Universiteit Leiden
The Netherlands

Opleiding Informatica

An aggregrative approach to time series classification

Aras Aslam Hayat

Supervisors:
Arno Knobbe & Saber Salehkaleybar

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

05/06/2023

Abstract

Time series classification is a part of supervised machine learning, where the aim is to predict or classify time series data into categories or classes. A time series is a sequence of measurements or observations that are collected over a specific timeframe. In this supervised setting, we have entire time series that are associated with a specific label. Dataset labelling is associated with each time series instance with its corresponding category or class, based on its observed characteristics.

The main objective here is to look for patterns, trends or other statistical measures within these temporal observations that can help us with the classification problem. The time series can vary across domains, such as sensor readings, audio fragments, or other temporal data.

Our investigation focuses on a total of six standardized background approaches. These approaches are known as: Whole series, Interval, Shapelet, Dictionary-based, Combinations, and Model-based. Furthermore, nine baseline algorithms are derived from these six approaches. The algorithms are: COTE, ST, Boss, EE, DTW_f, TSF, TSBF, LPS, and MM. These approaches and algorithms will serve as our benchmark to calculate the accuracy of the classification model's ability to predict or classify the labels of unseen data points.

In this thesis, we will be looking at an alternative approach to time series classification, which is by using an algorithm that is based on aggregative metrics. To measure the effectiveness of this aggregative approach, we will be conducting a comparative analysis with the other existing algorithms. We will do this by using a benchmark of 85 datasets where not only our aggregative algorithm but also all the other algorithms have been tested. Our aim is to show the effectiveness of the aggregative metrics in accurately assigning labels to the data points within the observations.

Contents

1 Introduction

Time series classification [25] is a fundamental task within the realm of supervised machine learning. Supervised machine learning is a subset of machine learning where labeled datasets will be used in order to train algorithms to predict or classify unseen data. Time series classification will use supervised machine learning techniques to build a model. The model is trained with labeled data points that are associated with its category or class [9]. This allows the model to assign a label to unseen observations based on the relevant information extracted from the labelled training data.

In the past, researchers have made several attempts to find algorithms that are effective in classifying time series observations. However, time series classification comes with some challenges that go beyond machine learning classifiers. These challenges include overfitting, underfitting, data misinterpretation, and biases in data distribution. These challenges need to be solved, and it is interesting to look at the capabilities of an algorithm that will try to mitigate these challenges.

Time series classification algorithms can be applied directly to process raw time series data or use preprocessed features derived from it. Raw data may contain noise, outliers, or irrelevant features, which can negatively affect classifier performance by masking critical patterns. This can also lead to high computational work in the case of large datasets, which in turn leads to increased time and cost expenses. To resolve these problems, we can use the indirect approach to apply time series classification. Using the indirect approach, we will use algorithms that will use feature engineering techniques to extract characteristics from the time series data as input for a standard machine learning classifier. This allows us to be less susceptible to noise or outliers, as well as being computationally more efficient. The aggregative algorithm is an indirect method that can be used for time series classification.

There are several classifiers that can help in the second step of this indirect approach, but in this thesis we have chosen the Random Forest classifier for the final prediction making [6]. This is due to its ability to being robust against noise and outliers, handling large datasets and providing us of a feature importance.

In this thesis, we study the performance of the aggregative algorithm on time series data. This algorithm summarizes a collection of values (in this case, the data points) into one single value. This is done with the help of aggregation functions. These aggregation functions will help us extract relevant information in order to capture the essential characteristics [5]. By collecting a lot of characteristics, these will be able to act as our extracted features. These features will be used as input for the Random Forest classifier. Finally, we are able to perform time series classification.

1.1 Thesis Overview

In this thesis, we will gain a deeper understanding of time series classification. We will be discussing the various techniques that are used to classify time series data accurately.

In Section 2, we will describe the underlying structure of the benchmark algorithms that have been used to conduct time series classification. We will also describe the datasets that have been used by

the algorithms.

In Section 3, we will explain the methodology that has been used. We shall go through the step-by-step approach and show all the aspects that have been examined and utilised in order to create the aggregative algorithm.

In Section 4, we will explain what types of experiments we have performed. These experiments will show the statistical results as well as the accuracy scores.

Section 5 is a continuation of Section 4, where we will explain what the results tell us and how these need to be interpreted. We will also explain how the aggregation algorithm can be interpreted in the comparative analysis among the other algorithms.

In Section 6, there will be a conclusion about the total performance of the aggregation algorithm in comparison to the other algorithms based on the benchmark datasets, and finally, we will be looking into future research and discussing critical points that could actually solve and improve the time series classification problems.

This thesis was written as a part of the Computer Science bachelor's degree at the Leiden Institute of Advanced Computer Science (LIACS) and supervised by Dr. A.J. Knobbe and Dr. S. Salehkaleybar.

2 Background

In this chapter, we will review time series classification techniques. Furthermore, in Section 2.2 the algorithms that result from these techniques are described.

The research in this thesis is inspired by one paper: *"The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances"* [5]. Many of the setting within this thesis will be based on this paper. We will also use the results from this paper for the comparative analysis of our aggregative algorithm.

2.1 Time Series Classification techniques

In the field of time series classification (TSC), various techniques are used to extract relevant information during the construction of a classifier. The performance of classifiers varies when they encounter different forms of time series data. An algorithm can perform effectively for one specific data type, however, its performance does not necessarily mean that it will generally perform well for other data types. This is the reason why it is important to use and look into different techniques when we try to extract relevant information.

TSC has different approaches for data extraction that have found applications in different research areas. Understanding the distinctions and similarities among the approaches is crucial for comprehending their practical implications. The approaches are categorised as follows:

1. Whole-Series: This is an approach where the collection of datapoints from the time series is treated as one single entity. When we compare two series with each other, we can either use similarity or distance measures. The similarity/distance measure will use all the information in the time series in order to search for similarities between different time series. To further improve the effectiveness of the distance measure in cases of small misalignments between different time series, elastic distance measurements can be used as a solution, such as the so-called Dynamic Time Warping [17]. This mechanism is able to mitigate these misalignments.

An advantage of this approach is that there is no need for feature extraction. The disadvantage of this approach revolves around dealing with different lengths of the time series during comparisons. Suppose we use time series for comparisons, it is essential for all of them to be of equal lengths. However, if time series of varying lengths are used, it may lead to inaccurate or biased comparisons. This technique is also highly affected by outliers, which can also have a negative impact on the overall performance.

To give an example of the overall structure of this approach, consider the four time series in Figure 1. Suppose that these time series are named t_0,t_1,t_2 and t_3. An algorithm will use the elastic distance measure between the different series to indicate which series are similar [5]. As a result, we consider t_0 showing the most similarities with t_1, and therefore the least similarities with t_2 and t_3. Based on this technique, we end up with a total of two pairs of series: the black lines (t_0 and t_1)

and the green lines(t_2 and t_3).

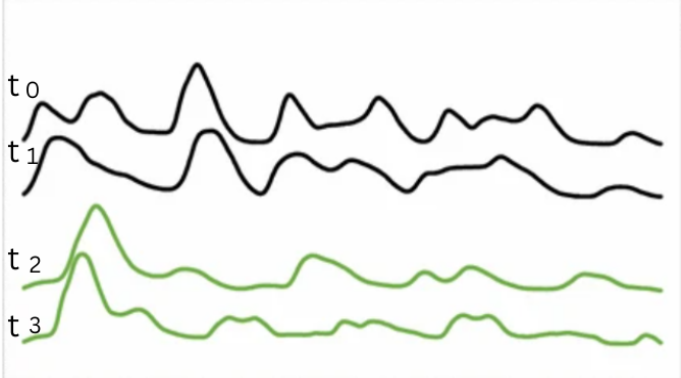

Figure 1: We can see four coloured lines. What we can observe is that there is a clear similarity between the black lines and the green lines. Source:[5].

2. Interval: Unlike the Whole-Series approach, which considers the entire time series, the interval approach will concentrate on one or more specific interval(s) in the time series based on certain characteristics. The background of the selected interval depends on these contiguous subsets of features or characteristics, each consisting of (unique) information that potentially distinguishes it from other intervals. These intervals can occur at random places within the time series. This is where the use of *phase-dependent* intervals becomes crucial. The phase-dependent intervals will distinguish themselves from other intervals because of the **relevant** information they gather between fixed time intervals. This relevant information can consist of statistical measurements that result in optimal accuracy or interpretability for the classifier [5]. This method is actually able to capture the time series at different stages of time, which means that we can focus more on the relevant data and eliminate portions with significant noise for certain conditions.

The challenging part of this technique can be finding the optimal setting for the interval. One of the reasons why this can be so difficult is the fact that the lengths of the intervals are hard to set correctly. Solutions for this problem could be fixed or dynamic intervals based on specific time series data.

Consider Figure 2, where we see an ECG during two heartbeats. If we look at this figure, we can see that there is a lot of movement of the three coloured lines (ventricular pressure, ventricular volume, and electrocardiogram). We notice that during specific times, there is a huge difference in the movement of these signals, particularly noticeable during the systole phases. These systole intervals could be marked as phase-dependent movements containing several unique features or characteristics. These features/characteristics can help in classifying the time series data.

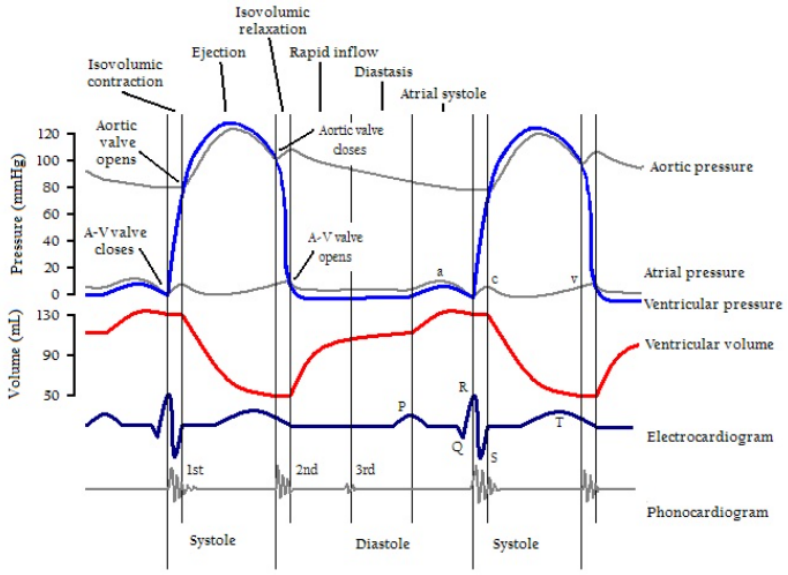

Figure 2: In the systole phases, we can see that there is a phase-dependent movement containing a lot of noise, which contains useful information about the class. Source: [2].

3. Shapelet: This is an approach that is based on a unique pattern or a discriminative subsequence within a time series that can be used for classification. For finding these patterns, it is not relevant where these patterns are localized. This allows us to detect **phase-independent** patterns that are unique for their presence or absence in the data series. This method is less susceptible to outliers. But a disadvantage of this method is that it is very computationally expensive to find differentiating patterns at the appropriate length.

To give an example of the workings of a Shapelet, consider Figure 3. On the left, we see two collections of time series, each representing several examples of walking on different surfaces, which in this case are either carpet (top) or cement (bottom). The challenge is to determine for a new time series (on the right) whether the surface is made of cement or carpet. First of all, Shapelet-based techniques will try to find unique or discriminative patterns. After having a total of 4 unique/discriminative patterns (marked in red), we will try to find which of the two collections on the left shows the best presence of these patterns. After we have found two distinctive patterns in the series on the right, we can see that for the intervals of the first pattern (x-axis, 0–25) and the second pattern (x-axis, 42–62), the carpet surface fits these patterns the best at these intervals. We can conclude that the target series also represents "Walking on Carpet".

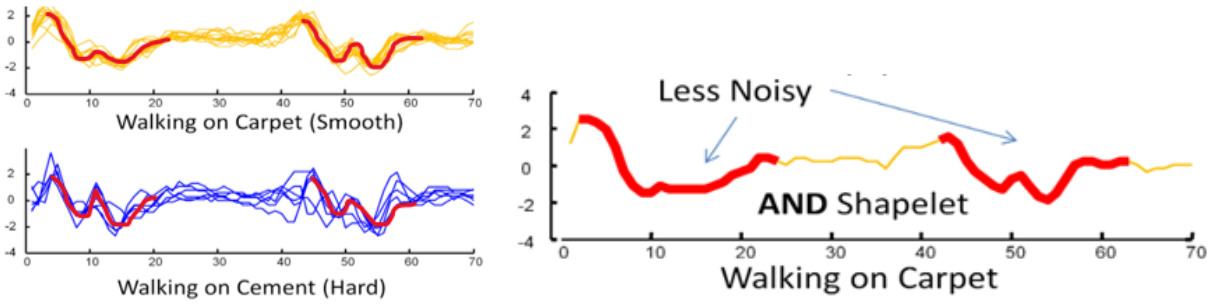

Figure 3: On the left, two collections of time series with class labels "Walking on Carpet" (top) and "Walking on Cement" (bottom). Based on the two patterns indicated in red, the target series on the right is likely of class "Walking on Carpet". Source: [2].

4. Dictionary based: This approach is, in essence, similar to a Shapelet. In the case of a Shapelet, we usually look at its presence or absence in the series. In the dictionary method, one looks at pre-determined sequences that form the elements of a dictionary. With the help of each subsequence, one focuses on the frequency of pattern recurrence rather than one single phase-independent pattern.

An example where we would benefit from using the dictionary-based approach is shown in Figure 4. We can see that we have a lot of noise in these series. If we tried to find a Shapelet in order to look for the presence or absence of this unique pattern, this would be almost impossible and confound this approach. A Shapelet could be applicable to multiple series due to the similarity of the pattern. This is due to the fact that it is hard to find a unique pattern. However, in the case of a dictionary-based approach, we would more likely get a more accurate performance because we focus on the frequency of the patterns based on a pre-determined sequence [5].

