time series C and Q.

Figure 2: Example of Dynamic Time Warping. Image taken from Keogh and Ratanamahatana (2005) and recolored.

A number of modifications to the DTW distance are proposed in the literature. These include Derivative Dynamic Time Warping (DDTW) and Weighted Dynamic Time Warping (WDTW). DDTW is proposed by Keogh and Pazzani (2001) and converts the time series into a series of first order differences. This avoids having a single point of one series to map to many points in another series, which can negatively impact the DTW distance. WDTW is proposed by Jeong et al. (2011) and uses a (logistic) weight function to add penalties to the LCM to favour reduced warping. This brings more balance between shape matching and alignment in time.

The time and space complexity of finding the DTW distance between two time series is $O(nm)$ and thus determining the distance matrix with this measure takes $O(nmN^2)$ time. This quadratic time complexity can make DTW a time consuming process, hence different methods are proposed in the literature to speed up the distance measure. These methods fall into three categories: adding constraints, abstracting the data and indexing.

Constraints are added by, for example, limiting the number of elements in the LCM that are evaluated. Two common constraints are the Sakoe-Chuba band (Sakoe and Chiba (1978)) and the Itakura Parallelogram (Itakura (1975)). When abstracting the data, the DTW algorithm is only run on a reduced representation of the data (Chu et al. (2002), Keogh and Pazzani (2000)). At last, indexing uses lower bounds to reduce the number of times the DTW algorithm has to be executed (Keogh and Ratanamahatana (2005), Kim et al. (2001)). We refer to the cited literature for further explanation of the techniques mentioned above.

It must be noted that all three categories of methods that speed up DTW may cause the resulting distance to deviate from the true DTW distance. Also, all three categories increase the speed of DTW by a constant factor, which still gives the process of determining the DTW distance a time complexity of $O(nm)$. To overcome this time complexity, Salvador and Chan (2007) introduced an approximation algorithm for DTW, called FastDTW, that has a time complexity of $O(\max\{n, m\})$. This approximation algorithm first reduces the dimensions of a time series by averaging adjacent pairs of points in time, then finds the minimum warping path on the reduced time series and at last refines the warping path through local adjustments by considering neighbouring cells in the original LCM.

Even though FastDTW has a linear time complexity, accurate approximations come with large constant factors in time complexity. This makes computing the FastDTW distance still considerably slower than, for example, computing the Euclidean distance. In the practical application of the distance measures in Section 5, we will therefore not use any of the alternative DTW methods.

2.2.2 Longest Common Subsequence (LCSS)

The LCSS similarity measure is based on the longest common subsequence problem, which aims at finding the longest subsequence that is common to two or more sequences. Here we define a subsequence to be a sequence that appears in the same relative order, but where the individual elements are not necessarily contiguous. Usually the problem is defined for discrete sequences, such as strings, where it searches for exact matches between sequences. To give an example, we consider two strings: $S_1 = ABCDGH$ and $S_2 = AEDFHR$. Here the longest common subsequence is ADH with size 3.

LCSS can also be applied to measure the similarity between two (or more) time series. Since time series usually contain real numbers, a match between two points in time is typically defined if the absolute difference between the two points is less than ϵ. The LCSS distance between two time series can be obtained by applying dynamic programming using the following recursion:

$$L(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + L(i-1, j-1) & \text{for } |x_i - y_j| < \epsilon \\ \max\{L(i-1,j), L(i,j-1)\} & \text{otherwise,} \end{cases} \tag{8}$$

where $1 \leq i \leq n$ and $1 \leq j \leq m$. The distance itself is now obtained by solving $L(n, m)$. We consider the scaled version of this distance in Equation (9), which scales between 0 and 1 for two time series of equal length. This scale is proposed by Ratanamahatana et al. (2005). Since the LCSS distance only regards similar points, it is robust to noise and outliers. Furthermore, Vlachos et al. (2002) have shown that LCSS does not satisfy the triangle inequality.

$$d_{LCSS}(x, y) = \frac{n + m - 2L(n,m)}{n + m}. \tag{9}$$

The time complexity of computing the LCSS distance is $O(nm)$. However, often a warping threshold δ is added to restrict the matching to a maximum difference in time ($|i-j| \leq \delta$). This improves the time complexity to $O((n + m)\delta)$ (Cassisi et al. (2012)). The recursion now becomes:

$$L(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + L(i-1, j-1) & \text{for } |x_i - y_j| < \epsilon \text{ and } |i-j| \leq \delta \\ \max\{L(i-1,j), L(i,j-1)\} & \text{otherwise.} \end{cases} \tag{10}$$

To give an example of the effects of ϵ and δ, we refer to Figure 3. Here we observe a minimum bounding envelope, which is created by the vertical boundaries of ϵ and the horizontal boundaries of δ. Time series points outside this envelope cannot be matched and the value of $L(n, m)$ is the sum of the number of points in time where the red line (time series y) lies within this envelope.

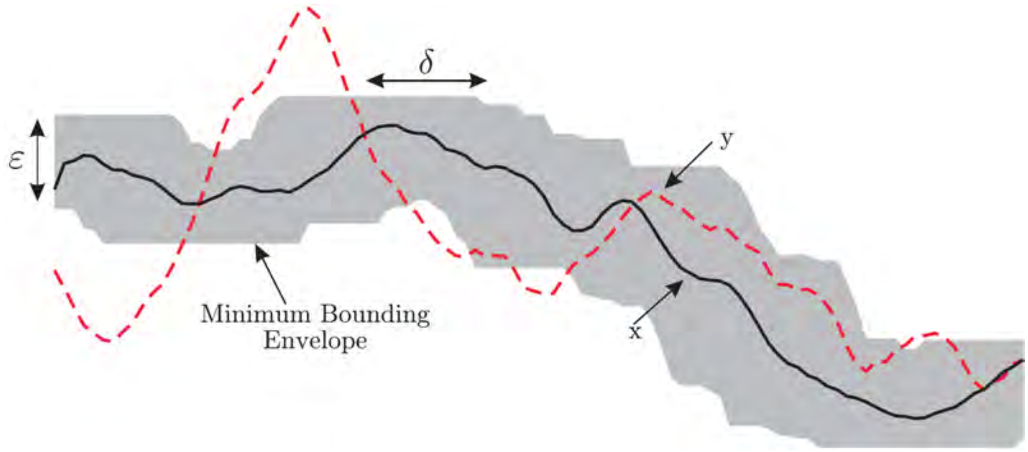


Figure 3: Example of the effects of ϵ and δ on the LCSS matching, image taken from Gorecki (2014).

2.3 Feature-based distances

In this subsection we discuss three different feature-based distances: the Discrete Fourier Transform, the Discrete Wavelet Transform and the Symbolic Aggregate approXimation. Feature-based distances first extract features from the time series and then measure the distance between these features. The three measures that we consider decompose the time series into sine waves, wavelets and strings, respectively. Feature-based distances are often applied to obtain a reduction in both dimensionality and noise, hence they are used when dealing with large (noisy) data sets.

In general clustering, a commonly used method for extracting features from data is Principal Component Analysis (PCA) (Jolliffe (2014)). This is a dimensionality reduction method that is used to explain the variance-covariance structure of a set of variables through linear combinations. While PCA appears to be an effective method in general clustering, where it reduces the number of dimensions while retaining most of the information, its performance appears to be less effective for time series clustering. This is due to the fact that PCA does not take into account the ordering of data, making it unable to capture time-evolving and time-shifted patterns (Li and Prakash (2011)). For this reason, we do not consider PCA in this subsection on feature-based distances.

2.3.1 Discrete Fourier Transform (DFT)

The Discrete Fourier Transform is a dimensionality reduction method that is introduced by Agrawal et al. (1993). It transforms the time series from a "time-domain" $x(t)$ to a "frequency-domain" representation $X(f)$. Here the collection of values of $X(f)$ at frequencies f are called the spectrum of $x(t)$. In other words: the Fourier transform decomposes a time series into all different cycles (consisting of amplitude, offset and rotation speed) that the series are composed off. The DFT is obtained by calculating the inner product between the time series and a sine wave and is defined as follows:

$$X(l) = \sum_{k=0}^{n-1} x_k e^{-\frac{i2\pi}{n}lk}, \tag{11}$$

for $x = x_0, ..., x_{n-1}$, $l = 0, ..., n-1$ and $i^2 = -1$. The resulting vector $X(l)$ is a vector with n complex numbers. It is also possible to transform a collection of frequencies $X(f)$ back to the "time-domain". This is done by the inverse DFT, which is defined as follows:

$$x_k = \frac{1}{n}\sum_{l=0}^{n-1} X(l) e^{\frac{i2\pi}{n}lk}, \tag{12}$$

for $x = x_0, ..., x_{n-1}$, $l = 0, ..., n-1$. Note that in Equations (11) and (12) we consider the indexes of the time series from 0 to $n-1$ instead of from 1 to n. This is done to be consistent with the literature on Fourier transforms.

According to Parseval's theorem, the DFT preserves the Euclidean distance between two time series (Hughes (1965)). This means that when all frequencies from the frequency domain $X(f)$ are used, the Euclidean distance between two Fourier transforms is equal to the Euclidean distance between the original time series of those transforms. This is caused by the fact that DFT is a linear transformation.

As indicated by Agrawal et al. (1993), most of the energy in real-world signals (time series) is concentrated in the low frequencies. This is where an advantage of the DFT arises: only a limited number of (low) frequencies are required to obtain a good approximation of the original time series. The DFT can thus be used to reduce the number of dimensions of a time series by only considering a limited number q $(q \leq n)$ of frequencies. According to the Nyquist–Shannon sampling theorem, only the first $\frac{n}{2}$ or fewer frequencies should be used (Strohmer and Tanner (2005)). However, it must be noted that every frequency that is omitted causes the approximation to diverge from the original time series. Therefore one does not want to approximate a time series with too few frequencies and thus usually the upper bound $q = \frac{n}{2}$ is used.

The Fourier distance also exploits the property of dimensionality reduction. It determines the DFT of both time series and then calculates the Euclidean distance only between the first q complex numbers (frequencies) of the Fourier transforms to approximate the Euclidean distance between the original time series. Besides the advantage of dimensionality reduction, disregarding the higher frequencies also denoises the time series in the approximation.

In Figure 4, the main idea behind the DFT is visualized. Here a time series (red line) is decomposed into the different sine waves (blue lines) that together form the original series. The brown bars on the frequency axis indicate the significance of every frequency in recreating the original time series. This plot on the magnitude-frequency axis is also called a periodogram. Observe that also in this example it holds that the lower frequencies have the highest significance in recreating the original series.

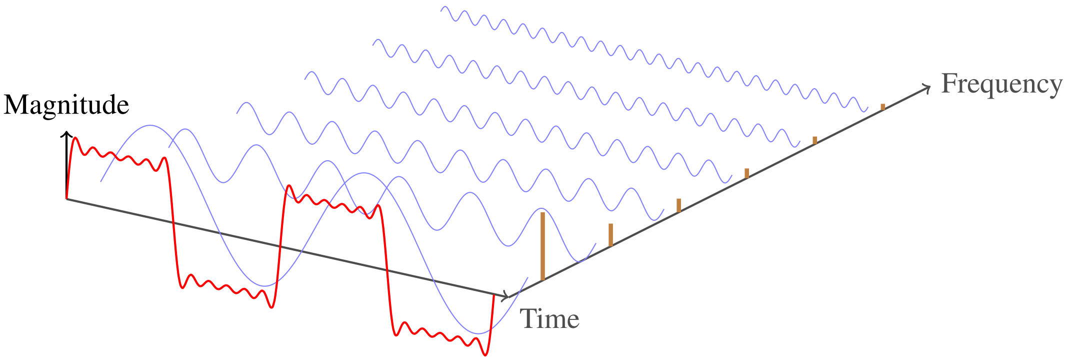


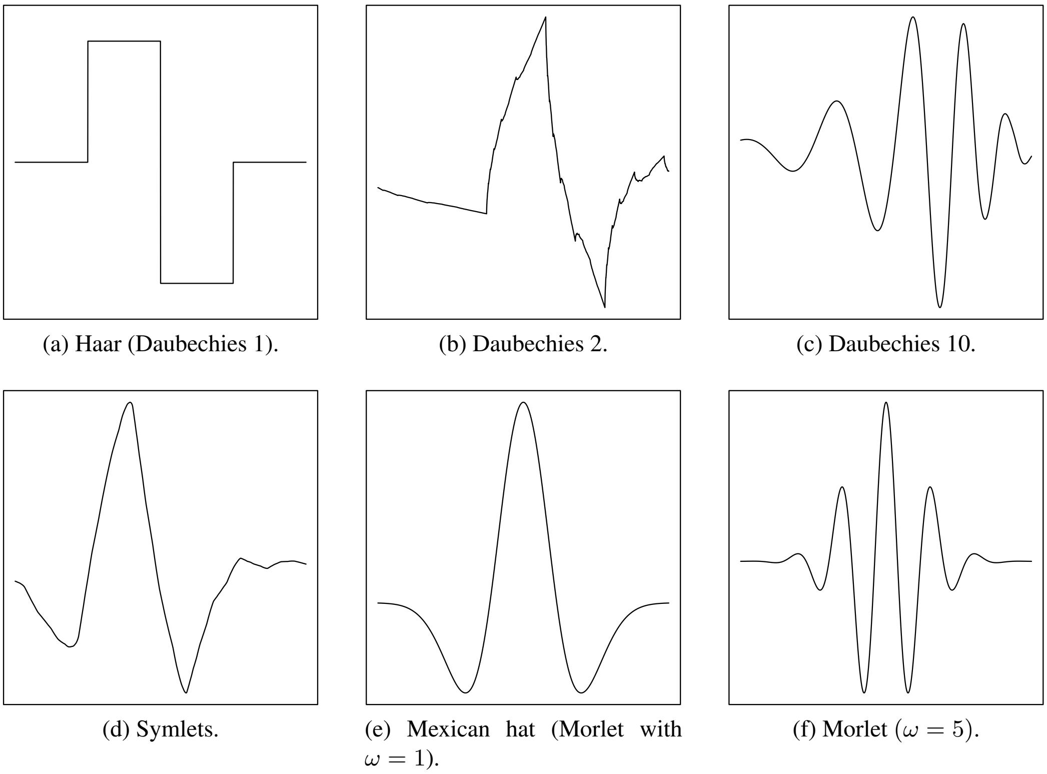
Figure 4: Example of a decomposition of a time series (red line) into the different sine waves (blue lines) that together form the time series. The brown bars represent the corresponding periodogram.

To illustrate the effects of omitting high frequencies to approximate and denoise a time series, we refer to Appendix B. Here an arbitrary time series (with $n = 105$) from our data set is detrended (linear trend) and standardized and plotted in Figure 20. The periodogram for the corresponding time series is plotted in Figure 21. By default, the periodogram function in R only considers the first $\frac{n}{2}$ frequencies. We observe that, as expected, the lower frequencies on the left have the highest significance. We now use the inverse DFT to approximate the original time series using different values of q. In Figure 22, the approximations of the original time series can be found for $q = 13, 26, 52$ and 105. Observe that the original time series are obtained for $q = 105$, since all frequencies from $X(f)$ are used. For $q = 13, 26$ and 52, we observe that high frequency spikes disappear as q decreases. According to the literature, for $n = 105$ we have to choose q around 52. This seems reasonable for this example, since we observe for $q = 52$ that some of the noise from the original series is removed while still retaining most of the general shape.

The time complexity of the DFT as defined in Equation (11) is $O(n^2)$ due to the matrix-vector multiplication. This time complexity can be improved to $O(n\log(n))$ by using Fast Fourier Transform (FFT) algorithms. FFT algorithms compute the same results as obtained by Equation (11), but by factorizing the DFT matrix into a product of sparse factors to avoid redundant calculations. The most commonly used FFT algorithm is the Cooley-Tukey algorithm (Cooley and Tukey (1965)).
Calculating the distance between two time series based on the Fourier coefficients has time complexity $O(q)$ and thus the whole process of determining the Fourier distance has time complexity $O(n\log(n))$. Determining the distance matrix with this measure takes $O(Nn\log(n)+qN^2)$ time. Note that we here multiplied the calculation time of the FFT ($O(n\log n)$) by N instead of N^2, as we only have to determine the Fourier transform of every time series once.

2.3.2 Discrete Wavelet Transform (DWT)

The Discrete Wavelet Transform is, just like the DFT, a dimensionality reduction method that also reduces noise. It decomposes a time series into a set of basis functions that are called wavelets. A wavelet is a rapidly decaying, wave-like oscillation with mean zero. Unlike the sine waves that are used in Fourier decomposition, wavelets only have a finite duration. Wavelets are defined by two functions: the wavelet function ψ and the scaling function φ. These are also referred to as the mother wavelet and father wavelet, respectively. The mother wavelet ψ characterizes the basic shape of the wavelet. In Figure 5, a selection of commonly used mother wavelets can be found. The father wavelet φ characterizes the scale of the wavelet. In Figure 6, a Morlet (mother) wavelet is plotted for different scales. Observe that scale is the wavelet equivalent of frequency.



(a) Haar (Daubechies 1).

(b) Daubechies 2.

(c) Daubechies 10.

(d) Symlets.

(e) Mexican hat (Morlet with $\omega = 1$).

(f) Morlet ($\omega = 5$).

Figure 5: A selection of commonly used mother wavelet functions or function families.

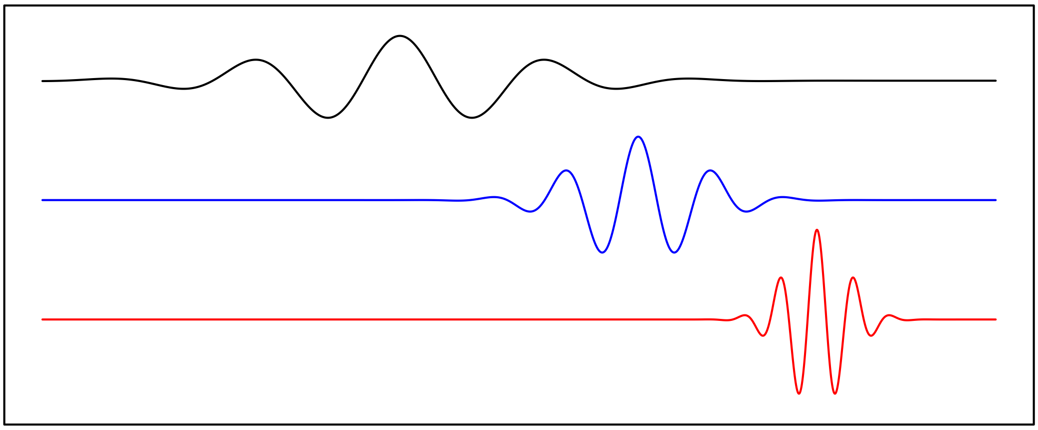


Figure 6: Wavelet function $Wave(\tau, s) = \sum_t x_t \frac{1}{\sqrt{s}} \psi^* \left(\frac{t-\tau}{s} \right)$ with mother wavelet function $\psi(t) = \pi^{-1/4} e^{i\omega t} e^{-t^2/2}$ using angular frequency $\omega = 5$ plotted for different scales s ($s = 1$ for black, $s = 2$ for blue and $s = 4$ for red) and different centers τ.

The DWT is obtained by successively passing a time series through high-pass and low-pass filters. This produces detail and approximation coefficients for different levels of decomposition. At each level of decomposition, the time series is down sampled by a factor of two using a half-band filter.

To give an example of a wavelet decomposition, we consider the simplest kind of wavelet function: the Haar wavelet (Haar (1910)). This example is taken from Chaovalit et al. (2011) and revised for visual purposes. The Haar wavelet and scaling function are defined as follows:

$$\psi(t) = \begin{cases} 1 & \text{if } 0 < t \leq \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} < t \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

$$\varphi(t) = \begin{cases} 1 & \text{if } 0 \leq t \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

The Haar (mother) wavelet is also plotted in Figure 5a. We consider a short time series S of length 8: $S = \{82, 75, 55, 64, 50, 42, 69, 59\}$. By applying the Pyramid algorithm (Mallat (1989)) we can decompose this time series into different levels of approximation and detail coefficients. These coefficients can be found in Table 3. The approximation coefficients (in bold) are indicated by $A_{i,j}$, where i indicates the level of decomposition and j the index of the coefficient. The detail coefficients are denoted by $D_{i,j}$ (in italic). The coefficient values are obtained by taking the pairwise averages and differences of the approximation level or original time series that lie one level lower:

$$A_{i,j} = \begin{cases} \frac{A_{i-1,2j} + A_{i-1,2j-1}}{2} & \text{if } i > 1 \\ \frac{t_{2j} + t_{2j-1}}{2} & \text{if } i = 1 \end{cases} \tag{15}$$

$$D_{i,j} = \begin{cases} \frac{A_{i-1,2j} - A_{i-1,2j-1}}{2} & \text{if } i > 1 \\ \frac{t_{2j} - t_{2j-1}}{2} & \text{if } i = 1, \end{cases} \tag{16}$$

where t_j is the value of the original time series on the j-th index.

Observe that in Table 3, due to the half-band filter, the total number of coefficients is halved for every level of decomposition. Also observe that the approximation coefficients of decomposition level i can be obtained by adding and subtracting the detail coefficients of decomposition level $i+1$ to and from the approximation coefficients of level $i+1$. Doing this in an iterative manner, one can reconstruct the original time series S by using only the highest level approximation coefficient (in our example: $A_{3,1}$) and all detail coefficients $D_{i,j}$. A vector of these coefficients together has the same length as the original time series. It must be noted that the above only holds if the dimension of the original time series is a power of 2. If this is not the case, equations (15) and (16) will at some point be applied to odd series, where three values are taken together instead of two. For example, when creating a decomposition of a time series with 105 points, the resulting 6 levels of decomposition have lengths 52, 26, 13, 6, 3 and 1. When reconstructing time series that are not a power of 2, slight errors may occur when trying to reconstruct the original time series.

Table 3: Time series S with three levels of the Haar wavelet decomposition. The table on the top contains the approximation and detail coefficients and the table on the bottom contains the notation of the corresponding values.

S	82	75	55	64	50	42	69	59
Level 1	**78.5**	**59.5**	**46**	**64**	*-3.5*	*4.5*	*-4*	*-5*
Level 2	**69**	**55**	*-9.5*	*9*				
Level 3	**62**	*-7*						

S	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Level 1	$\mathbf{A_{1,1}}$	$\mathbf{A_{1,2}}$	$\mathbf{A_{1,3}}$	$\mathbf{A_{1,4}}$	$D_{1,1}$	$D_{1,2}$	$D_{1,3}$	$D_{1,4}$
Level 2	$\mathbf{A_{2,1}}$	$\mathbf{A_{2,2}}$	$D_{2,1}$	$D_{2,2}$				
Level 3	$\mathbf{A_{3,1}}$	$D_{3,1}$						

Reduction of dimensionality can be achieved by omitting the detail coefficients of lower levels of decomposition. Every time de detail coefficients of a level are omitted, the dimension is halved. In Figures 7 and 8, the approximations of time series S can be found when omitting the first and the first two detail levels of detail coefficients, respectively.

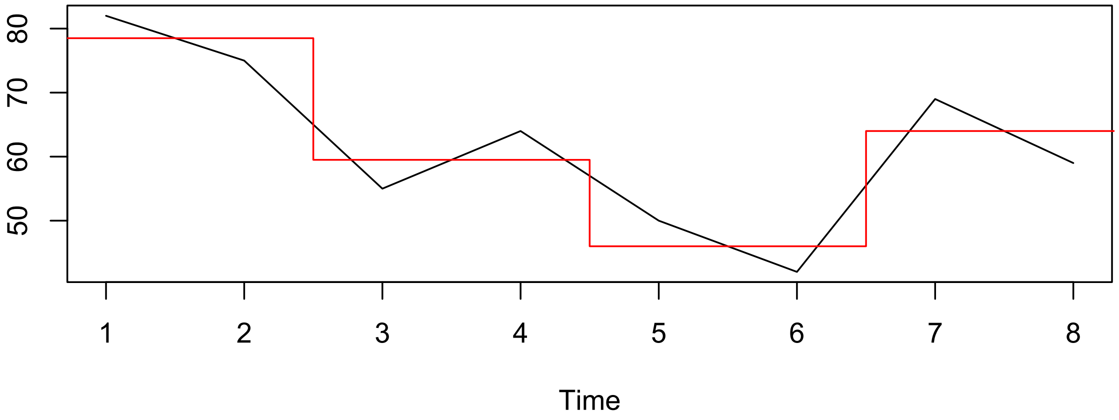


Figure 7: Approximation (in red) of the original time series S (in black) when using only the detail coefficients of decomposition levels 2 and 3 (and thus omitting $D_{1,j}$). The approximation is done with coefficients $\{A_{3,1}, D_{3,1}, D_{2,1}, D_{2,2}\}$ and thus the dimension of the approximation is 4 (compared to 8 dimensions for the original series).

