Master thesis

Unsupervised segmentation of biosignal-based time-series

Knut Joar Strømmen

Robotics and Intelligent Systems
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2020

Thesis

Unsupervised segmentation of biosignal-based time-series

Knut Joar Strømmen

Abstract

The rise of the Internet of Things (IoT) and the development of more compact and less power-hungry sensors have led to an increasing amount of data from various modalities. To analyze a large amount of continuously recorded time series data at scale, it is often beneficial to separate the data into homogenous segments before further analysis or classification. One common way to do time series segmentation is to find the transitions that separate the different segments. These transitions are in the literature referred to as changepoints, and the problem of discovering them is referred to as change point detection (CPD). Long time series data is hard and time-consuming to label; thus, having a CPD algorithm that works in an unsupervised fashion is highly beneficial.

This thesis provides an overview of automatic change point detection, time series segmentation, and previous research concerning representation learning in the context of CPD. The main contribution of this thesis is the *Latent Space Unsupervised Semantic Segmentation* (LS-USS) model, which is an unsupervised segmentation algorithm that is hyperparameter lite, domain agnostic, and works well with multidimensional data. This model utilizes an autoencoder for extracting constrained feature encodings from time-series data before using the similarity between neighboring feature encodings in time to do changepoint detection and segmentation.

The LS-USS model is compared with other state-of-the-art algorithms on multiple datasets. The results show that the model performs similarly or better than other state-of-the-art models. To demonstrate a practical application of LS-USS, a dataset containing activity data from non-depressed and depressed participants is used. Using the LS-USS algorithm as a preprocessing step before classification significantly outperforms the previous best classification results on this dataset.

Table of Contents

List of Figures

List of Tables

Preface

It has been a long year due to the ongoing pandemic. Luckily, this thesis has kept me busy!

The work undertaken has allowed me to combine concepts from various fields, which has really made me appreciate the diverse education acquired through five years at IFI. It has also helped me learn to manage high complexity, and I will never again underestimate the amount of time that goes to documenting and writing when doing scientific work.

I want to give a huge thanks to Ulysse for being interested and supportive through the whole thesis. I have learned a lot from you, and I am forever grateful for all the help.

Thanks to Jim for having an eye for details and for reminding me to *Concentrate on where you feel you get "fish" and "return home" before it gets too dark and difficult to get back to the dock.*

I would also like to thank my friends. Both for motivating me to write and for distracting me from writing.

Lastly, a well-deserved thanks to my family for showing compassion and helping me out under stressful periods.

1 Introduction

Humans use the cognitive functions of attention and memory to make constrained representations about the world that are beneficial for relevant goal attainments [1]. An important skill when driving a car is to differentiate between the road and the roadside. Contrastingly, identifying every nook and cranny that the road is comprised of would be useless. Instead, having a constrained encoding of the road based on high-level features such as texture, color and contrasts are much more helpful. A sufficient change in the feature encoding would indicate that the road lane has ended and that the roadside has started. These transition between objects in visual data is often referred to as edges. If one has detected all the edges between objects in an image, extracting the objects is trivial. In time-series data mining, a similar concept to edges called changepoints is utilized. A changepoint represents a transition between different states in the process that generates the time series data. The problem of finding these transitions is referred to as Change point detection (CPD).

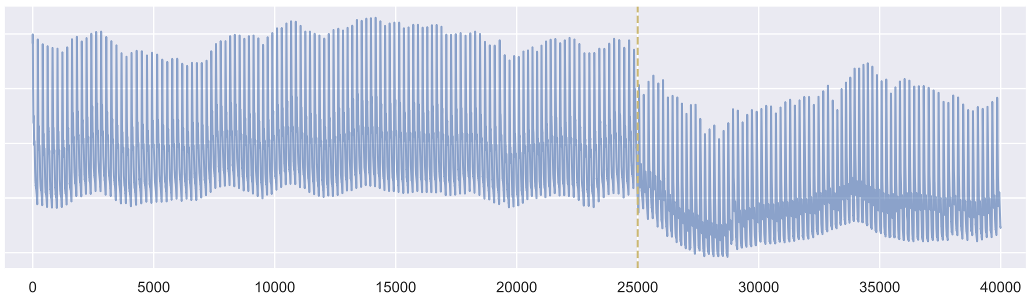

Figure 1: Example of a changepoint occurrence at timestep 25,000. From a dataset in the UCR Time Series Semantic Segmentation Archive [2] where a subject was lying on a tilt table with foot support. At timestep 25,000 the tilt table was rapidly rotated to the stand-up position. The figure shows the volunteers ABP (Arterial Blood Pressure).

1.1 Applications of Change Point Detection

CPD algorithms are getting increasingly useful due to the rise of the Internet of Things (IoT) and the development of more compact and less power-hungry sensors, which has led to an increasing amount of long-time series data from various modalities. A device category that is becoming more ubiquitous is wearable devices (wearables) attached to the user’s body. Examples of such devices are smartphones, smartwatches, fitness trackers, and sensors implemented in textiles. The various kinds of wearables’ sensors can include accelerometers, gyroscopes, heart rate sensors, and galvanic skin response monitors. The data extracted from these sensors can then be employed to derive various helpful metrics such as calories burned, heart rate, blood pressure, physical activity, sleep patterns, and even mental states.