Figure 4 shows several series that show the effects of an earthquake. We have a pre-determined sequence of upward and downward movements. The number of these movements indicates the power of an earthquake. What we can observe is that there is a recurrent pattern of a spike quickly moving upwards and downwards. If we focus on the frequency of this pattern, we can see that the series at the first level has the most pattern occurrences. Therefore, we can conclude that this series can be classified as the earthquake with the most powerful impact.

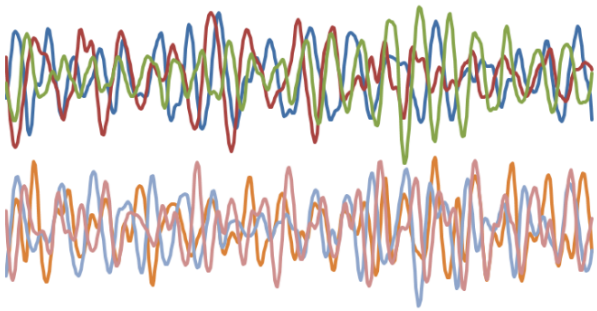

Figure 4: We see a frequent pattern of upwards and downwards movements for the datasets, which indicate the power of the eartquakes. Source: [2].

5. Combinations: This is an approach that is a combination of several of the methods that have been discussed previously. Combining multiple techniques can lead to more computational needs and higher complexity in the implementation of this approach, but on the other hand, it can also lead to better classification performance, being less sensitive to outliers, and performing better on different domains of the data because of the variations in classifying unseen data [5].

An example of such a combination can be seen in Figure 5, which represent a certain movement for the right and left hand. We will use a combination of the Whole Series approach and the Shapelet-based approach. First of all, we will use a distance measure to see if we have a lot of similarities between the blue and yellow lines (left). After completing this first section of the approach, we will look deeper into the series by using a Shapelet-based approach. On the chart (right) we can see that we used 2 unique patterns that have each been localized for a total of 2 times in the chart. By using this combination approach, we managed to extract features more efficiently, which made us less sensitive to outliers and resulted in better classification performance. Eventually, we conclude that we have a total of 3 identical gestures for the hands.

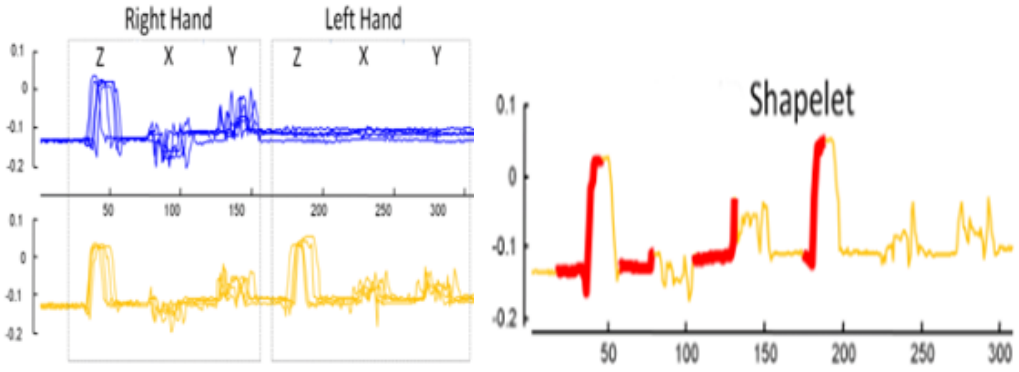

Figure 5: This represents a combination of the Whole-Series approach (left) and a Shapelet-based approach (right). We determine an identical movement of the right hand for a total of two times and one for the left hand. Source: [5].

6. Model-based: This approach applies a generative model to time series, such as kernel or autoregressive models. The generative model tries to capture the underlying characteristics, such as coefficients or parameters, of the time series. Afterwards, these models are compared to each other to find similarities. These extracted features can then be used to train a classifier in order to label unseen observations.

This technique is also flexible in applying a model that gives the best results when we try to classify the data. But this can confound this technique because an inappropriate selection could lead to misleading assumptions about the dataset, which are not representative of the labelling of unseen data and can negatively affect it [5]. In Figure 6, we have an autoregressive model (AR) [15], including some time series data. The autoregressive model will try to predict future values based on a linear combination of past values. These past values are the values that are presented in Figure 6. By analyzing the relationship between these past values and using these values in a formula, the model will classify unseen time series data.

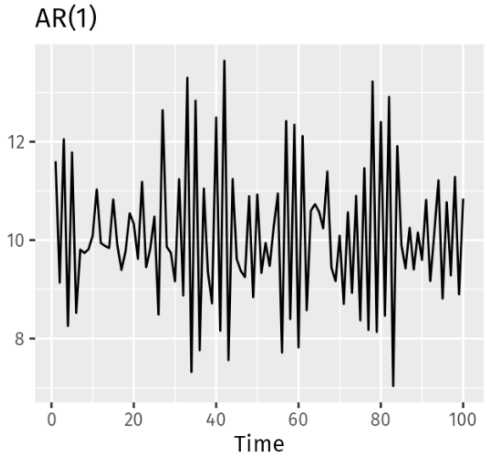

Figure 6: This is an autoregressive model that is used to predict variables based on previously seen values. Source: `www.otexts.com`

.

Besides these techniques, there are also other possibilities, such as deep learning [12] or the k-nearest-neighbour [22] approach. However, in this thesis, we will mainly focus on the approaches described before. The main reason for this is that we will use nine algorithms for comparative analysis. These algorithms are all based on the approaches that we have broadly discussed above.

2.2 Related Work

The techniques that have been discussed before serve as the foundation for creating a lot of algorithms that can be used for time series classification. In this thesis, we will be focusing on nine different algorithms, which we will use for comparative analysis against our own algorithm.

The reason we will use these nine algorithms is based on their performance. In the paper [5], the researchers have analysed a lot of different algorithms. What they found was that based on their benchmark classifiers (1-NN DTW and Rotation Forest), only nine performed better than these classifiers. That is why we will also use these nine algorithms. To get a better and deeper understanding of each algorithm, we will briefly outline their characteristics.

Collection of Transformation Ensembles (COTE): The base foundation approach is Combination. This combination consists of several domains, such as time, autocorrelation, power spectrum, and shapelet [5]. We can use the different techniques and their strengths in order to extract relevant features from the time series data. This combination allows the algorithm to perform relatively well in time series classification problems [3].

Shapelet Transform (ST): The base foundation approach in this case is the Shapelet. The Shapelet will try to find the discriminative subsequences that separate the time series data from each other. The shapelet tries to find the k best shapelets, which will be used to create a transformed dataset. In this dataset, the information about the distance between a time series and a shapelet is stored, which means that these act as features containing the information for the interpretability of the time series classification problem [14].
Next, a combination of an ensemble of classifiers is used, such as k-Nearest Neighbour and Decision Trees [5], and we assign a weight-based cross-validation on the training accuracy. This means that we can make accurate predictions based on a weighted vote within these classifiers for the best solution.

Bag of Symbolic Fourier Approximation Symbols (BOSS): This is an algorithm that is based on the Dictionary-based technique. Time series data can contain random variations, also called outliers or noise, that can influence the performance of an algorithm. This algorithm aims to apply noise reduction to the time series. It will search for substructures within the series and remove them from the original time series. After these steps, the quantization will take place. In this step, the time series data will be grouped based on pre-determined subsequences. Finally, we are left with a string that can be used for algorithm matching.
This whole process is called 'Symbolic Fourier Approximation' (SFA) decomposition [4]. We have transformed the time series data into a string. The string consists of subsequences. These subsequences represent the symbolic representation as a bag-of-symbols. The bag-of-symbols method is used as our feature for frequency of pattern recurrence recognition. The number of occurrences will

be stored as a feature vector [5]. With a final classification algorithm, such as k-nearest-neighbours [22], the time series classification labelling will be based on these vectors.

Elastic Ensembles (EE): This algorithm is based on the Whole Series approach. The algorithm itself is an ensemble of a total of 11 Nearest Neighbour (NN) classifiers, where each uses various elastic distance measures for comparing time series data. An example of these classifiers is the following: Euclidean distance and the full window DTW [5]. The Euclidean distance is a measurement that is used to determine the similarities or dissimilarities between time series. This is done by calculating the distance between two datapoints, summarizing them, and taking the root square of the total sum for a certain interval. Dynamic Time Warping (DTW) uses a technique to find the optimal alignment between datapoints by warping time. This means that it can compress or stretch parts of the series to find the optimal match between the time series. It aims to minimize the total distance between the datapoints. This is particularly useful when comparing time series with different lengths to look for similarities.
This algorithm uses a weighted voting scheme and cross-validation among the classifiers to determine which of these classifiers fits the best given the dataset. This is done based on the training set accuracy, meaning which algorithm performed the best on the specific time series data. It uses 1-NN classifiers combined with the weighted structure to choose the best classifier [19]. This is also the reason why this algorithm is named elastic ensembles, because it is flexible enough to adjust as best as possible to a dataset based on the internal structure of the algorithm. This ensemble of classifiers is, in general, more accurate at labelling unseen data correctly than a single classifier.

Dynamic Time Warping as Feature Extraction (DTW_f): This is an algorithm that is based on a combination of Whole Series and Dictionary-based approaches. This algorithm consists of the basic Dynamic Time Warping (DTW) [17] and the Symbolic Aggregate Approximation (SAX) [18]. The training set is transformed into a set of features. Each feature will be compared by means of the window of DTW, that measures the distance between different pairs captures characteristics. Additionally, another set of features will be created where the optimal distances of the DTW cases is stored. After this application, we continue with the next method called SAX. SAX will transform the numeric values of the features in discrete symbols, the so-called word. The time series is than transformed to a collection of words. Furthermore, a histogram is created that counts the frequency of each word. This information is concatenated with the characteristics of the window DTW features. This new dataset is used by a machine learning algorithm for the classification of unseen observations such as a Support Vector Machine (SVP) [5].

Time Series Forest (TSF): This is an algorithm whose base foundation is the Interval approach. The base method is a Random Forest-like approach. Random intervals from the time series will be chosen. Next, time series data (intervals) containing important statistical information will be computed. By using a number of overall statistics for each of these interval features, multiple trees will be randomly created. By using this method to create tree structures, we can gain multiple insights from the time series data [11].
This makes this algorithm suitable for time series data that contains a large number of features. When we have to classify time series data, these will be passed through multiple decision trees, and finally, there will be a majority vote of the tree ensemble to determine the final prediction.

Time Series Bag of Features (TSBF): This algorithm is also based on the Interval approach as its foundation. It is also an extension of the TSF that was discussed earlier. The algorithm consists of four stages. The first stage is based on the fact that we create new intervals of the data. Next, subseries of various lengths and at arbitrary locations are chosen within these intervals. Furthermore, we will add the overall statistics of the subseries to each of these interval-based features.
The following stage will use a random forest, which will be trained using the newly created subseries. Eventually, a probability class of estimates for each subseries will be created. The third stage will combine the class probabilities and form a bag of patterns of subseries for each series [8]. Here, we have extracted patterns or characteristics of the time series data and counted the occurrences of these features. The algorithm aims to ensure that the out-of-bag error is minimal before entering the next stage. The out-of-bag-error in this case is the performance of the classifier based on the future unseen observations. In the final stage, a Random Forest classifier will be built based on the bag of features [5].

Learned Pattern Similiarity (LPS): This algorithm builds upon the Interval approach as its foundation. In this algorithm, LPS uses a regression model rather than a classification model. This means that we are actually looking for a correlation between the subseries rather than pattern recognition, which you would normally see when working with a classification model [7]. We will use an interval structure where subseries will be created once again. These random subseries will then be combined to create a new set of attributes. These attributes are then randomly selected and used for the input for an ensemble of regression trees. The leaf nodes of these regression trees will be concatenated for the classification of the unseen data [5]. Finally, new cases of time series data will be classified based on a 1-NN classification [22] and the concatenated regression tree nodes.

Move-Split-Merge (MSM): This is an algorithm whose foundation is the Whole Series approach. MSM is used to measure the similarities between different time series, which is fulfilled by transforming the first time series into the second time series. The algorithm is built on three different fundamental operations: Move, Split, and Merge. Move is an operation that changes the value of a single element; the split operation aims to convert a single element into two successive elements; and finally, the merge operation merges two successive elements into one [5].
Every time that an operation is obligatory, there is an associated cost. The aim here is to obtain the cheapest sequence of operations for the transformation of the time series into a target series. We can use these similarity measures to classify time series data [1].

3 Methods

In this chapter, we will explain the concept of an aggregation algorithm and its fundamental principles. We will also provide an overview of the steps we took to develop our aggregative approach. We will show how we have worked with the raw datasets, extracted them as whole time series, and later examined subsequences of the time series data. Additionally, we will explain the workings of the Random Forest classifier, which serves as our machine learning method.

3.1 Aggregation of Time Series

Time series have a different format in comparison to the normal tabular structure of machine learning, where rows represent example and columns features. In the case of time series, this is somewhat different. Firstly, the time series can have varying lengths. This can be a challenge in the case of handling these features because we don't have an equal distribution of features. Secondly, the information in this case is not structured or organized. The useful information can be localized in different parts of the series, whether it is in the beginning or at the end or somewhere in between. However, our aim is to extract predictive information in order to classify the time series data. This can be done through different algorithms which are based on different approaches (e.g., COTE or Shapelet). In this thesis, we will introduce our approach of extracting predictive information that is useful in order to classify the data, which is by using *aggregation*.