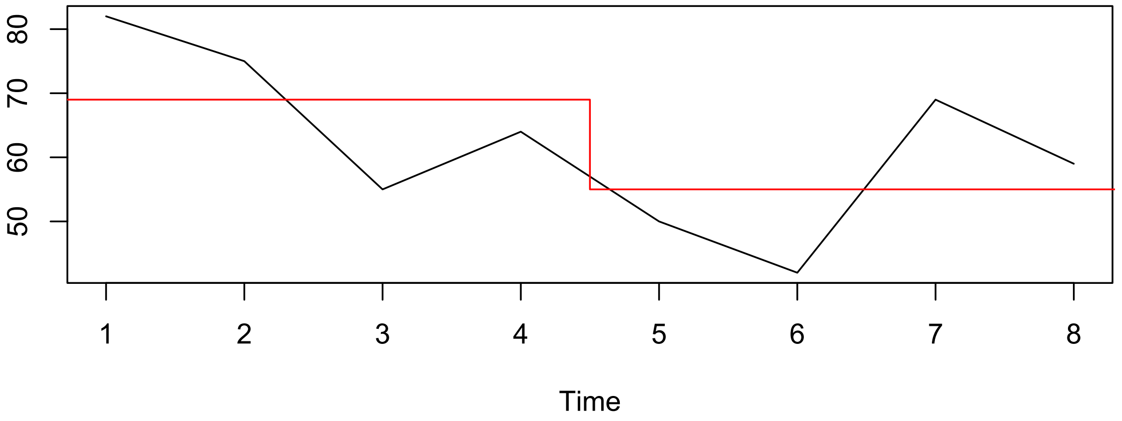


Figure 8: Approximation (in red) of the original time series S (in black) when using only the detail coefficients of decomposition level 3 (and thus omitting $D_{1,j}$ and $D_{2,j}$). The approximation is done with coefficients $\{A_{3,1}, D_{3,1}\}$ and thus the dimension of the approximation is 2 (compared to 8 dimensions for the original series).

In Appendix C, we applied wavelet decomposition to a time series from our data set. Here the same time series is used as in Appendix B. Two types of wavelets are considered: Haar (as in the example above) and Daubechies 10. As expected, we observe that when using all detail levels the original time series is reconstructed. Note that omitting detail levels does not only reduce the number of dimensions, but also the noise.

A main advantage DWT has over DFT is that it not only captures frequency information of a time series but also information on the location in time. This advantage can be visualized by plotting the power spectrum of a time series. In Figure 9, the wavelet power spectrum is plotted for the time series that is used in Appendices B and C. This spectrum indicates the power (transformation of the significance) of a set of periods for different locations in time. Observe that, in this case, long periods (around 50) are significant for the whole time interval. However, the shorter periods seem to only be significant for limited time intervals. While the DFT assigns sine waves over the whole interval, the DWT can assign wavelets specifically to the locations in time where a given period is significant, hence the advantage. For further explanation on how the significance in the power spectrum is calculated, we refer to Torrence and Compo (1998).

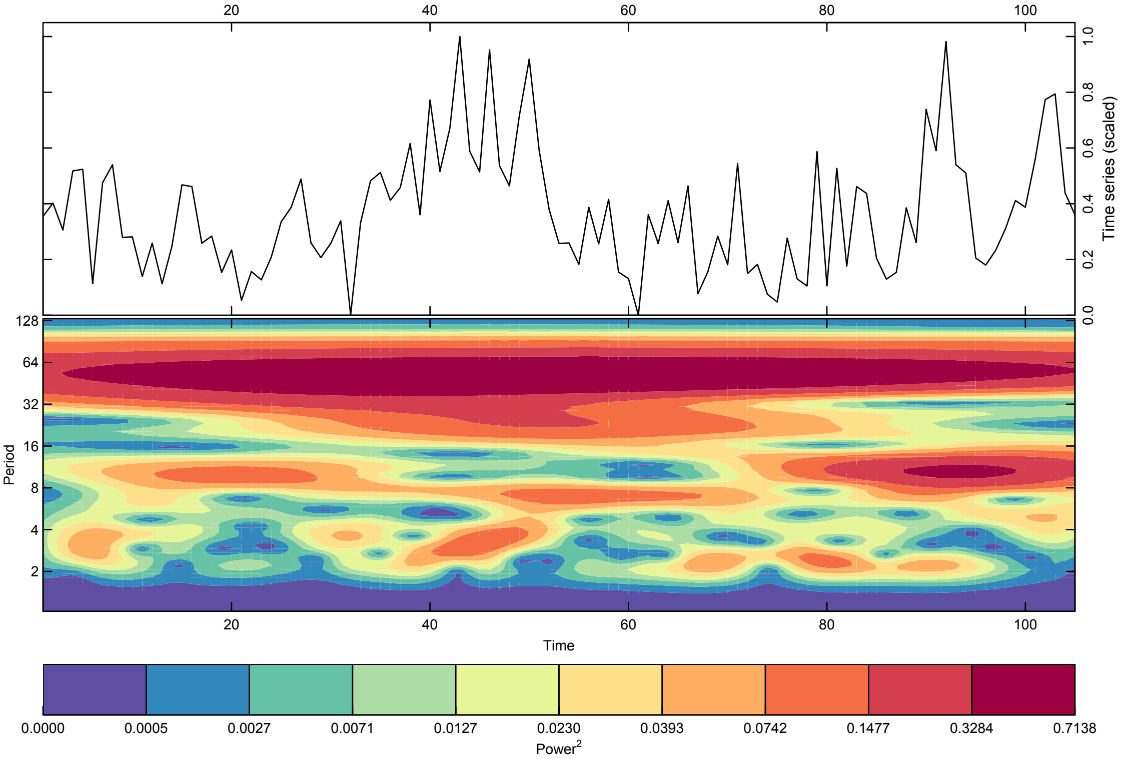


Figure 9: Wavelet power spectrum (bottom) for the time series (top)that is used in Appendices B and C. Plotted using the *wavelet.plot* function from the **dplR** package in R (Bunn et al. (2017)).

When applying the DWT to obtain an approximation of a time series, the number of detail levels to use and the chosen mother wavelet are both important factors. The number of detail levels to use is a trade-off between the number of dimensions and the amount of detail in the resulting approximation and thus this depends on the practical application of the wavelet transform. For choosing the "optimal" mother wavelet (shape), a number of different qualitative and quantitative methods have been applied in previous research (Ngui et al. (2013)). A common qualitative method is to simply pick a mother wavelet that has a similar shape compared to the original time series. A common quantitative method is to compute the maximum cross-correlation coefficient between a selected mother wavelet and the original time series and to compare this value for different mother wavelets. However, it is concluded by Ngui et al. (2013) that the mother wavelet that is most similar to the original series is not always the optimal wavelet.

The time complexity of the DWT when using the Pyramid algorithm is $O(n)$ (Nason (2008)). Once the DWT is calculated, it takes $b = \frac{n}{2^l}$ operations to calculate the Euclidean distance between two wavelet approximations. Here l is the number of detail levels that are omitted in the approximation. Note that $b \leq n$, as l cannot be negative.
When determining a distance matrix with DWT distances, determining the DWT for all time series takes $O(nN)$ time and calculating the distance matrix takes $O(bN^2)$ time. Together this process thus has a time complexity of $O(nN + bN^2)$.

2.3.3 Symbolic Aggregate approXimation (SAX)

SAX is a symbolic representation of time series that is proposed by Lin et al. (2007). They observed that most existing symbolic representations for time series suffer from two fatal flaws. The first flaw is that the dimensionality of the representations is equal to the original time series. Data mining algorithms often scale poorly with dimensionality and thus this is not desired. The second flaw concerns distance measures that can be defined on the symbolic representations. Often these distances have little correlation with the distance measures that are defined on the original data. With SAX, Lin et al. (2007) introduced a new symbolic representation that overcomes these two flaws.

The algorithm for obtaining the SAX representation consists of two steps: reducing the time series using Piecewise Aggregate Approximation (PAA) and assigning a letter or discrete label to every PAA element. But before running the SAX algorithm, first the time series are normalized, since series with different offsets and amplitudes are meaningless to compare (Keogh and Kasetty (2003)).

When converting the time series to the PAA representation, the series are reduced to a w-dimensional space ($w \leq n$). This is done by a vector $\bar{X} = \bar{x}_1, ..., \bar{x}_w$ where the elements are calculated by:

$$\bar{x}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x_j. \tag{17}$$

In other words, the dimension of the series is reduced from n to w by dividing the data into w equally sized parts. The mean value of every part is calculated and these values form vector $\bar{X}$. Examples of the PAA reductions of time series can be found in Figures 11 and 12. Here we have taken two time series from our data set which we both normalized.

The first part of the SAX algorithm, reduction using PAA, divides the time series horizontally. The second step assigns labels to all w parts by dividing the vertical axis into a regions. Note that the time series are normalized and thus have a Gaussian distribution (Larsen and Marx (2001)). Therefore $B = \beta_1, ..., \beta_{a-1}$ breakpoints are determined to divide the $N(0,1)$ Gaussian curve into a equal-sized areas. β_0 and β_a are $-\infty$ and ∞, respectively. Next, a discrete label is assigned to every region between the breakpoints with the following rule: $\hat{x}_i = \alpha_j \Leftrightarrow \beta_{j-1} \leq \bar{x}_i < \beta_j$. These labels α are either from an alphabet or a set of binary numbers with cardinality a. For example, if we choose an alphabet with cardinality 4, we obtain $\alpha_1 = a, \alpha_2 = b, \alpha_3 = c$ and $\alpha_4 = d$. The set of labels $\hat{X} = \hat{x}_1, ..., \hat{x}_w$ forms the SAX representation of the original time series. In Figure 13 an example of the SAX representation can be found. Here the two PAA reductions from Figures 11 and 12 are converted to the SAX representation with cardinality 8. Note that both time series are now represented by strings of the letters $a, b, ..., h$.

To find the distance between two SAX representations, the following equation is used:

$$MINDIST(\hat{X}, \hat{Y}) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (dist(\hat{x}_i, \hat{y}_i))^2}, \tag{18}$$

where the $dist$ function uses a lookup table that is based on the breakpoints β. The values in this lookup table are determined in the following way:

$$cell_{r,c} = \begin{cases} 0 & \text{if } |r - c| \leq 1 \\ \beta_{\max\{r,c\}-1} - \beta_{\min\{r,c\}} & \text{otherwise.} \end{cases} \tag{19}$$

In Table 4, an example of a lookup table with cardinality 8 can be found. The breakpoints from Figure 13 that are used to determine the values in this lookup table are also shown in Figure 10. Note that the distance between two adjacent regions is zero. The distance found by the $MINDIST$ function is a lower bound approximation to the Euclidean distance. Increasing the number of SAX symbols w or the cardinality a makes this lower bound tighter, but at the cost of an increase in calculation time.

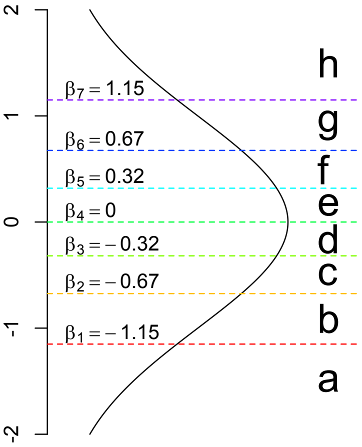


Figure 10: Gaussian curve with breakpoints for cardinality 8.

	a	b	c	d	e	f	g	h
a	0	0	0.48	0.83	1.15	1.47	1.82	2.30
b	0	0	0	0.36	0.67	0.99	1.35	1.82
c	0.48	0	0	0	0.32	0.64	0.99	1.47
d	0.83	0.36	0	0	0	0.32	0.67	1.15
e	1.15	0.67	0.32	0	0	0	0.36	0.83
f	1.47	0.99	0.64	0.32	0	0	0	0.48
g	1.82	1.35	0.99	0.67	0.36	0	0	0
h	2.30	1.82	1.47	1.15	0.83	0.48	0	0

Table 4: SAX lookup table for an alphabet with cardinality 8.

We now consider the time complexity of SAX by breaking up the different steps in the algorithm. The first part is PAA reduction, which has a time complexity of $O(w \cdot \frac{n}{w}) = O(n)$. The second part, converting the PAA reductions to the SAX representation, has a time complexity of $O(w \cdot \log(a)))$. At last, calculating the $MINDIST$ between two SAX representations has time complexity $O(w)$. The whole process of determining the SAX distance thus has a time complexity of $O(n + w\log(a))$. However, if the SAX representation of the time series is already available, this time complexity is $O(w)$. We can thus observe that the main advantage of SAX arises when we have to use a single time series to determine multiple distances (e.g. when determining a distance matrix), since we only have to determine the SAX representation once. When determining a distance matrix with SAX distances, determining the SAX representation for all time series takes $O(Nw\log(a))$ time and calculating the distance matrix takes $O(wN^2)$ time. Together this process thus has a time complexity of $O(Nw\log(a) + wN^2)$.

Shieh and Keogh (2009) have proposed a variation on SAX, called iSAX, that allows indexing and mining of up to one hundred million time series. Later, Camerra et al. (2010) improved this algorithm and introduced iSAX2.0, which allows indexing and mining of up to a billion time series.

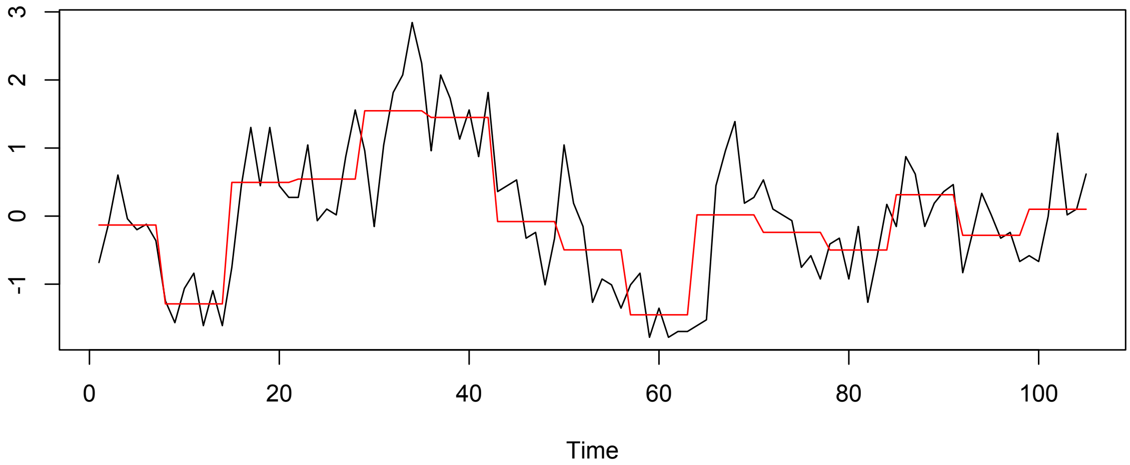


Figure 11: PAA reduction of series x. The dimension $n = 105$ is reduced to $w = 15$.

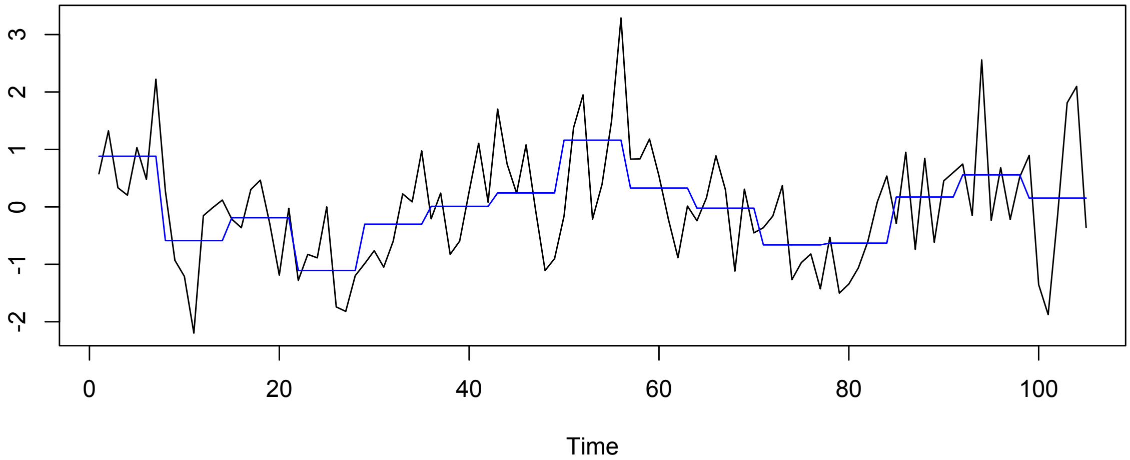


Figure 12: PAA reduction of series y. The dimension $n = 105$ is reduced to $w = 15$.

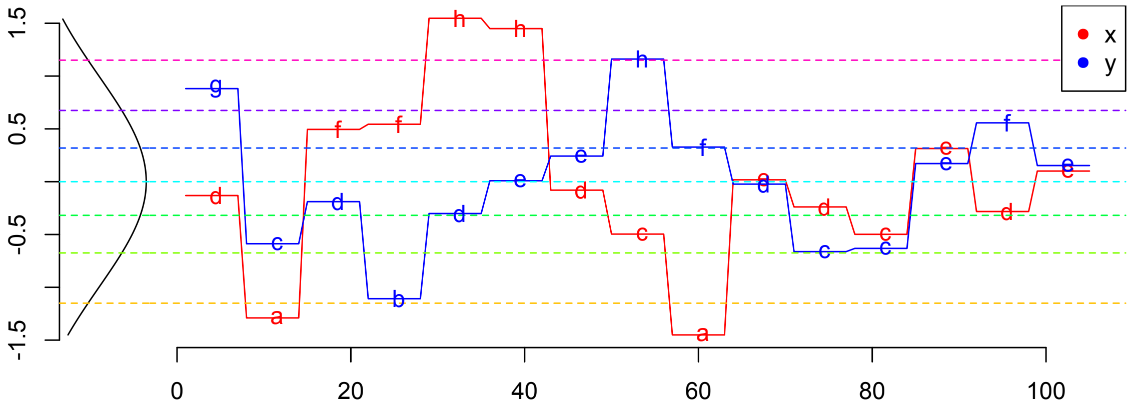


Figure 13: SAX representation of series x and y. $n = 105, w = 15$ and $a = 8$.

2.4 Ensemble schemes

In the previous subsections, different distance measures are considered individually. However, previous research on time series classification indicates that ensemble schemes of different distance measures often outperform individual distance measures when it comes to grouping time series (Zhou et al. (2015); Gorecki (2017); Lines and Bagnall (2015)). Only elastic distance measures are considered in these papers, since they "generally provide more accurate classification accuracies compared to non-elastic measures" (Gorecki (2017)). While this is the case, it must also be noted that elastic distance measures generally come with long computation times and thus combinations of these will be time-consuming.

Due to the promising results in previous research, we will also consider an ensemble scheme in this research. The ensemble scheme that we consider is referred to as the Linear Combination Method in Zhou et al. (2015), which combines distance matrices in a linear way. The resulting distance matrix $D_{ensemble}$ can be formulated by:

$$D_{ensemble} = \frac{1}{p} \sum_{ms \in MS} D_{ms}, \tag{20}$$

where MS is the set of distance measures that are considered in the ensemble scheme and p is the number of distance measures in MS. A variation to this approach is also considered in Gorecki (2017), where a linear combination of DTW, DDTW and LCSS distance measures is proposed with variable weights instead of equal weights. Besides elastic distance measures, we will also consider feature-based distance measures in the ensemble schemes to investigate their performance when being ensembled.

2.5 Comparing the distance measures

We now review and compare the different distance measures that are considered in this section, based on their theoretical time complexity, performance and properties. An overview of the time complexities for each distance measure can be found in Table 5.

Table 5: Overview of the time complexity of the different distance measures that are considered in this section. It is assumed that $n \geq m$. See the previous subsections for explanation on the variables q, b, w and a.

	Distance measure time complexity	**Distance matrix time complexity**
Shape-based distances		
Lock-step measures		
Minkowski ($\forall p$)	$O(n)$	$O(nN^2)$
Pearson correlation	$O(n)$	$O(nN^2)$
Elastic measures		
Dynamic Time Warping	$O(nm)$	$O(nmN^2)$
Longest Common Subsequence	$O(nm)$	$O(nmN^2)$
Feature-based distances		
Discrete Fourier Transform[1]	$O(n\log(n))$	$O(Nn\log(n) + qN^2)$
Discrete Wavelet Transform[2]	$O(n)$	$O(nN + bN^2)$
SAX representation[3]	$O(n + w\log(a))$	$O(Nw\log(a) + wN^2)$

In general clustering, often traditional lock-step distance measures are used, such as the Euclidean distance or Pearson correlation distance. While these work fine for a wide variety of clustering applications, when clustering time series they are often outperformed by other distance measures. This is due to some limitations that lock-step measures possess: they do not account for shifts in time and may not process noise and outliers in a desirable way.

To overcome the limitations of the lock-step measures, elastic measures are introduced, such as DTW and LCSS. Both distance measures warp the time series to account for shifting in time. LCSS also accounts for noise and outliers, depending on the threshold setting. This makes LCSS more robust than DTW under noisy conditions (Vlachos et al. (2002)). Furthermore, it must be noted that the local distance measure used in DTW is Euclidean, whereas LCSS uses the Manhattan distance as local distance measure.

Although elastic distance measures generally outperform lock-step distance measures for measuring similarity between time series, they also come with a significant increase in calculation time. This may make elastic measures unsuitable, depending on the size of the data set and the length of the time series. Furthermore, it is shown by Shieh and Keogh (2008) and Wang et al. (2010) that the classification accuracy difference between the Euclidean distance and elastic distance measures converges to zero as the number of time series increase. The speed of this convergence also depends on the time series data that is used.

[1] If the DFT representation is already available, the distance measure time complexity is $O(q)$ and the distance matrix time complexity is $O(qN^2)$.

[2] If the DWT representation is already available, the distance measure time complexity is $O(b)$ and the distance matrix time complexity is $O(bN^2)$.

[3] If the SAX representation is already available, the distance measure time complexity is $O(w)$ and the distance matrix time complexity is $O(wN^2)$.

Not only elastic measures may become inconvenient when the number of time series or the length of the individual time series becomes too large. Even lock-step measures may at some point require too much calculation time for determining the distance matrix. To reduce this calculation time, dimensionality reduction methods can be used. The three feature-based distance measures that we described in this subsection (DFT, DWT and SAX) are all examples of dimensionality reduction methods. Here the number of dimensions is reduced by decomposing the time series into different features: sine waves for DFT, wavelets for DWT and string representations for SAX. Both DFT and DWT are powerful in spotting periodicity in time series. Here DWT has an advantage over DFT, as it not only captures frequency information of a time series, but also information on the location in time. In addition, DWT is faster than DFT. However, it must be noted that both measures compute quite different transforms. Also, DFT is less complex compared to DWT, making it easier (faster) to interpret for new users compared to DWT.

Besides reducing the number of dimensions, all three feature-based distance measures also reduce noise by leaving out certain levels of detail in the approximations. Once time series are decomposed by any of the three feature based measures mentioned above, the Euclidean distance is used to approximate the distance between two time series. Out of the three distance measures, SAX is the least expensive in terms of calculation time (Lin et al. (2007)). If even SAX is taking too long, one could try faster variations of this algorithm, such as iSAX and iSAX2.0 (Shieh and Keogh (2009); Camerra et al. (2010)). It must be noted that feature based distance measures can also be relevant to use when calculation time (and thus dimensionality) is not an issue, as they then still provide noise reduction.

In Figure 14, a flowchart can be found on which distance measures to try when one wants to compute a distance matrix for the purpose of time series clustering. Here we only consider the seven distance measures that we discussed in this section. At the start, we assume a user has a set of time series data. This data should be preprocessed based on the goal of the clustering assignment. We refer to Subsection 5.1.2 for more explanation on this topic. After preprocessing, we suggest to always start by trying a lock-step distance measure such as the Euclidean distance or the Pearson correlation distance. If the actual difference in values of attributes is important, the Euclidean distance (or another Minkowski distance) should be tried. If general trend or shape similarity is important, one should go with a correlation distance. Note that the Pearson correlation distance and the Euclidean distance are essentially computing the same values when the data is standardized in the preprocessing step. Based on the computation time of the lock-step distance measure(s), we suggest to either try elastic measures (if the computation time is not taking too long) or feature-based measures (if the computation time takes too long). If the individual elastic measures, such as DTW or LCSS, are not taking too long for computing the distance matrix, it is suggested to try ensemble schemes as well. However, if the individual elastic measures are taking too long, we suggest the user to stop. Note that in this case the user still has a distance matrix from a lock-step distance measure. For the feature-based measures, it is suggested to first try using DWT or DFT and try SAX only if these measures do not compute the distance matrix within a desired time frame.