The time-series data gathered from these devices are often challenging and time-consuming to label. Say one wants to continuously monitor heart rate data and accelerometer data (recorded in three spatial directions) sampled at 50Hz from an activity watch worn by several participants. In this case, each participant would generate 720000 datapoints every hour. Having people manually supervising or labeling these data streams is for most real-world applications, unfeasible. To make use of data at this scale, having unsupervised CPD algorithms that can detect when the data changes become highly beneficial. Often, the most useful information lies in the areas where there is a change in the times series data, while long monotonous data contain little information. In the example above, one could use CPD algorithms to automatically find anomalies in a person's heart rate based on the activity level and heart rate. In [3] they made a CPD algorithm for heart rate trend monitoring that outperformed the change detection by clinicians in real time. Another use-case of CPD could be fault detection in factory machines, as these machines often perform repetitious tasks. By, for example, monitoring their power consumption, one can use a change point detection algorithm to detect anomalies which could indicate a malfunctioning machine automatically.

Another application of unsupervised CPD is time series segmentation, which is the main focus of this thesis. If all the changepoints in the time series that separate the different segments are detected, all the segments that the signal is comprised of are also found. When dealing with a large amount of continuously recorded data, it is often beneficial to separate the data into homogenous segments before further analysis or classification. It would be impossible to classify activities based on activity data without having information about when the different activities are performed. By first segmenting the signal, one could then train a classifier to detect the class of each segment based on the statistics in the segment. There is plenty of previous work done on human activity analysis in the context of CPD [4]. Another similar use-case would be to segment out uninteresting data which contains little useful information for a specific application. One example of this could be filtering out data captured from an activity watch when not being worn.

Some time series data is weakly labeled in that they include imprecise or inexact labels. An example of this could be recording EMG-data from a person instructed to hold five hand gestures for 15 seconds each in one continuous recording session. Here, a new segment in the data will occur approximately every 15 seconds, but one would not know the exact location of the actual gesture change since humans cannot robotically perform tasks at the exact time specified. To get a more precise segmentation, one could use CPD algorithms on a local area around every 15-second mark to pinpoint the exact time of a gesture change.

This thesis is associated with INTROMAT (INtroducing personalized TReatment Of Mental health problems using Adaptive Technology) [5], a Research Council of Norway funded research project. The project's vision is to investigate how the use of ICT (Information and Communications Technology) can improve public mental health. One of the INTROMAT project goals is to develop an unobtrusive Mental Health Monitoring System (MHMS) using sensors, surveys, and machine learning. The following section will provide information about mental health and how CPD algorithms can be utilized in the context of MHMS.

1.2 Mental Health Monitoring Systems

Mental health is a major problem worldwide. In 2010 mental health problems were the leading cause of years lived with disability(YLD) worldwide, and depressive and anxiety disorders were among the most frequent disorders [6], [7]. Living with a mental disorder can be challenging in many ways, as it reduces the ability to carry out everyday tasks, which often leads to social, economic, and emotional difficulties. It will affect the friends and family of the patient since it can be hard to see a person one cares for dealing with a mental illness. Mental health problems also affect society in lost taxes due to work impairment and treatment costs. World Economic Forum estimated that the cumulative global effect of mental disorders in terms of lost economic output could amount to US $16 trillion in the next 20 years, equivalent to 25% of global GDP in 2010 [8].

Much research has been done on mental health monitoring systems (MHMS) [9], but much more is needed to get to a fully autonomous patient monitoring system. Wearables are, as mentioned, great for continuous data collection, and recent studies show that they hold great potential for unobtrusive data gathering of biological signals from people suffering from mental disorders [9]. These devices have the advantage of being privacy-minded as it is hard to identify a specific person from data gathered using them. Another benefit of wearables is that they can be worn all day and are not bound to specific locations or settings. Since these devices are becoming increasingly ubiquitous, it is less expensive to deploy due to decreased manufacturing costs. In addition to this, many people already use them for other purposes such as physical health and fitness, making it less stigmatic to use for MHMS. According to Statista Consumer Surveys, 30 percent of U.S. consumers owned a fitness band, and 44 percent of U.S. consumers used their sport and fitness gadgets daily[10]. Smartphones, which are pretty common these days, have a lot of the same sensors used in fitness trackers and can therefore also be a useful tool for monitoring the mental state of a person, in addition to being a potential tool for self-reporting and interventions.

As MHMS utilizing wearables can generate a large amount of data, it is crucial to have algorithms that can extract useful information without supervision. One application of CPD in the context of mental health could be to find changepoints in the data that could suggest a change in behavior or mental state. Another possible application is to use unsupervised CPD to find homogenous segments in the data before extracting statistical features from each segment which is fed to a classifier trained to distinguish between different activities. As an example, there is evidence that depression leads to reduced daytime motor activity and increased nighttime activity compared to healthy individuals [11], so being able to separate between wakefulness and sleep could be beneficial for detecting depressed patients.

In this thesis, a CPD algorithm is used to separate activity data into natural segments, which are then used to classify non-depressed and depressed participants. More details on this follow in the next chapter, which will go through the main contributions presented in the thesis.

1.3 Summary of the Work

This thesis aims to develop unsupervised semantic segmentation algorithms that are hyperparameter lite, domain agnostic, and work well with multidimensional data. In addition to this, it is preferable if the algorithms are fast and can work on online streaming data.

Being hyperparameter lite and domain agnostic means the algorithm can be deployed on various time series data-mining problems without needing extensive tuning to perform well.

Real-world data is often multidimensional, meaning that data is sampled from various sources and sensors that often contain multiple channels. It is crucial that the algorithms can utilize this data effectively and ensure that the most helpful information is not drowned out by data sources containing less information.

Time series data is often recorded over a long time, making algorithms that are fast and scales well to big data crucial. Algorithms with a high time complexity that works great on short time series data would quickly get useless for problems such as continuous monitoring of people suffering from mental health issues, which entails data captured over long time intervals. Many real-world applications depend on fast changepoint detection to execute time-sensitive actions, which means that algorithms that cannot handle online streaming data will struggle, as they would require a complete rerun every time a new data point is added.

