Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrics and Operations Research

Evaluating elastic distance measures and ensemble methods for time series clustering

Jesse van der Ende (622894)

Supervisor:	Jeffrey Durieux
Second assessor:	Max Welz
Date final version:	July 1st, 2024

Abstract

Clustering is an unsupervised machine learning technique that groups similar objects together without the use of labels. With the emerging need for the analysis of time series data, techniques like clustering have received increased attention for performing exploratory data analysis. This paper investigates the use of various elastic distance measures for partitional time series clustering algorithms. It evaluates nine elastic distance measures within the k-means and k-medoids frameworks, aiming to offer a comprehensive description and summary by replicating the findings from Holder, Middlehurst and Bagnall (2024). Additionally, this paper adds to the literature by using two ensemble methods alongside these elastic distances to evaluate whether the general increase in performance of ensemble methods for non-temporal data also translates to time series. Our results, obtained by applying the models to a subset of the UCR archive, show some deviation from the findings of Holder et al. (2024). Consistent with Holder et al. (2024), the Move-Split-Merge (MSM) distance ranks as one of the best-performing distances. Contrary to their findings, Edit distance with Real Penalty (ERP) also comes out as one of the top performers in our analysis while Time Warp Edit (TWE) is worse than both MSM and ERP. The analysis of the ensemble methods reveals promising results for the Cluster-based Similarity Partitioning Algorithm (CSPA), but the performance of CSPA is dependent on a variety of factors such as the ensemble selection process.

1 Introduction

With the recent advancements in big data, the need for information extraction methods from complex datasets has increased (Aghabozorgi, Shirkhorshidi & Wah, 2015). Real-world applications can store data for a long time, such that many fields store this data in a time series format. The lack of labels in these datasets makes the use of supervised classification methods often not feasible, increasing the need for unsupervised techniques on time series data.

This paper focuses on clustering, which is one of the most well-known unsupervised machine learning techniques (Bijuraj, 2013). Clustering serves various purposes, such as exploratory data analysis, and is utilized in fields many fields, including pattern recognition, image analysis, finance and bioinformatics. While it can be applied to many types of data, our focus is only on using cluster algorithms for time series data.

Similar to non-temporal data, clustering time series involves grouping together objects in such a way that those within the same group (or cluster) are more similar to each other than to those in other groups. However, due to the temporal nature of time series, algorithms developed for non-temporal data fail to provide accurate clusters. Many alternatives to these algorithms have been developed in the literature, but this paper focuses solely on using elastic distance measures for distance-based partitional clustering algorithms. These elastic distances allow for alignment across the time axis, making them well-suited for time series data.

Our aim is to deliver a concise evaluation of nine commonly used elastic distance measures by replicating and validating the findings presented in Holder et al. (2024), which is a similar study on elastic distances. Building on this foundation, we further investigate how ensemble methods such as the Cluster-based Similarity Partitioning Algorithm (CSPA) can enhance the clustering accuracy of these elastic distance functions. Ensemble methods, which combine multiple models to improve overall performance, have shown success in the field of time series classification with elastic distance functions (Lines & Bagnall, 2015), but remain under-explored in the context of time series clustering.

The data used in this study comes from the UCR archive (Dau et al., 2019), a widely recognized repository of pre-labelled time series datasets. Two partitioning clustering algorithms, k-means and k-medoids, are applied to a subset of the archive to address the following research question:

How do various elastic distance measures, comprising of (weighted/derivative) Dynamic Time Warping, Longest Common Subsequence, Edit distance on Real Sequences, Edit distance with Real Penalty, Move-Split-Merge and Time Warp Edit, compare in their effectiveness for time series clustering using k-means and k-medoids algorithms, and how do ensemble methods like the Cluster-based

Similarity Partitioning Algorithm (CSPA) and simpler voting methods enhance the clustering performance compared to individual distance measures?

The results show that the Move-Split-Merge (MSM) and Edit distance with Real Penalty (ERP) distances perform best in terms of accuracy, while Longest Common Subsequence (LCSS) and Edit distance on Real Sequences (EDR) are among the worst performers. These results deviate from Holder et al. (2024), where ERP was significantly worse than MSM and TWE was also among the best performers. In line with Holder et al. (2024), DTW was significantly outperformed by other distances such as MSM, giving additional support for their notion that DTW may not be the ideal standard benchmark for time series clustering. Furthermore, the mutual performance of the distance measures was independent of the clustering algorithm, with k-medoids generally outperforming k-means.

For the ensemble methods, the experiments show that only the CSPA method could improve the results significantly. While this yields interesting directions for further research, it is important to note that using ensemble methods for time series clustering is not straightforward as the performance is heavily influenced by both the selection process and the final clustering algorithm.

The remainder of this paper adheres to the following structure. In Section 2, we review the literature. In Section 3, the data is described, along with some important notes regarding its usage. Section 4 outlines the methodology for the clustering algorithms, distance measures, ensemble methods and performance measures. Finally, the experimental results are presented in Section 5, followed by the conclusion in Section 6.

2 Literature Review

As mentioned in the previous section, this paper focuses on using partitional algorithms in the framework of time series clustering. These partitional algorithms divide the data into distinct non-overlapping clusters, but many other approaches to time series clustering have been proposed in the literature.

One of these approaches is the hierarchical clustering method, which constructs embedded clusterings that can be represented as a tree. Łuczak (2016) investigates the use of Dynamic Time Warping (DTW) and Derivative Dynamic Time Warping (DDTW) for hierarchical clustering methods for time series. Their results show that a combination of DTW and DDTW into a new distance function significantly outperforms the individual distance measures. Another hierarchical clustering approach is proposed by Pasupathi, Shanmuganathan, Madasamy, Yesudhas and Kim (2021), where the use of DTW together with Paradigmatic Time Sequences to perform trend analysis is investigated.

Alternatively, density-based approaches such as DBSCAN are used for clustering (Ester, Kriegel, Sander & Xu, 1996). These approaches rely on density instead of distance measures to cluster objects together, making them robust to noise and outliers. Because DBSCAN is quite sensitive to its input parameters, Sawant (2014) introduced adaptive methods which automatically select these parameters. They found evidence that these adaptive methods perform considerably well, mitigating the time-consuming task of setting and tuning the parameters.

Instead of using the raw time series as input for the clustering algorithms, it is also possible to perform feature extraction prior to clustering. An example of this is Räsänen and Kolehmainen (2009), where the features were extracted based on statistical measures such as mean and standard deviation. Others, such as Zhang, Ho, Zhang and Lin (2006), have used different feature extraction algorithms based on orthogonal wavelets. Both papers showed increased accuracy when compared to clustering on the raw set, illustrating the increase in performance that can be made by using feature extraction.

This study is not the first evaluation of time series clustering procedures. One example of another review is Aghabozorgi et al. (2015), where a theoretical overview of commonly used time series clustering algorithms is given. A more experimental approach for reviewing time series clustering algorithms is employed in Javed, Lee and Rizzo (2020), where partitional, hierarchical and density-based

clustering algorithms are employed together with various types of distance measures.

An analysis very similar to this study was done in Bagnall, Lines, Bostrom, Large and Keogh (2017). This paper evaluated the use of different elastic distance measures for time series classification instead of clustering. They used the same UCR archive that is employed in our study and found that the DTW distance provides a hard-to-beat benchmark for classification. However, the Collective of Transformation-Based Ensembles (COTE) method developed in Bagnall, Lines, Hills and Bostrom (2015) proved to be significantly more accurate than DTW.

Multiple ensemble approaches for clustering have been proposed in the literature. Strehl and Ghosh (2002) proposed three different graph-based ensemble methods. First, they proposed the Cluster-based Similarity Partitioning Algorithm (CSPA), which is also utilized in this study. This approach combines multiple clusterings by representing them as a undirected graph and uses a minimal cut partitioning algorithm to obtain the final clusters. Second, Strehl and Ghosh (2002) proposed the HyperGraph Partitioning Algorithm, in which the ensemble is represented by a hypergraph that is again partitioned into the final clusters. Lastly, they proposed the Meta-Clustering Algorithm (MCLA), which also makes use of a hypergraph but clusters the hyperedges together to obtain a so-called meta-clustering. This meta-clustering representation is then used to obtain the final clusters. All three ensemble methods are known to be able to improve results compared to individual clustering algorithms.

Another approach to cluster ensembles has been proposed by Wang, Shan and Banerjee (2011). They proposed the Bayesian Cluster Ensemble (BCE) model, which does not represent the ensemble with a graph but instead uses a Bayesian probabilistic framework. By assuming that each object in the dataset has a mixed membership to the different clusters in the final clustering, the BCE model uses a discrete probability distribution to extract the final clusters with the largest posterior probability. Wang et al. (2011) have shown that this method can handle uncertainty in the clustering process and is able to deal with missing data.

Finally, Alqurashi and Wang (2019) proposed the Adaptive Clustering Ensemble (ACE) method. In the ACE method, the most similar clusters are aggregated in an iterative procedure after which the objects with high uncertainty are dealt with to ensure that each object is assigned to only one cluster. This method has been shown to outperform other cluster algorithms such as MCLA and can deal with a variable number of clusters generated by the ensemble members.

3 Data

The data used in this paper was retrieved from the UCR time series archive[1]. This archive contains a total of 128 pre-labeled time series datasets divided into 15 different types. For each dataset, there is a default train and test split available, which is utilized in this paper to facilitate easy reproduction.

The maintainers of the repository have established some best practices for using the UCR data (Dau et al., 2019). They advised to use all available datasets to prevent 'cherry picking'. However, due to the time-consuming nature of the algorithms that are used in this study, it is not feasible to comply with this recommendation for the given time constraints. Therefore, only a subset of datasets was selected from the archive. A complete overview of the used datasets and their characteristics is given in Appendix A. These particular datasets were selected because they contain relatively few training instances and/or feature short time series lengths.

Selecting only certain datasets can influence the results. As such, we must exercise caution when drawing our conclusions. Still, Dau et al. (2019) noted that choosing a subset can give general insights, provided that the choice of datasets is adequately justified (for instance, because of time constraints).

Concerning the normalization of time series data, it is commonly considered best practice to apply methods to both normalized and raw data (Lima & Souza, 2023). However, running the algorithms

[1] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

twice is not feasible in our given time frame. Therefore, we opt to normalize all data, since normalization is only discouraged very rarely (Dau et al., 2019). Additionally, some datasets in the repository have already been normalized, meaning that normalizing the data ensures equal treatment of all datasets.

4 Methodology

4.1 Clustering algorithms

This section will discuss the k-means and k-medoids algorithms, which are utilized to perform the time series clustering. Throughout this paper, clustering algorithms are often referred to as clusterers, whereas a set of clusters produced by these clusterers is named a clustering.

Before discussing the clustering algorithms, we introduce some mathematical notation. Let x_j denote a single observation, such that $\mathbf{x} = (x_1, \ldots, x_m)$ denotes a time series of m observations. It is assumed that the time series are univariate and of equal length, implying that x_j are scalars and m is the same for each time series in a dataset. Then, let $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$ denote the dataset of n time series which needs to be divided into k clusters. The output of the clustering algorithms, denoted by $C = \{C_1, C_2, \ldots, C_k\}$, contains k sets of observations which represent the clusters.

4.1.1 K-means

The first clustering algorithm that will be used is the well-known k-means algorithm (MacQueen, 1967), also commonly referred to as Lloyd's algorithm (Lloyd, 1982). This algorithm partitions the data based on the distances between the objects. Central in k-means is the idea of representing each cluster with a so-called centroid, which is calculated by taking the mean of all points belonging to that cluster.

The k-means algorithm is initialized by selecting k initial centroids through a specific method (details of this procedure in Section 4.1.3). Then, an iterative process starts where each object will be assigned to the closest cluster centroid and the centroids are updated. This process continues until a convergence criterion is met, which is often based on a measure known as *inertia* (Ikotun, Ezugwu, Abualigah, Abuhaija & Heming, 2023). This *inertia* denotes the sum of distances between each data point and its centroid, and for this study convergence is achieved when the change in inertia falls below a certain threshold. Furthermore, to limit computation times, a maximum number of iterations is set. The pseudocode for the k-means procedure can be found in Algorithm 2 in Appendix B.

One general problem with k-means is that it is possible to obtain empty clusters (Ikotun et al., 2023). Therefore, each time an empty cluster is encountered, an object is assigned to this cluster. To reduce *inertia* the most, this object is chosen to be the object that has the largest distance to the centroid of the currently assigned cluster.

4.1.2 K-medoids

An adaptation of k-means, known as k-medoids, has been proposed by Kaufman and Rousseeuw (1987). This algorithm makes use of medoids, which are the data points in the clusters with the minimum average dissimilarity to all other points in the assigned cluster. By using medoids instead of centroids to represent the clusters, k-medoids resolves the sensitivity of k-means to outliers (Ikotun et al., 2023). By using both k-means and k-medoids, we can test which of the two algorithms performs best for time series clustering and if the accuracy of the different distance measures is (partly) dependent on the clustering algorithm.

The k-medoids algorithm proposed in Kaufman and Rousseeuw (1987) is implemented by using the Partitioning Around Medoids (PAM) method, which swaps medoids with a data point if the swap decreases the total dissimilarity. However, in a similar manner to k-means, the k-medoids algorithm

used in this paper is implemented using Lloyds algorithm (Lloyd, 1982). This means that, with some minor changes, Algorithm 2 is also applicable to k-medoids. When updating the cluster centres, instead of the mean we find the data point which minimizes the total dissimilarity. Furthermore, the calculation of the distances between data points is done once, prior to the iterative procedure. Handling empty clusters is not needed for k-medoids, as each medoid is an actual data point and is therefore always assigned to its own cluster.

Just like for k-means, convergence for k-medoids is achieved when either a certain number of iterations has been reached or the change in *intertia* between iterations falls below a certain threshold.

4.1.3 Initialization

Because both partitional clustering algorithms are sensitive to initialization issues, examining the effect of different initialization procedures can be crucial for the performance of the clusterers (Ikotun et al., 2023). The problem of initialisation has two aspects: determining the value of k and determining the initial centres of the clusters.

In their review on existing techniques to determine k, Xu et al. (2016) stated that most of these techniques require the algorithm to be run multiple times for different values of k. Because the computation of elastic distances can be computationally expensive, running the algorithms multiple times is not feasible in our time frame. While some proposed methods do not require multiple runs, such as I-k-means (Ismkhan, 2018) and X-means (Pelleg & Moore, 2000), we decided to follow the direction of Holder et al. (2024) and assume the number of clusters is known beforehand.