Aggregation is a process that will combine a collection of values (in this case, the time series datapoints) into one single value. This process is accomplished using *aggregation functions*. A few examples of aggregation functions are *sum, max* and *min*. Each function summarizes the underlying structure of the time series. For example, in the case of *sum*, you add up all the datapoints into one single value. The *min* function, for instance, will look at a range of numbers and will return the smallest number.

In our approach, aggregation is used as a form of feature extraction. The aggregation functions will be used to deal with the challenges of times series data as mentioned above: varying lengths and information that is localized at different areas within the time series. Each of these different aggregation functions will produce one feature that carries useful information about the time series. By using many forms of different aggregations functions, we will produce many features that describe the time series data in order for us to classify them as correctly as possible. By having a sufficient number of aggregation functions, the time series data is effectively transformed into the desired tabular format. Now that we have a tabular format, we are able to use a range of different classical machine learning algorithm in order to classify the time series data. In our case, we opted for the Random Forest classifier for the final classification phase.

We have used a range of aggregation functions in order to get the tabular format. We use the following collection of standard aggregation functions:

- Sum: Adding up all the values and returning one single value.

- Max: Finding the largest value in a range of total values.
- Min: Finding the smallest value in a range of total values.
- Mean: Calculating the average value by adding up all the values and dividing by the total numbers of values.
- Median: The middle values in a sorted sequence.
- Standard Deviation: Measures the spread of values around the mean. It describes the variability in a dataset.
- Variance: Measures how much the values vary from the mean.
- Quantile (0.1): The value below which 10% of the data falls.
- Quantile (0.25): The value below which 25% (first quartile) of the data falls. Understanding the lower ranges of the data.
- Quantile (0.3): The value below which 30% of the data falls.
- Quantile (0.4): The value below which 40% of the data falls.
- Quantile (0.5): The value below which 50% of the data falls, also known as the median.
- Quantile (0.6): The value below which 60% of the data falls.
- Quantile (0.75): The value below which 75% (third quartile) of the data falls. Understanding the upper ranges of the data.
- Quantile (0.8): The value below which 80% of the data falls.
- Quantile (0.9): The value below which 90% of the data falls.
- Quantile (0.95): The value below which 95% of the data falls.
- Quantile (0.98): The value below which 98% of the data falls.
- Quantile (0.99): The value below which 99% of the data falls.
- Quantile (1.0): The maximum value in the dataset.
- Range: The difference between the maximum and the minimum.
- Interquartile range: The difference between the Quantile (0.75) and Quantile (0.25). This results in the spread of the middle 50% of the data.
- Generalized mean: A single representative value that indicates the central tendency based on a set of numbers.

The generalized mean, also called the pth power mean, uses the following formula:

$$M_p = \left[\left(\frac{1}{N} \right) * \sum_{i=1}^{N} (x_i^p) \right]^{1/p}.$$

N is the total number of values in a time series. x_i are the individual points in the time series. p determines the type of generalized mean we want to calculate. In our case, this is a non-zero power of 2.

Suposse that we have a total of three positive real numbers $x = \{1,2,3\}$. Furthermore, the value of $p = 2$ and $n = 3$. This implies that we need to use the quadratic mean formula. Now we get the following:

$$M_2 = [\tfrac{1}{3} * (1^2 + 2^2 + 3^2)]^{1/2},$$
$$M_2 = [\tfrac{1}{3} * (1 + 4 + 9)]^{1/2},$$
$$M_2 = [\tfrac{1}{3} * 14]^{1/2},$$

$$M_2 = [\tfrac{14}{3}]^{1/2}.$$
$$M_2 \approx \sqrt{4.67} \approx 2.16.$$

The quadratic mean-value 2.16 tells us about the central tendency of the values in a dataset, especially focused on the magnitude of large values. A larger value of p would make the generalized mean tend even more to the largest value.

The aggregation functions that have been mentioned above form the total range of aggregation functions that will be used for the classification of the time series data. In the following section, we will introduce an alternative approach towards the time series data beside the whole-series aggregation. This alternative approach involves subsequences within the time series data.

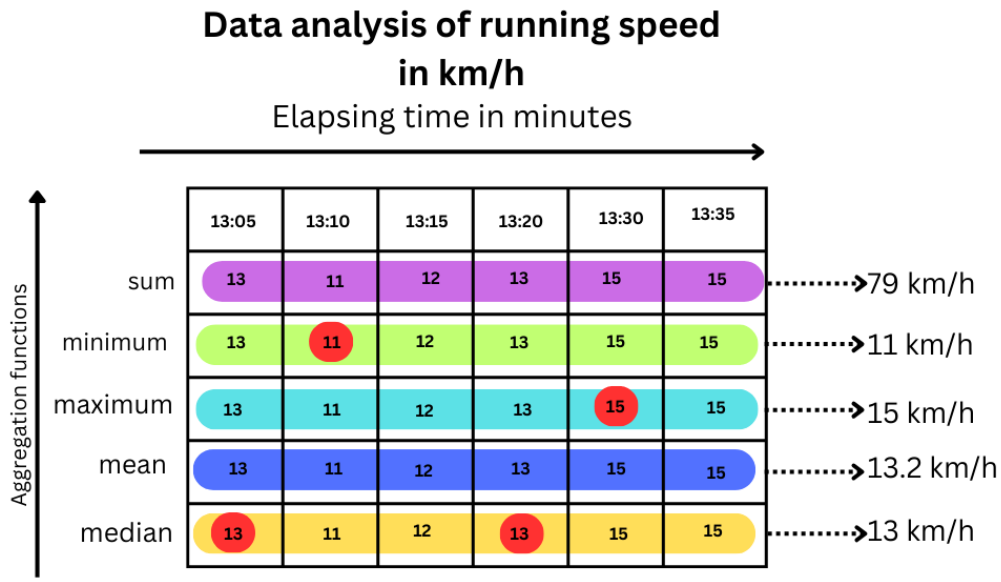

Figure 7: This is a schematic representation of the aggregation functions (leftside of the table) that have been applied to the "Data analysis of running speed" as feature exttraction for the time series. Furthermore, the results are represented as one single value (rightside of the table), which can be used as predictive information in order to clasify data.

3.1.1 Conditional Aggregation of Time Series Data

Following the application of different aggregation functions to the *entire* time series and their extracted features that carry useful information, our next step involves investigating the internal structure of the series. This is achieved by creating smaller chunks or subsequences. A subsequence in this case is an interval specified by a starting and end point where we have consecutive datapoints. The specification of subsequences depends on the following two input values: the total number of

subsequences n and the stride S.
It is important to note that we aim for an equal number of subsequences when we want to compare features from one time series against other time series. If we would simply use the same division that is responsible for the amount of subsequences for every time series, it will result in varying number of subsequences per time series. To mitigate this problem, we try to use a fixed standard of 20 subsequences per time series.
Moreover, the stride S determines the interval used to extract useful information by aggregation, based on a ratio of the subsequence length l. If $S = 1$, it means the subsequences are adjacent without overlap, and the features are extracted independently from each subsequence, moving on to the next with no overlap. This technique only considers relationships within each subsequence. However, to capture relationships that span across subsequences, the stride must be smaller than the subsequence length. For instance, with $S = 0.5$, consecutive subsequences overlap by 50%, allowing for more information to be captured across overlapping windows using aggregation functions. This approach can mitigate the risk of important time series segments being split between subsequences, while also generating more features.

To calculate the number of subsequences, we use the following formula:

$$l = \frac{N}{1 + S(n-1)},$$

where:

- S is the length of the stride.
- N is the length of a time series.
- n is the number of subsequences.
- l is the length of each subsequence.

This formula determines the length of each single subsequence, ensuring a consistent total number of n subsequences.

As an illustration, consider the following parameter settings: $n = 20$ and $S = 1$ (no overlap). Consider the time series of length $N = 200$, using the formula:

$$l = \frac{200}{1 + 1(20-1)} = \frac{200}{20} = 10$$

The value $l = 10$ corresponds to the length of each subsequence, ensuring a total of 20 subsequences with equal intervals when $S = 1$.

If we take this same concept again, where we modify the following parameter setting for $S = 0.5$, which introduces a 50% overlap between the subsequences, we get the following calculation:

$$l = \frac{200}{1 + 0.5(20-1)} = \frac{200}{9.5} \approx 21$$

The value $l = 21$ corresponds to the length of each subsequence, ensuring a total of 10 subsequences with equal intervals of 50% when $S = 0.5$.

Next, all the standard aggregation functions described in Section 3.1 will be used for each individual subsequence. This means that for each subsequence, we will gain an additional 23 different features. In the case of an equal distribution of the time series in 20 subsequences, this results in a total of 460 different features. Adding the 23 features derived from the entire time series results in 483 different features carrying potentially useful information that will contribute to the labelling of unseen observations.

In conclusion, we are able to use aggregation functions in order to extract many features that describe the time series. We can do this for the entire time series, the subsequences within the time series and subsequences with overlap.

3.2 Classification

To accurately label the unseen time series data, the involvement of a classifier is essential. As we have mentioned before, we use aggregation as a form of feature extraction. This is realized with the use of the aggregation function that produces these features. These features describe the time series. The aggregation is actually a pre-processing phase. It transforms the time series into a tabular format, which can then be used by a regular classification algorithm from the supervised machine learning literature to solve the classification problem.

Supervised machine learning is a type of machine learning where an algorithm is trained based on labelled time series data. Supervised machine learning, given a dataset, contains input features with an output or target variable. The goal of the algorithm is to learn from this relationship in order to predict the target variable for unseen observations. In this case, we opted for the Random Forest classifier (RFc) [6] as our final classification algorithm. The reason for this choice depends on some features of the Random Forest classifier. The Random Forest classifier can produce the Feature importance. This indicates what input features of the Random Forest have the highest contribution for making predictions for unseen observations. This makes it easier for us to adjust the Random Forest classifier for further optimization. Another feature is that we can handle large time series efficiently. These time series can contain a lot of outlier or noise. The Random Forest classifier is robust to outliers or noise, which makes it suitable for the time series classification. These are several reasons why we have used the Random Forest classifier.

The Random Forest classifier will not use all of the input features. It will use a subset of these input features. This will be done using a technique called bagging. The algorithm will create subsets based on original input features, which consist of randomly chosen subsets with replacement. This means that we create subsets that can contain repeated features. Next, each of the subsets will be used to create a single decision tree.The nodes in the decision tree will use the features from the subsets as splitting criteria to create the decision tree. In the end, we have a decision tree with leaf nodes, which contain the final predictions of the subset. By using multiple subsets, we can create multiple

decision trees. The Random Forest makes use of majority voting, as can be seen in Figure 8. Each decision tree will make a prediction for the unseen observation. All of these predictions will be counted, and the prediction with the majority of votes among all the trees will be selected as the final prediction. The Random Forest classifier is now able to make predictions for unseen observations.

These reasons also explain why we use the Random Forest classifier instead of a Decision Tree. A Decision Tree consists of only one tree and is deterministic, which means it can easily overfit the data. This results in a simpler, more interpretable tree regarding to the complex structure of a Random Forest classifier [23]. However, this simplicity comes at the cost of lower accuracy and robustness compared to the Random Forest classifier .

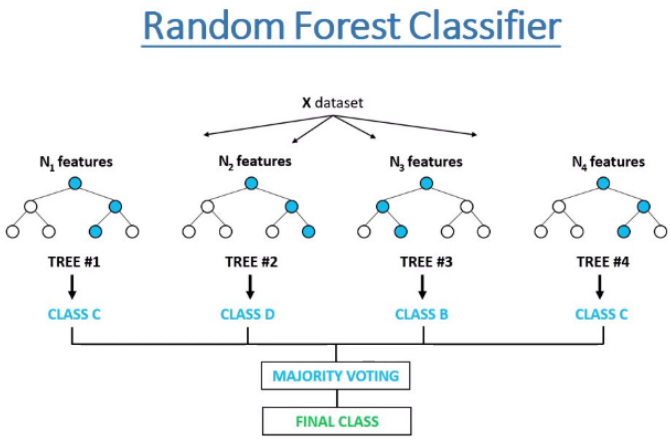

Figure 8: A schematic view of the input features of the Random Forest classifier and the final decision of the target value based on the collective decision. Source: [6]

.

4 Experiments

In this chapter, we begin by exploring the datasets used in our experiments, describing their characteristics and domains. These datasets serve as the foundation for our evaluations.
Furthermore, we will provide experimental setups for aggregation, asses the effectiveness of features responsible for the accuracy scores, and explore various parameter settings for conditional aggregation.
Additionally, we describe statistical tests such as the Friedman test and the Nemenyi test to provide comparative analysis among different approaches. These tests ensure reliable conclusions about the performance of the various algorithms.

4.1 Datasets

In this thesis, we will utilise a time series classification repository that provides us with a total collection of 85 datasets, which have specifically been designed for time series classification algorithms. This repository is maintained by the University of California (UCR) and the University of East Anglia (UEA). This is a group of time series researchers that consists of Chang Wei Tan, Anthony Bagnall, Christoph Bergmeir, Daniel Schmidt, Eamonn Keogh, François Petitjean, and Geoff Webb [2]. The main reason that we will use these specific datasets, is the fact that they have been used by the nine algorithms discussed in Section 2.2 to test their performances and hence serve as a benchmark for our experimentation and evaluation purposes.
These datasets have different characteristics. First of all, the datasets cover various real-world domains such as healthcare, food, movements, and more. Secondly, the datasets vary in terms of lengths, train/test set sizes, and the number of classes. Lastly, the datasets are categorised into different types, such as motion, sensors, audio, and so on. This makes the datasets suitable for experimentation purposes while covering different conditions to test the limits of a time series classification algorithm.
It is important to note that certain datasets perform better with specific algorithms. Thus, an algorithm that performs highly accurately across a handful of datasets with certain settings does not imply that it is overall the best performing algorithm. The performance is determined by different factors, such as the dataset types or lengths, that play a major role in determining the algorithm's efficiency [5].
To ensure that the evaluation of the Random Forest classifier is reliable and applicable, we consider the methodology for dividing the datasets into training and test sets. It is important to choose a well-balanced division in order to fairly test the performance of the supervised learning algorithm. In our implementation, the dataset is divided into a default setting of 75% as a subset for the training set and 25% as a subset for the test set [16].