One state-of-the-art algorithm that meets many of these requirements is *Fast Low-cost Unipotent Semantic Segmentation* [12] (FLUSS). The fundamental thought process behind the development of FLUSS is that subsequences (small snippets of the time series data extracted around each time step in the time series) in similar segments are more similar than subsequences belonging to other segments. Thus, parts of the time series containing many similar subsequences are likely to be a segment. The main contributions of this thesis are inspired by the FLUSS algorithm and try to improve upon the basic concepts presented in that paper.

FLUSS handles multidimensional data by calculating the likelihood of a change point at each time step for each channel independently before taking the average likelihood over all the channels. When you have many channels, some might not contain information that is helpful for segmenting out activities, or they might be heavily correlated. When taking the average over all the channels, these “not so useful” channels can drown out the useful channels as they are all equally weighted when taking the average. The authors acknowledge this problem and recommend that when dealing with high dimensional data one should do a search or manually find the most useful subset of channels and remove the rest.

As time series data is often sampled from multiple sources and sensors, it would be helpful to have an algorithm that can automatically extract the most useful information from high-dimensional times series data before doing segmentation. This is the primary motivation behind one of the main contributions of this thesis; the *Latent Space Unsupervised Semantic Segmentation* (LS-USS) model. LS-USS is based on the same basic concepts as FLUSS, but instead of using the similarity between subsequences, it finds the similarity between lower-dimensional encodings of the multidimensional subsequences. These encodings are found by feeding the subsequences through an autoencoder trained in an unsupervised fashion. The idea is that the dimensionality reduction learned by the autoencoder will remove redundant and correlated information between channels, which in turn will improve the segmentation. Through extensive testing conducted on both artificial and real-world datasets from various domains and sensors, it was found that LS-USS delivers on par or better segmentation scores compared to other state-of-the-art algorithms.

One challenge in applying unsupervised algorithms is the selection of hyperparameters. FLUSS only requires us to define the subsequence length, which, depending on the context, is not always trivial to define. While it was shown that FLUSS is not overly sensitive to its value, for practical application of FLUSS, this remains an important consideration. It was found that FLUSS seems to perform best when the subsequence length is set to approximately one period of the signal. One of the contributions of this thesis is to augment FLUSS with Welch's method to automatically determine an appropriate subsequence length, thus making FLUSS completely hyperparameter-free. Welch’s method is a method for estimating the frequency contents of a signal. From the frequency content, one can find the peak frequency, which makes finding the main periodicity of the signal trivial. This method will in this thesis be referred to as FLUSS-Welch. This algorithm is tested on the UCR Time Series Semantic Segmentation Archive[2], which includes 21 datasets from many domains. The results show that FLUSS-Welch achieves similar results to specifying the subsequence length manually.

The algorithms developed in this thesis should be relatively domain agnostic. Nevertheless, as this thesis is associated with INTROMAT, it would be beneficial to test the algorithm on more datasets from mental health research, but these datasets are often difficult to get access to due to the sensitive nature of personal data from people suffering from mental health issues. To showcase a practical application of the algorithms developed in the context of MHMS, an open dataset from the INTROMAT project called Depresjon [13] containing data from non-depressed and depressed participants is used. Here the goal is to see if using segments found by unsupervised segmentation can increase depression detection. The results show that using the LS-USS segmentation algorithm before classification clearly outperforms the previous best classification results on the dataset.

1.4 Chapters

1.4.1 Background

The background chapter starts with more details on CPD and provides an overview of the previous research done on classical CPD (2.1). Section 2.2 will go through a concept called matrix profiles, an essential building block for both FLUSS and LS-USS. In section 2.3, FLUSS will be covered in more detail.

Section 2.4 is about representation learning and relevant research regarding representation learning in the context of CPD. Representation learning, or feature learning, concerns techniques for automatically extracting feature encodings/representations from raw data. One common example of representation learning methods is autoencoders, which are highly relevant for the development of LS-USS.

The evaluation metrics used are essential to get a fair comparison between different CPD algorithms. Thus, section 2.5 gives an overview of various evaluation metrics with a discussion of their strength and weaknesses when used in CPD problems.

The last chapter in the Background, Section 2.5, will give a brief overview of Spectral Density Estimation. This chapter is relevant for FLUSS-Welch as it uses the spectral density estimator Welch’s method to find an appropriate subsequence length for FLUSS.

1.4.2 Methods

The methods chapter details the new methods developed for this thesis and the implementations of some state-of-the-art CPD algorithms presented in the background chapter. The chapter is divided into three main sections:

Section 3.1 describes the scalers used to scale the data before doing segmentation.

Section 3.2 details the CPD algorithms implemented in this thesis.

Section 3.3 covers the changepoint extraction algorithms implemented. These algorithms are used to decide what constitutes a changepoint based on the output from the CPD algorithms.

1.4.3 Datasets

The datasets chapter includes descriptions of five datasets used to test the methods implemented in this thesis. The three datasets presented in section 4.1 to 4.3 contains long multidimensional data which will be used to compare LS-USS against other state-of-the-art CPD algorithms.

The Depresjon dataset, which will be used to showcase a practical application of LS-USS in the context of MHMS, is detailed in section 4.4.

Section 4.5 presents the UCR Time Series Semantic Segmentation Archive[2] that includes 21 datasets from many different domains, ideal for testing FLUSS-Welch as all the datasets contain human-specified subsequence sizes.