For the initialisation of the cluster centres, three different procedures are considered. First, we consider Forgy's method (Forgy, 1965), which randomly assigns each data point to the clusters and then computes the cluster centres as initial centres.

Second, we consider random initialisation (MacQueen, 1967), which chooses the initial centres randomly from the data. The difference between random initialisation and Forgy's method is that random initialisation has some theoretical support, while Forgy's method has none. By selecting centres based on a uniform distribution, we obtain good potential candidates for the final cluster centres as they most likely come from regions with a high density.

Lastly, we consider the k-means++ algorithm (Arthur & Vassilvitskii, 2006). This algorithm takes the first centre randomly from the data and selects the rest with a certain probability. If we let $md(\mathbf{x})$ denote the minimum distance from point $\mathbf{x}$ to previously selected centres, then the probability for $\mathbf{x}$ to be selected next is given by

$$\frac{md(\mathbf{x})^2}{\sum_{j=1}^{n} md(\mathbf{x}_j)^2}. \tag{1}$$

The intuition behind this probability is that if a data point is close to other cluster centres, we select it with a lower probability. This may result in more well-separated cluster centres, which could benefit the clustering procedure.

It is noted by Celebi, Kingravi and Vela (2013) that these three initialization techniques are often outperformed by other methods such as PCA-Part (Su & Dy, 2007) and Bradley and Fayyad's method (Bradley & Fayyad, 1998). However, since they are simple to implement and do not increase computation times significantly, they still can give us valuable insights regarding the effect of initialization on the performance of the distance measures. Additionally, using multiple initialization techniques gives us additional candidates for the ensemble methods (see Section 4.3).

4.2 Distance measures

The clustering algorithms of the previous section work with the distances between two objects. Often, for non-time-series data, the Euclidean distance is used as a standard for these distance-based procedures. However, when using the Euclidean distance for time-series data, two points are compared

directly without allowing for any flexibility in alignment. This makes the Euclidean distance sensitive to distortions in time and less suited for time-series data where temporal variability regularly occurs (Keogh & Ratanamahatana, 2005).

To deal with time distortions, we aim to use the same nine elastic distance measures as Holder et al. (2024). These elastic distance measures match points across sequences, allowing for a flexible alignment which is more robust to distortions in time. Lines and Bagnall (2015) have shown that using these elastic distance measures can significantly improve the accuracy of time series classification. The objective is to assess whether these improvements are also present for clustering and to validate the results from Holder et al. (2024). Table 1 shows the nine different elastic distance measures that will be used, together with the Euclidean distance as a benchmark.

Distance	Abbreviation	Source
Euclidean distance	ED	-
Dynamic Time Warping	DTW	Ratanamahatana and Keogh (2005)
Derivative DTW	DDTW	Keogh and Pazzani (2001)
Weighted DTW	WDTW	Jeong, Jeong and Omitaomu (2011)
Derivative WDTW	DWDTW	Jeong et al. (2011)
Longest Common Subsequence	LCSS	Soleimani and Abessi (2020)
Edit distance on Real Sequences	EDR	Chen, Özsu and Oria (2005)
Edit distance with Real Penalty	ERP	Chen and Ng (2004)
Move-Split-Merge	MSM	Stefan, Athitsos and Das (2013)
Time Warp Edit	TWE	Marteau (2009)

Table 1: List of the ten distance functions reviewed in Holder et al. (2024).

To simplify the notation, we follow the notation of Holder et al. (2024) by denoting two time series of length m with $\mathbf{a} = (a_1, \ldots, a_m)$ and $\mathbf{b} = (b_1, \ldots, b_m)$, where $\mathbf{a}, \mathbf{b} \in X$. The Euclidean distance d_{ED} between these two time series is then given by

$$d_{\text{ED}}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2}. \tag{2}$$

4.2.1 Dynamic Time Warping

One of the most widely used elastic distance measures for time series is Dynamic Time Warping (DTW; Keogh & Pazzani, 2001). By minimizing the distance between two time series, DTW allows for a non-linear mapping of one time series to the other. Because of this so-called 'warping', DTW can achieve a good alignment for two time series which have approximately the same shape but do not line up on the time-axis (Keogh & Ratanamahatana, 2005).

To align two time series $\mathbf{a}$ and $\mathbf{b}$, DTW uses an $m \times m$ distance matrix M. Each element of the matrix $M_{i,j}$ denotes the squared difference between a_i and b_j, which is used to compute the distance of a so-called warping path. A warping path, denoted by $P =< (e_1^P, f_1^P), (e_1^P, f_2^P), ..., (e_s^P, f_s^P) >$, is a set of elements of size s from M that defines a mapping between $\mathbf{a}$ and $\mathbf{b}$. To be valid, the warping path must start and finish in diagonally opposite corners of the matrix M, such that it must hold that $(e_1^P, f_1^P) = (1, 1)$ and $(e_s^P, f_s^P) = (m, m)$. Furthermore, the path must be continuous and monotonic, meaning that $0 \leq e_{i+1}^P - e_i^P \leq 1$ and $0 \leq f_{i+1}^P - f_i^P \leq 1$ for all $1 \leq i \leq m$.

There are exponentially valid warping paths, but for the DTW distance we are interested in the one that has the minimum total distance. Obtaining this minimum warping path can be time-consuming and therefore the amount of warping is constrained by using the Sakoe-Chiba band (Sakoe & Chiba, 1978). This band restricts the window of the warping to be around a certain width of the diagonal of M. The width is specified by the window size w, which is the percentage of cells the warping path is allowed to deviate from the diagonal.

Let P^{w*} denote the warping path with the minimal total distance for a window size of w. Furthermore, let $d_{P^{w*}}(M, w)$ denote the total distance of this path for a given distance matrix M of two time series $\mathbf{a}$ and $\mathbf{b}$. The final DTW distance $d_{\text{DTW}}(\mathbf{a}, \mathbf{b}, w)$ is then given by

$$d_{\text{DTW}}(\mathbf{a}, \mathbf{b}, w) = d_{P^w}(M, w) = \sum_{i=1}^{s} M_{e_i^{P^{w*}}, f_i^{P^{w*}}}. \tag{3}$$

This DTW distance is obtained by using a dynamic programming formulation given in Algorithm 3 in Appendix B.

4.2.2 Derivative Dynamic Time Warping

Numerous modifications of the DTW distance have been proposed by the literature. While the DTW distance is successful in aligning two time series which have distortions on the time axis, it is less suited for scenarios when the time series also differ in the y-axis. Therefore, Keogh and Pazzani (2001) have proposed an alternative called Derivative Dynamic Time Warping (DDTW), which considers the shape of the times series instead of the actual values. To do this, they transform the time series into an approximation of the derivative of the series.

Let $\mathbf{a}' = (a_2', a_3', \ldots, a_{m-1}')$ denoted the transformed series of time series $\mathbf{a}$, where a_i' is defined as in Eq. 4 for $1 < i < m$.

$$a_i' = \frac{(a_i - a_{i-1}) + (a_{i+1} - a_{i-1})/2}{2} \tag{4}$$

The DDTW distance is defined as the DTW of the transformed series, such that it is given by

$$d_{\text{DDTW}}(\mathbf{a}, \mathbf{b}, w) = d_{\text{DTW}}(\mathbf{a}', \mathbf{b}', w). \tag{5}$$

Just like DTW, the DDTW distance can be obtained by using Algorithm 3 in Appendix B, but now on the transformed series.

4.2.3 Weighted Dynamic Time Warping

Another modification of the DTW distance was proposed by Jeong et al. (2011). This new method, called Weighted Dynamic Time Warping (WDTW), weights the different points in the time series to adjust for the relative significance of two points in different phases of the time series. By weighting the time series in this way, Jeong et al. (2011) claimed that they could reduce the mapping of multiple points from one series to one single point in the other series, potentially benefiting the accuracy of time series classification algorithms.

The key idea is that two points get a smaller weight penalty when the phase difference between them is small and vice versa. These weights are given by a logistic function, as given in Eq. 6. Here, $w_{\max}$ defines the upper bound for the weights (set to 1 in this paper) and g is a parameter that controls the curvature of the weight function.

$$w(x) = \frac{w_{\max}}{1 + e^{-g \cdot (x - m/2)}} \tag{6}$$

The actual weight penalties are imposed when calculating the distance matrix M. For each warping distance $|i - j|$, a penalty of $w(|i - j|)$ is imposed, such that the elements of the new distance matrix M^{WDTW} become

$$M_{i,j}^{WDTW} = w(|i - j|) \cdot (a_i - b_i)^2. \tag{7}$$

By using this matrix as input for the dynamic programming formulation from Algorithm 3, we can obtain the WDTW distance. It is worth noting that using a reduced window size w does not benefit WDTW, as this reduced search space is already accounted for in the weighting process. The final

WDTW distance measure is therefore given by

$$d_{\text{WDTW}}(\mathbf{a}, \mathbf{b}, g) = d_{P^{w*}}(M^{WDTW}, w = 1) = \sum_{i=1}^{s} M^{WDTW}_{e_i^{P^{w*}}, f_i^{P^{w*}}}. \tag{8}$$

Jeong et al. (2011) also proposed a combination of WDTW and DDTW called Weighted Derivative DTW (WDDTW). This distance is defined in Eq. 9.

$$d_{\text{WDDTW}}(\mathbf{a}, \mathbf{b}, g) = d_{\text{WDTW}}(\mathbf{a}', \mathbf{b}', g) \tag{9}$$

4.2.4 Longest Common Subsequence

Instead of warping a point from a series onto a point from the other series, we can also view the alignment process as finding the common elements in two series. Often used for this is the Longest Common Subsequence (LCSS) method, which uses an edit-based approach to measure the distance between two series (Aghabozorgi et al., 2015). By finding the longest subsequence of matching elements, LCSS can define a distance between two time series. An advantage of looking at the alignment process in this way is that it makes the distance measure quite robust to noise (Chen et al., 2005).

Originally, the LCSS method was developed for series with discrete elements (Soleimani & Abessi, 2020). For discrete elements, two elements match if they are equal. However, for a series with continuous elements, it is likely that two values will never exactly be equal. Therefore, it is necessary to redefine the notion of a match to construct the longest common subsequence. In this case, two values are considered matching if their absolute difference is less than a certain threshold ϵ (Soleimani & Abessi, 2020).

To obtain the longest common subsequence of two time series $\mathbf{a}$ and $\mathbf{b}$, we construct a matrix $L \in \mathbb{R}^{(m+1)\times(m+1)}$ which is indexed from 0. Each element in this matrix represents the LCSS for two subseries of $\mathbf{a}$ and $\mathbf{b}$, meaning that $L_{m,m}$ represents the final LCSS for the complete series. The matrix is defined by Eq. 10, where we can observe that the LCSS is incremented if two elements match. If the two elements do not match, then a previously seen LCSS is carried over.

$$L_{i,j} = \begin{cases} 0 & \text{if } i = 0 \vee j = 0. \\ 1 + L_{i-1,j-1} & \text{if } |a_i - b_j| < \epsilon \wedge i \geq 1 \wedge j \geq 1. \\ \max(L_{i-1,j}, L_{i,j-1}) & \text{if } |a_i - b_j| \geq \epsilon \wedge i \geq 1 \wedge j \geq 1. \end{cases} \tag{10}$$

The LCSS is a measure of similarity, while for the clustering algorithms a distance is needed. To obtain an actual distance function, we can transform this similarity such that we obtain our final distance measure in Eq. 11. Here, $L_{m,m}(\epsilon)$ denotes the last element of matrix L for given $\mathbf{a}$, $\mathbf{b}$ and ϵ.

$$d_{\text{LCSS}}(\mathbf{a}, \mathbf{b}, \epsilon) = 1 - \frac{L_{m,m}(\epsilon)}{m} \tag{11}$$

4.2.5 Edit distance on Real Sequences

An adaptation of LCSS has been proposed in Chen et al. (2005), called Edit distance on Real Sequences (EDR). Both LCSS and EDR use a threshold ϵ to define when two elements match but use a different counting mechanism. Where LCSS just counts the number of matches to construct a similarity measure, EDR assigns a constant cost for inserting, deleting or replacing an element in a series. This way, we directly obtain a distance measure that seeks the minimum number of edit operations to transform one series into another series (Chen et al., 2005)

Just like for LCSS, we construct a matrix $E \in \mathbb{R}^{(m+1)\times(m+1)}$ indexed from 0 that represents the EDR for multiple subseries of $\mathbf{a}$ and $\mathbf{b}$. The matrix is defined in Eq. 12, where $E_{m,m}$ denotes the final

EDR distance between **a** and **b**. The intuition behind this matrix is that moving diagonally across the matrix can be seen as matching two elements. If these elements indeed match the cost of matching is 0. However, if they are not matching we have to replace one of the elements for a cost of 1. When moving horizontally or vertically in the matrix, we have to insert or delete an element from one of the series, also resulting in a cost of 1. This way, EDR describes the cost of transforming one series into another.

$$E_{i,j} = \begin{cases} 0 & \text{if } i = 0 \vee j = 0. \\ \min(E_{i-1,j-1} + 0, E_{i-1,j} + 1, E_{i,j-1} + 1) & \text{if } |a_i - b_j| < \epsilon \wedge i \geq 1 \wedge j \geq 1. \\ \min(E_{i-1,j-1} + 1, E_{i-1,j} + 1, E_{i,j-1} + 1) & \text{if } |a_i - b_j| \geq \epsilon \wedge i \geq 1 \wedge j \geq 1. \end{cases} \tag{12}$$

The final EDR distance is defined in Eq. 13, with $E_{m,m}(\epsilon)$ denoting the last element of matrix E given **a**, **b** and ϵ.

$$d_{\text{EDR}}(\mathbf{a}, \mathbf{b}, \epsilon) = E_{m,m}(\epsilon) \tag{13}$$

4.2.6 Edit distance with Real Penalty

Another alternative to LCSS and EDR was described in Chen and Ng (2004), where they proposed the Edit distance with Real Penalty (ERP). Similar to EDR, ERP assigns costs for inserting, deleting or replacing an element. However, unlike EDR, these costs are not constant and dependent on the element itself. One advantage is that ERP satisfies the triangle inequality, which can speed up computations in for example the k-Nearest-Neighbour algorithm (Chen & Ng, 2004).