4.2 Fine tuning accuracy score

We will present a table showing the accuracy score based on the nine benchmark algorithms for each dataset discussed in Section 4.1. The *accuracy score* is a metric to evaluate the performance of a classification model. It represents the correct fraction of predictions made by a classifier. Additionally, we will include the aggregation algorithm along with its corresponding accuracy score. This will be conducted by a total of 3 experiments:

- Ag1: Conditional aggregation for subsets of the entire time series *without* overlap.
- Ag2: Conditional aggregation for subsets of the entire time series *with* overlap.
- Ag3: Aggregation on the entire time series.

All experiments will use the standard aggregation functions in order to extract features from the time series, as mentioned in Section 3.1.

First of all, we will use conditional aggregation for the time series **without overlap**. This method will involve the entire time series with a form of conditional aggregation, as we discussed and described in Section 3.1.1. In our experimental setup, the window count will be equal to 20. The stride will be set to 1, which means that we will move with the exact same window size for each of the windows.

Secondly, we will use conditional aggregation for the time series **with overlap**. In this experimental setup, we will use the entire time series with a form of conditional aggregation, including different parameter settings. The window count will be equal to 10, implying that each window size will cover a larger range than in the previous experiment. The stride will be set to 0.5. This feature extraction approach will result in the additional information being captured from the adjacent windows.

Thirdly, we will perform aggregation on the entire time series. We will treat the entire time series as one single entity that is used to extract features by means of aggregation.

All experiments will eventually use their required features as the input for the Random Forest classifier. These scores are represented in Table 2:

	Datasets	COTE	ST	BOSS	EE	DTW_f	TSF	TSBF	LPS	MSM	Ag1	Ag2	Ag3
1	Adiac	**0.810**	0.768	0.749	0.665	0.605	0.707	0.727	0.765	0.636	0.790	0.795	0.710
2	ArrowHead	0.877	0.851	0.875	0.860	0.776	0.789	0.801	0.806	0.815	**0.906**	0.898	0.683
3	Beef	**0.764**	0.736	0.615	0.532	0.546	0.648	0.554	0.520	0.474	0.600	0.680	0.373
4	BeetleFly	0.921	0.875	**0.949**	0.823	0.853	0.842	0.799	0.893	0.794	0.740	0.800	0.900
5	BirdChicken	0.941	0.927	0.984	0.848	0.865	0.839	0.902	0.854	0.866	**1.000**	0.900	0.860
6	Car	0.899	**0.902**	0.855	0.799	0.851	0.758	0.795	0.836	0.841	0.760	0.880	0.680
7	CBF	0.998	0.986	0.998	0.993	0.979	0.958	0.977	0.984	0.972	**1.000**	0.996	0.661
8	ChlorineConcentration	0.736	0.682	0.660	0.659	0.658	0.719	0.683	0.642	0.626	**0.988**	0.968	0.765
9	CinCECGtorso	0.983	0.918	0.900	0.946	0.714	0.974	0.716	0.743	0.935	**1.000**	0.997	0.881
10	Coffee	**1.000**	0.995	0.989	0.989	0.973	0.989	0.982	0.950	0.945	**1.000**	0.986	0.771
11	Computers	0.770	0.785	0.802	0.732	0.659	0.768	0.765	0.726	0.713	**0.854**	0.792	0.786
12	CricketX	**0.814**	0.777	0.764	0.801	0.769	0.691	0.731	0.696	0.778	0.736	0.751	0.541
13	CricketY	**0.815**	0.762	0.749	0.794	0.756	0.688	0.728	0.706	0.760	0.722	0.793	0.493
14	CricketZ	**0.827**	0.798	0.776	0.804	0.785	0.707	0.738	0.714	0.779	0.737	0.747	0.571
15	DiatomSizeReduction	0.925	0.911	0.939	0.946	0.942	0.941	0.890	0.915	0.939	**1.000**	0.988	0.975
16	DistalPhalanxOAG	0.805	**0.829**	0.815	0.768	0.796	0.809	0.816	0.767	0.756	0.821	0.807	0.809
17	DistalPhalanxOC	0.821	0.819	0.814	0.768	0.760	0.813	0.812	0.742	0.754	0.815	**0.854**	0.798
18	DistalPhalanxTW	0.693	0.690	0.673	0.654	0.658	0.686	0.690	0.618	0.618	0.714	**0.763**	0.704
19	Earthquakes	0.747	0.737	0.746	0.735	0.747	0.747	0.747	0.668	0.695	**0.829**	0.760	0.781
20	ECG200	0.873	0.840	**0.890**	0.881	0.819	0.868	0.847	0.807	0.877	0.836	0.836	0.760
21	ECG5000	0.946	0.943	0.940	0.939	0.940	0.944	0.938	0.917	0.930	**0.953**	0.946	0.927
22	ECGFiveDays	0.986	0.955	0.983	0.847	0.907	0.922	0.849	0.840	0.879	**0.993**	0.991	0.982
23	ElectricDevices	0.883	**0.895**	0.800	0.831	0.874	0.804	0.808	0.853	0.825	0.856	0.856	0.732
24	FaceAll	**0.990**	0.968	0.974	0.976	0.963	0.949	0.942	0.962	0.986	0.958	0.965	0.659
25	FaceFour	0.850	0.794	0.996	0.879	0.909	0.891	0.862	0.889	0.920	**1.000**	**1.000**	0.629
26	FacesUCR	0.967	0.909	0.951	0.948	0.889	0.897	0.849	0.910	**0.970**	0.961	0.969	0.671
27	Fiftywords	0.801	0.713	0.702	**0.821**	0.748	0.728	0.744	0.776	0.817	0.707	0.705	0.327
28	Fish	0.962	**0.974**	0.969	0.913	0.931	0.807	0.913	0.912	0.897	0.839	0.839	0.755
29	FordA	0.955	**0.965**	0.920	0.751	0.884	0.816	0.831	0.869	0.725	0.694	0.668	0.579
30	FordB	**0.929**	0.915	0.911	0.757	0.843	0.790	0.751	0.852	0.730	0.731	0.710	0.605
31	GunPoint	0.992	0.999	0.994	0.974	0.964	0.962	0.965	0.972	0.948	**1.000**	**1.000**	0.960
32	Ham	0.805	0.808	**0.836**	0.763	0.795	0.795	0.711	0.685	0.745	0.793	0.741	0.700
33	HandOutlines	0.894	**0.924**	0.903	0.880	0.915	0.909	0.879	0.868	0.864	0.913	0.916	0.802
34	Haptics	0.517	0.512	0.459	0.451	0.464	0.467	0.463	0.415	0.444	0.517	**0.529**	0.403
35	Herring	0.632	**0.653**	0.605	0.566	0.609	0.606	0.590	0.549	0.559	0.456	0.506	0.619
36	InlineSkate	0.526	0.393	0.503	0.476	0.382	0.379	0.377	0.449	0.455	**0.578**	0.561	0.561
37	InsectWingbeatSound	0.639	0.617	0.510	0.581	0.602	0.613	0.616	0.519	0.570	0.672	**0.733**	0.269
38	ItalyPowerDemand	0.970	0.953	0.866	0.951	0.948	0.958	0.926	0.914	0.936	**0.974**	0.985	0.866
39	LargeKitchenAppliances	0.900	**0.933**	0.837	0.816	0.823	0.644	0.551	0.680	0.749	0.740	0.755	0.805
40	Lightning2	0.785	0.659	0.810	0.835	0.710	0.757	0.760	0.757	0.792	**0.877**	0.677	0.697
41	Lightning7	**0.799**	0.724	0.666	0.763	0.671	0.723	0.680	0.631	0.713	0.694	0.767	0.528
42	Mallat	0.974	0.972	0.949	0.961	0.929	0.937	0.951	0.908	0.918	0.994	**1.000**	0.927
43	Meat	0.981	0.966	0.980	0.978	0.983	0.978	0.983	0.968	0.977	0.967	**1.000**	0.987

Table 1: The benchmark algorithms including the three experimental setups. Each dataset is represented by the accuracy score across all the different algorithms. This table presents the first half of the datasets (1-43). The best performing algorithm per dataset is in bold.

	Datasets	COTE	ST	BOSS	EE	DTW_f	TSF	TSBF	LPS	MSM	Ag1	Ag2	Ag3
44	MedicalImages	0.785	0.691	0.715	0.761	0.701	0.757	0.701	0.710	0.757	0.785	**0.831**	0.700
45	MiddlePhalanxOAG	0.801	**0.815**	0.808	0.782	0.798	0.794	0.800	0.770	0.751	0.722	0.721	0.722
46	MiddlePhalanxOC	0.722	0.694	0.666	0.609	0.581	0.676	0.673	0.597	0.560	**0.838**	0.827	0.744
47	MiddlePhalanxTW	0.587	0.579	0.537	0.525	0.519	0.577	0.568	0.503	0.499	0.571	**0.672**	0.637
48	MoteStrain	0.902	0.882	0.846	0.875	0.891	0.874	0.886	0.917	0.880	0.953	**0.974**	0.855
49	NonInvFetalECGThorax1	0.929	**0.947**	0.841	0.849	0.877	0.880	0.842	0.807	0.818	0.889	0.898	0.788
50	NonInvFetalECGThorax2	0.946	**0.954**	0.904	0.914	0.898	0.914	0.862	0.826	0.894	0.932	0.915	0.845
51	OliveOil	0.901	0.881	0.870	0.879	0.864	0.883	0.864	0.892	0.872	**1.000**	0.800	0.867
52	OSULeaf	0.949	0.934	**0.967**	0.812	0.809	0.637	0.678	0.763	0.787	0.672	0.690	0.569
53	PhalangesOutlinesCorrect	0.783	0.794	0.821	0.780	0.793	0.804	0.825	0.790	0.760	0.837	**0.850**	0.791
54	Phoneme	**0.362**	0.329	0.256	0.299	0.220	0.211	0.278	0.245	0.275	0.345	0.340	0.227
55	Plane	**1.000**	**1.000**	0.998	**1.000**	0.996	0.994	0.993	**1.000**	0.999	**1.000**	**1.000**	0.981
56	ProximalPhalanxOAG	0.871	0.881	0.867	0.839	0.829	0.847	0.861	0.851	0.806	**0.882**	0.834	0.846
57	ProximalPhalanxOC	0.848	0.841	0.819	0.805	0.824	0.846	0.842	0.800	0.769	**0.871**	0.827	0.845
58	ProximalPhalanxTW	0.815	0.803	0.773	0.759	0.774	0.808	0.798	0.722	0.729	**0.837**	0.818	0.726
59	RefrigerationDevices	0.742	0.761	**0.785**	0.676	0.656	0.615	0.638	0.675	0.704	0.697	0.694	0.694
60	ScreenType	0.651	**0.676**	0.586	0.554	0.499	0.573	0.538	0.506	0.493	0.589	0.579	0.556
61	ShapeletSim	0.964	0.934	**1.000**	0.827	0.888	0.510	0.913	0.874	0.850	0.456	0.464	0.528
62	ShapesAll	**0.911**	0.854	0.909	0.886	0.796	0.800	0.853	0.885	0.875	0.793	0.816	0.699
63	SmallKitchenAppliances	0.788	0.802	0.750	0.703	0.753	0.813	0.674	0.724	0.717	0.773	**0.828**	0.781
64	SonyAIBORobotSurface1	0.899	0.888	0.897	0.794	0.884	0.845	0.839	0.842	0.764	0.967	**0.987**	0.927
65	SonyAIBORobotSurface2	0.960	0.924	0.888	0.870	0.859	0.856	0.825	0.851	0.877	0.936	0.943	0.802
66	StarlightCurves	**0.980**	0.977	0.978	0.941	0.960	0.969	0.978	0.968	0.882	0.971	0.969	0.957
67	Strawberry	0.963	0.968	0.970	0.959	0.970	0.963	0.968	0.963	0.958	0.963	**0.976**	0.951
68	SwedishLeaf	**0.967**	0.939	0.918	0.916	0.885	0.892	0.908	0.926	0.887	0.898	0.895	0.769
69	Symbols	0.953	0.862	0.961	0.957	0.930	0.888	0.944	0.960	0.952	**0.991**	0.984	0.791
70	SyntheticControl	**0.999**	0.987	0.968	0.994	0.986	0.990	0.987	0.972	0.982	0.973	0.979	0.425
71	ToeSegmentation1	0.934	**0.954**	0.929	0.788	0.922	0.661	0.858	0.841	0.821	0.851	0.925	0.872
72	ToeSegmentation2	0.951	0.947	**0.960**	0.907	0.904	0.782	0.886	0.926	0.895	0.857	0.867	0.905
73	Trace	**1.000**	**1.000**	**1.000**	0.996	0.997	0.998	0.981	0.966	0.956	0.980	0.980	**1.000**
74	TwoLeadECG	0.983	0.984	0.985	0.958	0.958	0.842	0.910	0.928	0.941	**1.000**	0.993	0.992
75	TwoPatterns	**1.000**	0.952	0.991	**1.000**	**1.000**	0.991	0.974	0.967	0.999	0.984	0.996	0.401
76	UWaveGestureLibraryX	0.831	0.806	0.753	0.805	0.806	0.806	0.834	0.819	0.775	0.832	**0.841**	0.452
77	UWaveGestureLibraryY	0.766	0.737	0.661	0.731	0.717	0.727	0.746	0.753	0.690	0.767	**0.768**	0.420
78	UWaveGestureLibraryZ	0.760	0.747	0.695	0.726	0.736	0.741	0.776	0.766	0.701	0.778	**0.796**	0.469
79	UWaveGestureLibraryAll	0.965	0.942	0.944	0.968	0.963	0.962	0.944	0.968	0.960	**0.969**	**0.969**	0.390
80	Wafer	0.999	**1.000**	0.999	0.997	0.996	0.997	0.996	0.995	0.996	0.999	**1.000**	0.990
81	Wine	0.904	**0.926**	0.912	0.887	0.892	0.881	0.879	0.884	0.884	0.893	0.893	0.921
82	WordSynonyms	0.748	0.582	0.659	**0.778**	0.674	0.643	0.669	0.728	0.773	0.721	0.728	0.350
83	Worms	0.725	0.719	**0.735**	0.644	0.673	0.628	0.668	0.642	0.616	0.646	0.594	0.542
84	WormsTwoClass	0.785	0.779	**0.810**	0.717	0.730	0.685	0.755	0.743	0.712	0.615	0.745	0.680
85	Yoga	0.898	0.823	0.910	0.885	0.863	0.867	0.835	0.874	0.888	**0.960**	0.953	0.851

Table 2: The benchmark algorithms including the three experimental setups. Each dataset is represented by the accuracy score across all the different algorithms. This table presents the second half of the datasets (44-85). The best performing algorithm per dataset is in bold.