1.4.4 Experiments

The experiments chapter shows the results of running the segmentation algorithms on the different datasets.

Section 5.1 details the experiments conducted on the datasets containing long multidimensional data and includes an in-depth comparison between the performance of the different segmentation algorithms.

Section 5.2 presents the results from using LS-USS on the Depresjon dataset as a preprocessing step before classifying depressed vs. non-depressed patients.

In section 5.2.2, the performance of FLUSS-Welch will be tested against using manually specified subsequence sizes on the UCR Time Series Semantic Segmentation Archive.

2 Background

This chapter will first give an overview of time-series changepoint detection (CPD). Then, a concept called the matrix profile is presented, which provides information about the similarity of the subsequences in a time series. Next, a state-of-the-art segmentation algorithm called FLUSS, built upon the matrix profile will be introduced. The section after that will focus on representation learning and present some examples from previous research of how it can be utilized in time-series segmentation. Then, there will be an overview of various evaluation metrics discussing their strengths and weaknesses when used in CPD problems. The last section centers around spectral density estimation, which is used to determine the frequency contents of a time series.

2.1 Time series change point detection

A changepoint (CP) represents a transition between different states in a process that generates time-series data [4]. Changepoint detection can be defined as the problem of hypothesis testing between the null hypothesis "no change occurs" and the alternative hypothesis "a change occurs" for each timestep in the time series. By finding these changepoints, one has indirectly segmented the time series, as the segments can be defined as the areas between the found changepoints.

2.1.1 Wanted Features

Besides raw performance in localizing changepoints, there are other aspects one needs to consider when comparing different CPD algorithms. The paper "A Survey of Methods for Time Series Change Point Detection" [5] introduces and describes some of the more central challenges for CPD algorithms.

One important feature to consider is whether the algorithm can be used for online data streams. Offline algorithms need the whole time series to do changepoint detection and cannot be updated with new data points without rerunning the algorithm on the whole dataset, which makes them unusable for streaming data. Online algorithms can run concurrently with the process. There will be a slight delay because of processing time, but it is usually sufficient that the datapoint is processed before a new one arrives. Different algorithms need varying amounts of data before detecting that a changepoint has occurred. To differentiate between them, the authors also define a third term called ε real-time algorithm. These need at least ε data points to detect a changepoint. A completely online algorithm would then be 1-real time since it only needs one data point to detect a change. It will often be a tradeoff between how real-time an algorithm is and the change point detection performance. The reason for this tradeoff is that offline algorithms can make decisions based on data from the whole time series, while online algorithms have to base the decision of there being a changepoint at the current timestep solely on previous observations.

Another critical factor is the scalability of the algorithm. Nowadays, as one has access to an increasing amount of data and often with high dimensionality, making an algorithm that is computationally efficient and scales well to large data sizes important. Having an algorithm where its space requirement and computation time grow exponentially with the length of the time series would be impractical when scaling to big data. Algorithms that can be interrupted and return the current best solution would be highly beneficial to meet the needs of applications with various time requirements. These algorithms are commonly referred to as anytime algorithms. In the survey paper [4] (from 2017), they did not find any interruptible CPD algorithms; thus, they considered anytime algorithms an avenue for future research. However, for matrix profiles (2.2), there has been developed an algorithm called scrimp++[14] that can calculate approximate matrix profiles to allow for use at “interactive speeds”. Scrimp++ was, in the original paper, not used for changepoint detection but for motif discovery which is another subject that will be touched on later in the thesis (2.1.3.6).

It is essential to consider what constraints and requirements the data impose on the choice of algorithm. Some might use statistical assumptions about stationarity or i.i.d. Some models cannot handle missing or multidimensional data. Others require extra information such as the number of changepoints or other knowledge about the underlying process that produces the time series.

2.1.2 Supervised CPD methods

Supervised learning uses labeled training data that consists of input-output pairs. The models will try to learn a mapping from input to output that generalizes to unseen data. There are two ways to apply this to CPD. One possible approach is to use binary classification, where the labels are “changepoint” and “no changepoint,” and train the model to recognize changes between states in the data. Alternatively, If the states are labeled into different classes, one can classify each data point in the time series based on class and extract the changepoint locations based on the class probability changes. Both techniques usually utilize the subsequences found by sliding a window over the time series as inputs to the model, where each subsequence is assigned a label/class. Most of the classifiers used in other machine learning tasks can also be used here. Some common classifiers are naïve Bayes, Support Vector Machines (SVMs), Gaussian Mixture Models (GMMs), Decision Trees, Hidden Markov Models, and Artificial Neural Networks (ANNs).

2.1.3 Unsupervised CPD methods

Unsupervised learning tries to find patterns in the input data without knowing the true label of the data. Labeling data can often be a time-consuming task. As time-series data is often long and hard to label visually based on just the time series data itself, finding changepoints in an unsupervised manner is of great interest. These algorithms will try to determine where a change point occurs based on the statistics of the data.

The survey paper [4] contains an overview of the main categories of unsupervised CPD algorithms in addition to implementing some of the algorithms in each category (depicted in Figure 2). For an in-depth comparison of the selected algorithms from each category, check out the original survey paper. The following sections will give a brief synopsis of each category.