To conclude, we can state that choosing a good distance measure is a trade-off between speed and accuracy. Elastic measures tend to outperform lock-step measures, but for large data sets this difference in performance becomes less significant. Feature-based measures are often used for large data sets, as they provide dimensionality reduction and thereby reduce the calculation time. If the calculation time is not an issue, one can try ensemble schemes of different distance measures. These generally outperform individual distance measures, but at the cost of an increase in calculation time.

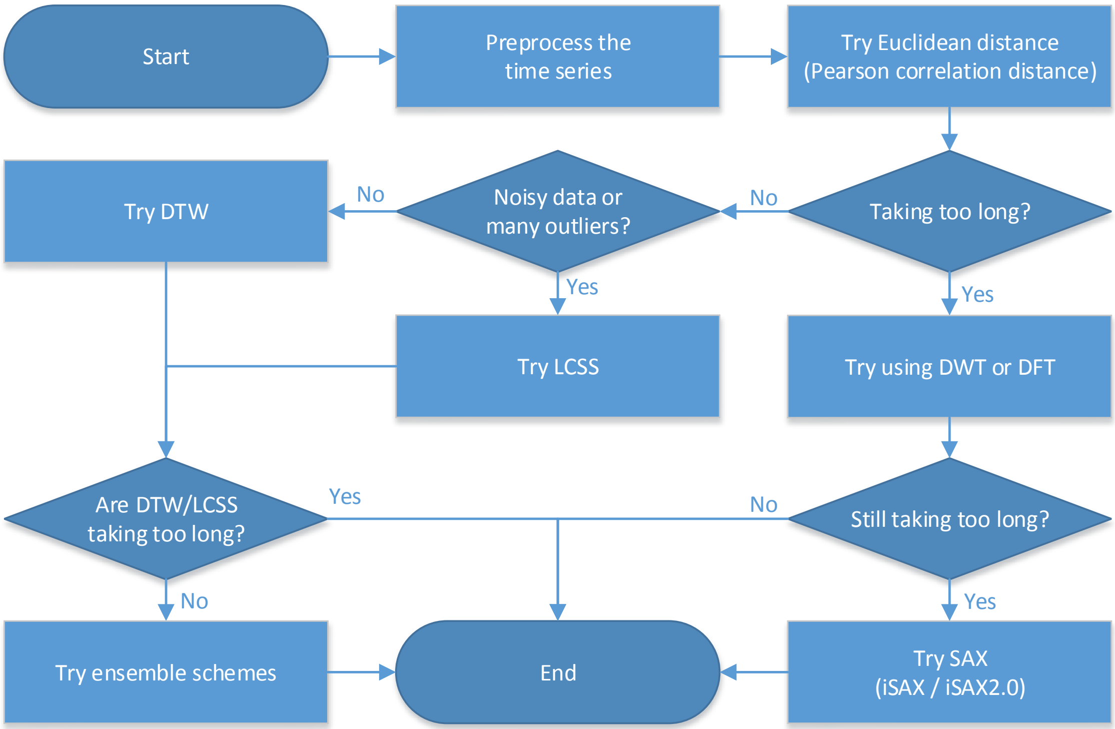


Figure 14: Flowchart of which distance measures to try when one wants to compute a distance matrix for the purpose of time series clustering. In this flowchart we only consider the seven distance measures we discussed in this thesis.

3 Clustering methods

In this section, we discuss a selection of methods for obtaining a clustering. Many methods are available in the literature and Sheikholeslami et al. (1998) classified the different clustering methods into four types: hierarchical clustering, partitional clustering, density-based clustering and grid-based clustering. It is concluded by Liao (2005) that for time series clustering, many general-purpose clustering methods can be applied and the choice of the distance measure is more important than the choice of the clustering method. We therefore only consider hierarchical and partitional clustering methods in this section, as these appear to be the most commonly used clustering methods in literature on time series clustering (Liao (2005); Rani et al. (2012); Sarda-Espinosa (2017); Aghabozorgi et al. (2015)). We first look into the two types of clustering methods individually and discuss some of the algorithms that are available. Afterwards, in Section 3.3, we compare the different methods.

Besides the four classes mentioned above, we can also make a distinction between hard clustering (also known as crisp clustering) and soft clustering (also known as fuzzy clustering). In hard clustering, we have non-overlapping clusters, where every observation belongs to exactly one cluster. In soft clustering, observations can belong to multiple clusters, often accompanied by probabilities of belonging to each cluster. For both types of clustering there are many different algorithms. However, we will only focus on hard clustering algorithms in this section.

3.1 Hierarchical clustering

Hierarchical clustering is a method for obtaining a clustering by building a hierarchy of clusters. This set of nested clusters is organized as a tree and can be visualized by a dendrogram. An example of a dendrogram for a hierarchical clustering of eight observations can be found in Figure 15. Here the numbers on the x-axis represent the eight different observations and the y-axis indicates the distance between different clusters.

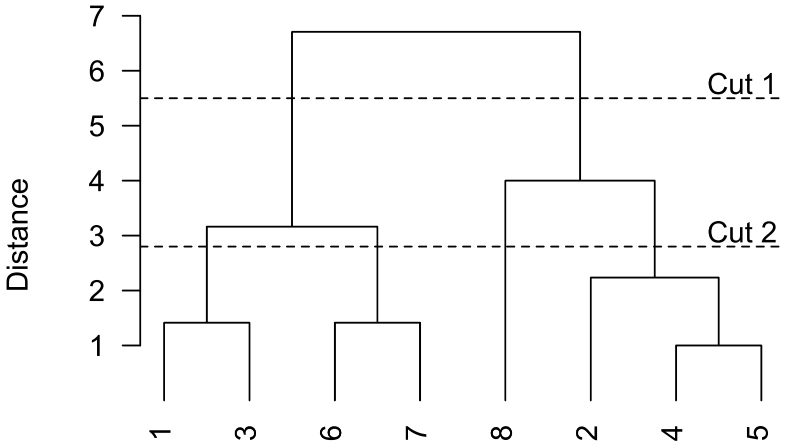


Figure 15: Example of a dendrogram for a hierarchical clustering with eight observations. Two examples of cuts are added for $k = 2$ (Cut 1) and $k = 4$ (Cut 2) clusters.

A clustering is obtained by making a cut in the dendrogram. In Figure 15, two cuts are added as an example. Cut 1 results in two clusters: $\{1, 3, 6, 7\}$ and $\{6, 2, 4, 5\}$. Cut 2 results in four clusters: $\{1, 3\}, \{6, 7\}, \{8\}$ and $\{2, 4, 5\}$.

There are two types of hierarchical clustering: agglomerative and divisive. Agglomerative clustering is a bottom-up approach that starts with all observations as individual clusters and, at each step, merges the closest pair of clusters. Divisive clustering is a top-down approach that starts with one cluster containing all observations and, at each step, splits a cluster until only singleton clusters of individual observations remain. In practice, agglomerative clustering is much more common than divisive clustering (Hastie et al. (2009)). In the next two subsections, we discuss both hierarchical clustering approaches.

An advantage of hierarchical clustering is that it does not require the number of clusters k to be predefined. Instead, one can obtain the clustering for any number of clusters $k \in \{1, ..., N\}$ by making a cut in the dendrogram at the corresponding number of clusters. Here N represents the total number of observations. A disadvantage is, however, that hierarchical clustering requires the distance matrix of all pairs of observations to be calculated. This can be computationally expensive, as the number of elements in this matrix grows with $O(N^2)$ (as seen in Section 2). Another disadvantage of hierarchical clustering is that it suffers from a lack of flexibility, since no adjustments to the tree can be performed once a split (divisive clustering) or a merger (agglomerative clustering) is done (Sarda-Espinosa (2017)).

3.1.1 Agglomerative clustering

Agglomerative clustering is a bottom-up approach that starts with all observations as individual clusters and, at each step, merges the closest pair of clusters. The standard algorithm for agglomerative clustering can be found in Algorithm 1. This algorithm starts with N clusters and iteratively merges clusters until only a single cluster remains.

Algorithm 1 Standard algorithm for agglomerative clustering.

```
1: Compute the distance matrix.
2: Initialize all $N$ observations to be the initial $N$ clusters.
3: while there is more than one cluster do
4:     Merge the closest two clusters.
5:     Update the distance matrix to reflect the merge from the previous step.
6: end while
7: return set of nested clusters.
```

In step 1, a distance matrix is calculated that contains the distances between all pairs of observations in the data. To compute these pairwise distances, any distance measure can be used (see also Section 2). The individual observations form the initial N different clusters.

Until only one cluster that contains all observations remains, the clusters are now iteratively merged. In each iteration, the pair of clusters with minimum distance in the distance matrix is merged. Once two clusters, cluster c_i and cluster c_j, are merged to form $c_i \cup c_j$ in step 4 of Algorithm 1, the distance matrix is updated to reflect this merge. This is done by deleting the rows and columns of clusters c_i and c_j from the distance matrix and by adding a row and column for the new cluster $c_i \cup c_j$ to the distance matrix. The distances between cluster $c_i \cup c_j$ and all other clusters are calculated using the Lance-Williams dissimilarity update formula (Murtagh and Contreras (2011)), where the distance between cluster $c_i \cup c_j$ and cluster c_k is defined by:

$$d(i \cup j, k) = \alpha_i d(i,k) + \alpha_j d(j,k) + \beta d(i,j) + \gamma |d(i,k) - d(j,k)|. \tag{21}$$

Here $d(q,r)$ is the distance between clusters c_q and c_r, $d(i \cup j, k)$ the distance between clusters $c_i \cup c_j$ and c_k and $\alpha_i, \alpha_j, \beta$ and γ are parameters that, together with the distance function, determine the method for agglomerative hierarchical clustering.

In this thesis we consider three (commonly used) methods for agglomerative hierarchical clustering: Single linkage, Complete linkage and Ward's method. The difference between these methods lies in the set of parameters that is chosen for the Lance-Williams dissimilarity update formula. Each set of parameters has a bias towards a type of hierarchy, which has pros and cons. In the next three paragraphs, the Lance-Williams update formula parameter values and the interpretations of all three methods that we consider can be found. Other methods include average linkage, McQuitty's method, the medoid method and the centroid method. For the Lance-Williams parameter values and the interpretations of these methods, we refer to Murtagh and Contreras (2011).

Single linkage is a method where the distance between merged cluster $c_i \cup c_j$ and cluster c_k is defined by $d(i \cup j, k) = \min\{d(i,k), d(j,k)\}$. The corresponding Lance-Williams dissimilarity update formula parameters are $\alpha_i = 0.5, \beta = 0$ and $\gamma = -0.5$. The advantage of Single linkage is its ability to find arbitrary shaped clusters (Jain and Dubes (1988)). A disadvantage is that it is highly sensitive to noise and outliers.

Complete linkage is a method where the distance between merged cluster $c_i \cup c_j$ and cluster c_k is defined by $d(i \cup j, k) = \max\{d(i,k), d(j,k)\}$. The corresponding Lance-Williams dissimilarity update formula parameters are $\alpha_i = 0.5, \beta = 0$ and $\gamma = 0.5$. The advantage of Complete linkage is that it is less influenced by noise and outliers compared to Single linkage. This comes at the cost, however, of being unable to deal with arbitrary shaped clusters. Furthermore, it tends to break large clusters.

Ward's method is an ANOVA based approach that is based on Ward's criterion (Ward (1963)). It defines the distance between two clusters c_i and c_j to be the increase in sum of squared error (SSE) that results from merging the two clusters: $d(i,j) = SSE_{i \cup j} - SSE_i - SSE_j$. The corresponding Lance-Williams dissimilarity update formula parameters are $\alpha_i = \frac{|i|+|k|}{|i|+|j|+|k|}, \beta = -\frac{|k|}{|i|+|j|+|k|}$ and $\gamma = 0$, where $|i|$ represents the number of observations in cluster i. When using Ward's method, it is noted by Murtagh and Contreras (2011) that the initial distance matrix should be calculated using (a distance measure proportional to) the Euclidean distance. An advantage of Ward's method is that it minimizes the total within-cluster variance.

What we can conclude from the three methods above, is that it depends on the data and the criteria for the clustering which method is most suitable. However, it is noted by Sarda-Espinosa (2017) that if the data can "easily" be grouped, all methods should provide similar results.

To execute steps 2 to 7 of Algorithm 1, many different algorithms have been proposed. For a comparative study of a selection of these algorithms, we refer to (Kuchaki Rafsanjani et al. (2012)). According to this study, the time complexity of steps 2 to 7 is $O(N^3)$, where N is the total number of observations in the data. This time complexity can be reduced to $O(N^2 \log N)$ by using priority queues. For Single linkage and Complete linkage, the time complexity can even be further reduced to $O(N^2)$. If we also take into account step 1 of Algorithm 1, this time complexity changes, based on the distance measure that is chosen. Note that generally the time complexity of step 1 is significantly higher than the time complexity of steps 2 to 7. This has an advantage in practice; once the distance matrix is computed, an agglomerative hierarchical clustering is often quickly obtained. This last property also makes it easy to try different methods for defining the linkage between clusters.

3.1.2 Divisive clustering

Divisive clustering is a top-down approach that starts with one cluster containing all observations and, at each step, splits a cluster until only singleton clusters of individual observations remain. Since the total number of possible splits is of $O(2^N)$, an exact algorithm for divisive clustering quickly becomes too time expensive. Therefore, different heuristics have been developed, such as DIANA (Kaufman and Rousseeuw (1990)) and Bisecting K-means (Cimiano et al. (2004)). These heuristics have time complexities similar to agglomerative clustering algorithms. Since divisive clustering is rarely used in practice and since we found no indication in the literature that it outperforms agglomerative clustering, we do not further address these heuristics in this thesis.

3.2 Partitional clustering

Partitional clustering is a type of clustering where all observations in the data are partitioned into k different clusters. Here, the number k has to be specified beforehand. It is stated by Sarda-Espinosa (2017) that partitional procedures can be regarded as combinatorial optimization problems, which aim at minimizing intracluster distance while maximizing intercluster distance. Finding a global optimum would require trying all possible clusterings, which is infeasible even for small data sets. Therefore, several heuristics for finding local optima are developed. In this subsection, we discuss four of these heuristics. Two of them, k-means and k-medoids, are commonly used partitional algorithms that build clusters around the means and medoids of observations, respectively. We also discuss two variations on k-medoids that are built for handling large data sets: CLARA and CLARANS.

3.2.1 *k*-means

k-means is a clustering method that aims to partition a set of numerical vectors (observations) into k clusters. Even though it was first proposed in 1955, it is still one of the most commonly used clustering methods (Steinhaus (1956); Jain (2010)). The general k-means algorithm can be found in Algorithm 2. In the next paragraphs we discuss the different steps of this algorithm.

Algorithm 2 k-means clustering algorithm (Forgy/Lloyd).

```
 1: Decide on a value $k$.
 2: Initialize $k$ cluster centers $\mu_1, ..., \mu_k$.
 3: while stopping condition not met do
 4:     for $i = 1 : N$ do
 5:         $c_i := \arg\min_l d(x_i, \mu_l)$
 6:     end for
 7:     for $j = 1 : k$ do
 8:         $\mu_j := \frac{\sum_{i=1}^N 1\{c_i=j\}x_i}{\sum_{i=1}^N 1\{c_i=j\}}$
 9:     end for
10: end while
11: return $c_1, ..., c_N$ and $\mu_1, ..., \mu_k$.
```

The first step is to decide on a value k. For this we refer to Section 4.

Once the number k is decided, the next step is to initialize k cluster centers. Two commonly used methods for this initialization are the Forgy method and the Random Partitioning method (Hamerly and Elkan (2002)). The Forgy method randomly selects k vectors (observations) from the set of observations and takes these as the initial centers. The Random Partitioning method randomly assigns each observation to one of the k clusters and then takes the means of the individual clusters as the initial centers. To avoid finding local optima when applying k-means, some implementations of the algorithm consider multiple (random) initializations. For an extensive overview and comparison of initialization methods we refer to Celebi et al. (2013).

The third step in the k-means algorithm is the while-loop that runs until the stopping criteria is met. Two commonly used stopping criteria are convergence and maximum number of iterations. Convergence can take multiple forms, for example when no observations are assigned a different cluster center (see step 5) or when the variance did not improve by at least a prespecified margin. Within the while-loop in step three, two for-loops can be found. The first for-loop (steps 4-6)

assigns to each observation x_i a label c_i that indicates the cluster whose center has minimum distance to the observation. Here the distance is usually taken to be the Euclidean distance. Other lock-step distance measures can be applied as well, however these generally do not give better results (Singh et al. (2013)). Furthermore, it is shown that elastic measures, such as DTW, generally do not give meaningful results when they are applied in k-means clustering (Niennattrakul and Ratanamahatana (2007)). This is caused by the fact that for these distance measures the triangle inequality does not hold. Note that the distances are calculated within the algorithm and thus no distance matrix has to be supplied for the k-means algorithm.
After the observations are assigned a cluster in the first for-loop, the second for-loop (steps 7-9) determines the new cluster centers. Observe that the centers are obtained by taking the average of all observations in a given cluster.
Once the stop condition for the while-loop is met, several sets of variables can be returned. We only included the two most important sets, which are the final cluster assignment $c_1, ..., c_N$ and the cluster centers $\mu_1, ..., \mu_k$.

We now consider the time complexity of Algorithm 2. Note that the first for-loop requires $O(Nkn)$ calculation time, where N is the number of n-dimensional observations and k is the number of clusters. The second for-loop has a time complexity of $O(Nk)$. Both for-loops are repeated in the while-loop for a number of iterations I, giving the whole algorithm a time complexity of $O(INkn)$. This time complexity is sometimes considered to be "linear", due to $I, k, n \ll N$. It must be noted that Algorithm 2 does not guarantee finding the optimal k-means solution, due to the possibility of local minima to be found. It is shown that if one does want to find the optimal solution, that this problem is NP-hard in general Euclidean space even for two clusters (Garey et al. (1982)). Therefore, heuristics are used instead.

Different variations of the heuristic in Algorithm 2 are proposed in the literature. Three commonly used k-means algorithms are the Hartigan-Wong algorithm (Hartigan and Wong (1979)), the Forgy/Lloyd algorithm (Lloyd (1982); Forgy (1965)) and the MacQueen algorithm (Macqueen (1967)). These are also the algorithms that are implemented in the *kmeans* function of the **stats** package in R. We note that while this function makes a distinction between the Forgy algorithm and the Lloyd algorithm, the same algorithm is applied when any of the two algorithms is called.

Algorithm 2 is the pseudo-code for the Forgy/Lloyd algorithm. This algorithm is a batch algorithm, where in every iteration the closest cluster center is recomputed for every observation. This property makes the Forgy/Lloyd algorithm well suited for large data sets. Two disadvantages of the Forgy/Lloyd algorithm are that it converges slower compared to the other two algorithms and that it has a possibility to create empty clusters (Morissette and Chartier (2013)).

The MacQueen algorithm is very similar to the Forgy/Lloyd algorithm. The main difference is that the MacQueen algorithm updates the centers every time an observation is assigned a different cluster, whereas the Forgy/Lloyd algorithm only updates the centers after all observations have been assigned a cluster. If we look at Algorithm 2, this means that an additional step is added between steps 5 and 6 that recalculates the two centers that are impacted if c_i changes in step 5. This adjustment allows faster initial convergence and usually only needs to iterate over all observations once to converge on a solution (Morissette and Chartier (2013)). It must be noted, though, that the final solution is sensitive to the order of the observations. Similar to the Forgy/Lloyd algorithm, the MacQueen algorithm (when using Euclidean distance) optimizes based on the total sum of squares.

The Hartigan-Wong algorithm optimizes based on a different criteria, namely the within-cluster sum of squares. This means that an observation may be assigned to a different cluster, even if it already belongs to the cluster with the closest center. Within the while-loop, the Hartigan-Wong algorithm iterates over the different observations and calculates for every observation the SSE within its cluster. Next, for every other cluster the SSE is calculated for the case that the observation is added to that cluster. If the SSE for any of the clusters is lower than the SSE of the cluster that the observation is currently in, the observation is swapped to the corresponding cluster. Just like the MacQueen algorithm, the Hartigan-Wong algorithm has fast initial convergence and is sensitive to the order of the observations.

A comparative research between the three algorithms in Slonim et al. (2013) concludes that all three algorithms are trivial to implement and similar in time complexity. Furthermore, they conclude that the Hartigan-Wong algorithm performs better in general compared to the other two algorithms. This is also stated in the *kmeans* function of the **stats** package in R, which is why the Hartigan-Wong algorithm is the default algorithm used in this package.

3.2.2 k-medoids

k-medoids is a clustering method related to k-means in the sense that its objective is to partition the data into k sets. The main difference between k-means and k-medoids is that in k-medoids observations are taken as medoids (centers). k-medoids is an NP-hard optimization problem and thus (just like for k-means) heuristics are used to obtain k-medoids partitions.

The most popular heuristic for k-medoids is the Partitioning Around Medoids (PAM) algorithm (Kaufman and Rousseeuw (1987)). The pseudo-code for this algorithm can be found in Algorithm 3. Note that the initialization of the algorithm is similar to that of Algorithm 2. Within the while-loop, however, the steps are different. Here swaps are considered in an iterative manner and if a swap decreases the total sum of distances between all observations and their closest medoid, the swap is made. While k-means may fail to converge for some distance measures, one can use k-medoids with any distance measure. Also, only the distances between the observations are needed for the algorithm (since the medoids are also observations). These can thus be determined prior to running the algorithm.

Algorithm 3 Partitioning Around Medoids (PAM) algorithm.

```
Decide on a value $k$.
Randomly choose $k$ observations as the initial medoids
while no change in the centroid assignment do
    for each medoid $c$ do
        for each non-medoid observation $o$ do
            if swapping $c$ and $o$ improves the solution then
                Swap $c$ and $o$
            end if
        end for
    end for
end while
return the $k$ medoids and the cluster assignment.
```

We now examine the time complexity of the PAM algorithm. Observe that in every iteration, for all k medoids, $(n-k)$ swaps are considered. Calculating the swapping cost for any of these swaps has time complexity $O(n-k)$. This gives every iteration in the PAM algorithm a time complexity of $O(k(N-k)^2)$ and thus the whole algorithm has a time complexity of $O(Ik(N-k)^2)$. Here I is the number of iterations, k the number of medoids (clusters) and N the total number of observations. Note that this time complexity does not include the time complexity of determining the distance matrix.

The main advantage k-medoids has over k-means is that it is more robust in the presence of noise and outliers (Kaufman and Rousseeuw (2008)). Also, being able to use any distance measure and calculate the distance matrix in advance is an advantage. A disadvantage k-medoids has over k-means is that it comes with a higher time complexity, scaling even worse when N increases.

3.2.3 Clustering LARge Applications (CLARA)

In the previous subsection, we have seen that the PAM algorithm for k-medoids clustering does not scale well for large data sets. To overcome this problem, Kaufman and Rousseeuw (1990) proposed a sampling-based method called Clustering LARge Applications (CLARA). This algorithm draws S random samples of observations of size z from the data set and applies PAM to each sample. After PAM is applied to a sample, the $N-z$ observations that are not in the sample are assigned to their closest medoid to obtain a clustering. In total, this thus gives S different clusterings. The best of these clusterings is returned by CLARA. The biggest strength of CLARA is that it deals with larger data sets than PAM, while still sharing the robustness property of the k-medoid framework. However, its solutions depend on the sampling, which is a trade-off between efficiency and clustering quality. The pseudo-code for the CLARA algorithm can be found in Algorithm 4.

Algorithm 4 Clustering LARge Applications (CLARA) algorithm.

```
Decide on values for $S, z$ and $k$.
for $s = 1 : S$ do
    Draw $z$ random observations from the data set to create a sample.
    Apply PAM on the sample.
    Assign the $N - z$ observations that are not in the sample to their closest medoid.
    Calculate the total sum of distances between all observations and their closest medoid.
    if total sum of distances is lowest found so far then
        Update the best clustering assignment to be the current assignment.
    end if
end for
return the best clustering assignment.
```

The time complexity for every iteration in Algorithm 4 is $O(kz^2 + k(N-k))$ (Sagvekar et al. (2013)). Since there are S samples and thus S iterations, the total time complexity is $O(kSz^2 + kS(N-k))$. The quality of a CLARA solution depends on the values that are chosen for S, z and k. Decreasing the values for S and z allows faster computations, but at the cost of a decrease in cluster quality. Experiments have reported that $S = 5$ and $z = 40 + 2k$ give satisfactory results, which is why in some literature these values are taken as default and the time complexity of CLARA is only given in terms of k and N (Sagvekar et al. (2013), Hesabi et al. (2015)). Note that it generally holds that $k, S, z \ll N$, making CLARA more efficient than PAM for large values of N. Also, not all distances between the observations in the data set are required in CLARA. For this reason, CLARA does not require the computation of a distance matrix of all observations in the data set.