4.3 Aggregation functions and their impact on classification

We will implement a total of 23 different aggregation functions, using a variety of diverse features in order to maximise the extraction of useful information. Each feature will carry unique characteristics that potentially contribute to the accurate labelling of unseen data. An interesting aspect to consider will be the contribution of each individual aggregation function in the final classification task.

To understand the impact of each individual aggregation function on the performance of a classifier, we will use a measurement tool within the Random Forest classifier called *feature importance*.

This function will assign a score to the input features that will be used by the Random Forest classifier during the construction of the classification model. These scores will indicate the level of contribution of each feature in the final classification process. These scores will offer valuable information and a better understanding of both the algorithm and the dataset we are examining.

Firstly, feature importance will show us the relevance of the functions in the predictive model. Secondly, it will provide insights into the background information used by the Random Forest classifier in order to accurately assign the target values to unseen observations. Moreover, feature importance will help us to potentially enhance the performance of the classifier. This will be achieved by identifying and eliminating the least relevant features, allowing us to focus more on the relevant ones.

Consider Figure 9. Here we have the dataset Gunpoint and its accuracy score displayed in the case of Aggregation without overlap, represented in Table 2. After applying the Random Forest classifier, we obtain an accuracy score of approximately 0.896. To determine which aggregation functions were responsible for this results, we used the *aggregatefunction* x_z , where x represents the total window count and z the specific window number, to display the top 10 most influential aggregation functions.

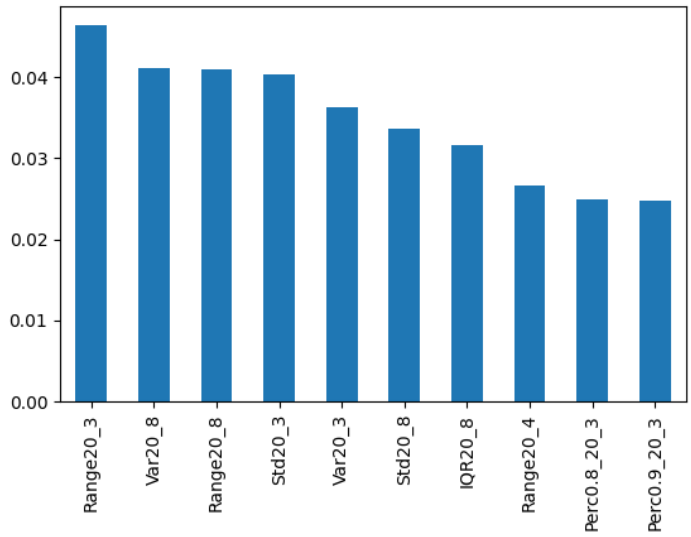

Figure 9: top 10 feature importance values of the Gunpoint dataset. Source: [5]

.

In Figure 9, we see a descending bar graph illustrating the impact of aggregation functions, with the most impactfull on the left side and the least impactfull on the right. The x-axis presents the names of the aggregation functions, and the y-axis indicates the level of influence for each aggregation function.

4.4 Running multiple subset instances

After examining different approaches for obtaining and enhancing the accuracy, the next step will be to investigate the influence of the parameters when using conditional aggregation and window counts. Previously, we used a benchmark where the subsequence count $n = 20$. The stride was set to 1.0, indicating no overlap of subsequences. In this case, the aggregation experiment referred to as Ag1 was applied. After this analysis, we compared this accuracy score to a run of an aggregation algorithm with overlap of the subsequences, adjusting the subsequence count $n = 10$ and the stride to 0.5, which corresponds to Ag2. Ag3, which aggregates the entire time series, is excluded here as it does not use conditional aggregation. This section focuses on methods using conditional aggregation to assess parameter impacts.

Increasing the window count might capture more relevant segments of the series, but it also reduces the length of each subsequence. Adjusting the stride impacts how much overlap occurs between subsequences, which can increase the number of features and provide more information. However, finding the optimal settings for these parameters is non-trivial. To determine if and when the accuracy decreases, we will experiment with different combinations of window counts and stride values, relative to benchmark parameters.

We will analyse settings corresponding to the following window count and stride:

- Ag1: window count = 20, stride = 1
- Ag4: window count = 5, stride = 1
- Ag6: window count = 10, stride = 1
- Ag8: window count = 30, stride = 1
- Ag10: window count = 40, stride = 1

Subsequently, we will also test the following window count and stride:

- Ag2: window count = 10, stride = 0.5
- Ag5: window count = 2.5, stride 0.5
- Ag7: window count = 5, stride 0.5
- Ag9: window count = 15, stride 0.5
- Ag11: window count = 20, stride 0.5

Using these parameter settings, we obtained the following results:

	Datasets	Ag1	Ag2	Ag4	Ag5	Ag6	Ag7	Ag8	Ag9	Ag10	Ag11
1	Adiac	0.785	0.791	0.752	0.794	0.755	0.793	**0.802**	0.757	0.745	0.799
2	ArrowHead	0.902	0.902	0.808	0.785	0.872	0.774	0.887	**0.943**	0.849	0.808
3	Beef	**0.760**	0.613	0.453	0.587	0.333	0.587	0.733	0.573	0.667	0.720
4	BeetleFly	0.760	0.500	0.600	**0.940**	0.880	0.880	0.900	0.900	0.700	0.800
5	BirdChicken	0.800	**1.000**	0.700	0.700	**1.000**	**1.000**	**1.000**	0.700	0.980	**1.000**
6	Car	0.740	0.833	0.773	**0.900**	0.653	0.767	0.727	0.620	0.787	0.813
7	CBF	**1.000**	**1.000**	0.997	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
8	ChlorineConcentration	0.994	0.978	0.933	0.904	0.950	0.953	0.989	0.990	**0.999**	0.997
9	CinCECGtorso	**1.000**	**1.000**	0.989	0.983	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
10	Coffee	**1.000**	0.943	**1.000**	0.643	**1.000**	**1.000**	**1.000**	**1.000**	0.986	**1.000**
11	Computers	0.784	0.803	0.797	0.768	0.763	0.794	**0.814**	0.810	0.790	0.797
12	CricketX	0.761	**0.781**	0.720	0.695	0.688	0.749	0.776	0.756	0.728	0.751
13	CricketY	0.721	0.757	0.707	0.714	0.732	0.741	0.743	**0.760**	0.734	0.734
14	CricketZ	0.758	0.790	0.751	0.635	0.764	**0.826**	0.728	0.758	0.748	0.755
15	DiatomSizeReduction	**1.000**	0.988	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.990
16	DistalPhalanxOAG	0.844	0.837	0.775	**0.868**	**0.868**	0.834	0.858	0.822	0.824	0.816
17	DistalPhalanxOC	0.797	**0.871**	**0.871**	0.860	0.834	0.821	0.852	0.847	0.835	0.864
18	DistalPhalanxTW	0.799	0.759	**0.806**	0.744	0.770	0.721	0.779	0.764	0.770	0.729
19	Earthquakes	0.776	0.784	0.802	0.755	0.771	0.790	**0.805**	0.793	0.802	0.791
20	ECG200	0.880	0.828	**0.912**	0.884	0.844	0.900	0.900	0.908	0.748	0.876
21	ECG5000	0.951	0.943	0.948	0.946	0.952	0.944	0.954	**0.960**	0.952	0.955
22	ECGFiveDays	0.994	0.995	0.995	0.996	**1.000**	0.995	0.989	0.995	0.977	0.995
23	ElectricDevices	0.850	0.857	0.847	0.846	0.848	**0.859**	0.837	0.845	0.827	**0.859**
24	FaceAll	0.952	0.966	0.929	0.912	0.950	0.947	0.955	**0.968**	0.940	0.967
25	FaceFour	0.929	**1.000**	**1.000**	0.936	**1.000**	**1.000**	**1.000**	0.993	**1.000**	**1.000**
26	FacesUCR	0.963	0.965	0.934	0.908	0.970	0.957	0.953	**0.972**	0.945	0.970
27	Fiftywords	0.741	0.722	0.678	0.622	0.666	0.719	0.698	**0.763**	0.705	0.677
28	Fish	0.732	**0.930**	0.866	0.848	0.805	0.834	0.891	0.859	0.802	0.884
29	FordA	0.698	0.696	0.658	0.622	0.670	0.658	0.750	0.713	**0.763**	0.698
30	FordB	**0.743**	0.711	0.672	0.669	0.687	0.677	0.725	0.737	0.722	0.740
31	GunPoint	**1.000**	**1.000**	0.920	**1.000**	0.988	0.964	0.996	0.996	**1.000**	0.960
32	Ham	0.833	0.793	0.844	0.656	0.730	0.800	**0.889**	0.770	0.844	0.837
33	HandOutlines	0.906	**0.924**	0.875	0.885	0.907	0.917	0.889	0.913	0.892	0.921
34	Haptics	0.467	0.503	0.509	0.478	0.448	0.522	0.453	**0.550**	0.438	0.533
35	Herring	0.625	0.556	0.594	**0.788**	0.681	0.644	0.525	0.500	0.525	0.675

Table 3: Aggregation algorithms including different parameter settings. Each dataset is represented by the accuracy across all the different algorithms. The best performing algorithm per dataset is in bold.

4.5 Friedman test

The Friedman test, invented by Milton Friedman, is a non-parametric statistical test that will be suitable for measuring the differences across a minimum of three or more occasions that will originate from a related sample. As a non-parametric statistical test, the Friedman test doesn't rely on assumptions about the underlying distribution. However, the Friedman test must meet the following requirements:

1. The data needs to be in ordinal or continuous form.

2. The data comes from a single sample, and the measurements must be from at least three or more different occasions.

3. The sample must be created by using a random sampling method.

4. The variables in the occasions must be mutually independent.

Ordinal data indicates attributes or variables that represent a ranking or a natural order. An example of the ordinal structure could be a Likert scale (e.g., strongly agree, agree, disagree, strongly disagree) or a ranking level (e.g., 1 as best, 5 as worst). Continuous variables represent numerical values within a continuous scale. Examples of these are height or weight measurements. Mutual independence of variables implies that the variables within these occasions do not influence each other's outcomes. Therefore, the output of one variable will not affect the output of another variable [20].

First of all, we will need a tabular format with different occasions derived from a related sample. In our experimental setup, the occasions will correspond to the algorithms and the related samples will be represented by the datasets. The rows will represent the different datasets. The columns will represent the benchmark algorithms, including our aggregation algorithm. To conduct a Friedman test, all of the columns will need to be of equal length.

After creating the tabular format, we will rank the variables across the rows. The ranking will be based on the highest accuracy score being ranked 1, the second highest 2, and so forth. In the case of a tie, we will rank the tied variables with the same numerical value based on the first variable. However, the counting will continue based on the total number of variables in the row. For example, if the first two variables tie and a row contains 10 variables, these two variables will be ranked 1 and the counting will continue from 3.

	Datasets	COTE	ST	BOSS	EE	DTW_F	TSF	TSBF	LPS	MSM	Ag1	Ag2	Ag3
1	Adiac	**1**	4	6	10	12	9	7	5	11	3	2	8
2	ArrowHead	3	6	4	5	11	10	9	8	7	**1**	2	12
3	Beef	**1**	2	5	9	8	4	7	10	11	6	3	12
4	BeetleFLy	2	5	**1**	8	6	7	10	4	11	12	9	3
5	BirdChicken	3	4	2	11	8	12	5	10	7	**1**	6	9
6	Car	2	**1**	4	8	5	11	9	7	6	10	3	12
7	CBF	2	6	2	5	8	11	9	7	10	**1**	4	12
8	ChlorineConcentration	4	7	8	9	10	5	6	11	12	**1**	2	3
9	CinCECGtorso	3	7	8	5	12	4	11	10	6	**1**	2	9
10	Coffee	**1**	3	4	4	9	4	8	10	11	**1**	7	12
11	Computers	6	5	2	9	12	7	8	10	11	**1**	3	4
12	CricketX	**1**	4	6	2	5	11	9	10	3	8	7	12
13	CricketY	**1**	4	7	2	6	11	8	10	5	9	3	12
14	CricketZ	**1**	3	6	2	4	11	8	10	5	9	7	12
15	DiatomSizeReduction	9	11	7	4	5	6	12	10	7	**1**	2	3
16	DistalPhalanxOAG	8	**1**	4	10	9	5	3	11	12	2	7	5
17	DistalPhalanxOC	2	3	5	9	10	6	7	12	11	4	**1**	8
18	DistalPhalanxTW	4	5	8	10	9	7	5	11	11	2	**1**	3
19	Earthquakes	4	9	8	10	4	4	4	12	11	**1**	3	2
20	ECG200	4	7	**1**	2	10	5	6	11	3	8	8	12

Table 4: The Ranked table based on the accuracy score.