Similar to both LCSS and EDR, the ERP method constructs a matrix $R \in \mathbb{R}^{(m+1)\times(m+1)}$ indexed from 0 that represents the ERP for multiple subseries of **a** and **b**. The matrix, which is defined in Eq. 14, shows that the cost of matching two elements is equal to the distance between these two elements. Intuitively this makes sense: if two elements are very similar the cost of matching them should be low and vice versa. Furthermore, when inserting or deleting an element, the cost is equal to the distance of that element to a constant parameter g. Defining the distance in this way ensures that ERP satisfies the triangle inequality.

$$R_{i,j} = \begin{cases} \sum_{k=1}^{m} |b_k - g| & \text{if } i = 0. \\ \sum_{k=1}^{m} |a_k - g| & \text{if } i \geq 1 \wedge j = 0. \\ \min(R_{i-1,j-1} + |a_i - b_j|, R_{i-1,j} + |a_i - g|, R_{i,j-1} + |b_i - g|) & \text{if } i \geq 1 \wedge j \geq 1. \end{cases} \tag{14}$$

The last element of the matrix R denotes the ERP distance between two time series **a** and **b** for a given value of g. If we denote this value with $R_{m,m}(g)$, we get that the final ERP distance is given by

$$d_{\text{ERP}}(\mathbf{a}, \mathbf{b}, g) = R_{m,m}(g). \tag{15}$$

4.2.7 Move-Split-Merge

One of the disadvantages of ERP is that it is highly dependent on the value of g and changing this can substantially affect the outcome (Stefan et al., 2013). Furthermore, ERP favours inserting and deleting values that are close to zero. Because of these disadvantages, the Move-Split-Merge (MSM) distance was introduced by Stefan et al. (2013). By using operations which are slightly different from insertion and deletion, MSM tries to mitigate the problems of ERP.

The main idea of MSM is that the cost of inserting or deleting a value should not be dependent on the absolute magnitude of the value (like in ERP), but rather should depend on both the inser-

ted/deleted value and the adjacent values. For example, inserting a value of 5 between a 4 and a 6 should cost less than inserting a value of 5 between two 0s.

To achieve this concept, MSM uses three main operations: Move, Split and Merge. The Move operation is the same as the substitution operation in ERP and replaces one element with another. The cost of this operation is equal to the absolute value of the difference between the new and old element. The Split operation repeats a certain value twice with as cost a constant parameter c. The Merge operation merges two successive values into one value and is therefore the inverse of the Split operation. The cost of the Merge operation is also equal to c.

$$Cost(x, y, z, c) = \begin{cases} c & \text{if } y \leq x \leq z \vee y \geq x \geq z. \\ c + \min(|x - y|, |x - z|) & \text{otherwise.} \end{cases} \tag{16}$$

Eq. 16 defines a cost function which summarizes the costs for the different operations. The underlying principle of this cost function is that inserting a value x between two values y and z which are very different from x requires an additional Move operation. This leads to an increased cost compared to when y and z are close to x. As the exact derivation of the cost function is out of the scope of this paper we will not discuss it further, but for interested readers we refer to Stefan et al. (2013).

By using Eq. 16, we can then obtain the MSM distance for given $\mathbf{a}$, $\mathbf{b}$ and c by using a similar dynamic programming algorithm as for DTW (given in Algorithm 4 in Appendix B). It is worth noting that, just like ERP, MSM satisfies the triangle inequality, yielding potential benefits regarding the optimization of algorithms and upper-bounding.

4.2.8 Time Warp Edit

The last elastic distance measure that will be evaluated is Time Warp Edit (TWE), proposed by Marteau (2009). This distance measure uses two types of parameters: a stiffness parameter γ which controls the warping along the time axis and a penalty parameter λ which penalises the deletion and insertion operations. A high γ results in a heavy penalization of warping between two points which differ a lot on the time axis, similar to the g parameter in WDTW. By using both γ and λ, TWE finds a balance between the characteristics of the warping and edit approaches. Furthermore, for $\lambda > 0$, TWE satisfies the triangle inequality (Marteau, 2009), yielding the same benefits as ERP and MSM.

The algorithm to obtain the TWE distance can be found in Algorithm 5 in Appendix B.

4.2.9 Parameter values

In the previous sections, nine distances were presented which all require the specification of one or more hyperparameters. Generally speaking, it is believed that tuning the hyperparameters of machine learning algorithms is essential to be able to accurately evaluate the performance. However, as shown by Holder et al. (2024), tuning the parameters of the distance functions did not significantly improve performance. For this reason, and due to the significant time required for hyperparameter tuning, we omit the tuning procedure and use the same values as used by Holder et al. (2024). An overview of the different parameter values is given in Appendix C.

4.3 Ensemble methods

While it is interesting to investigate which elastic distance measure performs best, ultimately we are interested in the best clustering of the dataset. All of the elastic distance measures utilized in this paper appear to have a solid theoretical foundation, suggesting that each may possess some validity for determining the optimal clustering. Therefore, two cluster ensemble methods are evaluated, which

can combine the outcomes of different cluster methods into one final clusterings. These ensemble methods have been known for being able to improve performance compared to individual clusterers (Ghosh & Acharya, 2011).

4.3.1 Simple Alignment Voting Method (SAVM)

The first ensemble method that will be considered is a method proposed by Zhou and Tang (2006). This approach uses an algorithm to align the clusters and then uses a (weighted) voting mechanism to determine the final clustering. A similar method has been used for time series classification in Lines and Bagnall (2015), where the implementation of voting mechanisms significantly improved the classifications generated by algorithms using various elastic distance measures. Given the similarity between the data and distance functions of Lines and Bagnall (2015) and this study, it would be worthwhile to investigate whether the improved accuracy of voting methods extends to time series clustering.

Let $\{C_1^{(a)}, C_2^{(a)}, \ldots, C_k^{(a)}\}$ and $\{C_1^{(b)}, C_2^{(b)}, \ldots, C_k^{(b)}\}$ denote the clusterings produced by two clusterers a and b respectively. Then, by counting the number of overlapping data points for each pair $C_i^{(a)}$ and $C_j^{(b)}$, we can pair the two clusters with the largest overlap. By repeating this procedure until all clusters have been paired, we obtain an alignment of two clusterings to which we can apply a voting mechanism. The algorithm for the alignment procedure is given in Algorithm 1.

Algorithm 1 The align process for two clusterers (adapted from: Zhou & Tang, 2006).

Input: $\{C_1^{(a)}, C_2^{(a)}, \ldots, C_k^{(a)}\}$ (clustering produced by clusterer a) and $\{C_1^{(b)}, C_2^{(b)}, \ldots, C_k^{(b)}\}$ (clustering produced by clusterer b).
Output: M (the aligned clusters).

Initialise matrix OVERLAP$\in \mathbb{R}^{(k\times k)}$ of zeros.
Let Count(x, y) be a function that counts the number of data points that appear in both x and y.
for $i = 1$ to k **do**
    for $j = 1$ to k **do**
        OVERLAP$_{ij}$ = Count$(C_i^{(a)}, C_j^{(b)})$
    end for
end for
$\Gamma = \emptyset$
$M = \emptyset$
while $\Gamma \neq \{C_1^{(b)}, C_2^{(b)}, \ldots, C_k^{(b)}\}$ **do**
    (u,v) = argmax(OVERLAP$_{ij}$)
    $M = M \cup \{(C_u^{(a)}, C_v^{(b)})\}$ ▷ Match clusters $C_u^{(a)}$ and $C_v^{(b)}$
    Delete OVERLAP$_{u*}$ ▷ Delete the row related to u
    Delete OVERLAP$_{*v}$ ▷ Delete the column related to v
    $\Gamma = \Gamma \cup \{C_v^{(b)}\}$
end while
return M

When dealing with $t > 2$ clusterers, this alignment process will not work directly. Therefore, one clusterer has to be selected as a baseline to which other clusterers are then aligned. Similar to Zhou and Tang (2006), we choose the baseline clusterer randomly.

To then finally combine the aligned clusterings, Zhou and Tang (2006) use four different voting schemes. First, they propose the *equal-voting* method, where each data point is assigned to the cluster in which it is most often present.

Second, they propose a *weighted-voting* method, where each clustering is given a weight based on the Normalized Mutual Information (NMI) criterion (Cover, 1999). Suppose we have two label vectors $\boldsymbol{\lambda}^{(a)}$ and $\boldsymbol{\lambda}^{(b)}$ representing the clusterings obtained by clusterers a and b. The corresponding NMI between these two clusterings, denoted by $\Phi^{NMI}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)})$, provides a measure which quantifies the

information shared between the two clusterings. Then, for a given clusterer m, we can compute the average NMI β_m with

$$\beta_m = \frac{1}{t-1} \sum_{l=1, l \neq m}^{t} \Phi^{NMI}(\boldsymbol{\lambda}^{(m)}, \boldsymbol{\lambda}^{(l)}) \quad (m = 1, 2, \ldots, t). \tag{17}$$

This β_m is a measure of the statistical information that the mth clusterer contains that has not been contained by other clusterers. The higher β_m, the less information the mth clusterer adds to the ensemble. The weights w_m in the voting method for clusterer m can then be computed as

$$w_m = \frac{1}{\beta_m Z} \quad (m = 1, 2, \ldots, t), \tag{18}$$

where Z is a normalization factor that ensures that all weights are positive and sum to 1.

Next to these two ensemble schemes, Zhou and Tang (2006) also use two selective approaches where they only select a subset of the clusterers. They do so by excluding the clusters which have a weight smaller than $\frac{1}{t}$. This results in four final ensemble schemes: *voting*, *weighted-voting*, *selective-voting* and *selective-weighted-voting*.

For the remainder of this paper, we will refer to this approach as the Simple Alignment Voting Method (SAVM). Please note that this name is created for ease of reference and is not part of the original authors' terminology.

4.3.2 Cluster-based Similarity Partitioning Algorithm (CSPA)

The second ensemble method we consider is the Cluster-based Similarity Partitioning Algorithm (CSPA; Strehl & Ghosh, 2002). This approach relies on the notion that two objects can be viewed similar if they are clustered in the same cluster and dissimilar otherwise. By counting the number of times each pair of objects is clustered together by the different ensemble members, one can obtain a new representation of the data to which any distance-based clustering algorithm can be applied.

CSPA works by constructing a $n \times n$ coassociation matrix S. In this matrix, each element $S_{i,j}$ denotes the fraction of clusterings in which i and j are in the same cluster. For example, given the four label vectors from different clusterers in Figure 1a, the corresponding coassociation matrix of Figure 1b can be computed. This matrix summarizes all information of the clusterings produced by the ensemble members, with the i-th row (or column) denoting a new representation of data point i.

	$\boldsymbol{\lambda}^{(1)}$	$\boldsymbol{\lambda}^{(2)}$	$\boldsymbol{\lambda}^{(3)}$	$\boldsymbol{\lambda}^{(4)}$
$\mathbf{x_1}$	1	3	1	1
$\mathbf{x_2}$	1	1	1	2
$\mathbf{x_3}$	2	2	2	3
$\mathbf{x_4}$	2	2	2	2
$\mathbf{x_5}$	3	1	3	3
$\mathbf{x_6}$	3	3	1	1

(a) Label vectors $\boldsymbol{\lambda}^{(1)}$, $\boldsymbol{\lambda}^{(2)}$, $\boldsymbol{\lambda}^{(3)}$ and $\boldsymbol{\lambda}^{(4)}$.

$$S = \begin{pmatrix} 1 & 0.5 & 0 & 0 & 0 & 0.75 \\ 0.5 & 1 & 0 & 0.25 & 0.25 & 0.25 \\ 0 & 0 & 1 & 0.75 & 0.25 & 0 \\ 0 & 0.25 & 0.75 & 1 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 1 & 0.25 \\ 0.75 & 0.25 & 0 & 0 & 0.25 & 1 \end{pmatrix}$$

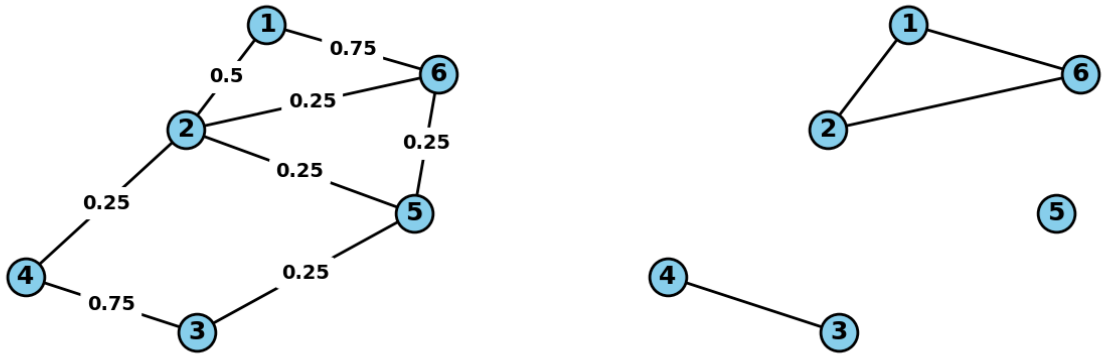
(b) Coassociation matrix S.

Figure 1: Illustrative example of the coassociation matrix with $n = 6$ and $t = 4$.

The final clustering is then obtained by applying any distance-based clustering algorithm to S. This paper uses two methods to perform the clustering. First, a simple k-means algorithm using the Euclidean distance is considered. Second, the graph-based method proposed by Strehl and Ghosh (2002) is used. In this last approach the coassociation matrix S is represented in an undirected graph. The nodes in this graph then represent the data points and the weights of the edges represent the

similarity between the start- and end-node as induced by S. The final clustering can then be obtained by partitioning this graph into k parts. To guarantee that the final clusters are highly similar, the partitioning process should remove edges with the lowest possible total weight.

Figure 2 shows an illustration of this partitioning process of the example of Figure 1. On the left, we see the graph representation of matrix S with the corresponding weights. By removing four edges like in the graph on the right, we obtain the partitioning of the graph that represents the final three clusters. Note that this partitioning is minimal, meaning that removing any other combination of edges to partition the graph in three parts will result in a higher total weight.



(a) Graph representation of S.

(b) Minimal partition of S into three clusters.

Figure 2: Example of the graph partition of data from Figure 1. Edges that begin and end at the same node are excluded for the sake of clarity.

To partition the graph and obtain the final clustering, this study follows the direction of Strehl and Ghosh (2002) and employs the graph partitioning algorithm introduced by Karypis and Kumar (1998). This algorithm is well-known for generating high-quality partitionings efficiently and is included in the METIS[2] package. An in-depth discussion of the algorithm is beyond the scope of this paper, but for more details we refer to Karypis and Kumar (1998).