3.2.4 Clustering Large Applications based on RANdomized Search (CLARANS)

CLARANS is a clustering technique proposed by Ng and Han (1994) that is based on the PAM and CLARA algorithms. Just like CLARA, it is a sampling-based method. However, instead of sampling different sets of observations, CLARANS dynamically samples from the solution space. This solution space contains $\binom{N}{k}$ different solutions, where every solution is a set of k medoids. To explain the CLARANS algorithm, we represent the solution space by a graph, where every node is a potential solution (set of k medoids). Two nodes in the graph are adjacent if they differ by one medoid. An example of this graph representation can be found in Figure 16. Note that every medoid has $n - k$ neighbours and thus every node (solution) has $k(n - k)$ neighbours.

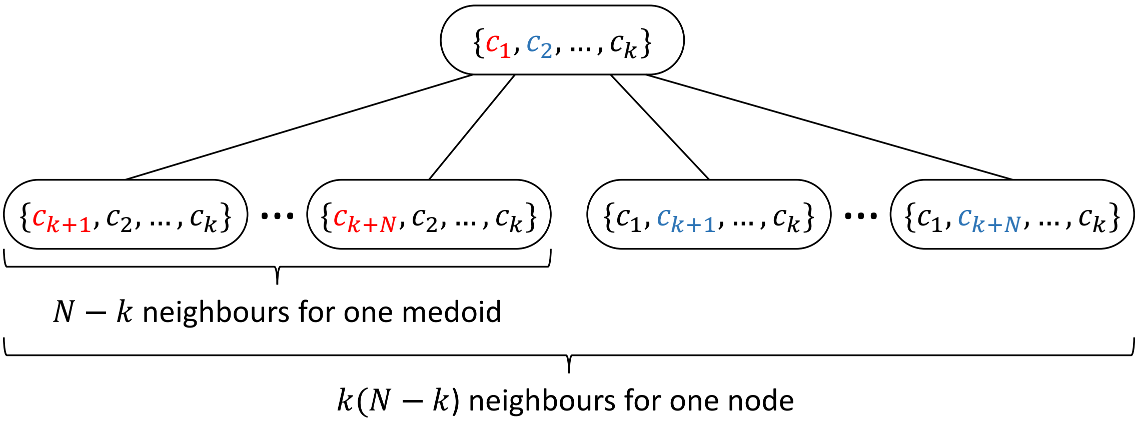

$\{c_1, c_2, \ldots, c_k\}$
$\{c_{k+1}, c_2, \ldots, c_k\}$
$\{c_{k+N}, c_2, \ldots, c_k\}$
$\{c_1, c_{k+1}, \ldots, c_k\}$
$\{c_1, c_{k+N}, \ldots, c_k\}$
$N - k$ neighbours for one medoid
$k(N - k)$ neighbours for one node

Figure 16: Example of the graph representation of the solution space that is used in the CLARANS algorithm.

The main idea behind the CLARANS algorithm is that for h random sets of k medoids it applies local search to find a local minimum for the costs, after which the best local minimum is returned. Here the cost of a solution is defined by the total sum of distances between all observations and their closest medoid.

The pseudo-code for the CLARANS algorithm can be found in Algorithm 5. As for PAM and CLARA, a value for k has to be decided on in advance. Furthermore, one has to decide on values for v and h. v is an integer that represents the maximum number of random neighbours with which each solution is compared and h is an integer that represents the total number of solutions to be sampled. Choosing the values for v and h is a trade-off between efficiency and clustering quality; higher values increase the search space and thus potentially improve the solutions, but at the cost of more computation time. Based on experiments, Ng and Han (1994) indicate that taking $h = 2$ and $v = \max\{250, 0.0125k(N - k)\}$ provides a good balance between running time and quality of the solutions.

Algorithm 5 Pseudo-code for the CLARANS algorithm.

```
Decide on values for $v, h$ and $k$.
for $i = 1 : h$ do
    Randomly select a node as the current node $C$ in the graph.
    Calculate the total sum of distances for current node $C$.
    Set counter $j$ to 1.
    while $j \leq v$ do
        Randomly select a neighbour $W$ of $C$.
        Calculate the total sum of distances for neighbour $W$.
        if total sum of distances for $W$ is lower than for $C$ then
            Assign $W$ as the current node $C$.
            Reset $j$ to 1.
        else
            $j = j + 1$.
        end if
    end while
    if total sum of distances for $C$ is lowest cost found so far then
        Update the best solution found so far.
    end if
end for
return the best solution that is found.
```

If one takes $h = 1$ and $v = k(N - k)$, essentially the PAM algorithm is obtained. However, we have seen that the PAM algorithm scales poorly with N, which is not desirable for large data sets. An advantage of CLARANS is that it is more scalable compared to PAM, as it allows v to be reduced to decrease the computation time. Ng and Han (1994) have shown that reducing the value of v makes CLARANS significantly faster compared to PAM, while still returning comparable clustering quality.

Ng and Han (1994) also compared CLARANS to CLARA. They indicated that CLARANS has an advantage over CLARA, as it does not confine the search space to a restricted area. In their experiments, CLARANS always found clusterings of better quality compared to those found by CLARA. In some cases, CLARANS took more time than CLARA. This is due to the time complexity of CLARANS, which is quadratic in N: $O(N^2)$ (Swarndeep Saket J (2016)). However, when the same calculation time is allowed for both algorithms, CLARANS is still better than CLARA. If the data set becomes too large, CLARANS may fail to enable clustering due to its quadratic time complexity (Hesabi et al. (2015)).

3.3 Comparing the clustering methods

We now compare the different hierarchical and partitional clustering methods that are discussed in this section. We conclude that there is no clear consensus in the literature on which of the two approaches produces better clusterings. However, there are some clear differences between the two approaches, which we will now address.

The main advantage hierarchical clustering has over partitional clustering, is that it does not require the number of clusters k as an input parameter. Instead, it constructs a hierarchy (tree) of clusters which can be cut to obtain clusterings for different values of k. Besides giving more freedom in choosing and adjusting the value of k, the hierarchical structure also gives more insight into the data compared to the 'flat' structure that is obtained in partitional clustering.

The main advantage partitional clustering has over hierarchical clustering, is that it does not (always) require the distance matrix to be computed. This advantage becomes more significant as the number of observations N grows, since the number of elements in a distance matrix grows with $O(N^2)$. If we do not consider the calculation time of the distance matrix, then still partitional clustering methods tend to outperform hierarchical clustering methods in terms of time complexity, as we can see in Table 6.

Table 6: Overview of the time complexities of the algorithms that are discussed in this section.

	Time complexity
Hierarchical clustering	
Agglomerative clustering[1]	$O(N^2 \log N)$
Divisive clustering[1]	$O(N^2 \log N)$
Partitional clustering	
k-means	$O(INkn)$
k-medoids	$O(Ik(N-k)^2)$
CLARA	$O(kSz^2 + kS(N-k))$
CLARANS	$O(N^2)$

For hierarchical clustering, one can choose between an agglomerative (bottom-up) approach or a divisive (top-down) approach. Here the divisive approach is rarely used, even though both approaches have heuristics with similar time complexities.

For partitional clustering, two commonly used methods are k-means and k-medoids. An advantage that k-means has over k-medoids is that it scales better with N. However, since it is designed for minimizing variance, k-means cannot handle arbitrary distances (in order to give meaningful results). For k-medoids, on the other hand, it is possible to use distance measures other than the Euclidean distance. However, k-medoids may fail to converge for some distance measures. Another advantage k-medoids has over k-means, is that it is more robust to outliers and noise.

CLARA and CLARANS are both variations of the PAM algorithm that use sampling methods to handle large data sets. According to the literature, CLARANS generally outperforms CLARA, but at the cost of a higher time complexity.

[1] The time complexity of the hierarchical methods does not include the time complexity of determining the distance matrix. Also, for Single linkage and Complete linkage in agglomerative clustering, the time complexity is only $O(N^2)$.

4 Determining the number of clusters

As we have seen in Section 3, many clustering methods require the number of clusters k as an input parameter in order to return a clustering. Nonhierarchical methods usually require k to be specified beforehand, whereas for hierarchical methods the value of k can be set afterwards. The number of clusters k is usually an unknown parameter, which the user has to specify based on prior knowledge (external cluster criterion, e.g. externally provided class labels) or based on an estimation (internal cluster criterion). Many different methods have been proposed to estimate the "optimal" number of clusters k^* for a clustering assignment.

Milligan and Cooper (1985) carried out a comparative study of 30 methods that estimate the number of clusters. Based on Monte Carlo simulations, they evaluated the performance of each method on artificial test data sets for which the optimal numbers of clusters are known in advance. The five methods that correctly estimated the number of clusters most frequently are (in order):

1. Calinski & Harabasz index
2. Je(2)/Je(1)
3. C index
4. Gamma index
5. Beale

In this section, we discuss seven different methods for estimating the number of clusters. These include the five methods above and two methods that we often encountered in implementations and articles on estimation methods: the Silhouette index and the Gap statistic.

An overview of the seven estimation methods that we discuss in the next subsections can be found in Table 7. Observe that for every method we indicate the category. The categories we consider are based on Gordon (1999), who made a distinction between global and local methods for determining the number of clusters. Global methods compare the criteria values for a range of k-values to determine the "optimal" number of clusters. Local methods determine the number of clusters by iteratively testing the hypothesis that a pair of clusters should be merged (or split). The remainder of this section will discuss the seven methods that are presented in Table 7. We first discuss the global methods, followed by the local methods. At last, a comparison between the different methods will be made in Subsection 4.3.

Table 7: Overview of the seven methods for estimating the number of clusters that we discuss in this section.

Estimation method	**Category**
Calinski-Harabasz index	Global
C index	Global
Gamma index	Global
Silhouette index	Global
Gap statistic	Global
Je(2)/Je(1) index	Local
Beale index	Local

4.1 Global methods

4.1.1 Calinski-Harabasz index

The Calinski-Harabasz index is proposed by Calinski and Harabasz (1974) and is calculated for every value of k by:

$$CH(k) = \frac{N-k}{k-1}\frac{BCSS(k)}{WCSS(k)}, \tag{22}$$

where N is the total number of observations, k the number of clusters, $BCSS$ the between cluster sum of squares and $WCSS$ the within cluster sum of squares.
The $BCSS$ is the weighted sum of differences between the cluster centroids and the overall centroid of the observations. Here the weights are the number of elements in each cluster. In Equation (23), the formula for the $BCSS$ can be found, where $|C_j|$ is the number of observations in cluster C_j, n the length of the observations, $\bar{c}_{lj}$ is the j-th index of the centroid of cluster l and $\bar{x}_j$ is the j-th index of the centroid of all observations.

$$BCSS(k) = \sum_{l=1}^{k}\sum_{j=1}^{n}|C_l|(\bar{c}_{lj} - \bar{x}_j)^2. \tag{23}$$

The $WCSS$ is the sum of the squared deviations from each observation and the cluster centroid the observation is in. In Equation (24), the formula for the $WCSS$ can be found, where C_l represents cluster l, n the length (number of dimensions) of the observations, x_{ij} the j-th index of observation i and $\bar{c}_{lj}$ the j-th index of the centroid of cluster l.

$$WCSS(k) = \sum_{l=1}^{k}\sum_{i\in C_l}\sum_{j=1}^{n}(x_{ij} - \bar{c}_{lj})^2. \tag{24}$$

The value of $CH(k)$ is analogue to the F-statistic in one-way ANOVA. For this F-statistic, it is known that $BCSS(k)$ has $k-1$ degrees of freedom and $WCSS(k)$ has $N-k$ degrees of freedom. This means that when k grows, $BCSS(k)$ should be proportional to $k-1$ and $WCSS(k)$ should be proportional to $N-k$. By scaling for those degrees of freedom as k grows, we obtain a measure for quantifying the effectiveness of the clustering where we minimize the distance within clusters and maximize the distance between clusters. Therefore, when applying the Calinski-Harabasz index, we want to find the value of k for which $CH(k)$ is maximized. The time complexity for computing the Calinski-Harabasz index is $O(nN)$ (Vendramin et al. (2010)).

4.1.2 C index

The C index is proposed by Hubert and Levin (1975) and is computed by using Equation (25).

$$C = \frac{S_w - S_{\min}}{S_{\max} - S_{\min}}, \tag{25}$$

where S_w is the sum of the N_w distances between all the pairs of observations within each cluster. For k clusters, the total number of pairs within each cluster is $N_w = \sum_{j=1}^{k}\frac{N_j(N_j-1)}{2}$, where N_j is the total number of observations in cluster j. $S_{\min}$ in Equation (25) is obtained by taking the sum of the N_w smallest distances between all the pairs of observations in the data set. Note that this also includes pairs that may not be within the same cluster. In total, there are $N_T = \frac{N(N-1)}{2}$ different pairs between all observations. $S_{\max}$ is obtained by taking the sum of the N_w largest distances between all the pairs of observations in the data set.

The "optimal" number of clusters is obtained by computing the C index for a range of values for k. The k that minimizes C indicates the optimal number of clusters. In terms of time and space complexity, it is indicated by Dimitriadou et al. (2002) that the C index can be prohibitive for large data sets. This is due to the fact that all pairwise distances have to be computed and stored. The total time complexity for determining the C index is $O(N^2(n + \log_2 N))$ (Vendramin et al. (2010)).

4.1.3 Gamma index

The Gamma index is proposed by Baker and Hubert (1975) and is an adaptation of the Gamma correlation statistic by Goodman and Kruskal (1954). The Gamma index can be calculated by Equation (26).

$$\Gamma = \frac{S_+ - S_-}{S_+ + S_-}, \tag{26}$$

where S_+ (S_-) represents the number of times a pair of observations that belong to the same cluster has a smaller (larger) distance than two observations that belong to different clusters. Note that value of the Γ can only take values in the range $[-1, 1]$. The formal definitions of S_+ and S_- can be found in Equations (27) and (28).

$$S_+ = \frac{1}{2}\sum_{l=1}^{k} \sum_{\substack{q,r \in C_l \\ q \neq r}} \frac{1}{2}\sum_{h=1}^{k} \sum_{\substack{s \in C_h \\ t \notin C_h}} \delta(d(q,r) < d(s,t)) \tag{27}$$

$$S_- = \frac{1}{2}\sum_{l=1}^{k} \sum_{\substack{q,r \in C_l \\ q \neq r}} \frac{1}{2}\sum_{h=1}^{k} \sum_{\substack{s \in C_h \\ t \notin C_h}} \delta(d(q,r) > d(s,t)) \tag{28}$$

where q, r, s and t are observations. Here q and r are assigned to the same cluster C_l and s and t are both assigned to different clusters. Furthermore, $\delta(\cdot) = 1$ if the inequality is satisfied and $\delta(\cdot) = 0$ otherwise. The factors $\frac{1}{2}$ are added to avoid counting pairs twice. Note that both Equations (27) and (28) are strict; the cases where the distances between pairs (q, r) and (s, t) are equal are not taken into account.

A good clustering (partitioning) is expected to have higher values of S_+, lower values of S_- and thereby higher values of Γ in Equation (26) (Vendramin et al. (2010)). Therefore, to obtain the "optimal" number of clusters according to the Gamma index, one should determine Γ for a range of values of k and select the k that maximizes the index. The time complexity for determining the Gamma index is estimated to be $O(nN^2 + \frac{N^4}{k})$ by Vendramin et al. (2010), making it a computationally expensive index for large values of N.

4.1.4 Silhouette index

The Silhouette index is proposed by Kaufman and Rousseeuw (1990) and is based on compactness and separation of clusters. It starts by measuring the silhouette of every observation i:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \tag{29}$$

where $a(i)$ is the average distance of observation i to all other observations in the same cluster. To calculate $b(i)$, we first calculate for every cluster, except for the one i is in, the average distance between observation i to all observations in the corresponding cluster. The value of $b(i)$ is now obtained by taking the minimum value of these average distances. The denominator in Equation (29) serves as a normalization term, scaling the value of $s(i)$ in the interval $[-1, 1]$. A silhouette value close to 1 indicates that the observation is assigned to the appropriate cluster, whereas a silhouette value near -1 indicates that the observation should be assigned to another cluster.

To obtain an index that describes the performance of a clustering, the overall average silhouette width can be calculated by taking the average of all individual silhouette values: $SWC = \frac{1}{N}\sum_{j=1}^{N} s(j)$. The "optimal" value of k, according to the Silhouette index, is the k that maximizes the value of silhouette width criterion SWC. The time complexity for determining the SWC is $O(nN^2)$ (Vendramin et al. (2010)).

4.1.5 Gap statistic

The Gap statistic is a measure that estimates the number of clusters by comparing the within-cluster dispersion with the dispersion that is expected under an appropriate null distribution. It is proposed by Tibshirani et al. (2000) and is designed to work for any clustering technique and distance measure. Below we explain the Gap statistic and apply the measure in three different examples that can be found in Appendix D.

The first step is to determine the within-cluster dispersion W_k for a range of values of k. The definition of W_k can be found in Equation (30).

$$W_k = \sum_{l=1}^{k} \frac{1}{2|C_l|} \sum_{q,r \in C_l} d(q, r), \tag{30}$$

where $|C_l|$ is the number of observations in cluster l and $d(q, r)$ is the distance between observations q and r. One could use W_k to determine the number of clusters by plotting the value for different numbers of clusters k. The "optimal" number of clusters is the k where after we were to add another cluster, the explanatory power of the clustering barely improves. This k acts as an 'elbow' in the graph. Hence this method is also called the elbow method. While the elbow method may indicate the correct optimal number of clusters, the k that acts as an "elbow" cannot always be clearly identified (Ketchen and Shook (1996)). Therefore, the Gap statistic has some additional steps to determine the number of clusters.

The second step is to generate B reference data sets from an appropriate reference null distribution. Tibshirani et al. (2000) indicated that usually the uniform distribution on the interval of the original data is an appropriate choice. The number of observations in every reference data set should be equal to the number of observations in the original data set. The precision of the Gap statistic improves as B increases, but implementations of the statistic suggest that usually 100 reference data sets are sufficient (Maechler et al. (2017)).

The third step is to determine the Gap statistic by using Equation (31).

$$Gap_k = \frac{1}{B}\sum_{b=1}^{B} \log(W_{kb}) - \log(W_k), \tag{31}$$

where W_{kb} is the value of W_k when using the distances from reference data set b. One can plot the values of Gap_k for different values of k and look for a maximum in order to determine the number of clusters. However, to add more certainty to this estimation, Tibshirani et al. (2000) included the standard deviation of the reference data sets into the estimation.

The fourth step is to determine the standard deviation sd_k of the values of $\log\{W_{kb}\}$:

$$sd_k = \sqrt{\frac{1}{B}\sum_{b=1}^{B}(\log(W_{kb}) - \bar{l})}, \tag{32}$$

where $\bar{l} = \frac{1}{B}\sum_{b=1}^{B}\log(W_{kb})$. A scaled version of this standard deviation, $s_k = \sqrt{1+\frac{1}{B}}sd_k$, is used in the final step for determining the number of clusters. In this final step, the "optimal" k is determined by finding the smallest k such that $Gap_k \geq Gap_{k+1} - s_{k+1}$. This last step is also referred to as the 1-standard-error method.

In Appendix D, we apply the Gap statistic on instances with 3, 5 and 1 cluster(s), respectively. From these examples we can confirm that the elbow method may not always be conclusive. This especially holds for cases where clusters lay close to each other (see Figure 26) or where there is no clear clustering structure (see Figure 27). We observe that the Gap statistic is still able to identify the correct number of clusters in all three examples.

At last, we discuss the time complexity of the Gap statistic. As with the analysis that is presented by Vendramin et al. (2010), we assume that computing the centroids takes $O(nN)$ time. Note that this time complexity may differ, depending on the clustering method that is chosen. Computing W_k has a time complexity of $O(nN)$, which becomes $O(BnN)$ when performed for all reference data sets B. Calculating the Gap_k value and its standard deviation both takes $O(B)$ time. In total, we can conclude that the Gap statistic has a time complexity of $O(BnN)$ time.

4.2 Local methods

4.2.1 Je(2)/Je(1) index

The Je(2)/Je(1) index, also referred to as the Duda index or Duda-Hart index, is a local method proposed by Duda and Hart (1973). It is an iterative method that determines the "optimal" number of clusters for a given hierarchical clustering. It does so by following the dendrogram of the hierarchical clustering (see Subsection 3.1), starting with all data in one cluster and increasing the number of clusters by one during every iteration. In every iteration, the method determines which cluster is the next cluster to be partitioned according to the dendrogram. For this cluster $\mathcal{D}$, the sum of squared errors is calculated, indicated by $Je(1)$:

$$Je(1) = \sum_{x\in\mathcal{D}} \parallel x - \bar{x} \parallel^2, \tag{33}$$

where $\bar{x}$ is the mean of all observations x in cluster $\mathcal{D}$. It also calculates the sum of squared errors of the two clusters $\mathcal{D}_1$ and $\mathcal{D}_2$ that would be created if cluster $\mathcal{D}$ were to be partitioned, indicated by $Je(2)$:

$$Je(2) = \sum_{i=1}^{2}\sum_{x\in\mathcal{D}_i} \parallel x - \bar{x}_i \parallel^2, \tag{34}$$

where $\bar{x}_i$ is the mean of all observations x in cluster $\mathcal{D}_i$. The "optimal" number of clusters is now determined by finding the smallest k such that

$$\frac{Je(2)}{Je(1)} \geq 1 - \frac{2}{\pi n} - z\sqrt{\frac{2\left(1-\frac{8}{\pi^2 n}\right)}{nN_{\mathcal{D}}}} = I_{Duda}, \tag{35}$$

where n is the number of dimensions of the data, $N_{\mathcal{D}}$ is the number of observations in cluster $\mathcal{D}$ and z is the cutoff value from a standard normal distribution specifying the significance level (Yan (2005)). Previous research on the value of z indicate that the best results are obtained when $z = 3.20$ (Milligan and Cooper (1985)).

At last we discuss the time complexity of the Je(2)/Je(1) index. We consider the case where the algorithm terminates when a value for k is found that causes the Je(2)/Je(1) index to be greater than or equal to I_{Duda}. In the worst case scenario, the algorithm has to determine Je(2)/Je(1) for N different clusters, where at every j-th partition there are $k = j$ clusters and $N - k$ observations in the cluster that has to be partitioned. Both $Je(1)$ and $Je(2)$ have a time complexity of $O(nN_i)$, where N_i is the number of observations in the cluster that has to be partitioned in the i-th iteration. Observe that in the worst case scenario we have that $N_1 = N, N_2 = N-1, N_3 = N-2, ..., N_N = 1$, where all the values of N_i sum up to $\frac{N(N+1)}{2}$. The total algorithm for determining the Je(2)/Je(1) index thus has a time complexity of $O(nN^2)$.

Some implementations of the Je(2)/Je(1) index do not run until a value k is found for which $\frac{Je(2)}{Je(1)} \geq I_{Duda}$, but instead calculate the index for a range of values k (Charrad et al. (2014)). This approach resembles the way in which global methods work. In this case it improves the time complexity to $O(knN)$ (or $O(nN)$ for every value of k that is considered), but it does not guarantee that an index value is found that is greater than or equal to I_{Duda}.

4.2.2 Beale index

The Beale index, proposed by Beale (1969), determines the "optimal" number of clusters very similarly to the Je(2)/Je(1) index. It is also defined only for hierarchical clustering and it follows the partitions in the order in which they occur in the dendrogram of the clustering. The only difference is the index that is considered at every partitioning in the algorithm. The index that is used for the Beale index is an F-statistic that is defined in Equation (36). This index uses $Je(1)$, $Je(2)$ and $N_{\mathcal{D}}$ as they are defined in Subsection 4.2.1.

$$F \equiv \left(\frac{Je(1) - Je(2)}{Je(2)} \right) \Bigg/ \left(\left(\frac{N_{\mathcal{D}} - 1}{N_{\mathcal{D}} - 2} \right) 2^{\frac{2}{n}} - 1 \right). \tag{36}$$

The "optimal" number of clusters is obtained by comparing F with an $F_{n,(N_{\mathcal{D}}-2)n}$ distribution. The null distribution that the cluster should not be partitioned is rejected is F is significantly large. Milligan and Cooper (1985) indicated that a significance level of 0.005 gives the best results. The time complexity of the Beale index is similar to the time complexity of the Je(2)/Je(1) index.

4.3 Comparing the indices

We now compare the different indices that we discussed in this section based on applicability, computation time and other properties. In Table 8, an overview can be found on for which distance measure(s) and clustering method(s) the seven methods that we discussed are applicable. Note that both local methods are limited to only the Euclidean distance and hierarchical clustering, while most global methods are applicable for any distance measure or clustering method. The only exception is the Calinski-Harabasz index, which is limited to using the Euclidean distance. Furthermore, we must note that the Calinski-Harabasz index, the C index and the Gamma index are not defined for $k = 1$.