In Table 4 we can see an example of a part of the datasets that have been ranked using the benchmark algorithms, including the experiments mentioned in Section 4.2. The best-performing algorithms are in bold. After ranking the entire table, these rankings will be summated column-wise for each algorithm. Next, we will apply the Friedman test using the following formula [21]:

$$F_R = \frac{12}{nk(k+1)} \sum R_i^2 - 3n(k+1)$$

- n will represent the number of algorithms.
- k will represent the number of datasets.
- $\sum R^2$ will represent the total sum of the ranks for each algorithm.

Once we obtain the result of the Friedman test, we will calculate the degrees of freedom (df) by the following formula: $df = k - 1$. Using the Friedman test result (F_r) and the degrees of freedom (df), we can derive the p-value from the chi-square distribution table.

4.6 Nemenyi Test

The Friedman test will determine if there is a significant difference among several algorithms based on their performance on the dataset. However, it will not specify which algorithm differs significantly from each other. To identify the significant differences between algorithms, we will use a post-hoc analysis. Once the Friedman test indicates a significant difference, we will apply the *Nemenyi test*. The Nemenyi test is a statistical test that will perform pairwise comparisons among the algorithms to identify which algorithms differ significantly from each other.
The input for the Nemenyi test will consist of the ranks of the algorithms, determined by the

performance on the datasets. This will be the same data used for the Friedman test. Additionally, we will calculate the average rank for each algorithm across all datasets. Next, we will use the following formula to calculate the Q-values:

$$Q = \frac{(R_i - R_j)}{\sqrt{\frac{k(k+1)}{6N}}}$$

- $R_i - R_j$ represent the absolute difference between the mean ranks of the pairwise algorithms i and j
- k will represent the number of algorithms.
- N will represent the number of datasets.

Once the Q-values are obtained using this formula, they will be multiplied by $\sqrt{2}$ [10]. The p-value will be obtained by using the studentized range distribution [24]. The p-values will be used to determine whether there is statistical significance between a pairwise comparison of the algorithms.

5 Results

In this chapter, we will explain the results of the experiments described in Chapter 4. We will provide a detailed explanation of the results and the insights gained regarding the benchmark algorithms with an emphasis on the aggregation algorithms. Furthermore, we will introduce a metric called the critical difference, which evaluates the performance of a single algorithm compared to all other algorithms, after conducting the statistical tests.

5.1 Performance score

In Section 4.2, we presented the benchmark table, which includes our three experimental setups. Each algorithm was tested on the 85 datasets, represented by Table 2, resulting in an accuracy. The accuracy is a rational number between 0 and 1. It describes the success rate of an algorithm in predicting the labelling of unseen data. An accuracy of 1 indicates perfect labelling, while an accuracy of 0 indicates completely incorrect labelling. When examining a dataset, we highlight the highest accuracy, and therefore the best-performing algorithm, in bold. If there are multiple best-performing algorithms, these are also in bold.

Furthermore, we use four different mathematical measures to extract useful information describing the performance of all the algorithms. These measures are:

- Average Accuracy: The sums of all the accuracies per dataset divided by the total number of datasets.
- Total Wins: The number of times an algorithm achieves the highest accuracy score, including ties, across all the datasets.
- Average Rank: The average ranking of each algorithm across all the datasets.
- Median Rank: The middle value of the ranks, indicating the central tendency of an algorithm's performance.

Using these measurements, we obtain the following table:

	COTE	ST	BOSS	EE	DTW_f	TSF	TSBF	LPS	MSM	Ag1	Ag2	Ag3
Avg Accuracy	**0.858**	0.838	0.833	0.812	0.805	0.796	0.799	0.795	0.796	0.832	0.835	0.715
Wins	18	17	10	4	1	0	0	1	1	**25**	22	1
Avg Rank	**3.059**	4.718	5.400	6.671	7.271	7.447	7.553	8.271	8.365	4.635	4.459	9.071
Median Rank	**3**	4	5	7	7	7	8	9	9	**3**	**3**	11

Table 5: Performance metrics for different algorithms

In Table 5, we see that the best-performing algorithm, based on the the highest average accuracy is COTE with an average accuracy of 0.858, followed by ST with 0.838, Ag2 (windows with overlap) with 0.835 and Ag1 (windows without overlap) with 0.832, completing the top four. The worst-performing algorithm is Ag3 (only entire dataset), with an average accuracy of 0.715.

Next, we look at the total wins per algorithm. Ag1 has the most wins (25), followed by Ag2 (22) and COTE (18). The algorithms with the least number of wins are DTW_f (1), TSF (0), TSFB (0), LPS (1), MSM (1), and Ag3 (1).

Low average ranks indicate better performance. The top three best-performing algorithms are COTE (3.059), followed by Ag2 (4.459) and Ag1 (4.635). The worst-performing algorithm is Ag3 with an average rank of 9.071.

Finally, the median rank represents the middle value of the rankings, where a lower score indicates a well-performing algorithm. COTE, Ag1, and Ag2 all have a median rank of 3. The worst median rank is assigned to Ag3: 11.

Overall, using these different measurements, we can clearly see that COTE, Ag1, and Ag2 are among the few best-performing algorithms for these datasets. This means that the COTE algorithm and the aggregation algorithms, both with and without overlap, perform well across the different datasets. However, the aggregation on the entire dataset (Ag3) is, in most cases, one of the worst-performing algorithm.

5.2 Friedman and Nemenyi test results

In Sections 4.5 and 4.6, we presented two statistical test: the Friedman test and the Nemenyi test. To interpret the results obtained from these statistical tests, we use a *significance level* α before conducting the tests. The significance level is the probability of rejecting the null hypothesis when the null hypothesis is true [13]. For the statistical tests, we use a significance level of $\alpha = 0.05$.

The Friedman test revolves around two hypotheses to interpret the p-value:

- **If** $p > \alpha$: Fail to reject the Null hypothesis (H_0). This suggests that there is not enough evidence to conclude a significant effect or difference.
- **If** $p \leq \alpha$: Reject the Alternative hypothesis (H_1). This indicates that the observed data is significantly different from what the null hypothesis predicts, suggesting there is an effect or a difference.

Upon conducting the Friedman test, based on results of Table 5, we obtained the following metrics:

Friedman statistic	p-value
254.2	3.65×10^{-48}

Table 6: Friedman chi-square test result.

We see a very large Friedman statistic of 254.2. Given the degrees of freedom and the chi-square table, this results in a p-value of 3.65×10^{-48}. Since $p \leq \alpha$, we reject the the Alternative hypothesis, suggesting that there is a significant difference among the benchmark algorithms and Ag1, Ag2 and Ag3.

To make a pairwise comparisons between algorithms, we use a post-hoc test called the Nemenyi test. Upon conducting the Nemenyi test, we obtain the following matrix of p-values:

	COTE	ST	BOSS	EE	DTW_f	TSF	TSBF	LPS	MSM	Ag1	Ag2	Ag3
COTE	1	0.139	**0.002**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	0.164	0.348	**0.001**
ST	0.139	1	0.9	**0.02**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	0.9	0.9	**0.001**
BOSS	**0.002**	0.9	1	0.481	**0.034**	**0.01**	**0.006**	**0.001**	**0.001**	0.9	0.882	**0.001**
EE	**0.001**	**0.021**	0.481	1	0.9	0.9	0.9	0.151	0.119	**0.016**	**0.004**	**0.001**
DTW_f	**0.001**	**0.001**	**0.034**	0.9	1	0.9	0.9	0.809	0.749	**0.001**	**0.001**	0.068
TSF	**0.001**	**0.001**	**0.01**	0.9	0.9	1	0.9	0.9	0.9	**0.001**	**0.001**	0.172
TSBF	**0.001**	**0.001**	**0.006**	0.9	0.9	0.9	1	0.9	0.9	**0.001**	**0.001**	0.236
LPS	**0.001**	**0.001**	**0.001**	0.151	0.809	0.9	0.9	1	0.9	**0.001**	**0.001**	0.9
MSM	**0.001**	**0.001**	**0.001**	0.119	0.749	0.9	0.9	0.9	1	**0.001**	**0.001**	0.9
Ag1	0.164	0.9	0.9	**0.016**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	1	0.9	**0.001**
Ag2	0.348	0.9	0.882	**0.004**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	0.9	1	**0.001**
Ag3	**0.001**	**0.001**	**0.001**	**0.001**	0.068	0.172	0.236	0.9	0.9	**0.001**	**0.001**	1

Table 7: Nemenyi test results represented as the p-values.

In Table 7, we see a table representing the p-values for pairwise algorithm comparisons. The diagonal line across the table always equals 1.0, as this is a pairwise comparison against itself. In cases where $p \leq \alpha$, the pairwise comparison is in bold. This implies that we reject the Alternative hypothesis, suggesting a significant difference between the pairwise comparisons. We conclude from Table 7 that in all other cases, we fail to reject the Null hypothesis, indicating that there is not enough evidence to conclude a significant difference. COTE is probably one of the best-performing algorithms, due to the fact that we encounter a significant difference for a total of eight times out of eleven.

5.3 Critical difference

After conducting the statistical tests, we first concluded that there is sufficient evidence for statistical significance among the algorithms. Secondly, we concluded that there is sufficient evidence for statistical significance among the pairwise comparisons of algorithms. From this top-down perspective, we concluded that in cases of statistical significance between the pairwise comparisons, one algorithm outperforms the other. However, we still do not know how an individual algorithm performs against all other algorithms. To examine the performance of a single algorithm compared to the others, we can use the *critical difference* [10]. We use the average rank of each algorithm. If the average rank of a single algorithm differs by at least the CD from another algorithm, we conclude that the algorithm is significantly different from the other. The following formula calculates the critical difference:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

- k: represent the number of algorithms
- N: represent the number of datasets
- q_α: represent the critical value given a significance level.

The critical value is derived from the Studentized Range statistic. Once this value is obtained, it is divided by $\sqrt{2}$ [10]. The α level for q_α is set to 0.05. After we set the α level, we can use the Studentized Range Q table:

df	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
26	2.907	3.514	3.880	4.141	4.345	4.511	4.652	4.773	4.880	4.975	5.061	5.139	5.211	5.277	5.339	5.396	5.450	5.500	5.548
27	2.902	3.506	3.870	4.130	4.333	4.498	4.638	4.758	4.864	4.959	5.044	5.122	5.193	5.259	5.320	5.377	5.430	5.480	5.528
28	2.897	3.499	3.861	4.120	4.322	4.486	4.625	4.745	4.850	4.944	5.029	5.106	5.177	5.242	5.302	5.359	5.412	5.462	5.509
29	2.892	3.493	3.853	4.111	4.311	4.475	4.613	4.732	4.837	4.930	5.014	5.091	5.161	5.226	5.286	5.342	5.395	5.445	5.491
30	2.888	3.486	3.845	4.102	4.301	4.464	4.601	4.720	4.824	4.917	5.001	5.077	5.147	5.211	5.271	5.327	5.379	5.429	5.475
31	2.884	3.481	3.838	4.094	4.292	4.454	4.591	4.709	4.812	4.905	4.988	5.064	5.134	5.198	5.257	5.313	5.365	5.414	5.460
32	2.881	3.475	3.832	4.086	4.284	4.445	4.581	4.698	4.802	4.894	4.976	5.052	5.121	5.185	5.244	5.299	5.351	5.400	5.445
33	2.877	3.470	3.825	4.079	4.276	4.436	4.572	4.689	4.791	4.883	4.965	5.040	5.109	5.173	5.232	5.287	5.338	5.386	5.432
34	2.874	3.465	3.820	4.072	4.268	4.428	4.563	4.680	4.782	4.873	4.955	5.030	5.098	5.161	5.220	5.275	5.326	5.374	5.420
35	2.871	3.461	3.814	4.066	4.261	4.421	4.555	4.671	4.773	4.863	4.945	5.020	5.088	5.151	5.209	5.264	5.315	5.362	5.408
36	2.868	3.457	3.809	4.060	4.255	4.414	4.547	4.663	4.764	4.855	4.936	5.010	5.078	5.141	5.199	5.253	5.304	5.352	5.397
37	2.865	3.453	3.804	4.054	4.249	4.407	4.540	4.655	4.756	4.846	4.927	5.001	5.069	5.131	5.189	5.243	5.294	5.341	5.386
38	2.863	3.449	3.799	4.049	4.243	4.400	4.533	4.648	4.749	4.838	4.919	4.993	5.060	5.122	5.180	5.234	5.284	5.331	5.376
39	2.861	3.445	3.795	4.044	4.237	4.394	4.527	4.641	4.741	4.831	4.911	4.985	5.052	5.114	5.171	5.225	5.275	5.322	5.367
40	2.858	3.442	3.791	4.039	4.232	4.388	4.521	4.634	4.735	4.824	4.904	4.977	5.044	5.106	5.163	5.216	5.266	5.313	5.358
48	2.843	3.420	3.764	4.008	4.197	4.351	4.481	4.592	4.690	4.777	4.856	4.927	4.993	5.053	5.109	5.161	5.210	5.256	5.299
60	2.829	3.399	3.737	3.977	4.163	4.314	4.441	4.550	4.646	4.732	4.808	4.878	4.942	5.001	5.056	5.107	5.154	5.199	5.241
80	2.814	3.377	3.711	3.947	4.129	4.277	4.402	4.509	4.603	4.686	4.761	4.829	4.892	4.949	5.003	5.052	5.099	5.142	5.183
120	2.800	3.356	3.685	3.917	4.096	4.241	4.363	4.468	4.560	4.641	4.714	4.781	4.842	4.898	4.950	4.998	5.043	5.086	5.126
240	2.786	3.335	3.659	3.887	4.063	4.205	4.324	4.427	4.517	4.596	4.668	4.733	4.792	4.847	4.897	4.944	4.988	5.030	5.069
inf	2.772	3.314	3.633	3.858	4.030	4.170	4.286	4.387	4.474	4.552	4.622	4.685	4.743	4.796	4.845	4.891	4.934	4.974	5.012

Table 8: Studentized Range Q table. Source: [10]

Given the α value of 0.05, we obtain the Studentized Range Q table shown in Table 8. On the x-axis, we observe the number of algorithms, and on the y-axis the degrees of freedom (df).