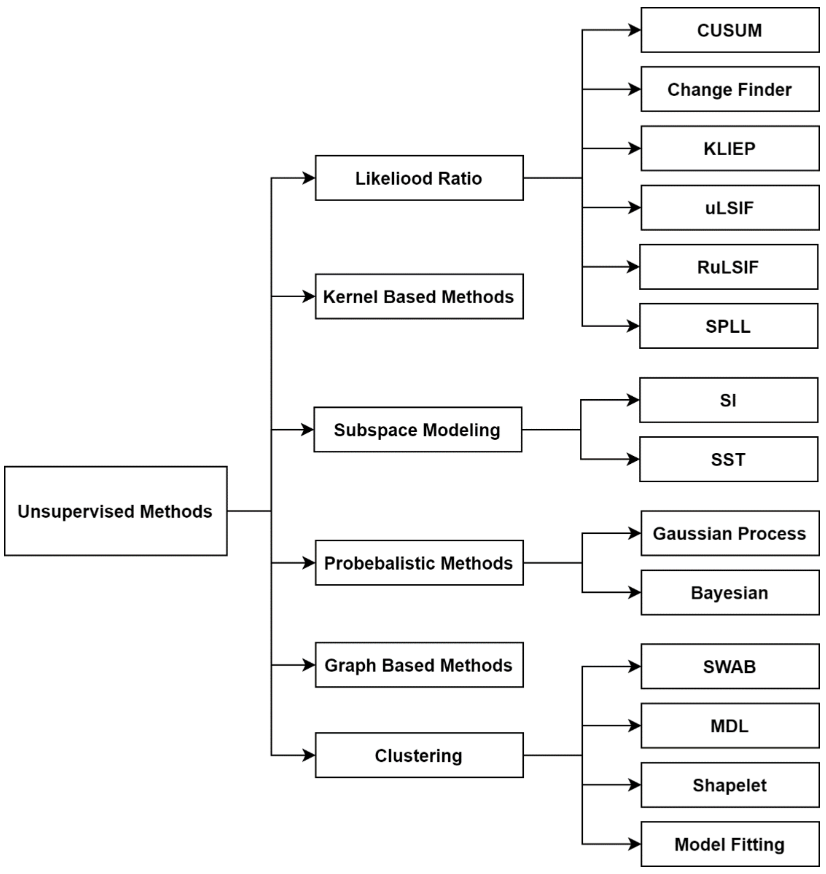

Figure 2: Unsupervised methods for change point detection. The paper [4] provides an overview of all the methods depicted in the diagram.

2.1.3.1 Likelihood Ratio Methods

One common way to detect changepoints is using Likelihood ratio methods. These methods compare the probability distributions of data before and after a potential changepoint. If the probability distributions are sufficiently different, it is concluded that the datapoint is a changepoint. Kullback-Leibler (KL) divergence is a common choice for measuring the dissimilarity between the two distributions.

2.1.3.2 Subspace Modeling

Another way to do unsupervised CPD is by using subspace modeling. Here one tries to find a mathematical model of the system states based on the finite amount of available data. The intuition is that a change in system state indicates a change point. Subspace modeling is closely related to System Identification, which is important in control theory [15].

2.1.3.3 Probabilistic methods

Probabilistic methods estimate the probability distribution of there being a changepoint at each time step. Some methods use a Bayesian approach to model the run-length distribution at each time step based on the observed data since the previous changepoint. The run-length represent the elapsed time since the last changepoint. When there is a spike in the probability of a 0-length run, a changepoint is likely to have occurred.

2.1.3.4 Kernel-based Methods

Kernel-based methods are most commonly used in supervised techniques such as SVMs but can also be used in unsupervised methods. The observations are first mapped onto a higher-dimensional feature space using kernel functions before using the Fisher discriminant ratio as a measure of the similarity between consecutive subsequences to determine if a new changepoint exists between them. The performance of kernel-based methods will be heavily dependent on the choice of kernel functions.

2.1.3.5 Graph-based methods

Another more recent approach to unsupervised CPD is graph-based methods. Here observations before and after a possible changepoint are divided into two parts. The number of edges in the graph that connects observations from the two parts decides whether a changepoint has occurred based on some threshold. Few edges increase the possibility of a changepoint. The graph is usually made using time series observations as nodes and edges connecting them based on a chosen distance metric. The background chapter will take a look at FLUSS [16], which is not specified by the authors as a graph method but uses the same concepts for doing time series segmentation.

2.1.3.6 Clustering methods

Clustering methods divide the data into a known or unknown number of clusters. If two data points close in time belong to different clusters, there is likely a changepoint between them. The work done on clustering of time series data can broadly be separated into two categories: whole clustering and subsequence clustering. Whole clustering is similar to regular clustering of discrete objects, which means it can only be used to cluster individual time series into clusters based on how similar they are. This technique is thus not applicable to time series segmentation, as the goal here is to segment one continuous time series. Subsequence clustering tries to cluster individual subsequences extracted from running a sliding window over the time series. Intuitively, this way of clustering seems to be useful for time series segmentation, as one could segment the time series based on which cluster center each subsequence belongs to. How to find these clusters, however, is a challenging problem.

Despite subsequence clustering being used as a subroutine in many other algorithms, the paper “Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research” [17] found that subsequence clustering produces essentially random cluster centers. The cluster centers are just sinusoids with random phases that average to the mean of the time series for any dataset used.

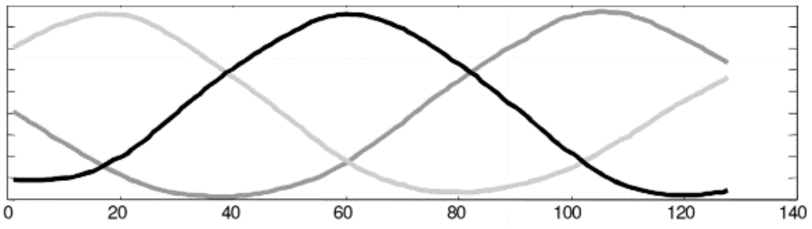

Figure 3: Plot depicting typical cluster centers generated by STS-clustering. From [17]

This was true for both k-means and hierarchical clustering, and for clustering based on a random sampling of a subset of subsequences. To get an intuition why this is the case, the NoGunGun dataset (depicted in Figure 4) from the UCR time series archive will be used.