Table 8: Overview of the distance measures and clustering methods that are applicable for all seven methods for estimating the number of clusters that we discuss in this section.

Estimation method	Category	Distance measure(s)	Clustering method(s)
Calinski-Harabasz index	Global	Euclidean	Any
C index	Global	Any	Any
Gamma index	Global	Any	Any
Silhouette index	Global	Any	Any
Gap statistic	Global	Any	Any
Je(2)/Je(1)	Local	Euclidean	Hierarchical
Beale	Local	Euclidean	Hierarchical

Even though the Calinski-Harabasz index has some limitations (see above), it is still one of the most widely used indices. This is due to its good performance according to previous research (e.g. Milligan and Cooper (1985)) and also its time complexity. In Table 9, an overview of the time complexities and criteria of the seven indices that we discussed can be found. The time complexities of the global methods do not account for the number of values of k for which the index has to be determined. We note that in most clustering assignments $n, k \ll N$. Observe that while most indices have a quadratic (or worse) time complexity in terms of the number of observations N, the Calinski-Harabasz index and Gap statistic both have a time complexity that is linear in N. However, the time complexity of the Gap statistic may still be proportional due to the choice of B. Also, we recall that the local methods can also be used in a global manner, making the time complexity of every iteration linear in N: $O(nN)$. The time complexity of the Gamma index appears to be the worst, as it has a term N^4 in it.

Table 9: Overview of the time complexity and criteria for all seven methods for estimating the number of clusters that we discuss in this section. n is the number of dimensions of the observations, N is the total number of observations, k is the number of clusters for which the Gamma index has to be calculated and B is the number of reference data sets that are generated for the Gap statistic.

Estimation method	Time Complexity	Criteria
Calinski-Harabasz index	$O(nN)$	Maximum value of index $CH(k)$
C index	$O(N^2(n + \log_2 N))$	Minimizing value of index C
Gamma index	$O(nN^2 + \frac{N^4}{k})$	Maximizing value of index Γ
Silhouette index	$O(nN^2)$	Maximizing value of index SWC
Gap statistic	$O(BnN)$	$\arg\min_k \{Gap_k \geq Gap_{k+1} - s_{k+1}\}$
Je(2)/Je(1) index	$O(nN^2)$	$\arg\min_k \{Je(2)/Je(1) \geq I_{Duda}\}$
Beale index	$O(nN^2)$	Smallest k for which H_0 not rejected

If we look at the criteria for the different indices, we can conclude that all indices that we discussed are built, in some way, around minimizing the within cluster distance and/or maximizing the between cluster distance. While this kind of similarities between indices can be found, Milligan and Cooper (1985) concluded that no general statement of cause can be given behind the performance of the different indices. They also concluded that the performance of an index depends on the data which it is used for.

In order to obtain a reliable estimation of the "optimal" number of clusters, it is suggested by Yan (2005) to apply several different indices instead of a single one. Charrad et al. (2014) used this idea to built a method that applies up to 30 different indices, after which the majority rule is applied to determine the "optimal" number of clusters. If most methods agree on a certain number of clusters, there is a good indication of a clear clustering structure in the data. If there is no or barely any agreement between the indices, Milligan (1996) commented that this might indicate that there is no significant cluster structure in the data. Note that while using multiple indices improves the reliability, the total computation time of the indices may become a problem for large data sets.

For time series clustering, we have seen in Section 2 that lock-step distance measures, such as the Euclidean distance, are often outperformed by elastic distance measures. To obtain a good estimation of the "optimal" number of clusters for time series clustering, we can thus conclude that indices that allow elastic distance measures are more reliable. This favours the C index, Gamma index, Silhouette method and Gap statistic over the other three indices that we discussed in this section.

5 Practical research

In the previous sections, we discussed and compared several methods for measuring similarity, obtaining clusterings and determining the number of clusters. In this section, we put a selection of these methods to the test by applying them to an actual time series data set. We do this in R (R Core Team (2016)), where we implement a selection of methods and compare them based on computation time and performance.

While it is easy to measure the performance of supervised learning algorithms, such as algorithms for classification problems, it is often hard to measure the performance of unsupervised learning algorithms, such as clustering algorithms. The reason for this, is that it is subjective what makes a clustering 'good'. The performance of a clustering depends on the goal and criteria of the clustering and may therefore differ per application.

Our initial approach was to quantify clustering quality by using classification data sets. For this, a selection of time series data sets were taken from the repositories of Dheeru and Karra Taniskidou (2017) and Anthony Bagnall and Keogh (2016). Here the idea was to compare the classification labels in the obtained clusters to see to what extent the clustering algorithms could create clusters of similarly labeled data. However, this approach appeared not to be feasible. First of all, it turned out to be a hard task to determine which label belongs to which cluster, as most clusters did not appear to have a dominant (common) label. Furthermore, overlaying the similarly labeled time series did not visually show logical groupings, whereas overlaying the time series within the clusters did. This result emphasizes the power of clustering, as it indicates that clustering can find better groupings compared to the groupings that are predefined by the labels. However, it also shows that our approach is not feasible for quantifying the quality of clustering algorithms.

To still be able to test clustering algorithms and quantify their performance, we set up an experiment with a clear clustering goal. This goal is to use clustering to obtain clusters that can be used for forecasting. Instead of making forecasts for all individual time series, only one forecast will be made for each cluster. This forecast is then applied to all time series in the corresponding cluster. To measure the quality of the clusterings, we take the error between the forecasts and the actual time series.

A practical example of the setup above could be a supplier who wants to determine future stocking levels. Making forecasts for all individual time series (products) may then be infeasible, as it could induce noise or require too much computation time. Using clustering instead could overcome these problems.

In the upcoming subsections, we further explain the setup of our experiment and present our findings. We start with Subsection 5.1, where we discuss the time series data set that is used for our experiment. Here we also discuss the preprocessing that is applied to our data set. Next, in Subsection 5.2, we discuss the R implementations and running times of all seven distance measures that we discussed in Section 2. These measures are applied to the time series data set in order to obtain the corresponding distance matrices, which are used in some of the clustering algorithms in this experiment. To determine the number of clusters in which the data should be partitioned, in Subsection 5.3 we apply the seven indices that we discussed in Section 4. At last, in Subsection 5.4, we further explain our setup to quantify clustering quality based on forecasting. We also present our results on the computation times and performance of the different algorithms for obtaining clusterings.

5.1 Data

In this subsection we discuss the time series data set that we use in our practical research. We start by giving some general background on the data in Subsection 5.1.1. Note that due to the confidentiality of the data, we can only provide limited information. In Subsection 5.1.2 we discuss how we filtered and preprocessed the data in order to obtain a suitable subset for our experiment.

5.1.1 Data description

The data that we use in this practical research originates from a large retailer, who is a client of PwC. The data has been used for a previous project PwC did for this client, however no (time series) clustering is performed during this project. Due to confidentiality, we cannot mention the name of the client and also we are limited in the information we can share on this data set.

The data set contains over one million rows of sales data, where each row represents one product in a given week. The unique key in the data is therefore formed by the article number in combination with the week number. The data in each row contains, among other things, information on the total number of sales, revenue and gross profit of the product during a specific week. In total, the data includes over 10,000 different products. The data ranges from week one of 2012 until week 29 of 2015, having a total time span of 185 weeks. There is thus a maximum of 185 rows of data available for each product.

5.1.2 Data preprocessing

As mentioned in Subsection 5.1.1, the data contains weekly time series data on three variables: sales, revenue and gross profit. Based on a data analysis on these variables, we conclude that the variable sales is the 'cleanest' out of the three variables. The reason for this is that the revenue and gross profit variables in some weeks contain negative values, while there is a positive number of sales. This is presumably caused by items that are returned during the corresponding week. For this experiment we therefore choose to only consider the weekly sales data of each product.

Initial tests on the data indicate that using the sales time series of all products comes with a large time complexity, where it could take hours up to even days to compute some distance matrices. Also, we observe that there are many products with zero or barely any sales during most of the weeks. Due to the time complexity and the fact that clusters of time series around zero are not very meaningful, we decided to only work with a subset of the data. In this subset we only take all products that have a positive number of sales in all 185 weeks. The subset that results from this filtering contains 2,381 sales time series. All analyses in the next subsections are applied only to this subset.

Before moving on to the actual clustering analysis, it is important to decide on which preprocessing should be applied to the data. Depending on the goal of the clustering, it may be desired to, for example, scale, detrend or deseasonalize the data in advance. In our case, we decided not to detrend or deseasonalize the data, since we want the clusters to capture this behavior instead. However, we do have to apply scaling, as the sales time series have a variety of ranges. Two common ways of scaling are normalization and standardization. In normalization, the data is scaled into a range of $[0, 1]$ by using the following formula:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}.$$

In standardization, the data is scaled to have mean 0 and a standard deviation of 1. This is done by using the following formula:

$$X_{std} = \frac{X - \mu}{\sigma},$$

where μ represents the mean and σ the standard deviation of all values in vector X. Both normalization and standardization have some drawbacks. In normalization, if there are outliers in the data, the 'normal' part of the data will be scaled to a small interval. In standardization, a drawback is that the scaled data is unbounded (unlike normalization). A suitable scaling should therefore be chosen depending on the goal of the clustering and the structure of the time series data. In our case, we choose to apply normalization on the data. This is because we want to cluster based on similar shapes in the time series rather than similar variances (which is the case for standardization). Therefore, each of the 2,381 individual time series is scaled to the interval $[0, 1]$.

Since our goal is to quantify clustering quality by comparing cluster forecasts with the actual sales data, we need to split the data into a training set and a test set. The training set will be used to obtain the clusterings and also to train the forecasting model on. The test set will be used to compare the actual sales values with the forecasted sales values. We decided to split our data at two years, which results in the training set containing the first 104 weeks (years 2012 and 2013) of each time series and the test set containing the remaining 81 weeks (2014 and a part of 2015). This enables the clusters to capture yearly trends and seasonality, while still having a large time span to forecast on.

5.2 Computing the distance matrices

In this subsection we discuss the R implementations of all seven distance measures that we discussed in Section 2. For each of the distance measures, we calculate the corresponding distance matrix of the time series data and measure the required computation time. We then compare the distance measures based on their computation times, to see whether these times are in line with the time complexities discussed in Section 2 and also to get a general idea on how the computation times scale with the number of time series N. All computations are performed on a Lenovo ThinkPad with a 2.1 GHz Intel Core i7 processor and 12GB of RAM.

We compute the distance matrices for two data sets: one containing all the data $(N = 2,381)$ and one containing only a random subset of $N = 500$ observations from the data set. For both data sets it holds that only the training set is considered, so both sets contain $n = 104$ points in time. In Section 2 we have seen that a distance matrix requires $\frac{N(N-1)}{2}$ distances to be calculated. In our case, this means that the data set with all time series requires 2,833,390 distances to be calculated and the subset requires 124,750 distances to be calculated. The expected scaling in calculation times for computing the distance matrices for the two data sets is thus $\frac{2{,}833{,}390}{124{,}750} \approx 22.7$. Below we first discuss the implementations of all distance measures in R, after which we present the computation times of all measures in Table 10.

Euclidean distance: Implemented using the *dist* function from the **stats** package. This function uses underlying C code to calculate the distance matrix, which makes it very time efficient.

Pearson correlation distance: Different variations of this distance measure are present in the literature and in packages that have implemented the Pearson correlation distance. However, these variations all come down to scaling the distance measure. For example, the *pearson.dist* function in the **hyperSpec** package (Beleites and Sergo (2017)) divides the measure by 2 to scale the distances in the interval [0,1] and the *diss* function in the **TSclust** package (Montero and Vilar (2014)) scales the distance to $d_{cor}(x, y) = \sqrt{2(1 - \rho(x, y))}$. This last modification makes the Pearson correlation distance equal to the Euclidean distance for the case where x and y are standardized (Borg and Groenen (2003)). Both the *pearson.dist* function and the *diss* function are applied to compute the distance matrix using the Pearson correlation distance. While essentially computing the same distance, there is a significant difference in calculation times for the two functions: *pearson.dist* calculated the distance matrix for the whole set in 0.44 seconds, while it took the *diss* function 202.30 seconds.

Dynamic Time Warping (DTW): Two different implementations of DTW are examined: using the *dist* function from the **proxy** package (Meyer and Buchta (2017)) and using the *diss* function from the **TSclust** package (Montero and Vilar (2014)). Due to the time complexity, a comparison between the two functions is performed on the subset of 500 observations. Here it took the *dist* function 206.69 seconds to obtain the distance matrix, while it took the *diss* function 332.81 seconds. We therefore used the *dist* function for computing the distance matrix for the data set with all observations.

Longest Common SubSequence (LCSS): For LCSS only one implementation is found, which is the *TSDatabaseDistances* function from the **TSdist** package (Mori et al. (2016b)). For the parameters we chose $\epsilon = 0.05$ and $\delta = 5$, meaning that the horizontal band is 5 time points and the vertical band is 0.05. These values seemed reasonable to us for our data set, as it requires time series to be relatively close in order to be considered similar by the measure.

Ensemble scheme: As discussed in Subsection 2.4, we only consider a linear ensemble scheme in this thesis. This ensemble scheme is obtained by adding the DTW distance to the LCSS distance. However, since both distances have different scales, the distances are first scaled to the interval $[0, 1]$ before being added to each other. The computation time of the addition is negligible compared to the computation times of DTW and LCSS and therefore the computation times of the ensemble scheme in Table 10 are simply obtained by adding the two individual computation times.

Discrete Fourier Transform (DFT): For the DFT we examined the *TSDatabaseDistances* function from the **TSdist** package. This function uses the Cooley-Tukey algorithm for calculating the Fast Fourier Transform, which increases the computation speed (see also Subsection 2.3.1). For the value of q, the number of (low) frequencies that are used for the approximation, we decided to use $q = 52$. This is based on the Nyquist-Shannon sampling theorem and the findings in Subsection 2.3.1.
While it is expected for the DFT to run faster compared to the Euclidean distance, it took the *TSDatabaseDistances* function 211.78 seconds to compute the distance matrix for the whole data set (compared to 1.55 seconds for the Euclidean distance). The cause of this difference became apparent when diving into the source code of the *TSDatabaseDistances* function; instead of computing the DFT for each time series once and then compute the distance matrix based on these transforms, the DFT is recalculated for every distance that is calculated in the distance matrix. To solve this inefficiency, we programmed our own implementation of the DFT. This implementation only took 20.95 seconds for computing the distance matrix for the whole data set. Note that the factor with which our function is faster compared to the *TSDatabaseDistances* function grows with N and thus increases for larger data sets. The fact that the Euclidean distance is still faster compared to the DFT distance can be explained by the differences in implementation, as for the DFT we could not use C code for computing the distance matrix (as is the case for the Euclidean distance).

Discrete Wavelet Transform (DWT): For the DWT, we examined the *diss* function from the **TSclust** package. This function has the option to compute the DWT, however the only wavelet that can be chosen is the Haar wavelet. Furthermore, the function does not provide the option to choose the number of decomposition levels to use, but instead decides on this number itself. We have rewritten the code of the *diss* function to enable choosing different wavelets and also to enable the user to choose the number of decomposition levels to use. Our tests indicate that this addition does not influence the computation time of the function.
To utilize the extra options that we programmed, we apply the DWT for two different wavelets: Haar and Daubechies 10. For both transforms we use decomposition level 2, which comes down to a reduction from 104 dimensions to 32 dimensions.

Symbolic Aggregate approXimation (SAX): For SAX, we examined the *diss* function of the **TSclust** package. Since initial tests indicated that the function is not programmed efficiently, we investigated the source code to find the cause. The reason why the function is inefficient for determining the SAX distance is similar to the cause we found at the DFT implementation discussed above; the decomposition is performed for every calculation of the distance matrix instead of decomposing all time series once and then using the decomposed series for determining the distance matrix. To overcome this inefficiency, we programmed our own implementation of the SAX distance. This implementation yields the same distances as the *diss* function, but uses significantly less computation time. For the subset of 500 time series, our own implementation took 0.56 seconds for computing the distance matrix, while the *diss* function took 35.65 seconds. This difference becomes more significant as N grows. For the parameters of the SAX distance, $w = 13$

and $a = 8$ are chosen. These values seem reasonable to us, based on the examples we provided in Subsection 2.3.3.

Table 10: Overview of the computation times of the different distance measures for determining the distance matrix. Time is given in seconds.

Distance measure	**Whole data set** $(N = 2,381)$	**Subset** $(N = 500)$
Euclidean	1,55	0,08
Pearson correlation	0,44	0,02
DTW	4330,91	206,69
LCSS	3876,05	171,67
Ensemble scheme	8206,96	378,36
DFT	20,95	0,85
DWT (Haar)	6,09	1,57
DWT (Daubechies 10)	6,20	1,09
SAX	10,23	0,56

In Table 10, an overview of the computation times of the different distance measures can be found for determining the distance matrices of the whole data set and the subset. We begin by observing that the expected scale of 22.7 between the two columns seems to hold. For DWT this scale appears to be less, we expect this difference to be caused by the start-up time of the DWT function. While the pearson correlation distance is the fastest out of all distances that we consider, we cannot be conclusive on whether this is always the case, due to the variation in computation times that we observed. However, what we can conclude, is that DTW and LCSS are significantly slower compared to the other distance measures, taking over 2,000 times longer compared to the Euclidean distance for our data set. Furthermore, we conclude that while being faster in theory, the dimensionality reduction methods that we considered are slower compared to the Euclidean distance in the current R implementations. However, this does not mean that these methods become irrelevant, since their noise reduction properties are still in effect.

5.3 Determining the number of clusters

In this subsection, we determine the number of clusters to choose for our data set of 2,381 time series. We do this by applying all seven indices from Section 4 on the data. For the indices that allow any distance measure as input, we determine the 'optimal' number of clusters for all distance measures from the previous subsection. Furthermore, based on the implementations that are available in R, we apply all indices for k-means clustering and agglomerative hierarchical clustering. For the hierarchical clustering, we apply both Complete linkage and Ward's method.

Below we first discuss the R implementations of the different indices, after which we present Table 11, which contains the optimal number of clusters as suggested by the different indices.

Calinski-Harabasz index, C index, Je(2)/Je(1) index & Beale index: All four obtained by using the *NbClust* function of the **NbClust** package (Charrad et al. (2014)). All four indices were calculated relatively fast (only a few seconds). The Je(2)/Je(1) index failed to converge to a solution when using hierarchical clustering with Ward's method. The Beale index failed to converge to a solution for the two hierarchical clusterings and could only find a solution for k-means clustering by using a significance level of 0.0005.

Gamma index: Implementation available in the *NbClust* function of the **NbClust** package. However, as also indicated by the *NbClust* function description, the index takes a long time to compute. In our case, we terminated the computation of the index, since it had not found the index value for a single number of clusters after running for over 20 minutes. We therefore do not consider the Gamma index in Table 11.

Silhouette index: We applied the Silhouette index by using the *silhouette* function of the **cluster** package (Maechler et al. (2017)). Calculating the index for our data only took a few seconds.

Gap statistic: Two implementations are examined: the *index.Gap* function from the **clusterSim** package (Walesiak and Dudek (2017)) and the *NbClust* function of the **NbClust** package. Both functions appear to be slow for computing the Gap statistic, taking tens of minutes even for low values of B (we used $B = 10$). We therefore decided not to include the Gap statistic in Table 11.

Table 11: Overview of the 'optimal' number of clusters for each combination of index, distance measure and clustering method that we considered. The range of possible k values was 5 to 30 clusters. An x indicates that the method was not applicable for the combination of methods and a minus sign indicates that the index did not converge to an optimal number of clusters.

Index	**Distance measure**	**Ward's method**	**Complete linkage**	**K-means**
Calinski-Harabasz	Euclidean	5	5	5
Beale	Euclidean	-	-	5
Je(2)/Je(1)	Euclidean	-	11	5
C	Euclidean	5	5	7
	Pearson correlation	30	23	x
	DTW	6	5	x
	LCSS	5	5	x
	Ensemble scheme	8	5	x
	DFT	5	5	x
	DWT (Haar)	30	7	x
	DWT (Daubechies 10)	26	5	x
	SAX	17	8	x
Silhouette	Euclidean	5	6	5
	Pearson correlation	5	5	x
	DTW	6	5	x
	LCSS	5	5	x
	Ensemble scheme	6	5	x
	DFT	5	5	x
	DWT (Haar)	5	8	x
	DWT (Daubechies 10)	5	6	x
	SAX	5	5	x

In Table 11, the 'optimal' number of clusters can be found for each combination of index, distance measure and clustering method that we considered. For the Silhouette method and the C index, all distance measures from the previous subsection are applied. The Calinski-Harabasz index, Beale index and Je(2)/Je(1) index are all only applicable for the Euclidean distance measure. While we did apply hierarchical clustering with Ward's method for every distance measure, we again have to mention that Ward's method is built around (a distance measure proportional to) the Euclidean distance.

By visual inspection, we decided that at least five clusters should be obtained from the clustering. We therefore set the range of k values for the different indices from 5 to 30 clusters. Note that many methods return five, the minimum number of k, as the 'optimal' number of clusters. Six, seven and eight clusters are also returned by a few indices, however no clear agreement between the indices can be detected. Even though five is returned by the majority of indices, the fact that the lowest possible k value is returned does not necessarily mean that this is the optimal k, but rather that no clear clustering structure is present in the data. Since no clear agreement between the indices is found, we decided to visually inspect the obtained clusters to pick the number of clusters (while keeping Table 11 in mind). Based on visual inspection of the obtained clusters, we decided to work with eight clusters in the remaining clustering analysis. We refer to Subsection 5.4 for visualizations of the eight clusters.

5.4 Using forecasting to compare cluster quality

In this last part of the practical research, we use forecasting to compare the quality of clusterings. To clarify how this is done, below an overview of the overall setup of this experiment can be found.

1. Normalize the time series data (all 185 weeks).
2. Decide on a distance measure.
3. Decide on the number of clusters.
4. Cluster the time series based on the first 104 weeks of time points.
5. Calculate the mean sales of each cluster.
6. For every cluster, make a forecast on the sales pattern of the last 81 weeks based on the cluster mean of the first 104 weeks.
7. Calculate the total difference between the cluster forecasts and the true sales patterns of the individual products.

Steps 1, 2 and 3 are discussed in the previous subsections. Note that in step 1, normalization is applied based on all 185 weeks instead of only applying it to the first 104 weeks. It could be argued that we here use 'information from the future', as the normalization of the first 104 time points could be influenced by the last 85 points in time. However, we still decided to normalize in this way, since it reduces the influence of outliers in the last 85 points in time on the error function in step 7. Furthermore, it allows us to limit the influence of clusters with a decreasing and increasing trend (see also step 6).

For step 4, we apply the following clustering methods to generate eight clusters:

Agglomerative hierarchical clustering: Implemented using the *hclust* function from the **stats** package. Based on tests on the linkage methods, Complete linkage and Ward's method seem to generate the best clustering and thus we only consider these two methods. Since hierarchical clustering is applicable on all distance measures, we apply it using all distance measures that are discussed in Subsection 5.2.

k-medoids: Implemented using the *pam* function from the **cluster** package (Maechler et al. (2017)). This function allows distance matrices to be given as input and thus we applied k-medoids on all distance measures that are discussed in Subsection 5.2.

k-means: Implemented using the *kmeans* function from the **stats** package. The algorithm that is used is the Hartigan-Wong algorithm, which is also chosen by default by the function. For the stopping condition, a maximum of 100 iterations is taken. Also, 100 random initializations are used to lower the probability of finding a local optimum. This does, however, increase the computation time, but this time is still in line with the other clustering methods, as we can also see in Table 12. As we have seen in Subsection 3.2.1, k-means is only applicable for the Euclidean distance.

CLARA: Implemented using the *clara* function from the **cluster** package (Maechler et al. (2017)). The sample size is taken to be $z = 40 + 2k = 56$, which is based on the literature study in Subsection 3.2.4. Since the method appeared to have fast computation times, the default number of samples of five is increased to $S = 1,000$. This is done to reduce the chance of finding local optima. While CLARA is applicable for all distance measures in theory, the implementation in the **cluster** package only supports the Euclidean distance. We therefore only apply this method for the Euclidean distance.

CLARANS: No implementations of CLARANS were found and thus we programmed this method ourselves. Based on the literature, the maximum number of neighbours v to compare each solution with is taken to be 250. The total number of solutions to be sampled is 10. The CLARANS function we implemented only supports the Euclidean distance, since it is expected to be slow for other distances.