One last note about CSPA is that it does not possess any weighting procedure, such that the information from all clusterings in the ensemble is used equally. This means that a bad clustering can influence the performance, making it important to think about the selection procedure of clusterings prior to fitting the method.

Due to time constraints, we have chosen to only implement a simple selection procedure which selects the best L clusterings based on the Davies Bouldin Index (Davies & Bouldin, 1979). This index is based on the ratio of within-cluster distances and between-cluster distances, with lower values indicating a better clustering. By running CSPA for multiple values of L, we aim to minimise the effect of bad clusterings on the performance of CSPA.

4.4 Performance measures

This paper uses six performance measures to assess the different clustering methods. Here, we will provide a short overview, but for a detailed discussion on the computation of these performance measures we refer to Appendix E.

The performance measures used are Clustering accuracy (CL-ACC), Rand Index (RI), Adjusted Rand Index (ARI), Mutual Information (MI), Normalised Mutual Information (NMI) and Adjusted Mutual Information (AMI). CL-ACC is a measure similar to classification accuracy and measures the

[2] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

percentage of predictions which are correct based on a default labelling. The RI and MI both assess the similarity between two sets of clusterings, where the RI is based on similarity counts (Hubert & Arabie, 1985) and the MI on information theory (Cover, 1999). Both metrics are useful for comparing clusterings from different algorithms as well as comparing a single algorithm's clustering with a default clustering. Their adjusted versions, ARI and AMI, adjust the default metrics for chance by ensuring that in expectation both measures are equal to 0 (Romano, Bailey, Nguyen & Verspoor, 2014). The NMI scales the default MI to be between 0 and 1.

To then determine if the differences in these performance metrics are statistically significant, we use a Wilcoxon signed-rank test (Demšar, 2006) together with the Holm correction (García & Herrera, 2008). These tests are used to form cliques of clusterings which are not statistically different at a significance level of 5%. The results are presented in critical difference diagrams and cliques are represented by a horizontal bar (see for example Figure 3).

5 Results

All experiments conducted in this paper were run using a computer with an AMD Ryzen 7 3700X @4.4 GHz processor and 32 GigaBytes of RAM. The experiments are performed in Python version 3.11.9, using the aeon[3] package for the clustering algorithms and the PyMetis[4] package as a wrapper for the METIS algorithm. For more details about the used packages, software and parameter settings we refer to Appendix D. Additionally, Appendix F contains some notes about the run times.

All of the methods applied in this paper are trained on the default train set from the UCR Archive. For the clustering algorithms, this means that the final centroids/medoids are updated only for the train set. For the ensemble methods, this means that the selection process and weight calculation are done on the train set only. The final evaluation of the performance of the different methods is done on the test set.

It should be noted that it is not necessary for clustering to evaluate the results on an unseen test set, unlike in the case of supervised classification for example. Consequently, for all the results of the test set in the subsequent sections, we have included corresponding figures and tables for the training set in Appendix G. Significant differences between the test and train sets will be noted in the text.

The remainder of this section is structured as follows. First, in Section 5.1, we compare the different initialization algorithms and investigate the difference in performance between k-means and k-medoids. Second, in Section 5.2, we provide an extensive comparison the elastic distance measures. Lastly, we compare the ensemble methods in Section 5.3 and provide a performance analysis in Section 5.4.

5.1 Comparison of the different initialisation and clustering algorithms

In Table 2, we can observe the accuracies of the different initialisation algorithms of both k-means and k-medoids for the different distance functions. On average, the Forgy method performs best for k-means, while the k-means++ method performs best for k-medoids. Additionally, k-medoids outperforms k-means for most of the distance functions.

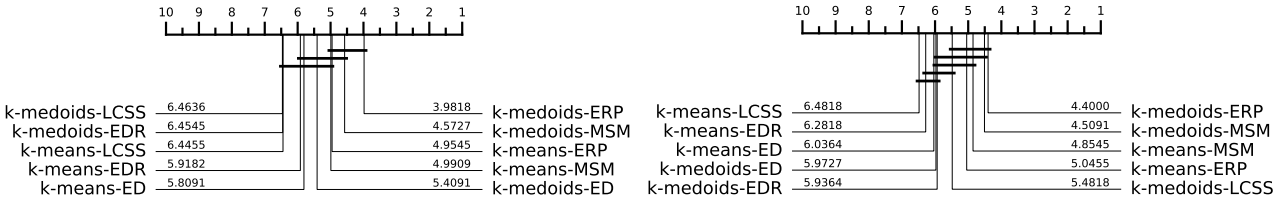
When looking at the performance across the different distance measures, it can be observed that the MSM measure ranks best for k-means, with ERP following as second. For k-medoids this is the other way around, with ERP ranking best and MSM ranking second best. In both clustering algorithms, the LCSS and EDR distance are among the worst performers. These results generally hold for all initialisation algorithms, suggesting that the influence of the initialisation on the performance of the distance measures is only minimal.

[3] https://github.com/aeon-toolkit/aeon
[4] https://github.com/inducer/pymetis

	k-means (%)			k-medoids (%)		
Distance	random	Forgy	k-means++	random	Forgy	k-means++
DTW	60.05	**60.85**	60.72	61.62	**62.78**	62.24
DDTW	60.78	**62.11**	61.25	60.48	**61.14**	61.03
ED	**60.68**	60.32	59.93	61.19	**62.01**	60.75
EDR	59.44	**60.01**	59.49	60.7	58.44	**61.0**
ERP	62.02	**62.62**	61.94	64.52	**66.41**	65.33
LCSS	**58.81**	58.79	58.56	60.21	58.33	**61.42**
MSM	62.62	62.89	**63.14**	**64.73**	63.17	64.57
TWE	**61.37**	60.48	60.33	62.08	61.78	**62.67**
WDTW	61.03	**62.50**	61.34	62.40	62.49	**62.74**
WDDTW	61.6	**61.81**	60.46	59.51	**61.66**	59.72
Average	60.84	**61.24**	60.72	61.74	61.82	**62.15**

Table 2: Accuracies of the different clustering algorithms for each of the distance functions on the test set. Bold is the highest accuracy across the three different initialisation algorithms.

To investigate whether the difference in performance between k-means and k-medoids is statistically significant, the accuracy ranks for the ED, ERP, MSM, LCSS and EDR distances are displayed in Figure 3. When using the Forgy initialisation, k-medoids ERP and k-medoids MSM are significantly better than all k-means clusterers except the one using the ERP distance. Moreover, with k-means++ initialization, no k-medoids clusterer is ranked lower than the k-means clusterer which utilizes the same distance measure. Not all differences between the different clusterers are statistically significant, but the results from Figure 3 suggest that k-medoids serves as a more robust benchmark for clustering compared to k-means. The results obtained from the train set are consistent with these conclusions, indicating that the models perform similarly on both splits (see Figure 7 in Appendix G). Additional support for the use of k-medoids over k-means is that k-medoids is computationally less expensive (see Table 10 in Appendix F).



(a) Forgy initialisation

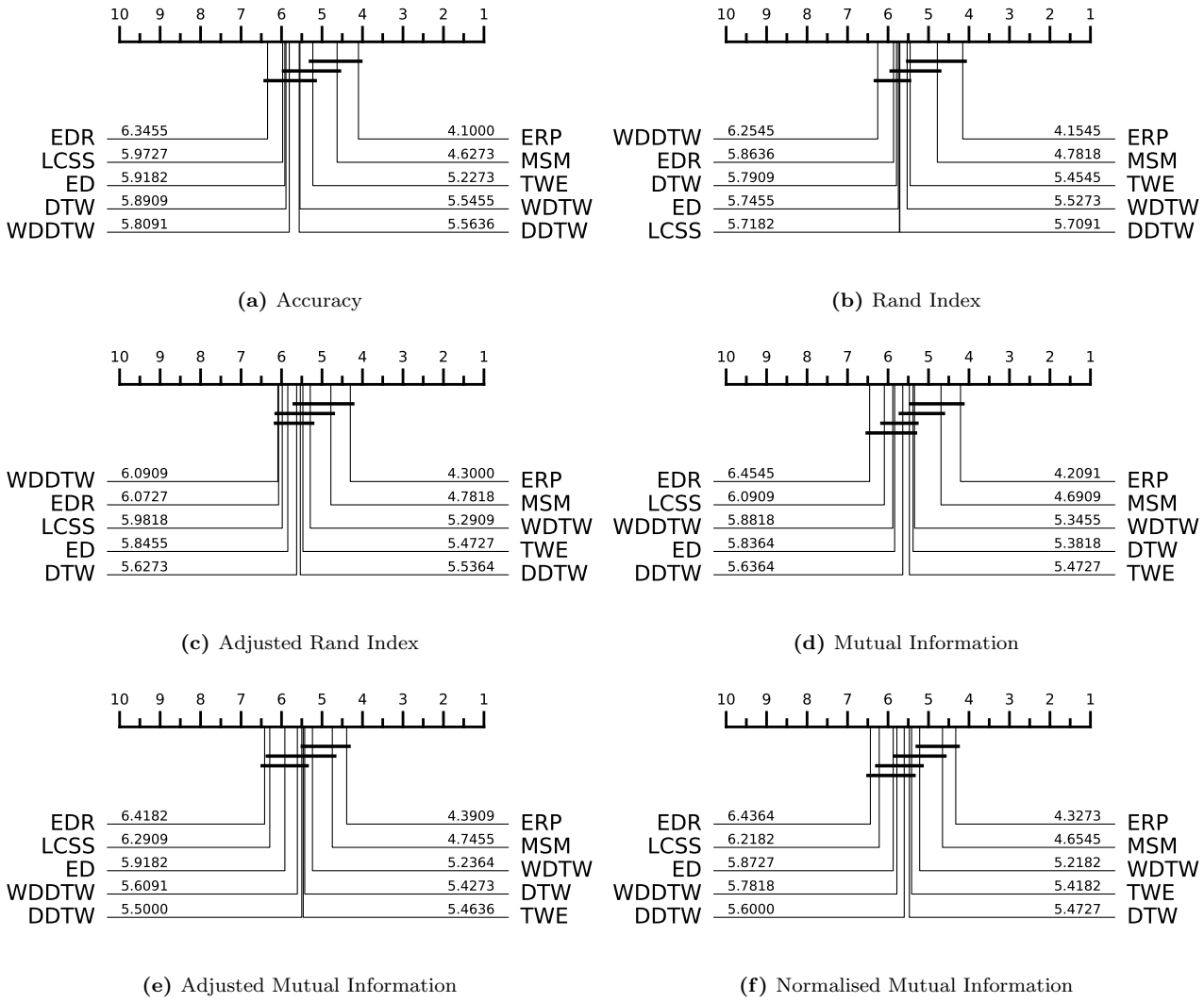
(b) k-means++ initialisation

Figure 3: Accuracies of k-means and k-medoids on the test set for the ED, ERP, MSM, LCSS and EDR distances. Results on the left are for the Forgy initialisation and on the right for the k-means++ initialisation.

5.2 Comparison of different distance measures

The results from the previous section suggest that MSM and ERP perform the best in terms of accuracy. In their similar analysis of elastic distances, Holder et al. (2024) found that TWE was also among the top performers, while ERP was found to be significantly less accurate than both MSM and TWE. To further investigate the differences in the elastic distances, we present the critical difference diagram for the ten distance measures for all performance metrics in Figure 4. The decision was made to display only the results for the k-medoids algorithm initialised by k-means++, although the other algorithms yielded comparable outcomes.

It can be observed that the ERP, MSM and TWE distances rank significantly higher than the other distances for two of the six performance measures. For other performance metrics, ERP and MSM continue to rank the highest, although the difference is not as pronounced as with accuracy. The EDR and LCSS distances are significantly worse than most of the top performers and are even



(a) Accuracy

(b) Rand Index

(c) Adjusted Rand Index

(d) Mutual Information

(e) Adjusted Mutual Information

(f) Normalised Mutual Information

Figure 4: Results for k-means++ k-medoids on the test set using ten different distance functions.

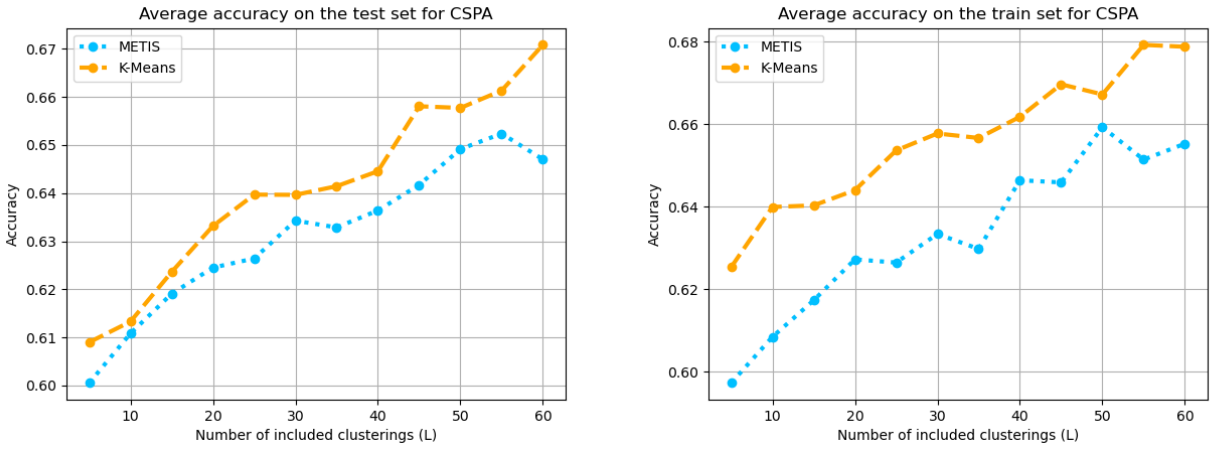
outperformed by ED. The WDTW measure also ranks reasonably high, being not significantly worse than ERP or MSM for most of the performance measures.

One key insight is that the DTW distance is significantly outperformed by some of the other distance measures, even though it is one of the most popular approaches for time series clustering or classification. This discovery aligns with the outcomes reported in Holder et al. (2024). Even with additional tuning of the window parameter and exploring different averaging methods such as DTW Barycentre Averaging (Petitjean, Ketterlin & Gançarski, 2011), Holder et al. (2024) could not beat the top distance metrics with DTW. The results from Figure 4 support their notion that DTW may not be the gold standard as depicted in the literature and using the MSM or ERP distance instead might be more appropriate.