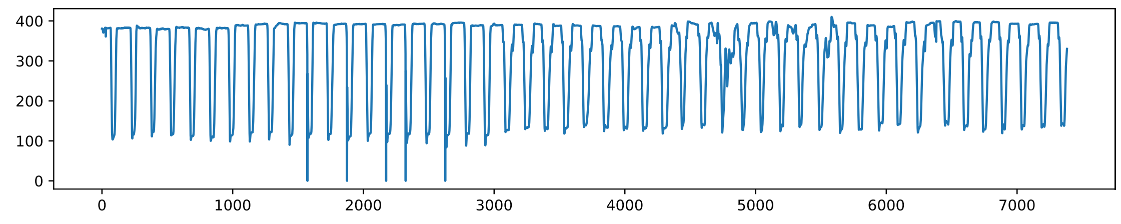

Figure 4: Plot showing the NoGunGun dataset from the UCR time series archive

As the cluster centers will be the average over all the subsequences in each cluster, this experiment tries to showcase the effect of averaging over multiple time series subsequences extracted from a sliding window. Here subsequences are extracted using a sliding window before taking the averages over n subsequence samples. The results presented in Figure 5 clearly show that given enough samples, the averages will converge to a straight line equal to the mean of the whole dataset. The out-of-phase sinusoid cluster centers mentioned above conforms to these results as their weighted average sums to a straight line. This is due to the inherent problem caused by using a sliding window to extract the subsequences. A detailed explanation of this phenomenon is provided in [17].

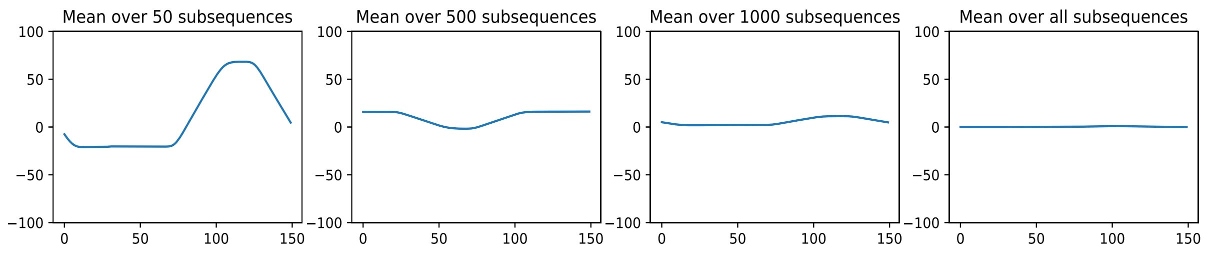

Figure 5: Plot showing the results of averaging over a selection of subsequences extracted from the NoGunGun dataset. The data is scaled to zero mean for the whole dataset.

There are no easy fixes to make subsequence clustering work, but the paper [17] introduces a novel method that makes it possible to do meaningful clustering of time series data. Instead of clustering every subsequence in the time series, this method clusters something called time series motifs. In the context of data mining, motifs are reoccurring sequences in the time series data (see example in Figure 6). These motifs can be seen as fingerprints for time series data, as all nonrandom time series data produced from the same process should contain some reoccurring patterns.

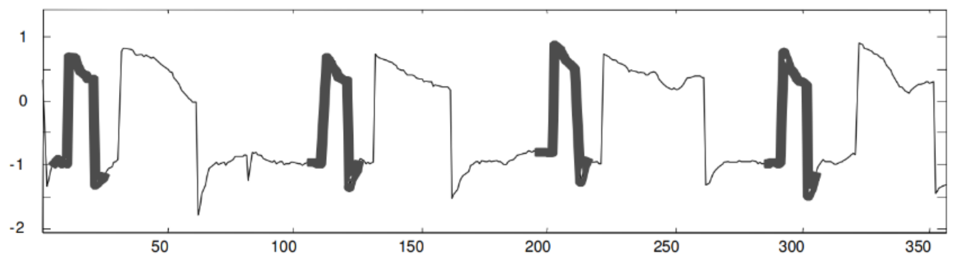

Figure 6: Example from [18] of a motif (marked in bold) that occurs four times. From [17]

The clustering algorithm presented extracts the top n motifs from the time series and does standard k-means clustering on these motifs. Notice that to do clustering in this way, the number of motifs extracted n has to be set much larger than the number of clusters k. The results were promising as the cluster centers looked like the different segments in the time series data they tested with. It is not surprising that this works since the method resembles doing whole clustering, in that it clusters short individual time series into clusters, instead of using all the subsequences extracted by a sliding window. This means that the cluster-centers are not averages over all the data but averages over motifs (similar patterns in the data).

One thing that has not been explained yet is how to discover these motifs. There are different ways of mining for motifs in time series data. One example of motif discovery is from a CPD algorithm presented in the paper “Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data” [19]. The CPD algorithm presented in [19] extracts and clusters motifs by finding motifs that minimize the number of bits required to represent the time series using Minimum Description Length (MDL). A more recent approach used for motif discovery called the matrix profile is detailed in the next chapter.