In step 6, we use ARIMA for forecasting. Since forecasting is not within the scope of this thesis and since we only use it as a tool for quantifying clustering quality, we will not go into depth on how ARIMA works. To obtain the forecasts, we use the *auto.arima* function from the **forecast** package. This functions returns the best ARIMA model according to the AICc value. The only parameter we adjusted for the *auto.arima* function is the value of D, for which we take $D = 1$. This adjustment ensures that strong seasonality is taken into account in the forecasting, as we will also see in plots later on. For further explanation on ARIMA forecasting and the *auto.arima* function in R, we refer to Hyndman and Khandakar (2008). Since we known from step 1 that the normalized true sales patterns lie within the interval $[0, 1]$, any forecast above 1 is scaled down to 1 and any forecast below 0 is scaled up to 0. This avoids forecasts of decreasing or increasing clusters to dominate the error function in step 7.

To quantify the clustering quality in step 7, we compute the total error between the forecast mean of each cluster and the true sales pattern of the individual products. This error is taken to be the absolute difference (Manhattan distance) between the forecast and the true series. Note that we here take the forecast and the true sales patterns from the normalized range of values, rather than taking the actual sales values. This is done to give each product equal weight.

A visualization of steps 5, 6 and 7 can be found in Figures 17, 18 and 19. Here one of the clusters obtained by the CLARANS algorithms is taken in Figure 17 and the corresponding ARIMA forecast can be found in Figure 18. In Figure 19, the mean of the forecast is plotted together with the normalized sales data of the forecasted period. The difference between these time series and the mean forecast yields us the total error for the given cluster.

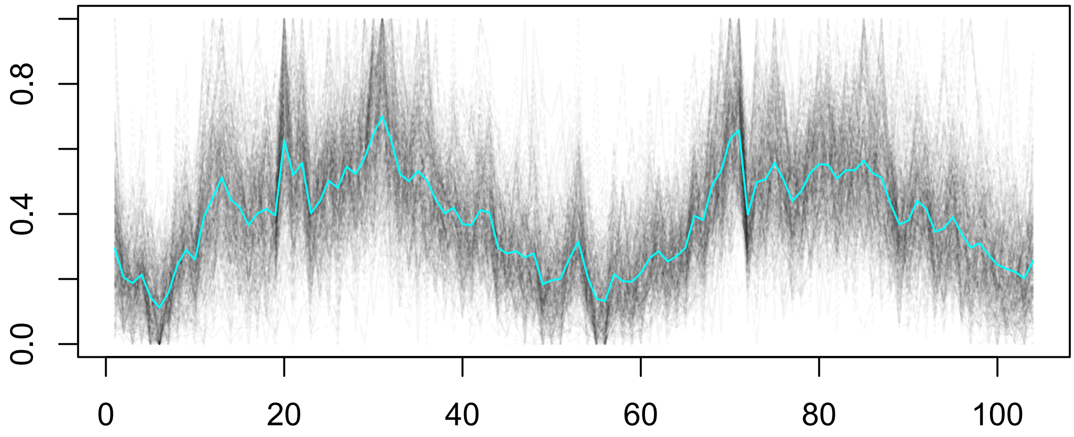


Figure 17: Example of step 5 of the experiment: determining the mean of all sales time series within a cluster. Only the first 104 time points are considered. The cluster is one of the clusters that is obtained when using CLARANS.

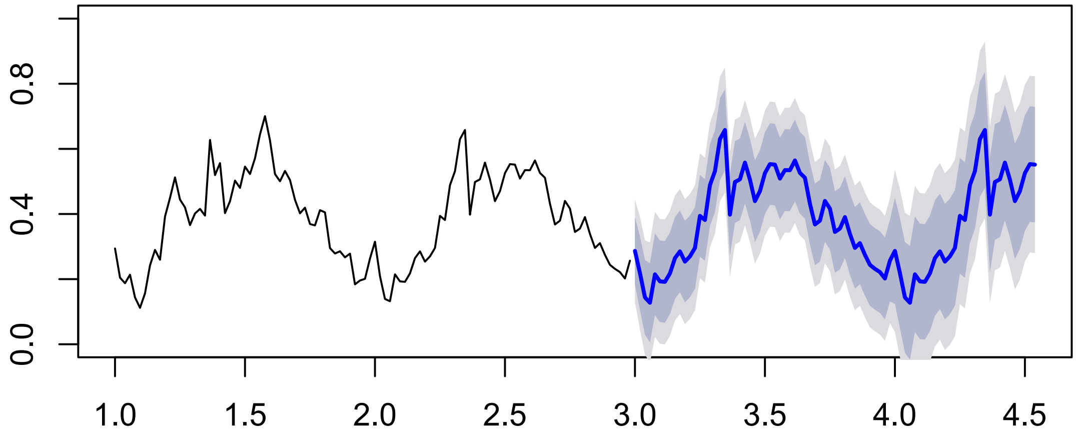


Figure 18: Example of step 6 of the experiment: using ARIMA to forecast 81 weeks. The cluster is one of the clusters that is obtained when using CLARANS.

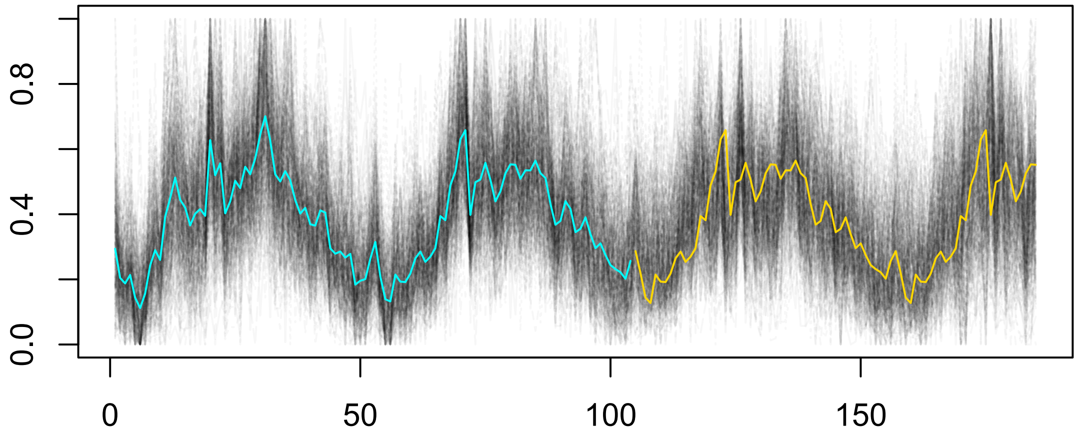


Figure 19: Example of step 7 of the experiment: comparing the forecast of the last 81 points in time (yellow line) with the true normalized time series. The cyan line on the left is the mean of the original cluster. The cluster is one of the clusters that is obtained when using CLARANS.

In Table 12, we present the computation times of each method for obtaining eight clusters using the parameters we discussed in this section. This table serves as an indication as to what one could expect when using these methods in practice. The hierarchical methods tend to be the fastest in generating the clustering, however we must note that these computation times do not include the computation of the individual distance matrices. This also holds for the k-medoids computation times. For the methods that are only applicable (in the implementations) when using the Euclidean distance, we observe that CLARANS is clearly slower compared to k-means and CLARA.

In Table 13, the total error (as defined above) for each method can be found. We note that all errors lie relatively close to each other. What is interesting, is that k-means, CLARA and CLARANS, which are based on the Euclidean distance, outperform most of the hierarchical clustering methods and k-medoids combinations that use more sophisticated distance measures. The hierarchical clustering appeared to perform best when using the DFT distance. The method that performs best in our experiment is CLARANS, followed by hierarchical agglomerative clustering using Ward's method and the DFT distance. The third best performing method in our experiment is k-means. The method that performs worst in our experiment is hierarchical clustering using Complete linkage and the DWT distance with the Daubechies 10 wavelet.

Table 12: Overview of the calculation times for obtaining eight clusters of all combinations of methods that we applied. Computation times are given in seconds.

Dist. measure	**Hierarchical Agglomerative Complete linkage**	**Hierarchical Agglomerative Ward's method**	**k-medoids**	**k-means**	**CLARA**	**CLARANS**
Euclidean	0.25	0.20	2.31	6.76	3.56	173.36
Pearson	0.24	0.19	2.48			
DTW	0.22	0.19	3.98			
LCSS	0.75	0.30	1.26			
Ensemble	0.46	0.26	1.22			
DFT	0.35	0.17	1.16			
DWT (Haar)	0.34	0.22	1.72			
DWT (Db 10)	0.22	0.19	1.76			
SAX	0.63	0.15	2.75			

Table 13: Total error between the cluster forecasts and the actual normalized sales patterns, given for all combinations of methods that we applied. The lowest error in each of the first three columns is underlined and the lowest error overall is shown in bold.

Dist. measure	**Hierarchical Agglomerative Complete linkage**	**Hierarchical Agglomerative Ward's method**	**k-medoids**	**k-means**	**CLARA**	**CLARANS**
Euclidean	29174	29143	28791	28531	28606	**28313**
Pearson	31884	30774	31858			
DTW	29762	29168	<u>28534</u>			
LCSS	32581	32574	29638			
Ensemble	32584	32278	30849			
DFT	<u>29064</u>	<u>28420</u>	28584			
DWT (Haar)	31946	31454	31611			
DWT (Db 10)	32625	31690	31614			
SAX	32081	31637	31306			

Besides comparing the computation times and the forecasting errors of different clustering assignments, we now investigate the obtained clusterings visually. For this, we plotted the obtained clusterings of the three methods that gave the lowest forecasting error in our experiment. We also plotted the clustering that is obtained by the method that gave the highest forecasting error. The plots of all four clusterings can be found in Appendix E.

When visually comparing the three best performing clustering assignments with the assignment that performed worst in our experiment in Appendix E, we observe three main differences. The first main difference lies in the cluster sizes. The three best performing clusterings mostly have clusters with sizes around 300-400, while the worst performing clustering has two large clusters that account for over 70% of the observations and other clusters with only a few (4, 12 and 23) observations. The second main difference is that, judging by the black lines in the background, more variance appears to be present between the time series of the clusters in the worst clustering. The third main difference is the ability of the clusterings to capture seasonality. In all three best performing clusterings, we clearly observe seasonality being captured. They all contain two or three clusters that contain seasonal peaks during the spring or summer. Furthermore, they all contain a small-sized cluster that has seasonal peaks during the winter months. This cluster with peaks during the winter is not present in the worst clustering. The other clusters with peaks during the spring and summer months are present, but with more moderate peaks.

6 Discussion

One of the most important factors in time series clustering is the distance measure that is chosen. Elastic measures such as DTW and LCSS outperform traditional lock-step measures, according to the literature, due to their ability to, for example, capture time warping and time shifting. However, time complexity plays a big role in time series clustering and therefore elastic distance measures quickly become infeasible as the number of time series grows. This also became apparent in our practical research, where for 2,381 time series it already took more than one hour to compute each of the distance matrices for the elastic measures. For large data sets, even traditional lock-step measures, such as the Euclidean distance, may become too time expensive.

To handle large data sets, two main approaches are suggested in this paper. The first approach is to use dimensionality reduction methods in order to reduce the number of dimensions (and noise) of the time series that are clustered. The second approach is to use sampling methods, such as CLARA and CLARANS, when constructing clusters. The first approach seems to be most effective when dealing with time series that have a large number of dimensions, whereas the second approach seems to be most effective when one deals with a large number of individual time series.

It appeared to be a difficult task to determine the optimal number of clusters for our experiment. The different indices that we considered did not agree on a certain number of clusters, indicating that our data might not be so suitable for a clustering analysis. Besides the indices that we considered in this thesis for determining the optimal number of clusters, there is still a wide variety of other indices available. We mainly based the choice of our indices on the article of Milligan and Cooper (1985), who based their results on experiments with 2, 3, 4 and 5 clusters. Since we were dealing with more clusters in our practical research, we cannot be sure that the indices that we used are the best to apply.

The results of our practical research indicate that the CLARANS algorithm using the Euclidean distance performs best for our clustering goal. However, we cannot make general conclusions based on this result on which clustering method performs best. What we can conclude, however, is that sampling methods show the potential of obtaining good clusterings in a reasonable amount of time. Further research could investigate whether the performance of sampling methods can be improved when they are applied in combination with elastic distance measures.

Since we included many different distance measures and clustering methods in our practical research, we were unable to discover the effects of all parameters that can be tweaked for these methods. For example, only one set of parameters is used for LCSS and only two different wavelets are tried when applying the DWT. We expect that further analysis on the parameters of the different distance measures could improve the results for these measures in our practical research. Also, in our analysis on distance measures, only shape-based and feature-based distance measures are considered. While these methods appear to be the most dominant in literature on time series clustering, further research could investigate the performance of other types of distance measures, such as edit-based and structure based distance measures.

To conclude, it depends on the size of the data set and the goal of the clustering assignment which distance measure, index for determining the number of clusters and clustering method are most suitable. We advise to always start by applying general clustering methods, like k-means, and decide based on the computation time and performance of these methods which distance measures or clustering methods could be applied to improve the results or computation time.

A Appendix: List of abbreviations

CLARANS	Clustering Large Applications based on RANdomized Search
CLARA	Clustering LARge Applications
DDTW	Derivative Dynamic Time Warping
DFT	Discrete Fourier Transform
DTW	Dynamic Time Warping
DWT	Discrete Wavelet Transform
ED	Euclidean Distance
FFT	Fast Fourier Transform
LCM	Local Cost Matrix
LCSS	Longes Common SubSequence
PAM	Partitioning Around Medoids
PCA	Principal Component Analysis
SAX	Symbolic Aggregate approXimation
WDWT	Weighted Dynamic Time Warping

B Appendix: Omitting high frequencies to approximate and denoise time series

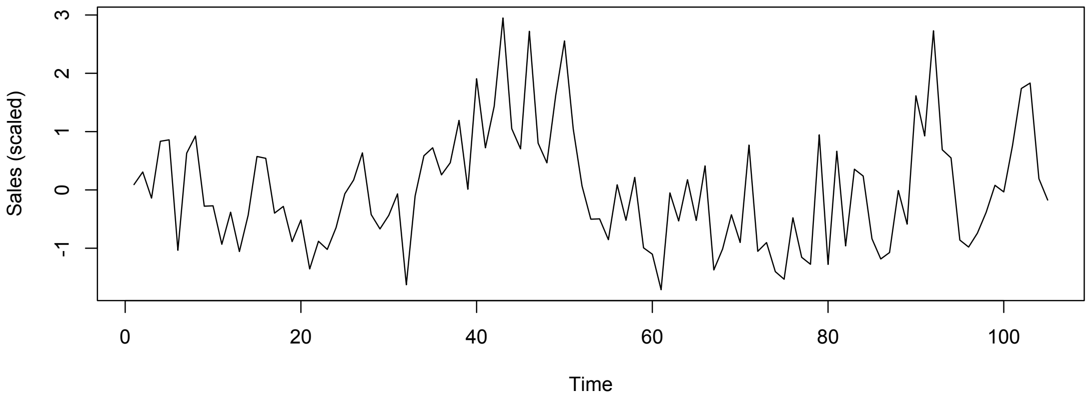


Figure 20: Plot of the scaled sales time series of an arbitrary product from our data set. For the scaling, the linear trend is removed and the series are standardized.

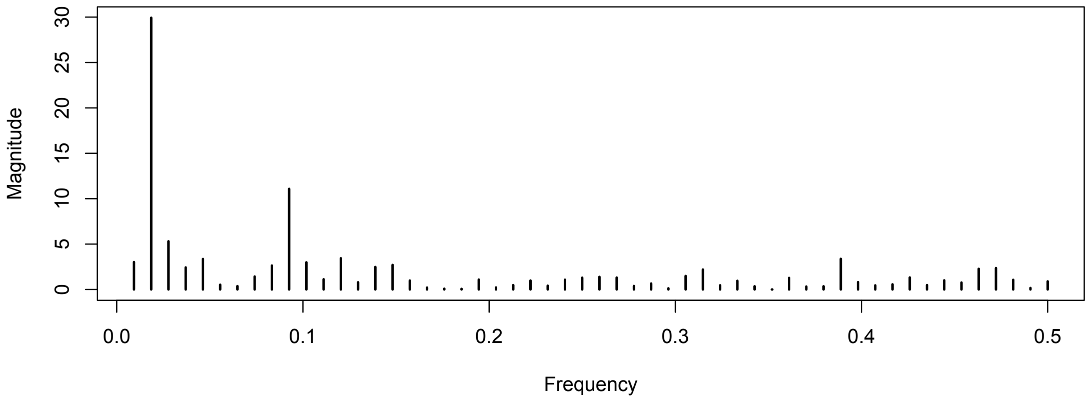


Figure 21: Periodogram for the time series in Figure 20.

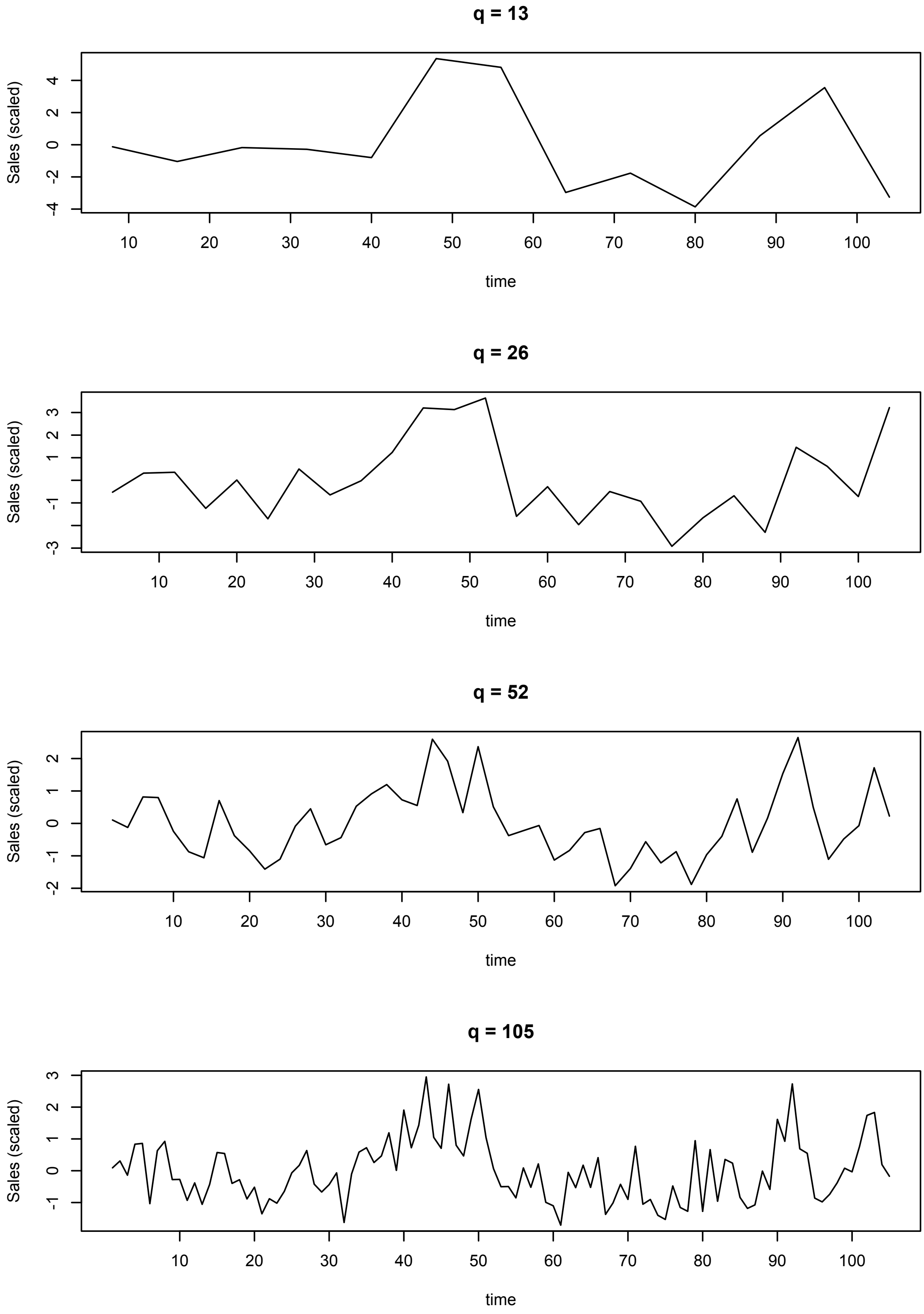


Figure 22: Approximations of the time series from Figure 20 when considering only the first $13, 26, 52$ and 105 frequencies from $X(f)$.

C Appendix: Omitting different levels of detail coefficients to approximate and denoise time series

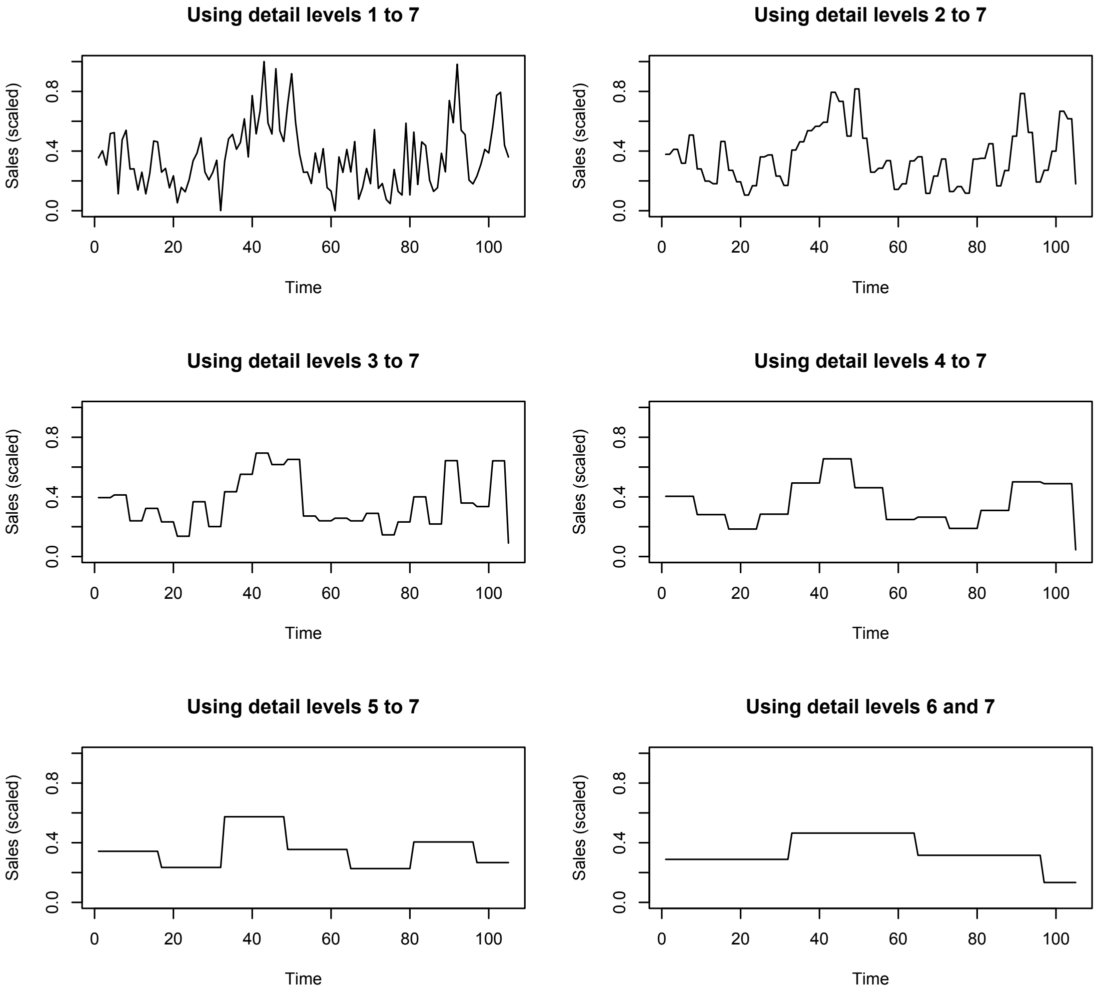


Figure 23: Approximation of a time series based on different levels of wavelet decomposition. In this example a time series of length 105 is used which has been supplemented with 23 dummy values (zeros) to obtain $2^7 = 128$ dimensions. This is done for visual purposes, as it allows perfect recreation of the original time series (using all detail levels). The dimensions of the approximations are, from left to right and from top to bottom, $128, 64, 32, 16, 8$ and 4. Only the approximations for the first 105 values are plotted. The Haar mother wavelet is used for this example.