5.3 Extension: ensemble methods

In this section we will present the performance of the ensemble methods proposed in Section 4.3. We will analyse the selection procedure of the CSPA method first, after which we compare both CSPA and SAVM with the best-performing individual clusterings.

The selection process of CSPA has been run for $L \in \{5, 10, \ldots, 60\}$ for both CSPA employing k-means and CSPA employing METIS. Figure 5 illustrates the average accuracy in the test and train set for varying numbers of clusterings included in the ensemble. The results show that increasing the number of included ensemble members has a positive impact on accuracy. However, when (almost) all clusterings are utilized, accuracy decreases in the training set for both CSPA methods. In the test set, this decline is only observed for METIS-CSPA. Additionally, k-means-CSPA generally outperforms METIS-CSPA for all values of K in both the train and test set.



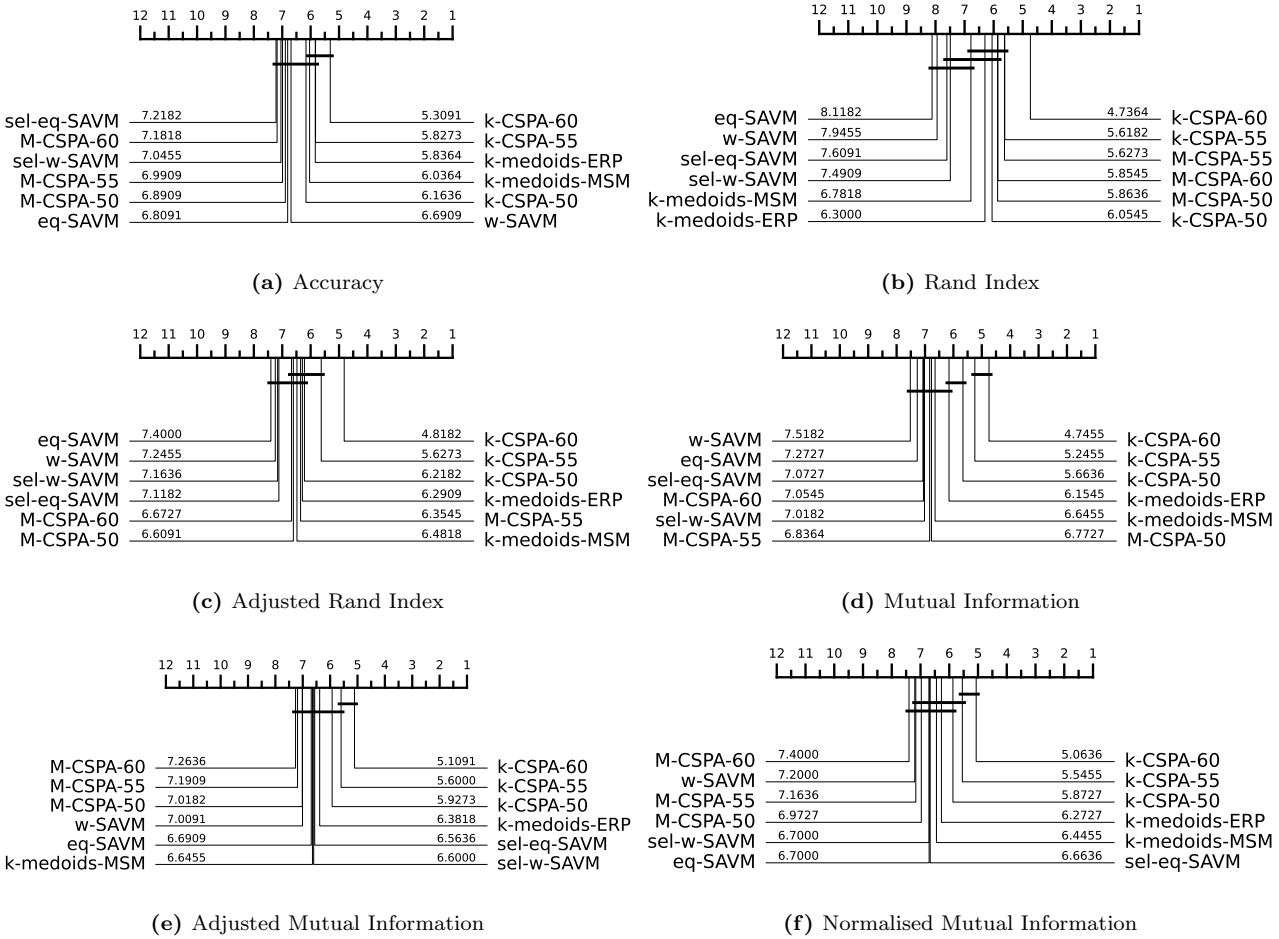
(a) Test set

(b) Train set

Figure 5: Average accuracy of k-means-CSPA and METIS-CSPA for different values of K. Results are on the left are for the test set and on the right for the train set.

For the comparison of CSPA with the best individual clusterings and SAVM, we will only provide the results for the $L = 50, 55, 60$. Figure 6 displays the results for both CSPA and SAVM against two of the best-performing individual clusterers. In line with the results in Figure 5, k-means-CSPA performs best when including all of the 60 clusterings. Furthermore, k-means-CSPA outperforms the two individual clusterers significantly for most of the performance measures. However, this result is dependent on the amount of clusterings included in the ensemble method, as the difference in performance is insignificant for $L = 50$. One, perhaps surprising, observation is that METIS-CSPA significantly underperforms compared to both k-means-CSPA and the individual methods. Given that both CSPA approaches utilize the same coassociation matrix, one might expect their performances to be comparable, which is not the case.

For SAVM, it is observed that each voting method is outperformed by the individual measures across all performance metrics. For most of the metrics, this difference is statistically significant. Furthermore, there is no significant difference between the performance of the four voting methods. The results on the train set give similar results, although the significance does not resemble the significance of the test set (see Figure 9 in Appendix G).



(a) Accuracy **(b)** Rand Index

(c) Adjusted Rand Index **(d)** Mutual Information

(e) Adjusted Mutual Information **(f)** Normalised Mutual Information

Figure 6: Results of the ensembles methods against k-medoids-MSM and k-medoids-ERP on the test set. The numbers succeeding 'CSPA' represent the quantity of clusterings incorporated in the selection process. The letters M and k preceding 'CSPA' denote the METIS and k-means algorithms, respectively. For SAVM, the voting schemes *equal*, *weighted*, *selective-equal*, and *selective-weighted* are abbreviated as *eq*, *w*, *sel-eq*, and *sel-w*.

5.4 Performance analysis

The previous sections have shown that the EDR and MSM distances are robust elastic distances which have a high average performance for all datasets. Furthermore, k-means-CSPA showed to perform even better than these individual measures. While these results give a good indication of the general performance, they do not provide insights into which approaches are best for data from a certain domain or with certain characteristics. Therefore, we will evaluate the clusterers across several different characteristics of the dataset.

To perform the analysis, we have to group the datasets with similar characteristics. However, only 55 out of the available 128 datasets were used in this study to ensure that the models were finished within the given time constraints. These 55 datasets were specifically selected for their smaller size, making it difficult to construct groups of datasets with varying characteristics. While the performance analysis still could yield valuable insights, the results presented here should be interpreted with caution.

First, we look at the performance of the clusterers across different categories of data. The UCR archive contains 15 different categories of time series datasets, 10 of which are present in our subset of 55 datasets. The datasets are grouped into the following groups: Image (group A, 17 datasets), Sensor (group B, 11 datasets), Simulated (group C, 7 datasets), Spectro (group D, 6 datasets) and Motion (Group E, 6 datasets). The remaining 8 datasets were excluded from this specific analysis because they could not be grouped to make a domain-specific group. For an explanation of the meaning of these different types, we refer to Dau et al. (2019).

Second, we choose to evaluate the performance of the clusterers for three other types of characteristics: the number of clusters in the dataset, the length of the time series and the number of training instances. The grouping of the datasets is given in Table 3.

Group:	A	B	C
Number of clusters	2-3 (8 datasets)	4-5 (7 datasets)	> 5 (10 datasets)
Length of the time series	< 200 (27 datasets)	200-450 (15 datasets)	> 450 (13 datasets)
Number of training instances	< 100 (36 datasets)	100-200 (9 datasets)	> 200 (10 datasets)

Table 3: Distribution of the datases across groups for three different characteristics.

The results of the performance analysis on the test set are given in Table 4. Results are presented only for k-medoids using the k-means++ initialisation, as this algorithm yielded the most accurate predictions. We learn for the majority of the groups that the k-CSPA-60 method performs best, which is expected based on our earlier conclusions. Furthermore, the DTW distance ranks best for datasets with a larger amount of training instances.

The MSM distance, which had a strong performance as an individual clusterer, does not achieve the highest ranking in any group. On the other hand, the ERP distance scores highly for datasets with a small number of clusters or small length, but is outperformed when the number of clusters or length increases. This indicates that the MSM distance is highly robust across all datasets, whereas the ERP measure excels in identifying well-defined clusters for datasets with a limited number of clusters and shorter lengths.

The observation that ERP demonstrates high accuracy on small datasets may explain the conflicting outcomes with Holder et al. (2024) work. Given that the majority of datasets in this study are small, the reported accuracy of ERP might be inflated when compared to its performance for larger datasets. This would explain why ERP is among the top performers in this paper, while in the work of Holder et al. (2024) ERP was significantly worse than MSM.

	Type					Clusters			Training instances			Length		
Accuracy(%)	A	B	C	D	E	A	B	C	A	B	C	A	B	C
k-CSPA-60	**63.55**	**73.76**	70.37	68.52	61.35	65.61	**73.02**	**68.52**	**67.71**	**75.51**	57.25	66.48	**70.74**	**64.12**
k-CSPA-55	61.91	71.31	70.05	68.05	61.15	65.55	67.39	67.46	66.61	74.59	56.81	66.41	67.61	63.86
k-CSPA-50	61.54	70.97	69.92	68.05	61.08	65.26	66.86	66.94	66.30	74.32	56.19	65.89	67.33	63.74
M-CSPA-60	59.75	69.68	70.69	66.94	59.60	64.96	60.87	66.41	65.86	72.71	53.33	64.59	68.38	60.70
M-CSPA-55	59.38	72.80	69.94	68.54	59.49	65.03	65.28	66.01	66.85	72.40	53.01	64.10	70.00	62.12
M-CSPA-50	59.58	71.22	70.83	68.08	59.85	64.76	64.19	66.01	66.11	72.29	53.99	64.21	68.50	62.25
eq-SAVM	60.30	68.98	60.46	**69.12**	60.92	63.29	67.58	62.17	64.28	68.36	57.05	63.63	68.05	58.54
sel-eq-SAVM	60.54	68.31	64.60	63.80	60.71	63.53	64.27	62.04	63.45	71.78	55.42	64.06	63.23	62.03
sel-w-SAVM	60.21	68.45	64.36	63.80	60.89	63.61	64.27	61.99	63.55	71.66	55.41	64.32	63.24	61.67
w-SAVM	60.37	69.44	61.50	68.28	61.25	63.65	67.03	62.24	64.25	69.62	57.09	64.45	67.58	58.18
DTW	56.33	65.50	**72.48**	64.36	63.18	62.42	58.02	64.49	61.51	70.48	**57.44**	62.89	65.00	57.70
DDTW	59.80	66.20	55.08	62.89	59.94	60.46	65.10	60.36	62.47	63.75	53.42	58.60	63.94	62.73
ED	55.87	66.88	57.77	67.09	59.00	62.53	60.53	54.12	62.17	68.10	49.00	59.53	64.28	59.21
EDR	55.64	68.75	52.77	62.01	64.23	62.89	61.88	53.22	62.06	68.40	50.56	61.23	62.99	58.25
ERP	58.34	72.62	71.70	65.55	**65.33**	**67.00**	63.46	60.27	66.37	70.84	56.61	**66.96**	67.68	59.21
LCSS	56.25	66.97	58.46	61.46	60.82	63.99	59.42	53.03	63.57	65.89	49.64	60.46	62.91	61.69
MSM	62.88	68.18	66.33	63.92	61.01	64.69	63.91	64.58	64.98	71.73	56.67	65.49	66.18	60.80
TWE	58.97	66.42	65.22	65.55	61.11	64.00	61.95	58.14	62.57	69.85	56.59	64.20	64.40	57.53
WDTW	56.41	68.56	72.46	64.59	61.45	62.07	62.88	65.19	62.81	68.68	57.18	63.06	66.45	57.81
WDDTW	59.15	62.68	59.40	64.32	55.63	58.68	61.53	62.39	60.22	63.60	54.42	58.74	62.15	58.97

Table 4: Accuracies of k-means++ k-medoids, CSPA and SAVM for the different groups of the performance analysis. Bold is the highest accuracy for each group. Results are on the test set.

6 Conclusion

In this paper, we have provided a detailed comparison of nine elastic distance measures used for time series clustering with k-means and k-medoids, thereby replicating the findings represented in Holder et al. (2024). Furthermore, we have evaluated the use of CSPA and SAVM ensemble methods for combining clusterings produced by these different elastic distance measures.

The study shows that the MSM and ERP distances rank among the best benchmarks for time series clustering. While MSM and ERP demonstrate a higher average performance than the other distances, the difference is not statistically significant for all of the used performance measures. These results were somewhat contradictory to Holder et al. (2024), where ERP was significantly worse than MSM and TWE was also among the best performers.

All of the used distance measures prove to be robust across various initialization procedures and clustering algorithms. The comparison between k-means and k-medoids reveals that k-medoids initialized by k-means++ outperforms k-means, although this difference is again not always statistically significant. Still, this suggests that k-medoids combined with MSM or ERP can be a candidate to serve as a robust time series clustering benchmark. We also demonstrate that the widely-used DTW measure is surpassed by the majority of other distance measures. This finding is consistent with the results presented in Holder et al. (2024), further supporting their notion that DTW should not be regarded as the definitive standard for time series clustering. The LCSS and EDR methods are the two worst-performing distances, being often worse than the Euclidean distance.