Based on the benchmark Table 2, we obtain the following values:

- k = 12
- N = 85
- $\alpha = 0.05$,

We first need the critical value. Based on the α of 0.05, k = 12, and an infinite degree of freedom, this results in a $q_{0.05}$ value of 4.622.

Dividing the value $q_{0.05}$ by $\sqrt{2}$ results in 3.268. By implementing this information into the formula for the critical difference, we receive the following result:

$$CD = q_\alpha \cdot \sqrt{\frac{k \cdot (k+1)}{6N}} = 3.268 \cdot \sqrt{\frac{12 \cdot (12+1)}{6 \cdot 85}} = 3.268 \cdot \sqrt{\frac{156}{510}} \approx 3.268 \cdot 0.5534 \approx 1.807$$

We can now compare each individual algorithm to all other algorithms and determine if the algorithm is statistically significantly better. We obtain the following figure:

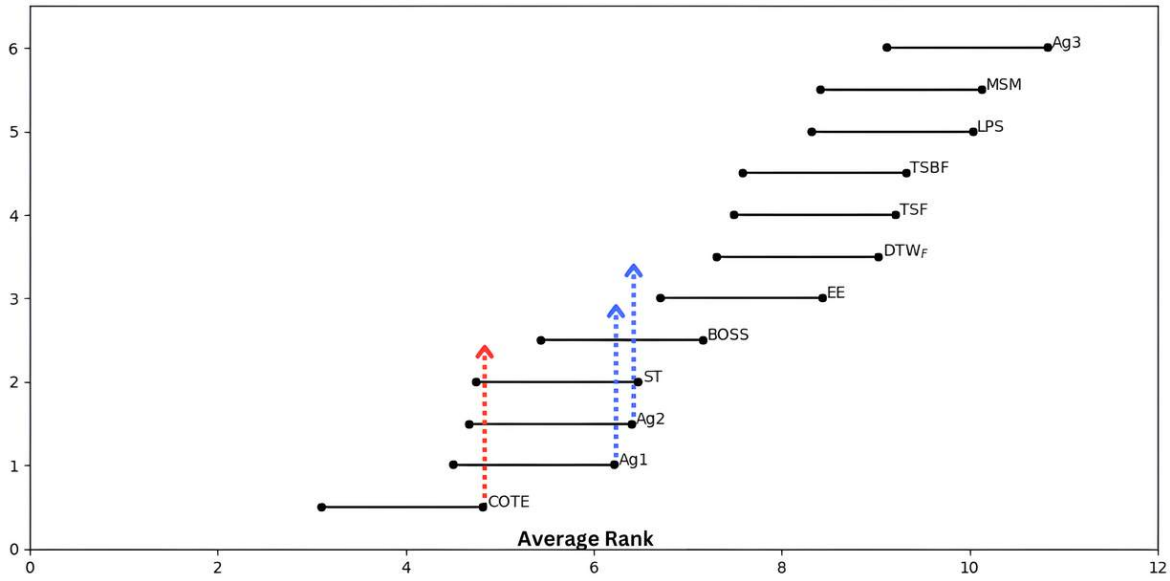

Figure 10: Critical Difference plot for the benchmark algorithms.

As you can see, we have a total of twelve algorithms. Each algorithm has a starting point on the x-axis, which is based on the average rank, derived from Table 5. Next, we add the value of the critical difference to each algorithm, which is represented by the horizontal lines. If an algorithm differs at least by the critical difference, it is considered statistically significant, which implies that the horizontal line of one algorithm does not overlap that of another.
Taking COTE as an example, we can observe that it is not statistically significantly better than Ag1, Ag2, and ST. However, it is statistically significantly better than all the other algorithms. To illustrate this, we have added several vertical lines to show if the horizontal lines overlap. Similarly, we can conclude that both Ag1 and Ag2 are not statistically significantly better than COTE, ST, and BOSS, but they statistically significantly outperform the other algorithms.

5.4 Feature importance score

In Section 4.3, we introduced a metric called feature importance. We describe the insights and benefits of using feature importance and provide a small example using the Gunpoint dataset 9. After applying the aggregation algorithms in combination with the benchmark algorithms on the 85 datasets, we observe that aggregation algorithms, both with and without overlap, perform quite well, as mentioned in Section 5.1. To further investigate these aggregation algorithms, we use the feature importance metric to identify the main aggregation functions responsible for the performance of these algorithms. These aggregation functions were previously discussed and analysed in Section 3.1.

To ensure a fair comparison of feature importances between Ag1 (without overlap) and Ag2 (with overlap), we standardized the total number of windows to 20 by adjusting the window sizes and stride settings. This standardization prevents potential bias in the Random Forest classifier caused by unequal information resulting from the windowing process. Standardizing the window count

ensured that differences in feature importance reflected the aggregation functions' characteristics rather than variations in available information.

To get a better understanding of the accuracies produced by the aggregation algorithms, we illustrate three bar graphs. Each bar graph represents the top-20 aggregation functions responsible for the performance of the aggregation algorithms. The figures contain a descending pattern, with the most influential aggregation function on the left and the least influential on the right. The horizontal x-axis contains the names of the aggregation functions. In cases of conditional aggregation, we use the following representation *function* x_y, where x illustrates the total window count and y illustrates the specific window number. We obtain the following three bar graphs:

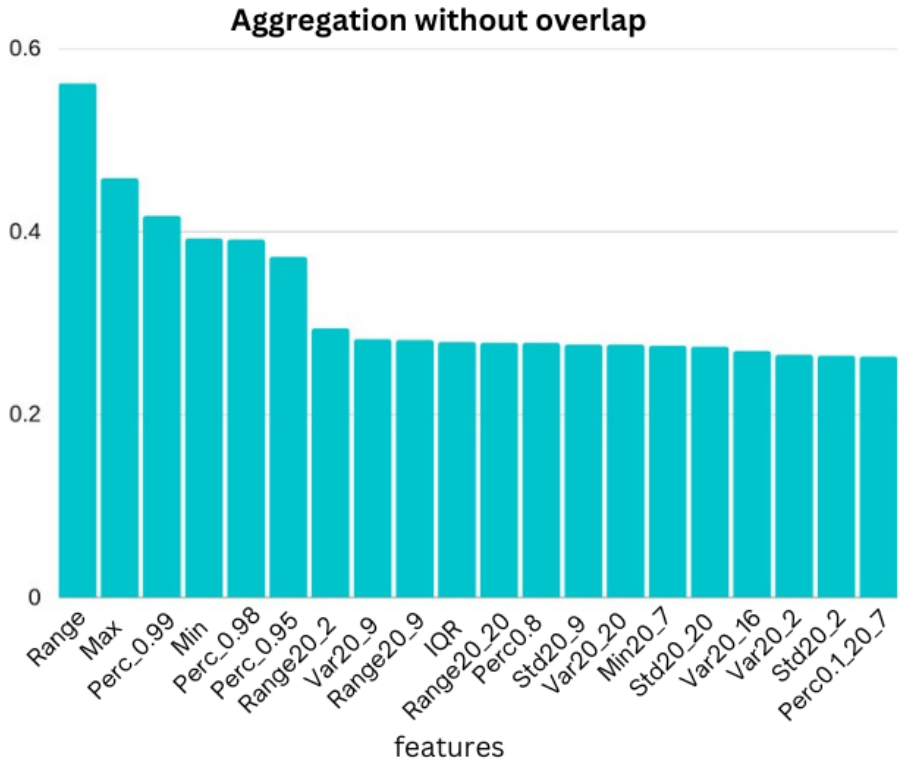

Figure 11: A bar chart illustrating the top 20 features importance's of the aggregation without overlap (Ag1). In some cases, aggregation functions are represented as follows: *function* x_y, where x illustrates the total window count and y the specific window number.

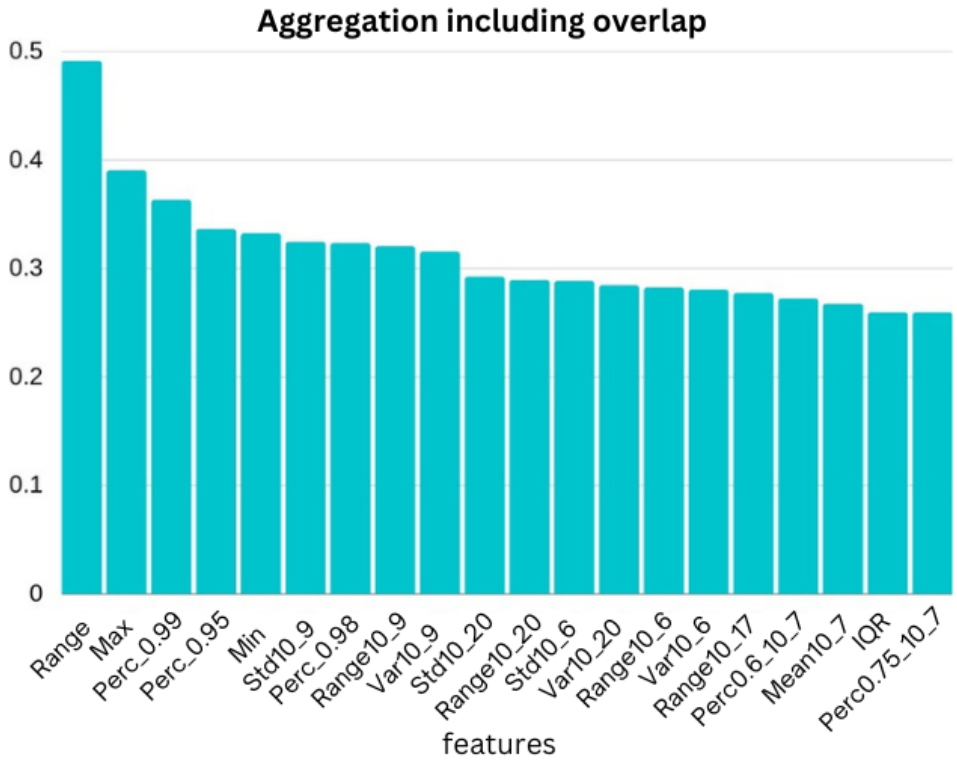

Figure 12: A bar chart illustrating the top 20 feature importances of the aggregation including overlap (Ag2). In some cases, aggregation functions are represented as follows: *function* x_y, where x illustrates the total window count and y the specific window number.

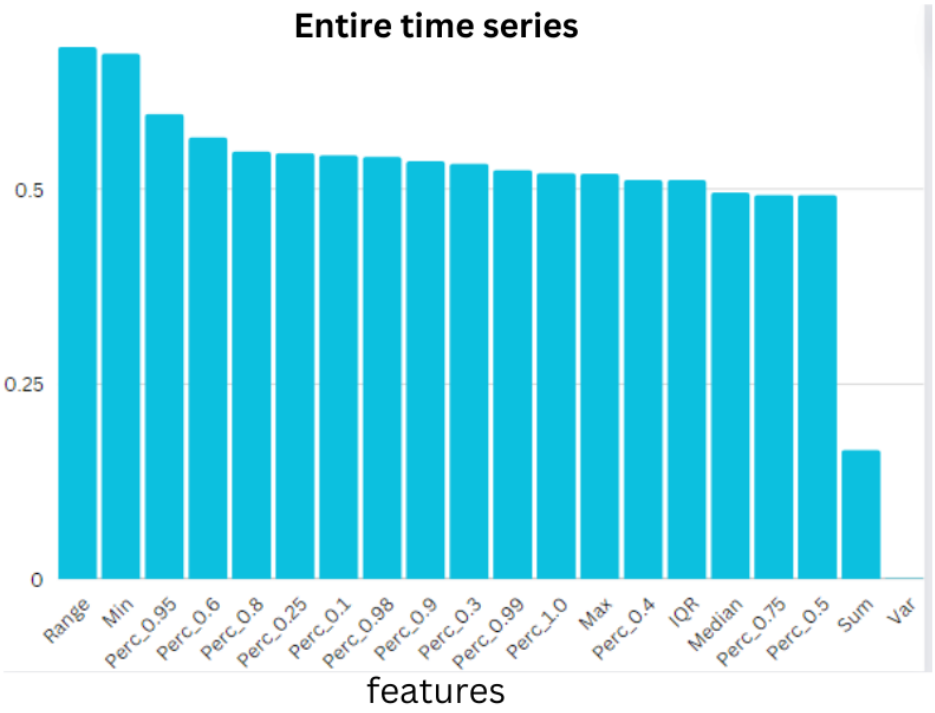

Figure 13: Top 20 feature importance based on the aggregation of the entire series (Ag3).

When examining the feature importances of the experimental setups, we notice that *Range* is the most influential feature in all three cases. Aggregation on the entire series is highly influenced by *Min* and the *Perc_0.95*. After these, the influence of the other features is very similar, except for the last two features.

Furthermore, we see that the two conditional aggregation algorithms show more similarities between each other than to aggregation on the entire series. The top three features are the same for both conditional aggregations, further completed by *Max* and *Perc_0.99*. Additionally, higher percentiles such as *Perc_0.98* and *Perc_0.95*, along with different range values like *Range_20_9* and *Range_10_9*, are mainly responsible for the classifier's performance rate.