2.2 Matrix profiles

One of the core concepts in this thesis is the matrix profile [20], a data structure for time series that facilitates change point detection and motif discovery. To calculate the matrix profile, one first has to find the all-subsequence set **A** from the time series **T** by utilizing a sliding window of length **m** to extract all the possible subsequences of **T**. After this the distance matrix is found by calculating the z-normalized Euclidian distance between every subsequence in **A** with every other subsequence in **A**. Figure 7 show what the resulting distance matrix looks like for the time series NoGunGun from the UCR time series archive. Each row in the distance matrix, referred to as the distance profile **D** depicts the distance from the subsequence at the current row index to every other subsequence in the time series. When the distance profile is calculated for every index, one naturally ends up with the distance matrix.

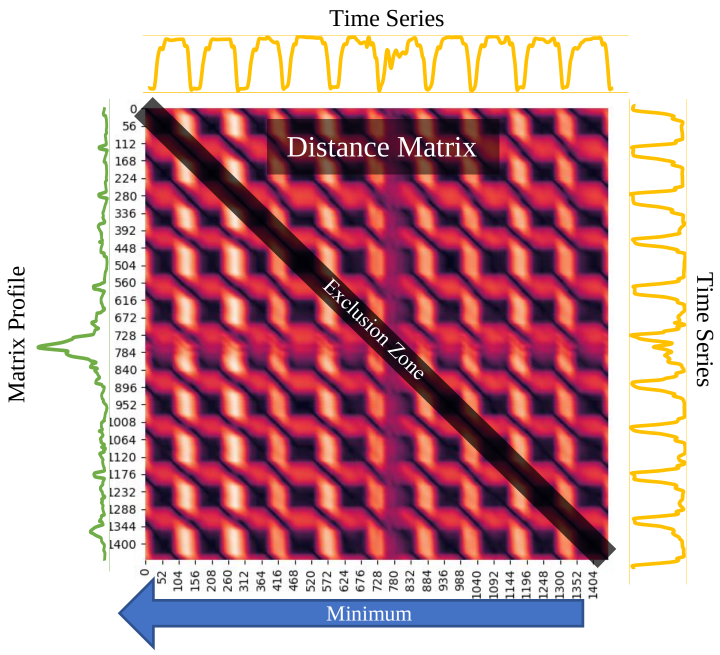

Figure 7: The distance matrix is a matrix constructed from all the distance profiles, which shows how similar each subsequence in the times series is to every other subsequence in the same time series. If one takes the minimum distance of each row, one will end up with the matrix profile. The original time series is shown in yellow and the matrix profile in green. The exclusion zone is to avoid trivial matches.

After the distance matrix is calculated, one can easily extract the matrix profile $\mathbf{P}_\mathbf{A}$, which is defined as the vector of Euclidian distances between all the subsequences in **A** with the nearest non-trivial neighbor in **A** itself [*]. The matrix profile can thus be made by extracting the smallest value from each row in the distance matrix, as this will be the distance to the nearest neighboring subsequence. There is, however, one problem with this, as the nearest neighbor to a subsequence will always be the subsequence itself because they are exactly the same. Also, subsequences that are extracted close in time will be nearly identical as well. To avoid these trivial matches, an exclusion zone around each index can be made by setting these areas to infinity. As seen in Figure 7, this exclusion zone will be along the diagonal of the distance matrix. When calculating the matrix profile P_A, one can also extract the index of the nearest neighbor for each row in the distance matrix to make the matrix profile index $\mathbf{I}_\mathbf{A}$, which is defined as the vector containing the indices of the nearest non-trivial neighbor in **A** for every subsequence in **A**. It can thus be seen as a time series containing pointers to the nearest neighbors for each subsequence. The matrix profile index $\mathbf{I}_\mathbf{A}$ will be important later as this is the basis for the segmentation algorithm FLUSS presented in section 2.3.

The matrix profile is a useful tool for motif and discord discovery. Motifs, as mentioned in 2.1.3.6, are reoccurring subsequences in the time series data. Discords can be seen as the opposite of motifs, as they are the subsequences that least resembles the other data in the time series. It is easy to get an intuition about the motifs and discords in the data by just looking at the matrix profile. In the areas with relatively low values, one knows that the subsequences in the original time series must have (at least one) relatively similar subsequence elsewhere in

[*] One can also calculate the matrix profile from two distinct time series, but this is not relevant for the purpose of this thesis

the data. These regions are reoccurring patterns and can thus be classified as motifs. While in the areas with relatively high values, one knows that the subsequence in the original time series must be a unique shape since it does not match any other subsequence. These areas are discords and can be considered anomalies in the data. Figure 8 shows an example of this. The time series data presented are mostly just repeated patterns (motifs), except for what seems like an anomaly (discord) ca. midway through. The matrix profiles are low in the areas where there seem to be motifs and high in the areas where there seem to be some kind of anomaly

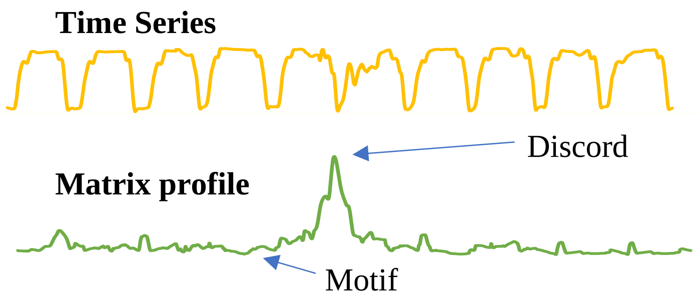

Figure 8: Plot depicting a time series and its corresponding matrix profile. The time series presented is an excerpt from the dataset NoGunGun from the UCR time series archive.