Our experiments reveal that the use of ensemble methods for time series clustering shows promising results for one of the two models. The voting-based method (SAVM) fails to improve the results of the individual distance measures, indicating that the results of Lines and Bagnall (2015) regarding voting-based methods for time series classification may not apply to time series clustering. Conversely, the CSPA model utilizing k-means is significantly better than the individual distance measures for most of the performance metrics. While this reveals a promising field for future research, the ensemble selection process and the final clustering algorithm used in CSPA both affected the results. This illustrates that the use of ensemble methods for time series clustering is not entirely straightforward.

It is important to acknowledge some of the main limitations of this study. Due to computational constraints, we were forced to use only a subset of the available datasets from the UCR archive. The implication of this is two-fold. Firstly, the selection of datasets from the archive was limited to smaller datasets, which could be a potential reason for the increased accuracy of ERP compared to related studies. Secondly, the use of fewer datasets results in larger standard errors during statistical testing, rendering some of the findings insignificant. In addition to computational limitations, this research only considers a straightforward selection process based solely on the Davies-Bouldin Index for the CSPA method. This decision allowed us to focus more on evaluating the different elastic distances and is motivated by the fact that each individual clustering achieved considerable performance. However, using a more advanced selection procedure could potentially improve results (Fred & Jain, 2005). This could explain why some of the ensemble methods failed to yield better results.

For further research, we therefore propose to apply a more sophisticated ensemble selection approach to the ensembles for time series clustering, such as the approach in Fern and Lin (2008). Fern and Lin (2008) use the concepts of quality and diversity based on the NMI in the selection process and show that this can significantly improve the results for the CSPA method. Furthermore, other ensemble methods such as the HyperGraph-Partitioning Algorithm (Fern & Brodley, 2004), Bayesian Cluster Ensembles (Wang et al., 2011) or the Hierarchical Agglomerative Approach (Fred & Jain, 2005) could be employed for time series clustering. Finally, it could be interesting to investigate the impact of setting the number of clusters for both the elastic distance measures and ensemble methods. Xu et al. (2016) provide a review of approaches that can be used for this. Examples of interesting methods are methods based on internal measures such as the Silhouette statistic and Gap statistic or methods based on external measures such as Jaccard Index or Hurbert statistic.

References

Aghabozorgi, S., Shirkhorshidi, A. S. & Wah, T. Y. (2015). Time-series clustering – a decade review. *Information Systems*, *53*, 16–38. doi: https://doi.org/10.1016/j.is.2015.04.007

Alqurashi, T. & Wang, W. (2019). Clustering ensemble method. *International Journal of Machine Learning and Cybernetics*, *10*(6), 1227–1246. doi: https://doi.org/10.1007/s13042-017-0756-7

Arthur, D. & Vassilvitskii, S. (2006). K-means++: The advantages of careful seeding. *Proc. of the Annu. ACM-SIAM Symp. on Discrete Algorithms*, *8*, 1027–1035. doi: https://doi.org/10.1145/1283383.1283494

Bagnall, A., Lines, J., Bostrom, A., Large, J. & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *31*(3), 606–660. doi: https://doi.org/10.1007/s10618-016-0483-9

Bagnall, A., Lines, J., Hills, J. & Bostrom, A. (2015). Time-series classification with cote: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, *27*(9), 2522–2535. doi: https://doi.org/10.1109/TKDE.2015.2416723

Bijuraj, L. (2013). Clustering and its applications. In *Proceedings of national conference on new horizons in it-ncnhit* (Vol. 169, p. 172).

Bradley, P. S. & Fayyad, U. M. (1998). Refining initial points for k-means clustering. In *Icml* (Vol. 98, pp. 91–99).

Celebi, M. E., Kingravi, H. A. & Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, *40*(1), 200–210. doi: https://doi.org/10.1016/j.eswa.2012.07.021

Chen, L. & Ng, R. (2004). On the marriage of lp-norms and edit distance. *Proceedings of the 30th international conference on very large data bases*, 792–803.

Chen, L., Özsu, M. T. & Oria, V. (2005). Robust and fast similarity search for moving object trajectories. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 491–502. doi: 10.1145/1066157.1066213

Cover, T. M. (1999). *Elements of information theory*. John Wiley & Sons.

Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., ... Keogh, E. (2019). The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, *6*(6), 1293–1305.

Davies, D. & Bouldin, D. (1979, 05). A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *PAMI-1*, 224–227. doi: 10.1109/TPAMI.1979.4766909

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, *7*, 1–30.

Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd* (Vol. 96, pp. 226–231).

Fern, X. Z. & Brodley, C. E. (2004). Solving cluster ensemble problems by bipartite graph partitioning. In *Proceedings of the twenty-first international conference on machine learning* (p. 36). Association for Computing Machinery. doi: 10.1145/1015330.1015414

Fern, X. Z. & Lin, W. (2008). Cluster ensemble selection. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *1*(3), 128–141. doi: https://doi.org/10.1002/sam.10008

Forgy, E. W. (1965). Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics*, *21*, 768–769.

Fred, A. & Jain, A. (2005, 07). Combining multiple clusterings using evidence accumulation. *IEEE transactions on pattern analysis and machine intelligence*, *27*, 835–850. doi: 10.1109/TPAMI.2005.113

García, S. & Herrera, F. (2008, 12). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research - JMLR*, *9*, 2677–2694.

Ghosh, J. & Acharya, A. (2011). Cluster ensembles. *WIREs Data Mining and Knowledge Discovery*, *1*(4), 305-315. doi: https://doi.org/10.1002/widm.32

Holder, C., Middlehurst, M. & Bagnall, A. (2024). A review and evaluation of elastic distance functions for time series clustering. *Knowledge and Information Systems*, *66*, 765–809. doi: https://doi.org/10.1007/s10115-023-01952-0

Hubert, L. & Arabie, P. (1985). Comparing partitions. *Journal of classification*, *2*, 193–218.

Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B. & Heming, J. (2023). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, *622*, 178–210. doi: https://doi.org/10.1016/j.ins.2022.11.139

Ismkhan, H. (2018). Ik-means-+: An iterative clustering algorithm based on an enhanced version of the k-means. *Pattern Recognition*, *79*, 402–413.

Javed, A., Lee, B. S. & Rizzo, D. M. (2020). A benchmark study on time series clustering. *Machine Learning with Applications*, *1*, 100001.

Jeong, Y.-S., Jeong, M. K. & Omitaomu, O. A. (2011). Weighted dynamic time warping for time series classification. *Pattern Recognition*, *44*(9), 2231–2240. (Computer Analysis of Images and Patterns) doi: https://doi.org/10.1016/j.patcog.2010.09.022

Karypis, G. & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, *20*(1), 359–392. doi: 10.1137/S1064827595287997

Kaufman, L. & Rousseeuw, P. (1987). Clustering by means of medoids. *Data Analysis based on the L1-Norm and Related Methods*, *31*, 405–416.

Keogh, E. J. & Pazzani, M. J. (2001). Derivative dynamic time warping. In *Proceedings of the 2001 siam international conference on data mining (sdm)* (pp. 1–11). doi: 10.1137/1.9781611972719.1

Keogh, E. J. & Ratanamahatana, C. A. (2005, 03). Exact indexing of dynamic time warping. *Knowledge and Information Systems*, *7*, 358–386. doi: https://doi.org/10.1007/s10115-004-0154-9

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, *52*.

Lima, F. T. & Souza, V. M. (2023). A large comparison of normalization methods on time series. *Big Data Research*, *34*, 100407. doi: https://doi.org/10.1016/j.bdr.2023.100407

Lines, J. & Bagnall, A. (2015). Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, *29*(3), 565–592. doi: https://doi.org/10.1007/s10618-014-0361-2

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Trans. Inf. Theory*, *28*, 129–136.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, *1*, 281–297.

Marteau, P.-F. (2009, 03). Time warp edit distance with stiffness adjustment for time series matching. *IEEE transactions on pattern analysis and machine intelligence*, *31*, 306–18. doi: 10.1109/TPAMI.2008.76

Pasupathi, S., Shanmuganathan, V., Madasamy, K., Yesudhas, H. R. & Kim, M. (2021). Trend analysis using agglomerative hierarchical clustering approach for time series big data. *The Journal of Supercomputing*, *77*(7), 6505–6524.

Pelleg, D. & Moore, A. W. (2000). X-means: Extending k-means with efficient estimation of the number of clusters. In *Icml* (Vol. 1, pp. 727–734).

Petitjean, F., Ketterlin, A. & Gançarski, P. (2011). A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, *44*(3), 678–693. doi: https://doi.org/10.1016/j.patcog.2010.09.013

Ratanamahatana, C. A. & Keogh, E. (2005). Three myths about dynamic time warping data mining. In *Proceedings of the 2005 siam international conference on data mining* (pp. 506–510).

Romano, S., Bailey, J., Nguyen, V. & Verspoor, K. (2014). Standardized mutual information for clustering comparisons: one step further in adjustment for chance. In *International conference on machine learning* (pp. 1143–1151).

Räsänen, T. & Kolehmainen, M. (2009, 04). Feature-based clustering for electricity use time series data. *Lecture Notes in Computer Science*, *5495*, 401–412. doi: 10.1007/978-3-642-04921-7_41

Sakoe, H. & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *26*, 159–165.

Sawant, K. (2014). Adaptive methods for determining dbscan parameters. *International Journal of Innovative Science, Engineering & Technology*, *1*(4), 329–334.

Soleimani, G. & Abessi, M. (2020). Dlcss: A new similarity measure for time series data mining. *Engineering Applications of Artificial Intelligence*, *92*, 103664. doi: https://doi.org/10.1016/j.engappai.2020.103664

Stefan, A., Athitsos, V. & Das, G. (2013, 06). The move-split-merge metric for time series. *Knowledge and Data Engineering, IEEE Transactions on*, *25*, 1425–1438. doi: 10.1109/TKDE.2012.88

Strehl, A. & Ghosh, J. (2002, 01). Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, *3*, 583–617. doi: 10.1162/153244303321897735

Su, T. & Dy, J. G. (2007). In search of deterministic methods for initializing k-means and gaussian mixture clustering. *Intelligent Data Analysis*, *11*(4), 319–338.

Wang, H., Shan, H. & Banerjee, A. (2011). Bayesian cluster ensembles. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *4*(1), 54–70.

Xu, S., Qiao, X., Zhu, L., Zhang, Y., Xue, C. & Li, L. (2016). Reviews on determining the number of clusters. *Applied Mathematics & Information Sciences*, *10*(4), 1493–1512.

Zhang, H., Ho, T. B., Zhang, Y. & Lin, M.-S. (2006). Unsupervised feature extraction for time series clustering using orthogonal wavelet transform. *Informatica*, *30*(3).

Zhou, Z.-H. & Tang, W. (2006). Clusterer ensemble. *Knowledge-Based Systems*, *19*(1), 77–83. doi: https://doi.org/10.1016/j.knosys.2005.11.003

Łuczak, M. (2016). Hierarchical clustering of time series data with parametric derivative dynamic time warping. *Expert Systems with Applications*, *62*, 116–130. doi: https://doi.org/10.1016/j.eswa.2016.06.012

A Data

Dataset	Type	Instances (train/test)	Length	No. classes
ArrowHead	Image	36/175	251	3
BME	Simulated	30/150	128	3
Beef	Spectro	30/30	470	5
BeetleFly	Image	20/20	512	2
BirdChicken	Image	20/20	512	2
CBF	Simulated	40/900	128	3
Car	Sensor	60/60	577	4
Chinatown	Traffic	20/343	24	2
Coffee	Spectro	28/28	286	2
DiatomSizeReduction	Image	16/306	345	4
DistalPhalanxOutlineAgeGroup	Image	400/139	512	2
DistalPhalanxTW	Image	400/139	80	6
ECG200	ECG	100/100	96	2
ECGFiveDays	ECG	23/861	136	2
FaceFour	Image	24/88	350	4
Fish	Image	175/175	463	7
FreezerRegularTrain	Sensor	150/2850	301	2
FreezerSmallTrain	Sensor	28/2850	301	2
GunPoint	Motion	50/150	150	2
GunPointAgeSpan	Motion	135/316	150	2
GunPointMaleVersusFemale	Motion	135/316	150	2
GunPointOldVersusYoung	Motion	135/316	150	2
Ham	Spectro	109/105	431	2
Herring	Image	64/64	512	2
InsectEPGRegularTrain	EPG	62/249	601	3
InsectEpgSmallTrain	EPG	17/249	601	3
ItalyPowerDemand	Sensor	67/1029	24	2
Lightning2	Sensor	67/1029	24	2
Lighting7	Sensor	70/73	319	7
Mallat	Simulated	55/2345	1024	8
Meat	Spectro	60/60	448	3
MedicalImages	Image	381/760	99	10
MiddlePhalanxOutlineAgeGroup	Image	400/154	80	3
MiddlePhalanxTW	Image	399/154	80	6
MoteStrain	Sensor	20/1252	84	2
OliveOil	Spectro	30/30	570	4
Plane	Sensor	105/105	144	7
PowerCons	Power	180/180	144	2
ProximalPhalanxOutlineAgeGroup	Image	400/205	80	3
ProximalPhalanxOutlineCorrect	Image	600/291	80	2
ProximalPhalanxTW	Image	400/205	80	6
Rock	Spectrum	20/50	2844	4
ShapeletSim	Simulated	20/180	500	2
SmoothSubspace	Simulated	150/150	15	3
SonyAIBORobotSurface1	Sensor	20/601	70	2
SonyAIBORobotSurface2	Sensor	27/953	65	2
Symbols	Image	25/995	398	6
SyntheticControl	Simulated	300/300	60	6
ToeSegmentation1	Motion	40/228	277	2
ToeSegmentation2	Motion	36/130	343	2
Trace	Sensor	100/100	275	4
TwoLeadECG	ECG	23/1139	82	2
UMD	Simulated	36/144	150	3
Wine	Spectro	57/54	234	2
Yoga	Image	300/300	426	2

Table 5: Datasets used in this paper from the UCR Archive

B Algorithms

B.1 K-means and K-medoids

Algorithm 2 displays the clustering algorithm for k-means. Note that the handling of empty clusters is not shown to provide a clear representation.

Algorithm 2 Pseudocode basic k-means algorithm (adapted from: Ikotun et al., 2023)

Input: $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$ (dataset to be clustered), $Y = \{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_k}\}$ (initial cluster centroids), k (number of clusters), *conv* (criterion to determine when the algorithm has converged).
Output: $C = \{C_1, C_2, \ldots, C_k\}$ (set of final clusters).