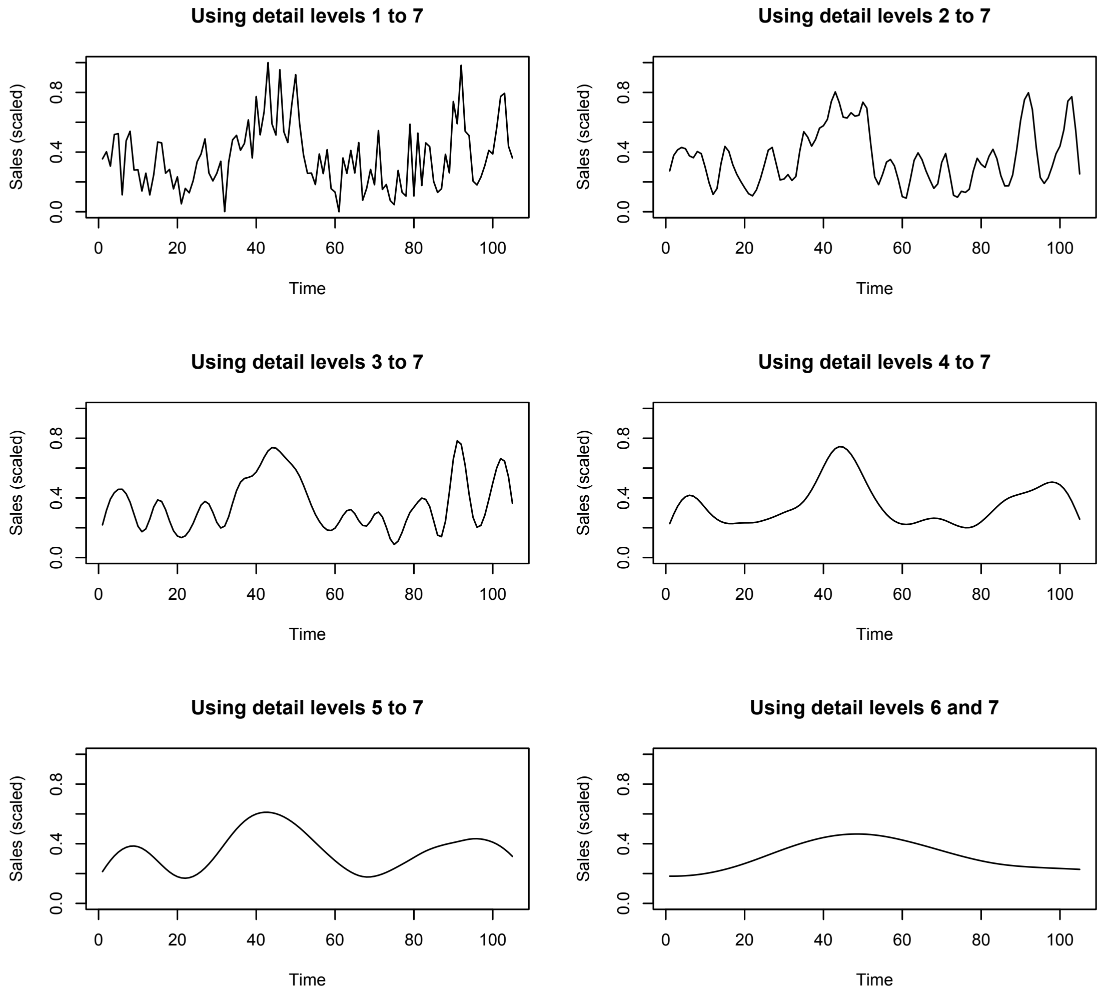
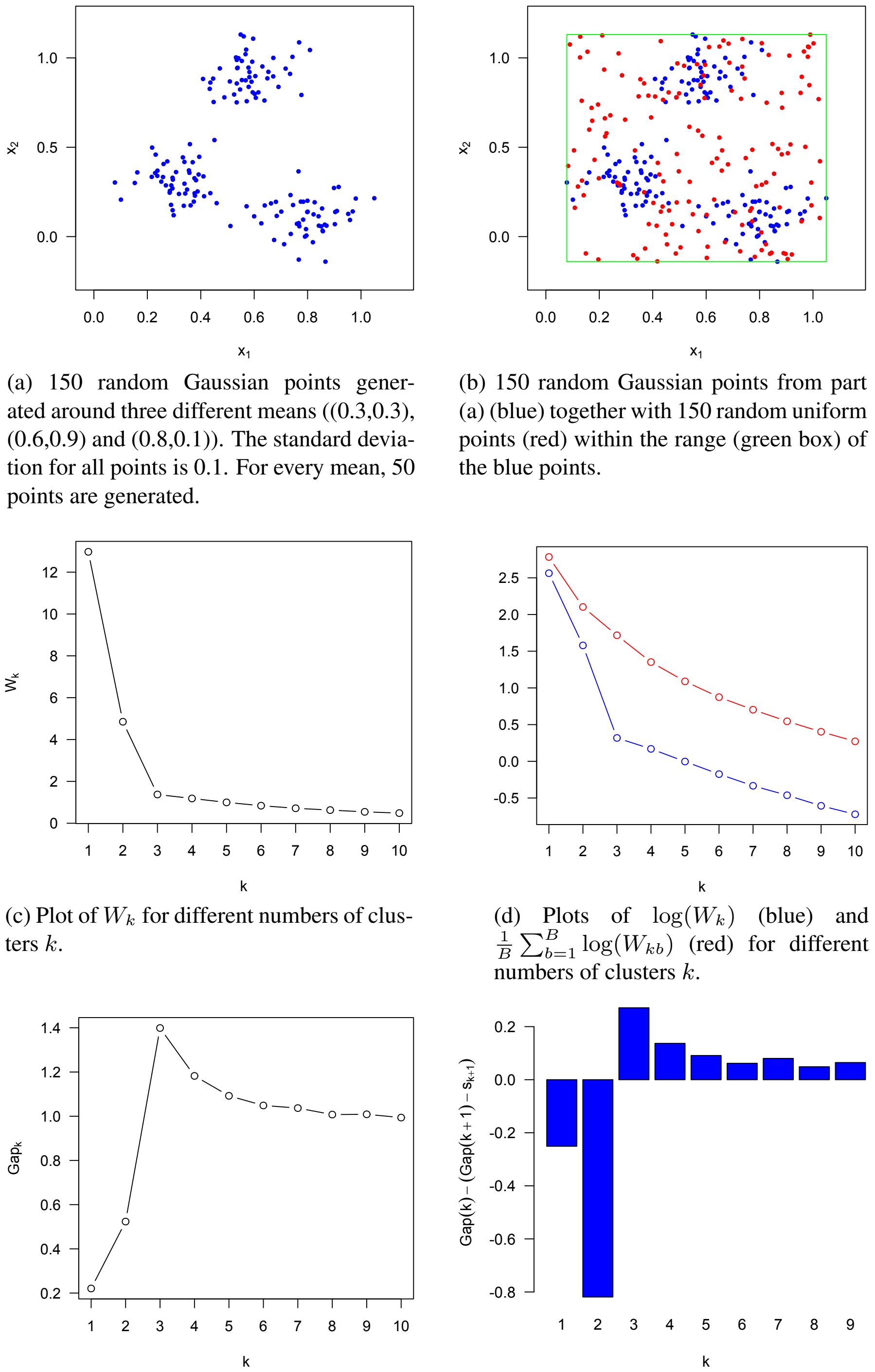


Figure 24: Approximation of a time series based on different levels of wavelet decomposition. In this example a time series of length 105 is used which has been supplemented with 23 dummy values (zeros) to obtain $2^7 = 128$ dimensions. This is done for visual purposes, as it allows perfect recreation of the original time series (using all detail levels). The dimensions of the approximations are, from left to right and from top to bottom, $128, 64, 32, 16, 8$ and 4. Only the approximations for the first 105 values are plotted. The Daubechies 10 mother wavelet is used for this example.

D Appendix: Examples of the Gap statistic for determining the number of clusters



(a) 150 random Gaussian points generated around three different means ((0.3,0.3), (0.6,0.9) and (0.8,0.1)). The standard deviation for all points is 0.1. For every mean, 50 points are generated.

(b) 150 random Gaussian points from part (a) (blue) together with 150 random uniform points (red) within the range (green box) of the blue points.

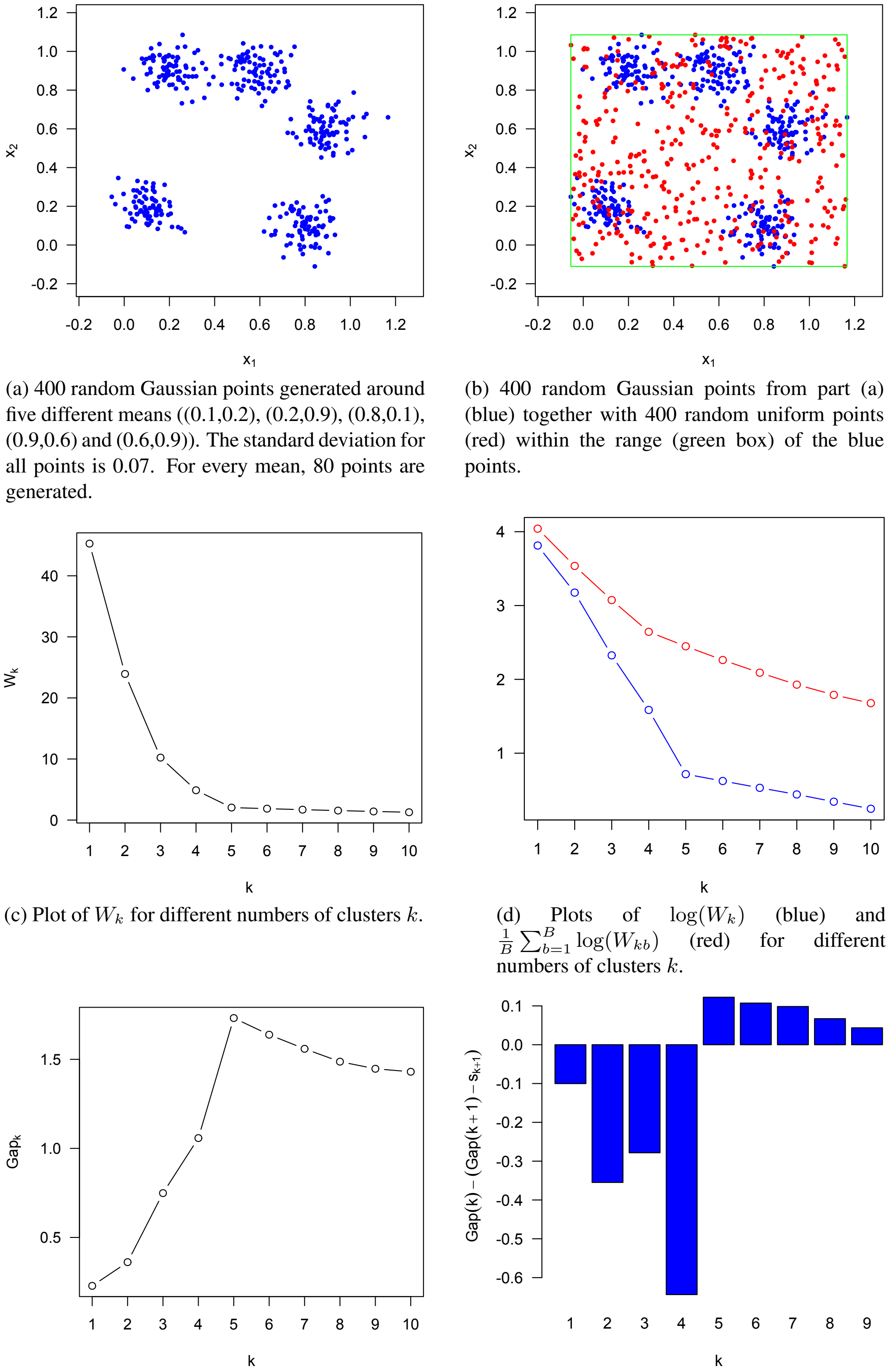
(c) Plot of W_k for different numbers of clusters k.

(d) Plots of $\log(W_k)$ (blue) and $\frac{1}{B}\sum_{b=1}^{B}\log(W_{kb})$ (red) for different numbers of clusters k.

(e) Value of Gap_k plotted for different numbers of clusters k.

(f) Barplot of the values that follow from the 1-standard-error method.

Figure 25: Example 1 of the different steps that are involved when determining the optimal number of clusters using the Gap statistic. Clustering is performed using K-means and $B = 100$ is used. Observe that the plots of W_k, Gap_k and the 1-standard-error method in parts (c), (e) and (f) all indicate that, indeed, three is the optimal number of clusters.



(a) 400 random Gaussian points generated around five different means ((0.1,0.2), (0.2,0.9), (0.8,0.1), (0.9,0.6) and (0.6,0.9)). The standard deviation for all points is 0.07. For every mean, 80 points are generated.

(b) 400 random Gaussian points from part (a) (blue) together with 400 random uniform points (red) within the range (green box) of the blue points.

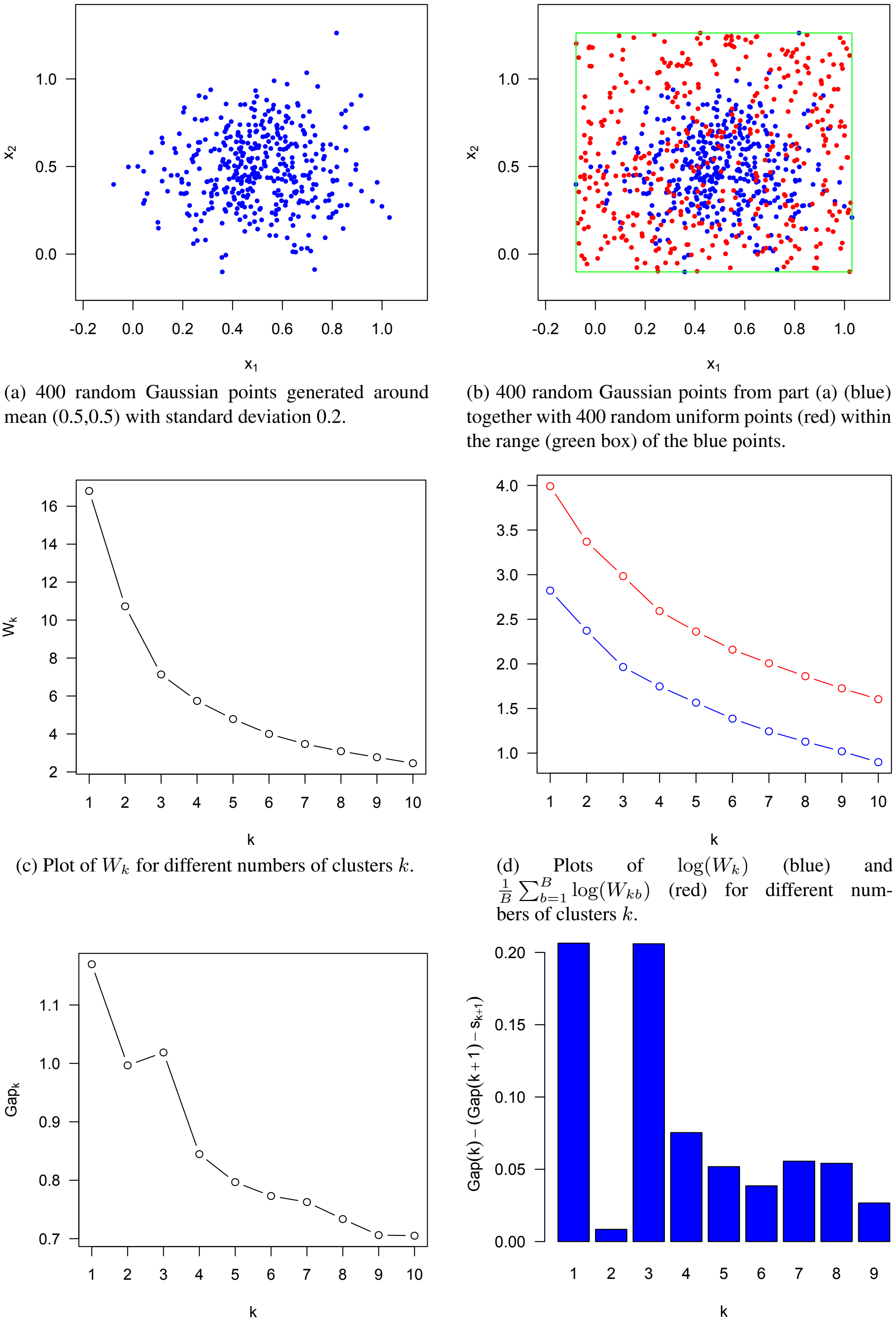
(c) Plot of W_k for different numbers of clusters k.

(d) Plots of $\log(W_k)$ (blue) and $\frac{1}{B}\sum_{b=1}^{B}\log(W_{kb})$ (red) for different numbers of clusters k.

(e) Value of Gap_k plotted for different numbers of clusters k.

(f) Barplot of the values that follow from the 1-standard-error method.

Figure 26: Example 2 of the different steps that are involved when determining the optimal number of clusters using the Gap statistic. Clustering is performed using K-means and $B = 100$ is used. Plots (e) and (f) indicate 5 to be the "optimal" number of clusters, while there is no clear "elbow" in (c).



(a) 400 random Gaussian points generated around mean (0.5,0.5) with standard deviation 0.2.

(b) 400 random Gaussian points from part (a) (blue) together with 400 random uniform points (red) within the range (green box) of the blue points.

(c) Plot of W_k for different numbers of clusters k.

(d) Plots of $\log(W_k)$ (blue) and $\frac{1}{B}\sum_{b=1}^{B}\log(W_{kb})$ (red) for different numbers of clusters k.

(e) Value of Gap_k plotted for different numbers of clusters k.

(f) Barplot of the values that follow from the 1-standard-error method.

Figure 27: Example 3 of the different steps that are involved when determining the optimal number of clusters using the Gap statistic. Clustering is performed using K-means and $B = 100$ is used. There is no clear elbow in (c). Plots (e) and (f) both indicate the right number of clusters, which is one.

E Appendix: Visualizations of the clusterings that are obtained in the experiment

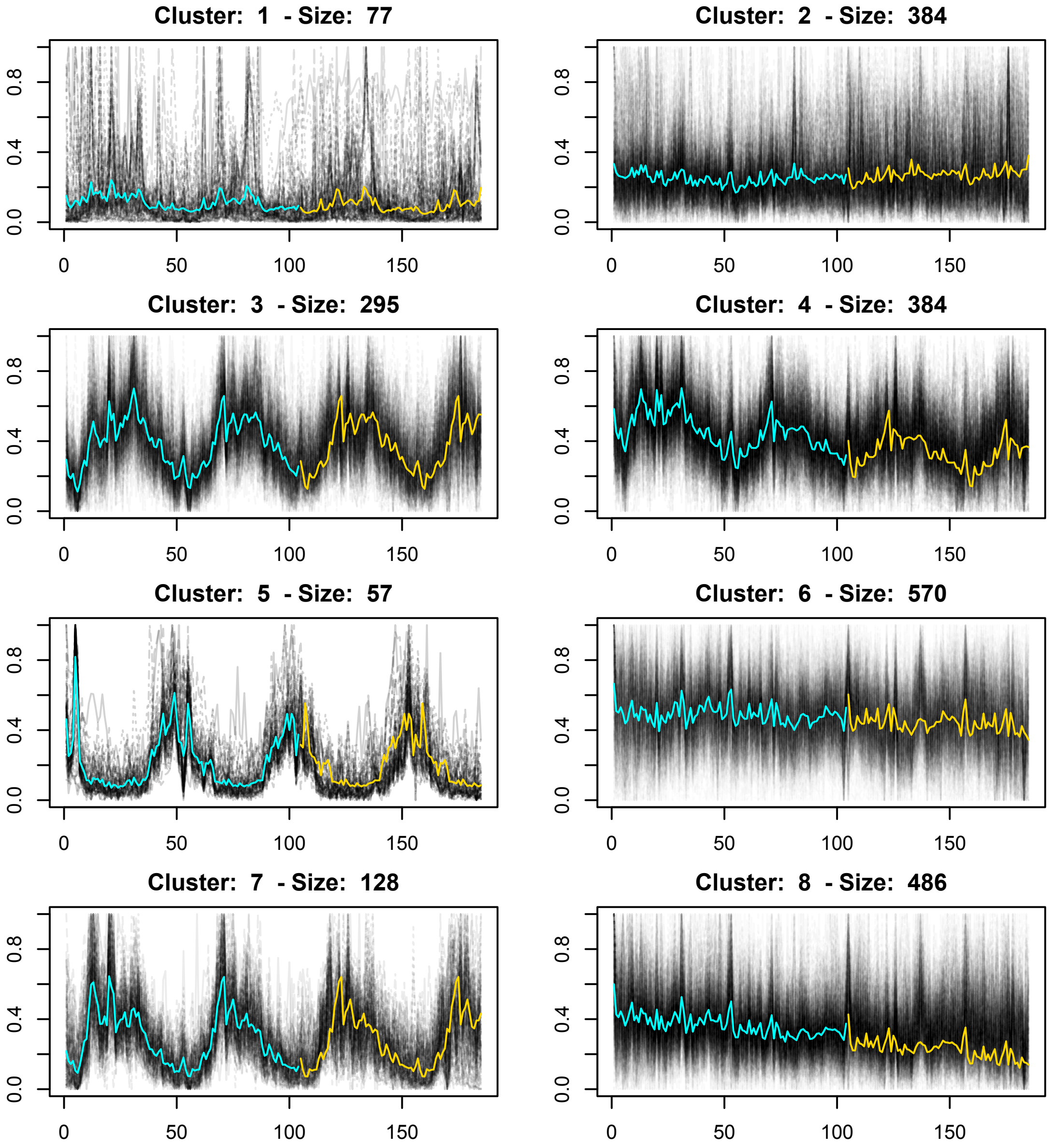


Figure 28: Visualization of the eight clusters that are obtained when applying the CLARANS clustering algorithm with Euclidean distance on our time series data set. This clustering assignment is the best in our experiments. The black lines in the background represent the time series within each cluster. The cyan line indicates the cluster mean of the first 104 points in time, which is used to obtain the ARIMA forecast. The ARIMA forecast is plotted by the yellow line. The size (number of time series) of each cluster is indicated above the plots.

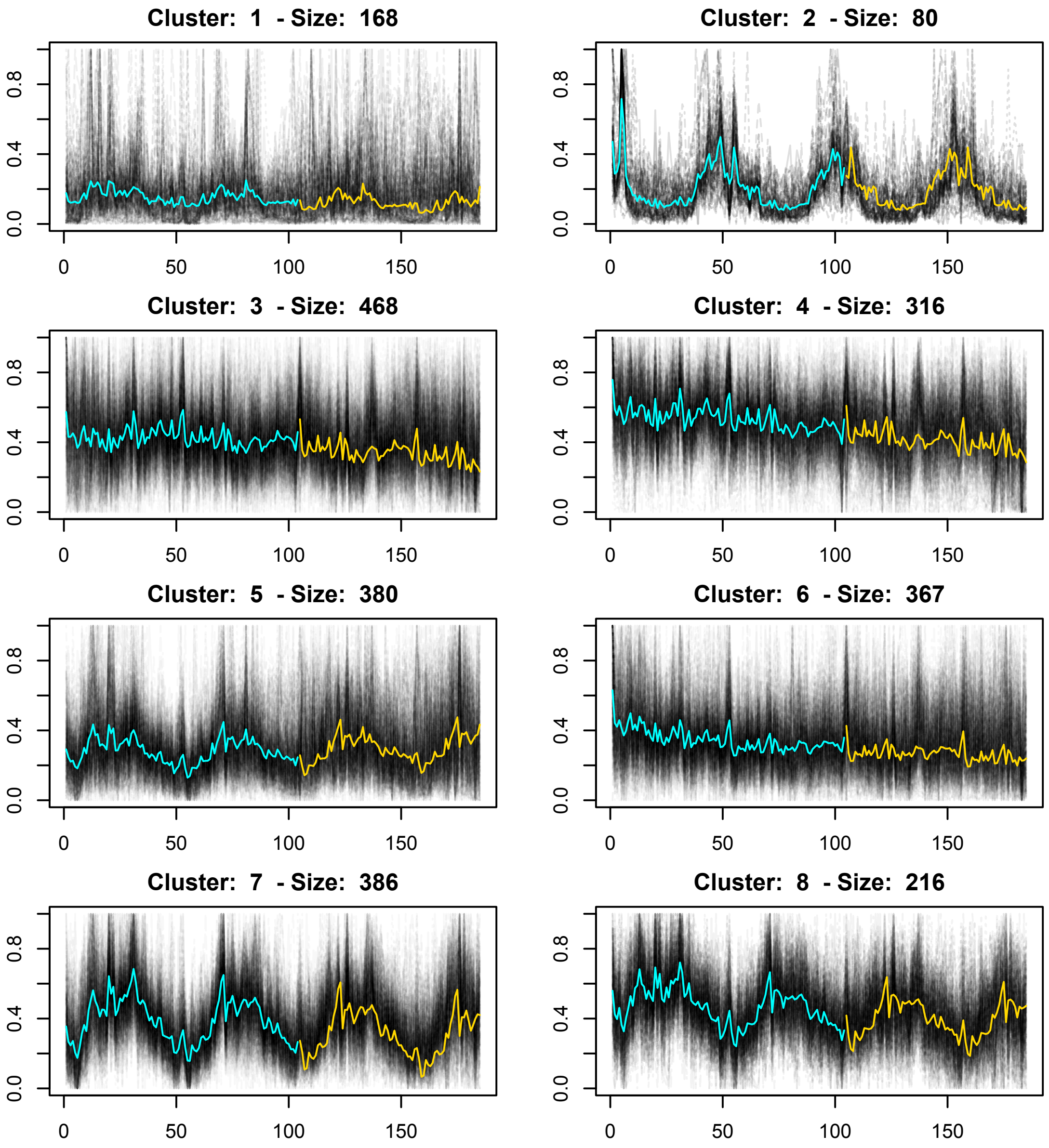


Figure 29: Visualization of the eight clusters that are obtained when applying the hierarchical agglomerative clustering algorithm using Ward's method with the DFT distance on our time series data set. This clustering assignment is the second best in our experiments. The black lines in the background represent the time series within each cluster. The cyan line indicates the cluster mean of the first 104 points in time, which is used to obtain the ARIMA forecast. The ARIMA forecast is plotted by the yellow line. The size (number of time series) of each cluster is indicated above the plots.