This explains why aggregation on the entire time series is less effective than aggregation with and without overlap. Although we still use the aggregation functions for the entire time series and they play a major roll for the performance of the classifier, the aggregation functions for conditional aggregation are so important that they outperform the aggregation on the entire series. This also highlights the difference between the aggregation algorithm with and without overlap, and the functions that determine the better performance rate in the case of aggregation with overlap.

5.5 Parameter influence

After examining the results for the benchmark algorithms including the experimental setups discussed above, we notice that aggregation algorithms, both with and without overlap, perform well in classifying unseen observations. To further investigate the aggregation algorithms, we also present the average accuracy, total wins, average rank, and median rank as metrics to determine how the parameter settings for the window count and stride influence the performance of the different algorithms.
Using the same performance metrics as before, we obtain the following table:

	Ag1	Ag2	Ag4	Ag5	Ag6	Ag7	Ag8	Ag9	Ag10	Ag11
Avg Accuracy	0.834	0.838	0.821	0.811	0.829	0.835	0.841	0.837	0.833	**0.842**
Wins	14	**20**	13	8	13	15	17	18	18	18
Avg Rank	4.882	4.412	6.282	7.024	5.353	5.165	4.871	4.212	5.071	**4.000**
Median Rank	5	4	7	8	5	6	5	4	5	**3**

Table 9: Performance metrics for different aggregation methods

When we look at the performance metrics in Table 9, we can see that Ag11 is the overall best-performing algorithm, achieving the highest average accuracy (0.842), the best average rank (4.000), and a median rank of 3. Ag2 follows Ag11 closely regarding the average accuracy (0.838) and leads in terms of the total wins with 20. Ag5 is the worst performing aggregation algorithm with the lowesthe lowest average accuracy (0.811), the highest average rank (7.024), and the fewest wins (8). We were also interested in the performance of the aggregation algorithms in terms of overlapping and non-overlapping cases. Based on the information in Table 9, we obtained the following two figures:

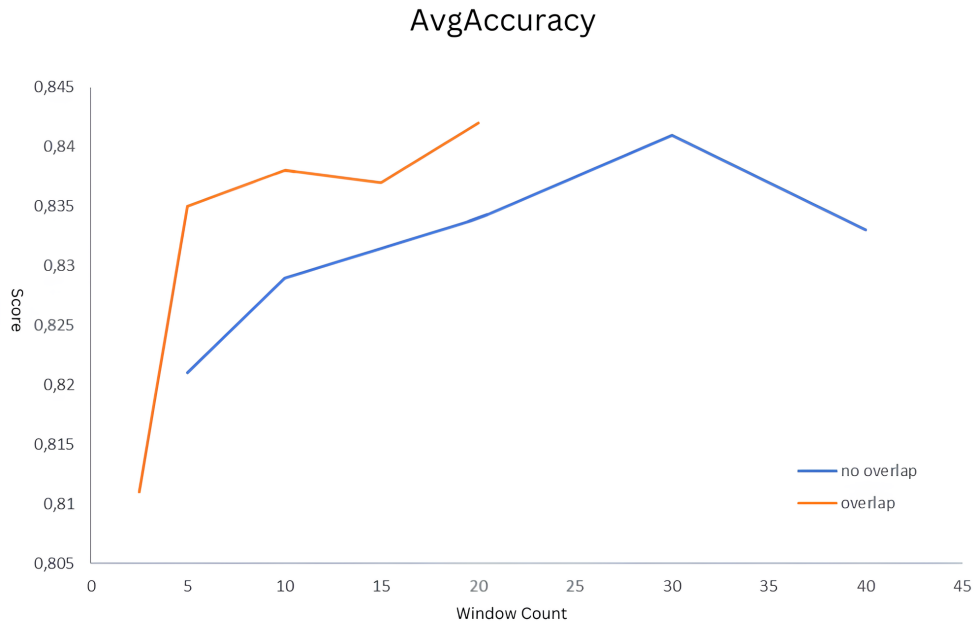

Figure 14: A line chart illustrating the average accuracy (AvgAccuracy) for aggregation algorithms containing overlap and no overlap. The x-axis represents the varying window counts, and the y-axis displays the average accuracy.

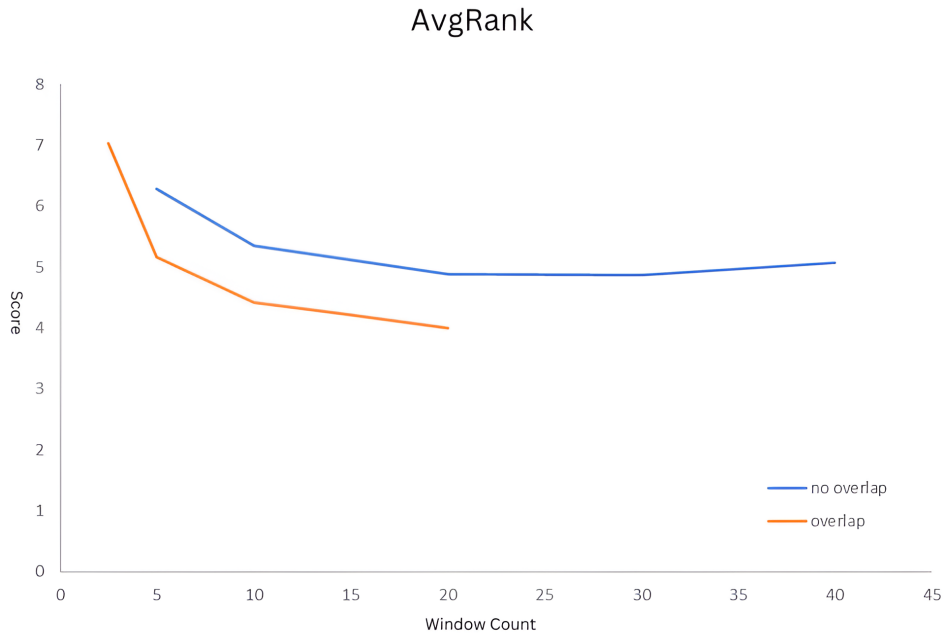

Figure 15: A line chart illustrating the average rank (AvgRank) for aggregation methods containing overlap and no overlap across varying window counts. The x-axis represents the varying window counts and the y-axis displays the average rank.

When observing Figure 14, we see that each overlapping aggregation algorithm (Ag2, Ag5, Ag7, Ag9, and Ag11) outperforms its non-overlapping counterpart (Ag1, Ag4, Ag6, Ag8, and Ag10) in a pairwise comparison, achieving a higher average accuracy score. As the window count gets larger than 30, we start to see a decline in the average accuracy in the case of no overlap, compared to overlap where we still achieve a higher average accuracy for larger window counts.

In Figure 15, we see the average rank for the aggregation algorithms in case of overlapping and non-overlapping cases. Also in this case, the overlapping cases outperform the non-overlapping cases. When the window count gets larger, we see the performance of the overlapping aggregation functions getting lower, whereas the average rank of the non-overlapping aggregation eventually stays the same and slightly increases in the end.

Based on these findings, we conclude that Ag11 and Ag2 are among the best-performing algorithms. These metrics also show that using parameter settings with overlap enhances the aggregation algorithms, resulting in higher accuracies compared to those without. The worst-performing aggregation algorithm compared to all the other algorithms is Ag5. The difference between the best and worst-performing algorithm in terms of the average accuracy is 0.031, which is relatively small. However, notable differences are seen in terms of total wins (10), the average rank (3.034) and the median value (4).

6 Conclusion

In this thesis, we have focused on the exploration of an aggregative approach to time series classification. We have used nine benchmark algorithms, which have acted as reliable resources for time series classification. The algorithms are designed to classify unseen time series data and accurately predict target labels associated with different classes or categories. Our research supports the conclusion that the aggregative approach demonstrates a high success rate for time series classification.

We have seen several statistics that confirm that the aggregative approach has an overall high success rate. Upon comparing the total wins of the benchmark aggregation algorithms, we observe that they have the highest number of wins, and rank second and third in terms of average rank among all other algorithms, after COTE. In addition to these metrics, we have also looked at other statistical measurements. The average scores of our aggregative approach outperform BOSS, EE, DTW_f, TSF, TSBF, LPS, and MSM, as mentioned in Table 5.

Furthermore, we analyzed the performance of the Random Forest classifier by evaluating feature importance. The objective was to determine the impact of the aggregation functions for the following aggregation algorithms: Ag1, Ag2, and Ag3. Our analysis revealed that aggregation functions applied to the entire series play a important role across all three aggregation algorithms, as mention in Section 5.4. However, the conditional aggregation functions demonstrated a substantial impact, resulting in a higher performance for both aggregation with and without overlap, compared to whole-series aggregation.

We also examined variants of aggregation algorithms and noted different results among them. We have seen that these algorithms perform highly sufficient in the various metrics when we look at the average accuracy score, total wins, average median rank, and average rank, as mentioned in Table 9.

To validate our insight and conclusions regarding the aggregation algorithms, we used a top-down approach by utilising the Friedman test, the Nemenyi test, and the critical difference analysis. These statistical tools indicated insufficient evidence to conclude that the conditional aggregation functions statistically outperform COTE, ST, and BOSS. However, they provided sufficient evidence to conclude that our conditional aggregation approach statistically outperforms the EE, DTW_f, TSF, TSBF, LPS, and MSM algorithms.

In light of these findings, the aggregative approach towards time series classification shows promising results by contributing to the advancement of time series classification methods and providing valuable lessons for researchers in this field. The growth and progress of time series classification from this thesis could be a promising tool for future work.

6.1 Further Research

Further research aimed at improving the performance of time series classification can be conducted. This can be done by mainly looking into the practical implications and generalizability of the aggregative approach. Practical implications can involve investigating domains such as healthcare, finance, or automation of processes, which could potentially lead to improved productivity and cost reductions.

Another interesting feature to investigate is the use of alternative aggregation functions. In this thesis, we have already used the most popular aggregation techniques. By examining other variations,

this could provide insights into different aggregation approaches that could enhance the performance of the overall aggregation algorithm. Alternative approaches, such as implementing different forms of percentile-based algorithms, will aim to capture additional statistical characteristics of the time series data to improve the aggregation functions. Another possibility could be using weighted aggregations, putting more emphasis on specific aggregation functions based on their feature importance and determine whether this will enhance the performance of the aggregation algorithms. Additionally, considering different types of classifiers can play a crucial role in optimising the aggregative approach. By experimenting with various classifiers, we may identify the most suitable classifier for the specific input features generated by different aggregation techniques, depending on the type of time series data.
The datasets that we have used in this research contain certain assumptions, such as that the series were of equal length, had real values, and contained no missing values. Developing solutions for such scenarios that do not meet these assumptions and investigating the best strategies for handling these problems can potentially open classification problems for other sections [5]. Understanding these implications and improving them can lead to further enhanced practical applications of the aggregative approach in real-world solutions.

References

[1] Vassilis Athitsos Alexandra Stefan and Gautam Das. The move-split-merge metric for time series. *IEEE Journals Magazine — IEEE Xplore*, June 2013.

[2] Jason Lines Aaron Bostrom James Large Anthony Bagnall, Eamonn Keogh and Matthew Middlehurst. Time series classification website. Available at: https://www.timeseriesclassification.com/dataset.php, 2018.

[3] Anthony Bagnall, Jason Lines, Jon Hills and Aaron Bostrom. Time-series classification with COTE: The collective of transformation-based ensembles. *IEEE Journals & Magazine*, September 2015.

[4] A. J. Bagnall and G. J. Janacek. A run length transformation for discriminating between auto regressive time series. *Journal of Classification*, 31(2):154–178, 2013.

[5] A. J. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. 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, 2016.

[6] Prashant Banerjee. Random forest classifier tutorial. https://statistics.laerd.com/spss-tutorials/friedman-test-using-spss-statistics.php, 2020.

[7] M. G. Baydogan and G. C. Runger. Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30(2):476–509, 2015.

[8] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. A bag-of-features framework to classify time series. *IEEE Journals Magazine*, 2013.

[9] Julianna Delua. IBM, what is supervised learning? https://www.ibm.com/topics/supervised-learning, 12 March 2021.

[10] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[11] H. Deng, G. C. Runger, E. Tuv, and M. Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

[12] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[13] Jim Frost. Hypothesis tests, significance levels, and p-values. https://statisticsbyjim.com/hypothesis-testing/hypothesis-tests-significance-levels-alpha-p-values/.

[14] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. J. Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2013.

[15] Rob J. Hyndman and George Athanasopoulos. Forecasting: Principles and Practice (2nd ed), Autoregressive Models, 31 May 2021.

[16] Javatpoint. Train and test datasets in machine learning. https://www.javatpoint.com/train-and-test-datasets-in-machine-learning.

[17] R. J. Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, 30(2):283–312, 2015.

[18] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[19] J. Lines and A. J. Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592, 2014.

[20] Dr Adam Lund and Mark Lund. Friedman test in spss statistics - how to run the procedure, understand the output using a relevant example. https://statistics.laerd.com/spss-tutorials/friedman-test-using-spss-statistics.php, 2018.

[21] P. Płoński. Random forest feature importance computed in 3 ways with Python. https://mljar.com/blog/feature-importance-in-random-forest/, 2020.

[22] Rajesh C. Sachdeo, Wanpracha Art Chaovalitwongse, and Ya-Ju Fan. On the time series k-nearest neighbor classification of abnormal brain activity. November 2007.

[23] A. Sharma. Random forest vs decision tree — which is right for you? https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/, June 12 2024.

[24] Maksim Terpilovskii. scikit-posthocs: Post hoc tests for pairwise multiple comparison of ranked distributions. https://github.com/maximtrp/scikit-posthocs/blob/master/scikit_posthocs/_posthocs.py, 2023.

[25] Timothy R. Dinger, Yuan-chi Chang, Raju Pavuluri and Shankar Subramanian. What is time series classification? https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification/, 25 January 2022.