1: Initialise distance matrix $D \in \mathbb{R}^{n \times k}$ of zeros.
2: Initialise k empty clusters $C_1, C_2, \ldots, C_k$.
3: $s = 0$
4: **while** *conv* == False **do**
5: **for** $j = 1$ to k **do**
6: $C_i = \emptyset$ ▷ Make clusters empty.
7: **end for**
8: **for** $i = 1$ to n **do**
9: **for** $j = 1$ to k **do**
10: $D_{ij} = \mathrm{d}(\mathbf{x_i}, \mathbf{y_j})$ ▷ $\mathrm{d}(\mathbf{x_i}, \mathbf{y_j})$ denotes the (Euclidean) distance between $\mathbf{x_i}$ and $\mathbf{y_j}$.
11: **end for**
12: Add $\mathbf{x_i}$ to $C_{\mathrm{argmin}(D_{i*})}$
13: **end for**
14: **for** $j = 1$ to k **do**
15: $\mathbf{y_i} = \mathrm{mean}(C_i)$ ▷ $\mathrm{mean}(C_i)$ denotes the average of all points in C_i.
16: **end for**
17: update *conv*. ▷ Update the convergence criterion.
18: **end while**
19: **return** $C = \{C_1, C_2, \ldots, C_k\}$

There are two adjustments needed to adjust the algorithm for k-medoids. First, when updating the cluster centres in line 15 in Algorithm 2, we do not use the mean but instead take the data point which minimizes the total dissimilarity. Second, the calculation of the distances between data points (lines 8-13 in Algorithm 2) is done once, prior to the iterative procedure.

B.2 Elastic distances algorithms

Algorithm 3 The dynamic programming algorithm to find the DTW distance for the Sakoe-Chiba band (Holder et al., 2024).

Input: $\mathbf{a}, \mathbf{b}$ (time series of length m), w (window size), M ($m \times m$ distance matrix of $\mathbf{a}$ and $\mathbf{b}$)
Output: $d_{\mathrm{DTW}}(\mathbf{a}, \mathbf{b}, w)$ (the DTW distance).

1: Initialise matrix $C \in \mathbb{R}^{(m+1)\times(m+1)}$ of zeros indexed from 0.
2: **for** $i = 1$ to m **do**
3: **for** $j = 1$ to m **do**
4: **if** $|i - j| < w \cdot m$ **then**
5: $C_{i,j} = M_{i,j} + \min(C_{i-1,j-1}, C_{i-1,j}, C_{i,j-1})$
6: **end if**
7: **end for**
8: **end for**
9: **return** $d_{\mathrm{DTW}}(\mathbf{a}, \mathbf{b}, w) = C_{m,m}$

Algorithm 4 The dynamic programming algorithm to find the MSM distance (Stefan et al., 2013).

Input: $\mathbf{a}, \mathbf{b}$ (time series of length m), c (cost of the Split/Merge operation)
Output: $d_{\mathrm{MSM}}(\mathbf{a}, \mathbf{b}, c)$ (the MSM distance).

Initialise matrix $C \in \mathbb{R}^{m \times m}$ of zeros.
$C_{1,1} = |a_1 - b_1|$
for $i = 2$ to m **do**
    $C_{i,1} = C_{i-1,1} + Cost(a_i, a_{i-1}, b_1, c)$ ▷ Initialise the first row.
end for
for $j = 2$ to m **do**
    $C_{1,j} = C_{1,j-1} + Cost(b_j, a_1, b_{j-1}, c)$ ▷ Initialise the first column.
end for
for $i = 2$ to m **do**
    for $j = 2$ to m **do**
        $C_{i,j} = \min(C_{i-1,j-1} + |a_i - b_j|, C_{i-1,j} + Cost(a_i, a_{i-1}, b_j, c), C_{i,j-1} + Cost(b_j, a_i, b_{j-1}, c))$
    end for
end for
return $d_{\mathrm{MSM}}(\mathbf{a}, \mathbf{b}, c) = C_{m,m}$

Algorithm 5 The dynamic programming algorithm to find the TWE distance (Marteau, 2009).

Input: $\mathbf{a}, \mathbf{b}$ (time series of length m), γ (the stiffness parameter), λ (the penalty parameter)
Output: $d_{\mathrm{TWE}}(\mathbf{a}, \mathbf{b}, \gamma, \lambda)$ (the TWE distance).

Initialise matrix $D \in \mathbb{R}^{(m+1) \times (m+1)}$ of zeros indexed from 0.
for $i = 1$ to m **do**
    $D_{i,0} = \infty$ ▷ Initialise the first row.
    $D_{0,i} = \infty$ ▷ Initialise the first column.
end for
for $i = 1$ to m **do**
    for $j = 1$ to m **do**
        $match = D_{i-1,j-1} + |a_i - b_j| + |a_{i-1} - b_{j-1}| + 2\gamma(|i - j|)$
        $insert = D_{i-1,j} + |a_i - a_{i-1}| + \lambda + \gamma$
        $delete = D_{i,j-1} + |b_j - b_{j-1}| + \lambda + \gamma$
        $D_{i,j} = \min(match, insert, delete)$
    end for
end for
return $d_{\mathrm{TWE}}(\mathbf{a}, \mathbf{b}, \gamma, \lambda) = D_{m,m}$

C Hyperparameter values

Distance	Parameter value
DTW	$w = 0.2$
DDTW	$w = 0.2$
WDTW	$g = 0.05$
DWDTW	$g = 0.05$
LCSS	$\epsilon = 0.05$
EDR	$\epsilon = 0.05$
ERP	$g = 0.05$
MSM	$c = 1$
TWE	$\gamma = 0.05, \lambda = 1$

Table 6: The hyperparameter values of the nine elastic distance measures as used in Holder et al. (2024).

D Software and packages

Like mentioned in Section 5, this paper uses the aeon package together with Python to perform the clustering algorithms for the elastic distance measures. At the time of writing, the most recent version of aeon was version 0.8.1. As this version contains some bugs for the k-means++ initialisation and does not support Forgy's method as an option for initialisation, an adjusted version of aeon was used. This version is available on GitHub[5], with some comments about the adjusted code.

For all experiments using a random number, the random state 1 was used. Furthermore, the default parameters provided by the packages are used across all models, except for the clustering algorithms. For the clustering algorithms, we adjust the maximum number of iterations to 100, which is lower compared to the default 300. It is unlikely that this has an impact on the results, as the default number of iterations in other packages such as 'tslearn' is 50. Furthermore, the clusterers converged in under 20 iterations for most of the datasets and no convergence warnings were obtained. Table 7 contains some additional information about the versions and usage of particular software utilized in this paper.

Table 7: Software and packages used in this paper.

Software/Package name:	Version:	Usage:
Python	3.11.9	Estimation of all models.
scikit-learn	1.4.2	Performing k-means for CSPA and calculation of performance metrics.
pandas	2.2.1	Data structures.
numpy	1.26.4	Array processing.
pymetis	2023.1.1	Performing the METIS algorithm.
aeon	0.8.1	Performing the elastic distance clusterings.
tsml-eval	0.4.0	Performing evaluation of models.
tsml	0.3.0	General framework for the machine learning models.

[5] https://github.com/jevdende/aeon-updated/

E Performance measures

This section will provide additional information and explain formulas for the performance measures. The notation in this section is consistent with the notation used in Section 4, but for the sake of clarity some of the notation will be restated. An overview and the literature on the measures are listed in Table 8.

Metric	Source
Clustering accuracy (CL-ACC)	Holder et al. (2024)
Rand Index (RI)	Hubert and Arabie (1985)
Adjusted Rand Index (ARI)	Hubert and Arabie (1985)
Mutual Information (MI)	Romano et al. (2014)
Adjusted Mutual Information (AMI)	Romano et al. (2014)
Normalized Mutual Information (NMI)	Romano et al. (2014)

Table 8: Performance measures used in this paper to evaluate the clusterings.

E.1 Clustering accuracy

The clustering accuracy is similar to classification accuracy, as it measures the percentage of correct predictions according to a predefined labelling. Let $\boldsymbol{\lambda}^{(j)}$ denote the label vector that represents the clusterings produced by clusterer j and let $\boldsymbol{\lambda}$ denote the actual label vector from the dataset. To compute the clustering accuracy, we have to assign each cluster to a certain class from the label vector $\boldsymbol{\lambda}$. By denoting S_k to be the set of all permutations of clusters and classes, we can compute the clustering accuracy as in Eq. 19.

$$\text{CL-ACC}(\boldsymbol{\lambda}^{(j)}, \boldsymbol{\lambda}) = \max_{\mathbf{s} \in \mathbf{S_k}} \frac{1}{|\boldsymbol{\lambda}|} \sum_{i=1}^{|\boldsymbol{\lambda}|} \begin{cases} 1, & \text{if } \lambda_i = \mathbf{s}(\lambda_i^{(j)}). \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

For a large number of clusters, taking every permutation can be computationally expensive. Therefore, combinatorial optimisation algorithms are employed to reduce the computation times. Just like Holder et al. (2024), the Hungarian algorithm (Kuhn, 1955) is employed on a cost matrix for this purpose.

E.2 Rand Index

The Rand Index (RI) is a similarity measure between two clusterings (Hubert & Arabie, 1985). Given clustering label vectors $\boldsymbol{\lambda}^{(a)}$ and $\boldsymbol{\lambda}^{(b)}$, let g denote the number of pairs of elements that are in the same cluster in both $\boldsymbol{\lambda}^{(a)}$ and $\boldsymbol{\lambda}^{(b)}$. Furthermore, let h denote the number of pairs of elements that are in different clusters in both $\boldsymbol{\lambda}^{(a)}$ and $\boldsymbol{\lambda}^{(b)}$. For a dataset with n observations, we can then compute the RI as in Eq 20.

$$\text{RI}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)}) = \frac{g + h}{\binom{n}{2}} \tag{20}$$

It can be observed that the RI is just the fraction of pairs that are either both in the same cluster or are both in different clusters for the two clusterings, with a higher value indicating more similar clusterings. By then taking one of the two clusterings to be the actual clustering as provided by the dataset, we obtain a measure for how similar our new clustering is to the actual clustering.

The adjusted version of RI called Adjusted RI (ARI), corrects the RI to adjust for change (Hubert & Arabie, 1985). Let v denote the number of pairs of elements that are in the same cluster in $\boldsymbol{\lambda}^{(a)}$,

but are in different clusters in $\boldsymbol{\lambda}^{(\boldsymbol{b})}$. Then, let w denote the number of pairs of elements that are in the same cluster in $\boldsymbol{\lambda}^{(\boldsymbol{b})}$, but are in different clusters in $\boldsymbol{\lambda}^{(\boldsymbol{A})}$. The ARI is then given as

$$\text{ARI}(\boldsymbol{\lambda}^{(\boldsymbol{a})}, \boldsymbol{\lambda}^{(\boldsymbol{b})}) = \frac{(g+v)\cdot(g+w)+(h+v)\cdot(h+w)}{\binom{n}{2}}. \tag{21}$$

E.3 Mutual Information

Just like the RI, the Mutual Information (MI) measures the similarity between two clusterings. However, instead of being based on counts, the MI is based on information theory (Romano et al., 2014). Let $n_i^{(a)}$ denote the number of objects in the i-th cluster of labeling $\boldsymbol{\lambda}^{(a)}$. Then, let $n_{i,j}$ denote the total number of points that are both in the i-th cluster of labelling $\boldsymbol{\lambda}^{(a)}$ and the j-th cluster of labelling $\boldsymbol{\lambda}^{(b)}$. Then, for a dataset with k clusters and n objects, we can compute the MI with

$$\phi^{\text{MI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)}) = \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{n_{i,j}}{n} \log\left(\frac{n \cdot n_{i,j}}{n_i^{(a)} n_j^{(b)}}\right). \tag{22}$$

The Normalised MI (NMI) is another measure that ensures that the MI is between 0 and 1, making it easier to interpret. The NMI achieves the normalisation by dividing the MI by the product of the entropy of the individual clusterings. Let $H(\boldsymbol{\lambda}^{(a)})$ denote the entropy for clustering $\boldsymbol{\lambda}^{(a)}$, such that it can be computed as

$$H(\boldsymbol{\lambda}^{(a)}) = \sum_{i=1}^{k} \frac{n_i^{(a)}}{n} \log\left(\frac{n_i^{(a)}}{n}\right). \tag{23}$$

Then, the final NMI is given by

$$\phi^{\text{NMI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)}) = \frac{\phi^{\text{MI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)})}{\sqrt{H(\boldsymbol{\lambda}^{(a)})H(\boldsymbol{\lambda}^{(b)})}}. \tag{24}$$

Both the MI and NMI are not corrected for change. Therefore, another version has been proposed called the Adjusted Mutual Information (AMI). The AMI uses the expected value of MI under the null hypothesis of $\boldsymbol{\lambda}^{(a)}$ and $\boldsymbol{\lambda}^{(b)}$ being independent. This expectation is denoted with $\mathbb{E}[\phi^{\text{MI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)})]$, but the derivation is out of the scope for this paper. For the interested readers, we refer to Romano et al. (2014). The final AMI is then computed as in Eq. 25.

$$\phi^{\text{AMI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)}) = \frac{\phi^{\text{MI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)}) - \mathbb{E}[\phi^{\text{MI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)})]}{\max(H(\boldsymbol{\lambda}^{(a)}), H(\boldsymbol{\lambda}^{(b)})) - \mathbb{E}[\phi^{\text{MI}}(\boldsymbol{\lambda}^{(a)}, \boldsymbol{\lambda}^{(b)})]} \tag{25}$$

F Run times

Table 9 contains the run times in hours for the different distance measures. As expected, the Euclidean distance has the lowest run time. The MSM distance performs second best, which is notable considering it is among the top-performing algorithms in terms of accuracy. The LCSS and EDR distances perform worst in terms of run times, just like they did in terms of accuracy.

Distance	Fit time	Predict time (train set)	Predict time (test set)
DTW	6.90	0.04	0.43
DDTW	13.37	0.04	0.42
ED	0.04	0.00	0.00
EDR	26.50	0.09	1.07
ERP	8.42	0.09	1.06
LCSS	23.67	0.09	1.07
MSM	1.40	0.02	0.29
TWE	15.75	0.33	3.81
DWTW	11.31	0.09	1.09
WDDTW	6.47	0.09	1.07
Total	113.84	0.90	10.31

Table 9: Run times in hours across the different distance measures.

When we look at the difference in run times between the two clustering algorithms in Table 11, we observe that k-medoids is more than 10 times as fast as k-means for the fitting procedure. Considering that k-medoids outperforms k-means in terms of accuracy, it further supports the argument that k-medoids should be used for a robust benchmark for time series clustering.