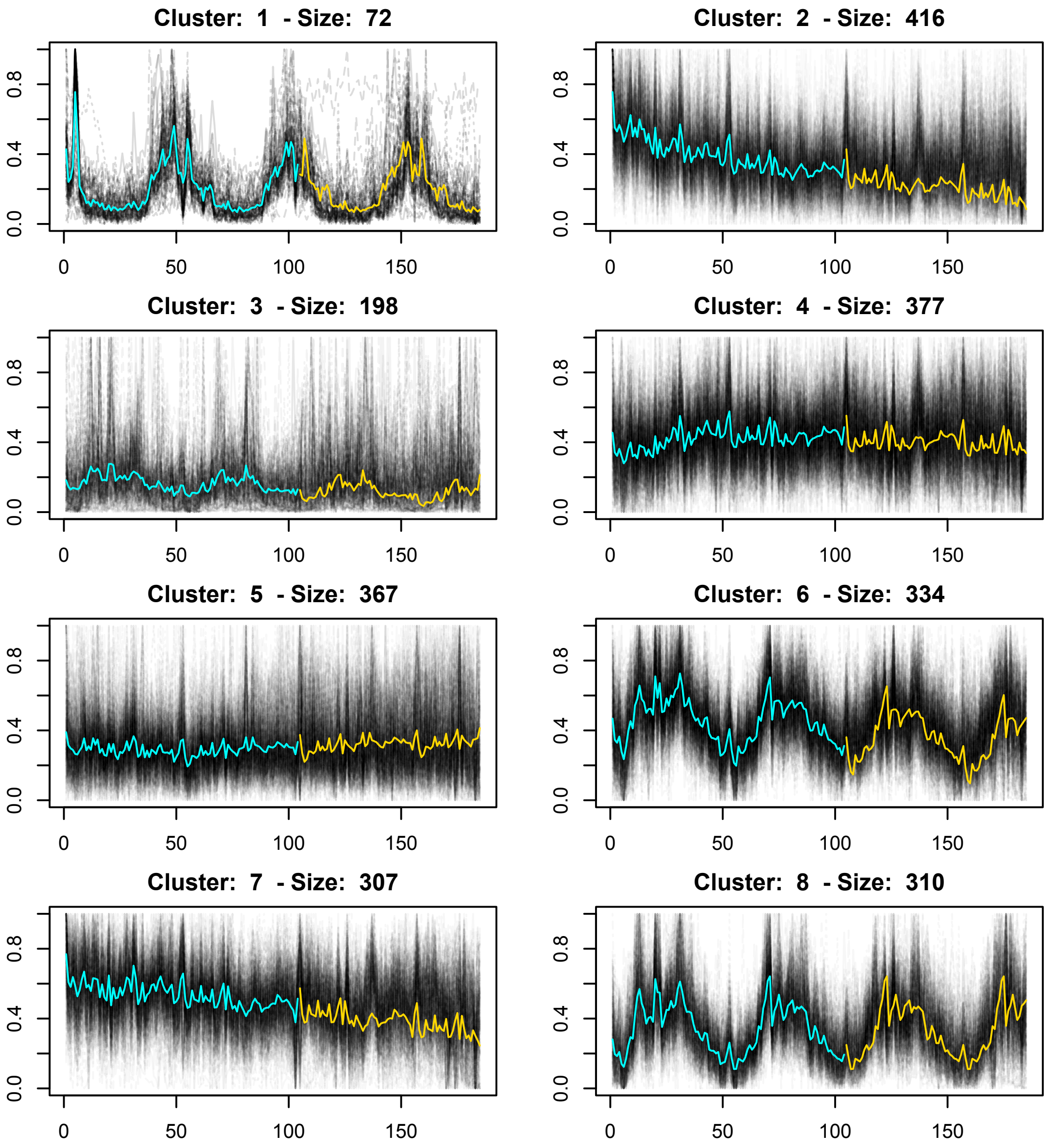


Figure 30: Visualization of the eight clusters that are obtained when applying the k-means clustering algorithm with Euclidean distance on our time series data set. This clustering assignment is the third best in our experiments. The black lines in the background represent the time series within each cluster. The cyan line indicates the cluster mean of the first 104 points in time, which is used to obtain the ARIMA forecast. The ARIMA forecast is plotted by the yellow line. The size (number of time series) of each cluster is indicated above the plots.

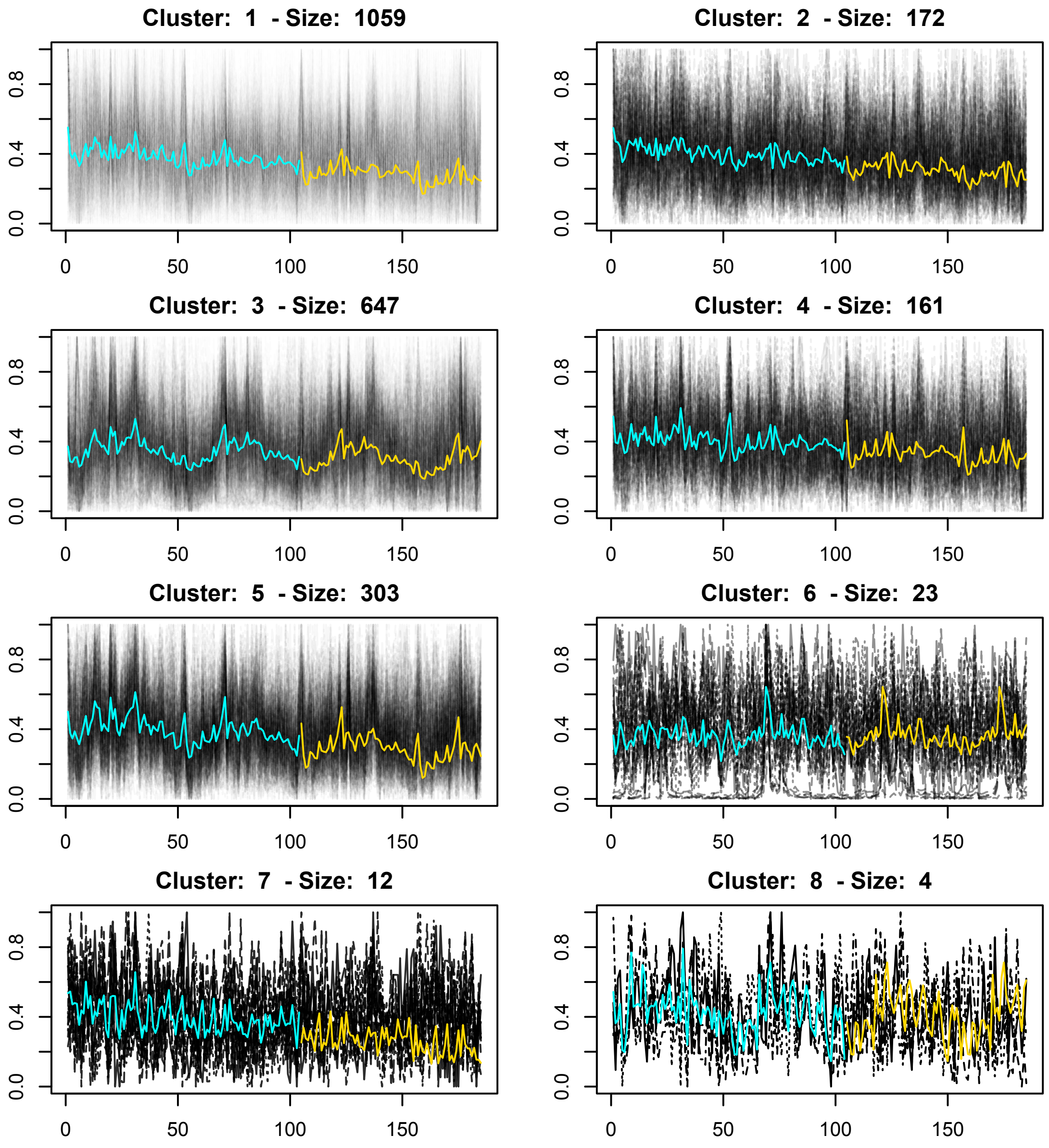


Figure 31: Visualization of the eight clusters that are obtained when applying the hierarchical clustering algorithm using Complete linkage and the DWT distance with the Daubechies 10 wavelet on our time series data set. This clustering assignment performed the worst in our experiments. The black lines in the background represent the time series within each cluster. The cyan line indicates the cluster mean of the first 104 points in time, which is used to obtain the ARIMA forecast. The ARIMA forecast is plotted by the yellow line. The size (number of time series) of each cluster is indicated above the plots.

References

Aach, J. and G. M. Church (2001). Aligning gene expression time series with time warping algorithms. *Bioinformatics 17 6*, 495–508.

Aghabozorgi, S., A. S. Shirkhorshidi, and T. Y. Wah (2015). Time-series clustering – a decade review. *Information Systems 53*, 16 – 38.

Agrawal, R., C. Faloutsos, and A. N. Swami (1993). Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, FODO '93, London, UK, UK, pp. 69–84. Springer-Verlag.

Anthony Bagnall, Jason Lines, W. V. and E. Keogh (2016). The uea & ucr time series classification repository. www.timeseriesclassification.com.

Baker, F. B. and L. J. Hubert (1975). Measuring the power of hierarchical cluster analysis. *Journal of the American Statistical Association 70*(349), 31–38.

Bar-Joseph, Z., G. Gerber, D. K. Gifford, T. S. Jaakkola, and I. Simon (2002). A new approach to analyzing gene expression time series data. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, RECOMB '02, New York, NY, USA, pp. 39–48. ACM.

Beale, E. (1969). *Euclidean Cluster Analysis*. Scientific Control Systems Limited.

Beleites, C. and V. Sergo (2017). *hyperSpec: a package to handle hyperspectral data sets in R*. hyperSpec. R package version 0.99-20171005.

Borg, I. and P. Groenen (2003). Modern multidimensional scaling: Theory and applications. *Journal of Educational Measurement 40*(3), 130.

Bunn, A., M. Korpela, F. Biondi, F. Campelo, P. Mérian, F. Qeadan, and C. Zang (2017). *dplR: Dendrochronology Program Library in R*. . R package version 1.6.6.

Calinski, T. and J. Harabasz (1974). A dendrite method for cluster analysis. *Communications in Statistics 3*(1), 1–27.

Camerra, A., T. Palpanas, J. Shieh, and E. Keogh (2010, Dec). isax 2.0: Indexing and mining one billion time series. In *2010 IEEE International Conference on Data Mining*, pp. 58–67.

Cassisi, C., P. Montalto, M. Aliotta, A. Cannata, and A. Pulvirenti (2012). Similarity measures and dimensionality reduction techniques for time series data mining. In A. Karahoca (Ed.), *Advances in Data Mining Knowledge Discovery and Applications*, Chapter 03, pp. 71 – 96. Rijeka: InTech.

Celebi, M. E., H. A. Kingravi, and P. A. Vela (2013, January). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Syst. Appl. 40*(1), 200–210.

Chaovalit, P., A. Gangopadhyay, G. Karabatis, and Z. Chen (2011, February). Discrete wavelet transform-based time series analysis and mining. *ACM Comput. Surv. 43*(2), 6:1–6:37.

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

Chu, S., E. Keogh, D. Hart, M. Pazzani, and Michael (2002). Iterative deepening dynamic time warping for time series. In *In Proc 2 nd SIAM International Conference on Data Mining*.

Cimiano, P., A. Hotho, and S. Staab (2004). Comparing conceptual, divisive and agglomerative clustering for learning taxonomies from text. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pp. 435–439. IOS Press.

Cooley, J. W. and J. W. Tukey (1965). An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comput. 19*, 297–301.

Dheeru, D. and E. Karra Taniskidou (2017). UCI machine learning repository.

Dimitriadou, E., S. Dolničar, and A. Weingessel (2002, Mar). An examination of indexes for determining the number of clusters in binary data sets. *Psychometrika 67*(1), 137–159.

Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. A Wiley Interscience Publication. Wiley.

Esling, P. and C. Agon (2012, December). Time-series data mining. *ACM Comput. Surv. 45*(1), 12:1–12:34.

Forgy, E. (1965). Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics 21*(3), 768–769.

Fujita, A., J. R. Sato, M. A. A. Demasi, M. C. Sogayar, C. E. Ferreira, and S. Miyano (2009). Comparing pearson, spearman and hoeffding's d measure for gene expression association analysis. *J. Bioinformatics and Computational Biology 7*(4), 663–684.

Garey, M., D. Johnson, and H. Witsenhausen (1982, Mar). The complexity of the generalized lloyd - max problem (corresp.). *IEEE Transactions on Information Theory 28*(2), 255–256.

Goodman, L. A. and W. H. Kruskal (1954, dec). Measures of association for cross classifications. *Journal of the American Statistical Association 49*(268), 732–764.

Gordon, A. (1999). *Classification, 2nd Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. CRC Press.

Gorecki, T. (2014). Using derivatives in a longest common subsequence dissimilarity measure for time series classification. *Pattern Recognition Letters 45*(Supplement C), 99 – 105.

Gorecki, T. (2017). Classification of time series using combination of dtw and lcss dissimilarity measures. *Communications in Statistics - Simulation and Computation 0*(0), 1–14.

Grabusts, P. and A. Borisov (2009). Clustering methodology for time series mining. *J. Riga Technical University 40*, 81–86.

Haar, A. (1910). Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen 69*(3), 331–371.

Hamerly, G. and C. Elkan (2002). Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, New York, NY, USA, pp. 600–607. ACM.

Hartigan, J. A. and M. A. Wong (1979). A K-means clustering algorithm. *Applied Statistics 28*, 100–108.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning: data mining, inference and prediction* (2 ed.). Springer.

Hesabi, Z. R., Z. Tari, A. M. Goscinski, A. Fahad, I. Khalil, and C. Queiroz (2015). Data summarization techniques for big data - A survey. In *Handbook on Data Centers*, pp. 1109–1152. Springer.

Hubert, L. and J. Levin (1975). *A General Statistical Framework for Assessing Categorical Clustering in Free Recall.* Theoretical papers. Wisconsin Research and Development Center for Cognitive Learning.

Hughes, H. K. (1965). The physical meaning of parseval's theorem. *American Journal of Physics 33*(2), 99–101.

Hyndman, R. and Y. Khandakar (2008). Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software, Articles 27*(3), 1–22.

Iglesias, F. and W. Kastner (2013). Analysis of similarity measures in times series clustering for the discovery of building energy patterns. *Energies 6*(2), 579–597.

Itakura, F. (1975, Feb). Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing 23*(1), 67–72.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters 31*(8), 651 – 666. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

Jain, A. K. and R. C. Dubes (1988). *Algorithms for Clustering Data.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Jeong, Y.-S., M. K. Jeong, and O. A. Omitaomu (2011). Weighted dynamic time warping for time series classification. *Pattern Recognition 44*(9), 2231 – 2240. Computer Analysis of Images and Patterns.

Jolliffe, I. (2014). *Principal Component Analysis*, Chapter 1 – 14, pp. 1 – 405. John Wiley & Sons, Ltd.

Kaufman, L. and P. Rousseeuw (1987, 01). Clustering by means of medoids. *PLOS Computational Biology 1*, 405–416.

Kaufman, L. and P. Rousseeuw (1990, 01). *Finding Groups in Data: An Introduction To Cluster Analysis*. John Wiley & Sons, Inc.

Kaufman, L. and P. J. Rousseeuw (2008). *Introduction*, pp. 1–67. John Wiley & Sons, Inc.

Keogh, E. and S. Kasetty (2003, Oct). On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Mining and Knowledge Discovery 7*(4), 349–371.

Keogh, E. and A. Ratanamahatana (2004). Everything you know about dynamic time warping is wrong. *3rd Workshop on Mining Temporal and Sequential Data, in conjunction with 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-2004), Seattle, WA 1*, 1 – 11.

Keogh, E. and C. A. Ratanamahatana (2005, March). Exact indexing of dynamic time warping. *Knowl. Inf. Syst. 7*(3), 358–386.

Keogh, E. J. and M. J. Pazzani (2000). Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, New York, NY, USA, pp. 285–289. ACM.

Keogh, E. J. and M. J. Pazzani (2001). Derivative dynamic time warping. In *In First SIAM International Conference on Data Mining (SDM'2001)*.

Ketchen, D. J. and C. L. Shook (1996). The application of cluster analysis in strategic management research: An analysis and critique. *Strategic Management Journal 17*(6), 441–458.

Kim, S.-W., S. Park, and W. W. Chu (2001). An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings 17th International Conference on Data Engineering*, pp. 607–614.

Kuchaki Rafsanjani, M., Z. Asghari, and N. Emami (2012, 01). A survey of hierarchical clustering algorithms. *The Journal of Mathematics and Computer Science 5*, 229–240.

Larsen, R. and M. Marx (2001). *An Introduction to Mathematical Statistics and Its Applications*. Number v. 1 in 1. Prentice Hall.

Li, L. and B. A. Prakash (2011). Time series clustering: Complex is simpler! In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, USA, pp. 185–192. Omnipress.

Liao, T. W. (2005). Clustering of time series data - a survey. *Pattern Recognition 38*(11), 1857 – 1874.

Lin, J., E. Keogh, L. Wei, and S. Lonardi (2007, Oct). Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery 15*(2), 107–144.

Lines, J. and A. Bagnall (2015, May). Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery 29*(3), 565–592.

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 129–137.

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297.

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

Mallat, S. G. (1989, Jul). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*(7), 674–693.

Meyer, D. and C. Buchta (2017). *proxy: Distance and Similarity Measures*. . R package version 0.4-19.

Mico, L. and J. Oncina (1998). Comparison of fast nearest neighbour classifiers for handwritten character recognition1work partially supported by the spanish grant cicyt tic97-0941.1. *Pattern Recognition Letters 19*(3), 351 – 356.

Milligan, G. (1996). *Clustering Validation: Results and Implications for Applied Analyses*. Reprint series. Max M. Fisher College of Business, Ohio State University.

Milligan, G. W. and M. C. Cooper (1985, Jun). An examination of procedures for determining the number of clusters in a data set. *Psychometrika 50*(2), 159–179.

Montero, P. and J. A. Vilar (2014). TSclust: An R package for time series clustering. *Journal of Statistical Software 62*(1), 1–43.

Mori, U., A. Mendiburu, and J. A. Lozano (2016a). Distance measures for time series in r: The TSdist package. *R journal 8*(2), 451–459.

Mori, U., A. Mendiburu, and J. A. Lozano (2016b). Distance measures for time series in r: The TSdist package. *R journal 8*(2), 451–459.

Morissette, L. and S. Chartier (2013, 02). The k-means clustering technique: General considerations and implementation in mathematica. *Tutorials in Quantitative Methods for Psychology 9*, 15–24.

Murtagh, F. and P. Contreras (2011). Methods of hierarchical clustering. *CoRR abs/1105.0121*, 61–64.

Nason, G. (2008). *Wavelet Methods in Statistics with R* (1 ed.). Springer Publishing Company, Incorporated. Page 26.

Ng, R. T. and J. Han (1994). Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, San Francisco, CA, USA, pp. 144–155. Morgan Kaufmann Publishers Inc.

Ngui, W. K., M. S. Leong, L. M. Hee, and A. M. Abdelrhman (2013, 11). Wavelet analysis: Mother wavelet selection methods. In *Advances in Manufacturing and Mechanical Engineering*, Volume 393 of *Applied Mechanics and Materials*, pp. 953–958. Trans Tech Publications.

Niennattrakul, V. and C. A. Ratanamahatana (2007). On clustering multimedia time series data using k-means and dynamic time warping. In *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*, MUE '07, Washington, DC, USA, pp. 733–738. IEEE Computer Society.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

Rani, S., G. Sikka, and T. W. Liao (2012). Recent techniques of clustering of time series data: A survey. In *International Journal of Computer Applications*, Volume 52.

Ratanamahatana, C. A., J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das (2005). *Mining Time Series Data*, pp. 1069–1103. Boston, MA: Springer US.

Sagvekar, V., V. Sagvekar, and K. Deorukhkar (2013, nov). Performance assessment of clarans: A method for clustering objects for spatial data mining. *Global journal of engineering, design and technology 2*(6), 1–8.

Sakoe, H. and S. Chiba (1978, Feb). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing 26*(1), 43–49.

Salvador, S. and P. Chan (2007, October). Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal. 11*(5), 561–580.

Sarda-Espinosa, A. (2017). Comparing time-series clustering algorithms in r using the dtwclust package. In *Manual of the R package dtwclust*.

Sheikholeslami, G., S. Chatterjee, and A. Zhang (1998). Wavecluster: A multi-resolution clustering approach for very large spatial databases. In ., pp. 428–439.

Shieh, J. and E. Keogh (2008). isax: Indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, New York, NY, USA, pp. 623–631. ACM.

Shieh, J. and E. Keogh (2009, Aug). isax: disk-aware mining and indexing of massive time series datasets. *Data Mining and Knowledge Discovery 19*(1), 24–57.

Singh, A., A. Yadav, and A. Rana (2013). K-means with three different distance metrics. In *International Journal of Computer Applications 67(10)*, pp. 13–17.

Slonim, N., E. Aharoni, and K. Crammer (2013). Hartigan's k-means versus lloyd's k-means: Is it time for a change? In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pp. 1677–1684. AAAI Press.

Steinhaus, H. (1956). Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci 1*, 801–804.

Strohmer, T. and J. Tanner (2005, 02). Implementations of shannon's sampling theorem: A time-frequency approach. *Sampl. Theory Signal Image Process 4*, 1–17.

Swarndeep Saket J, D. S. P. (2016, jun). An overview of partitioning algorithms in clustering techniques. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) 5*(6), 1943–1946.

Tibshirani, R., G. Walther, and T. Hastie (2000). Estimating the number of clusters in a dataset via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63*, 411–423.

Torrence, C. and G. P. Compo (1998). A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society 79*(1), 61–78.

Vendramin, L., R. J. G. B. Campello, and E. R. Hruschka (2010, August). Relative clustering validity criteria: A comparative overview. *Stat. Anal. Data Min. 3*(4), 209–235.

Vlachos, M., G. Kollios, and D. Gunopulos (2002). Discovering similar multidimensional trajectories. In *Proceedings 18th International Conference on Data Engineering*, pp. 673–684.

Walesiak, M. and A. Dudek (2017). *clusterSim: Searching for Optimal Clustering Procedure for a Data Set.* . R package version 0.47-1.

Wang, X., H. Ding, G. Trajcevski, P. Scheuermann, and E. J. Keogh (2010). Experimental comparison of representation methods and distance measures for time series data. *CoRR abs/1012.2789*, 1 – 56.

Ward, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association 58*(301), 236–244.

Xi, X., E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana (2006). Fast time series classification using numerosity reduction. In *In ICML'06*, pp. 1033–1040.

Xu, W., Y. Hou, Y. S. Hung, and Y. Zou (2013, January). A comparative analysis of spearman's rho and kendall's tau in normal and contaminated normal models. *Signal Process. 93*(1), 261–276.

Yan, M. (2005, 01). Methods of determining the number of clusters in a data set and a new clustering criterion. *Virginia Polytechnic Institute and State University 1*, 1–120.

Yi, B.-K., H. V. Jagadish, and C. Faloutsos (1998, Feb). Efficient retrieval of similar time sequences under time warping. In *Proceedings 14th International Conference on Data Engineering*, pp. 201–208.

Zhou, J., S.-F. Zhu, X. Huang, and Y. Zhang (2015, Jul). Enhancing time series clustering by incorporating multiple distance measures with semi-supervised learning. *Journal of Computer Science and Technology 30*(4), 859–873.