Algorithm	Fit time	Predict time (train set)	Predict time (test set)
k-means	105.28	0.45	5.14
k-medoids	8.56	0.45	5.16

Table 10: Run times in hours across the different clustering algorithms.

Finally, examining the run times of the various initialization algorithms in Table 11, we observe that the random and Forgy initialization methods have similar fit times, with k-means++ being marginally slower.

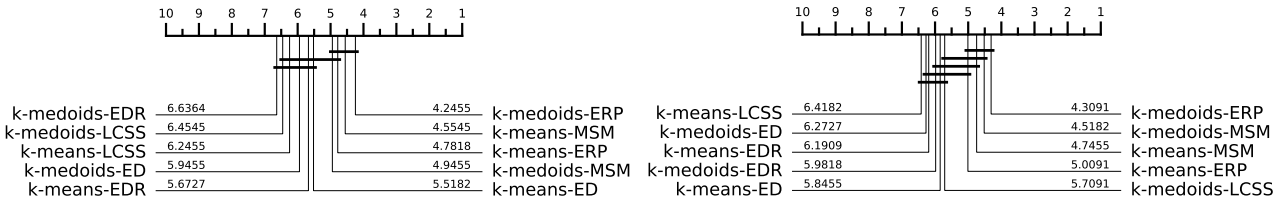
Initialisation	Fit time	Predict time (train set)	Predict time (test set)
random	36.34	0.3	3.50
Forgy	36.58	0.3	3.41
k-means++	40.92	0.3	3.39

Table 11: Run times in hours across the different initialisation algorithms.

G Results on the train set

	k-means (%)			k-medoids (%)		
Distance	random	Forgy	k-means++	random	Forgy	k-means++
DTW	63.75	64.24	64.59	65.29	65.66	65.85
DDTW	60.95	62.47	61.48	62.92	63.65	62.57
ED	63.11	63.12	62.88	63.42	63.15	62.47
EDR	62.01	62.18	61.32	61.96	59.49	62.94
ERP	64.7	65.08	64.61	67.22	67.88	68.28
LCSS	60.07	61.3	60.8	61.74	60.11	63.67
MSM	65.71	65.96	65.33	66.43	64.65	67.22
TWE	64.24	63.68	64.12	65.06	63.65	65.48
WDTW	63.47	64.57	64.23	65.23	64.68	65.25
WDDTW	63.25	62.96	61.71	62.68	65.24	63.17
Average	63.13	63.56	63.11	64.19	63.82	64.69

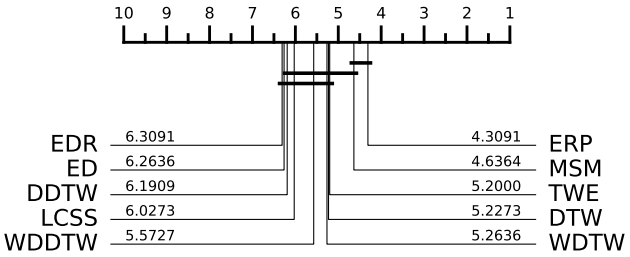
Table 12: Accuracies of the different clustering algorithms for each of the distance functions in the train set. Bold is the highest accuracy across the different initialisation algorithms.



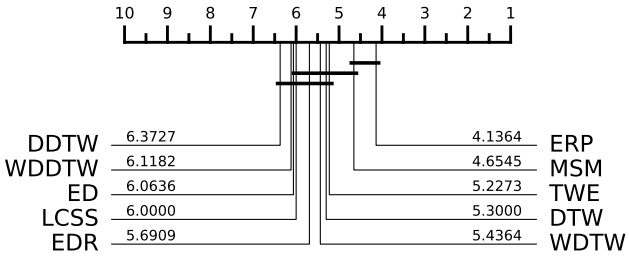
(a) Forgy initialisation

(b) k-means++ initialisation

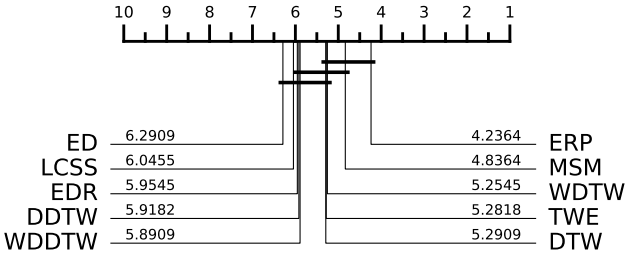
Figure 7: Accuracies of k-means and k-medoids on the train set for the ED, ERP, MSM, LCSS and EDR distances. Results on the left are for the Forgy initialisation and on the right for the k-means++ initialisation.



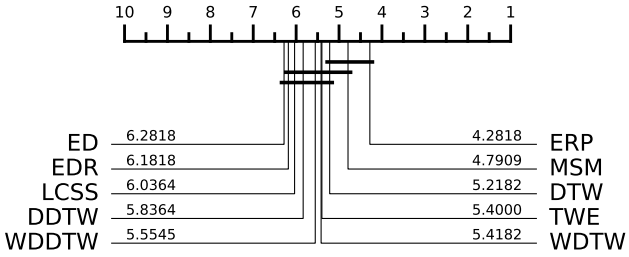
(a) Accuracy



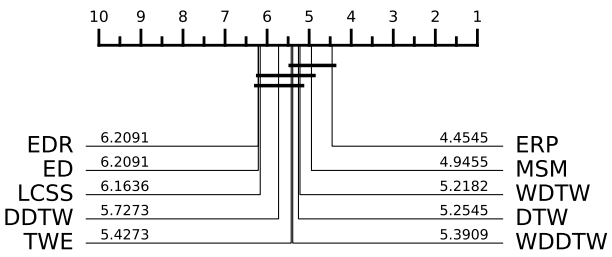
(b) Rand Index



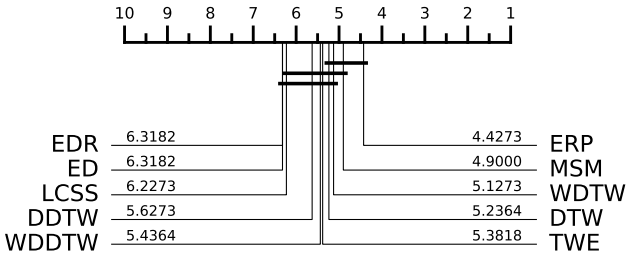
(c) Adjusted Rand Index



(d) Mutual Information

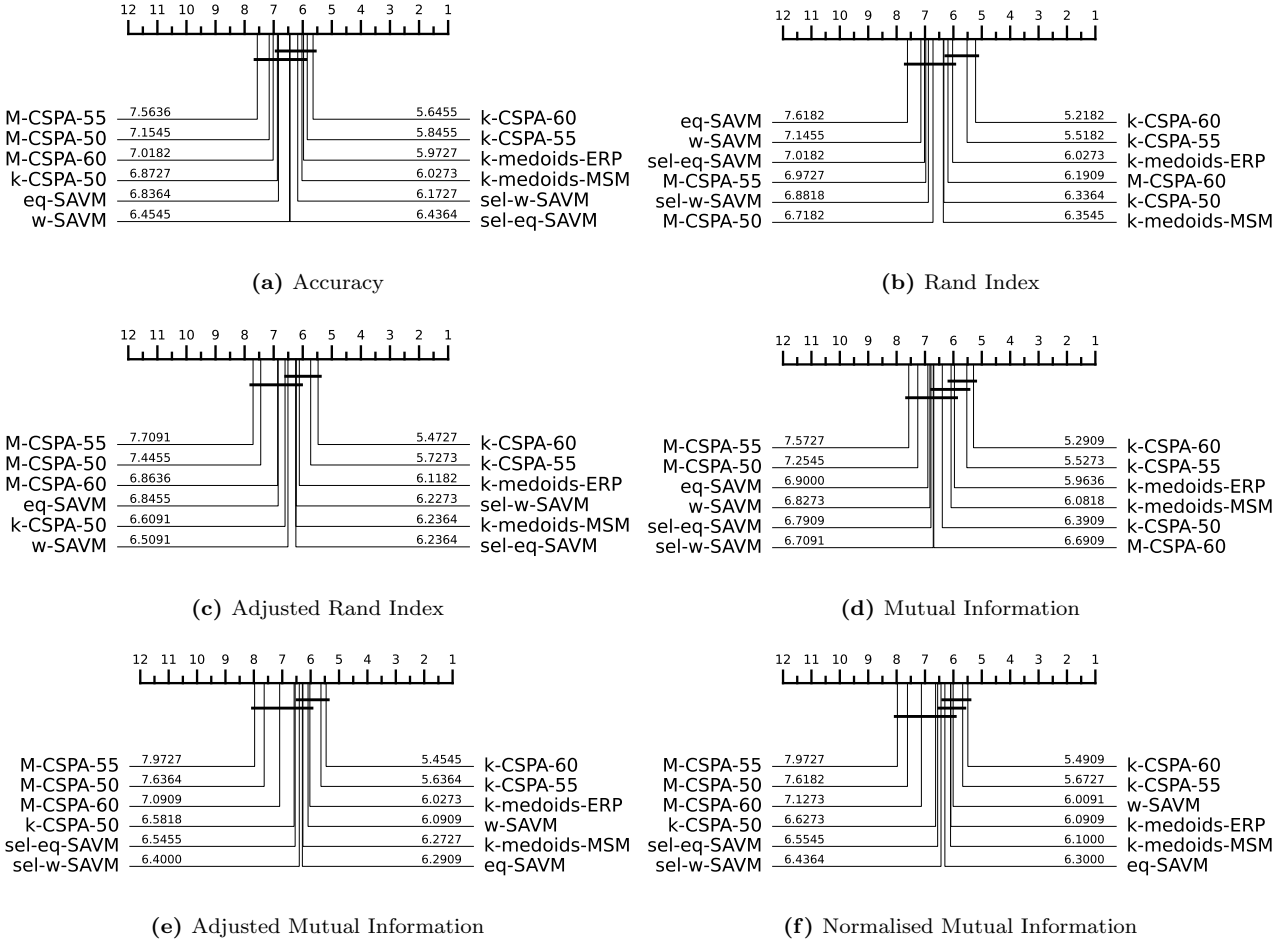


(e) Adjusted Mutual Information



(f) Normalised Mutual Information

Figure 8: Results for k-means++ k-medoids on the train set using ten different distance functions.



(a) Accuracy

(b) Rand Index

(c) Adjusted Rand Index

(d) Mutual Information

(e) Adjusted Mutual Information

(f) Normalised Mutual Information

Figure 9: Results of the ensembles methods against k-medoids-MSM and k-medoids-ERP on the train set. The numbers succeeding 'CSPA' represent the quantity of clusterings incorporated in the selection process. The letters M and k preceding 'CSPA' denote the METIS and k-means algorithms, respectively. For SAVM, the voting schemes *equal*, *weighted*, *selective-equal*, and *selective-weighted* are abbreviated as *eq*, *w*, *sel-eq*, and *sel-w*, respectively.

	Type					Clusters			Training instances			Length		
Accuracy(%)	A	B	C	D	E	A	B	C	A	B	C	A	B	C
k-CSPA-60	**64.10**	71.55	69.78	**70.93**	63.74	68.97	64.99	**65.76**	68.66	**73.86**	59.71	67.91	69.99	**65.39**
k-CSPA-55	63.73	71.55	71.55	70.47	63.74	69.14	64.99	65.37	68.88	73.66	59.32	68.15	69.62	65.51
k-CSPA-50	62.59	70.90	69.21	70.78	63.33	67.79	64.99	63.89	67.07	73.56	59.32	66.94	68.52	64.20
M-CSPA-60	59.70	74.28	66.17	68.43	59.97	67.01	63.80	61.12	67.04	71.87	54.36	65.12	68.09	63.42
M-CSPA-55	58.87	73.99	65.65	68.43	58.53	66.51	64.15	60.61	66.78	70.67	54.28	64.72	67.32	63.49
M-CSPA-50	59.43	72.82	70.46	68.56	60.24	68.21	61.89	60.09	67.04	74.61	54.12	66.70	66.80	63.32
eq-SAVM	65.05	72.51	59.29	68.00	60.26	66.61	66.30	62.30	67.06	67.40	59.78	65.29	69.89	62.10
sel-eq-SAVM	64.65	70.60	64.42	69.49	63.45	68.22	64.21	62.31	67.31	72.57	58.85	66.74	68.17	64.64
sel-w-SAVM	64.65	71.49	64.47	69.49	63.03	68.30	65.40	62.51	67.68	72.57	58.89	67.10	68.10	65.03
w-SAVM	65.44	73.24	60.50	69.11	63.33	67.83	67.01	62.48	67.78	70.55	59.65	66.88	69.96	62.80
DTW	61.77	70.96	76.66	64.64	62.91	67.41	61.40	63.04	66.02	70.46	61.08	67.15	67.16	61.64
DDTW	63.72	66.21	57.05	63.31	55.48	62.83	61.48	62.33	63.16	63.50	59.58	63.04	60.89	63.52
ED	59.09	67.38	62.63	66.51	58.28	64.95	60.67	54.34	64.81	65.90	50.99	61.95	65.59	59.96
EDR	58.08	70.25	55.27	63.07	64.66	64.12	65.31	56.83	65.71	66.30	49.97	61.27	66.80	61.96
ERP	59.51	**74.61**	**78.48**	68.31	**66.38**	**70.58**	66.57	60.74	**69.46**	73.33	59.49	**71.51**	69.24	60.46
LCSS	58.14	68.79	60.95	61.12	64.00	65.88	64.57	54.65	66.71	66.14	50.51	62.18	65.94	64.14
MSM	63.55	71.62	71.67	68.33	62.65	67.92	**68.56**	63.59	68.27	74.17	57.17	68.22	**70.19**	61.69
TWE	61.08	69.87	69.95	68.62	62.94	66.87	64.57	60.83	65.77	72.09	58.47	66.57	68.85	59.32
WDTW	61.24	71.24	76.49	62.46	62.21	65.90	62.48	64.73	65.23	69.43	**61.57**	66.92	66.68	60.14
WDDTW	65.39	66.77	62.78	62.17	55.65	62.91	61.54	65.28	63.66	63.41	61.16	63.66	61.46	64.12

Table 13: Accuracies of k-means++ k-medoids, CSPA and SAVM for the different groups of the performance analysis. Bold is the highest accuracy for each group. Results are on the train set.