Due to these qualities, the matrix profiles can be applied to a multitude of time series change point detection problems. It could, for example, be used directly for fault or anomaly detection in various domains by looking for discord in the data. Some examples of this could be monitoring for abrupt changes in the financial markets, finding anomalies in factory machines, or finding irregularities in the activity patterns of people suffering from a mental health disorder. The motifs extracted by this algorithm can also be used for segmentation based on clustering, as seen in the previous chapter. In this thesis, it will be used as a crucial building block for the segmentation algorithms FLUSS and LS-USS.

To deploy change point detection algorithms on real-world problems, it is important that the algorithms scale well to large amounts of data. Calculating the Euclidean distance between every subsequence in a time series to every other subsequence in the same time series (as done in Figure 7) will quickly result in high computation time. There would also quickly be a problem with storing the distance matrix as this algorithm requires $O(n^2)$ space. Calculating the matrix profile for a time series with a length of 1 million samples would need around 4TB of memory. The following sections will detail how to calculate the matrix profile more efficiently.

2.2.1 Mueen’s algorithm for similarity search

Mueen’s algorithm for similarity search (MASS) is a method for finding the distance from one query subsequence **Q** in **T** to every other subsequence in **T**. The distance measure in this algorithm is the z normalized Euclidian distance.

The convolutional theorem states that convolution in time is equal to multiplication in the frequency domain. Taking the dot product between a subsequence **A[i]** and the time series **T** in the frequency domain is thus the same as taking the dot product between each subsequence

A[i] and every other subsequence in the set **A**. The time complexity of convolution is O(nm) in the time domain and O(nlogn) in the frequency domain, making it possible to drastically reduce the number of calculations by detouring into the frequency domain. Doing this will also make the computation time independent of the subsequence size used. The algorithm presented does precisely this by using the Fast Fourier transform on both the query **Q** and the time-series, multiplying the resulting spectra, before finally using the Inverse Fast Fourier transform on the product. The result is a fast and accurate way of calculating the dot product between **Q** and all the subsequences in **T**.

Algorithm name: Sliding Dot Product

SlidingDotProduct(Q,T)

```
Inputs:
    Q – query
    T – Time series
Outputs:
    QT – the sliding dot product between Q and T

1. n = Length(T), M = Length(Q)
2. $T_a$ = T appended with n zeros
3. $Q_r$ = Reverse(Q)
4. $Q_{ra}$ = $Q_r$ appended with 2n-m zeros
5. $Q_{raf}$ = FFT($Q_{ra}$), $T_{af}$ = FFT($T_a$)
6. QT = InverseFFT(Elementwisemulitplication($Q_{raf}$, $T_{af}$))
7. return QT
```

Figure 9: Outline of the SlidingDotProduct [20]

$$D[i] = \sqrt{2m\left(1 - \frac{QT[i] - m\mu_Q M_T[i]}{m\sigma_Q \Sigma_T[i]}\right)}$$

Equation 1: The z-normalized distance.

The formula for calculating the z normalized Euclidean distance for every index in the distance profile is presented in Equation 1. In addition to the dot product **QT**, a few other statistics needs to be calculated: $\boldsymbol{m}$ is the subsequence length, $\mu_{\boldsymbol{Q}}$ is the mean of **Q**, $\boldsymbol{M_T}$ is the mean of **A[i]**, $\sigma_{\boldsymbol{Q}}$ is the standard deviation of **Q**, and $\boldsymbol{\Sigma_T}[\boldsymbol{i}]$ is the standard deviation of **A[i]**. These statistics are also calculated efficiently, using a technique from [21]. The total time complexity for MASS is O(nlogn). The MASS algorithm is outlined in pseudocode in Figure 10.

Algorithm name: Mueen’s algorithm for similarity search (MASS)

```
MASS(Q,T)
Inputs:
        Q – query
        T – Time series
Outputs:
        D – A distance profile D of every query Q
    1. QT = SlidingDotProducts(Q,T)
    2. $\mu_Q$, $\sigma_Q$, $M_T$ $\Sigma_T$ = ComputeMeanStd(Q,T)
    3. D = CalcualteDistanceProfile(Q,T,QT, $\mu_Q$, $\sigma_Q$, $M_T$ $\Sigma_T$) // Equation 8
    4. return D
```

Figure 10: Outline of MASS

2.2.2 Scalable Time-series Anytime Matrix Profile

MASS is useful for finding the similarity between one query subsequence **Q** and every other subsequence in the time series. To obtain the full matrix profile $\mathbf{P_A}$ and the corresponding matrix profile index $\mathbf{I_A}$, one needs to find the nearest neighbor for every sub-sequence, which is what the *Scalable Time-series Anytime Matrix Profile* (STAMP) algorithm solves. STAMP queries every subsequence in the all-subsequence set **A** against all subsequences in **A** itself. For each query, a distance profile **D** is produced. The matrix profile for that location is updated by adding the minimum distance in **D**, and the matrix profile index is updated by adding the index of the minimum distance in **D**. When this is done for all the queries, the algorithm returns the full matrix profile. For an outline of the algorithm, take a look at Figure 11. The time complexity of this algorithm is $O(n^2 logn)$ since the MASS algorithm($O(nlogn)$) is used n times to calculate the matrix profile, but based on empirical results, the algorithms growth rate is roughly $O(n^2)$, which the authors attribute to FFT being exceptionally well optimized.

```
Algorithm name:  Scalable Time-series Anytime Matrix Profile (STAMP)

STAMP($T_A$,m)
 Inputs:
        $T_A$ – time series
        m – subsequence length
 Outputs:
        $P_A$,$I_A$ – The incrementally updated matrix profile and the associated matrix profile index
   1. $P_A$=infs, $I_A$ = zeros
   2. for idx=0 : length($T_A$)-m
   3.    D = MASS(A[idx],$T_A$)
   4.    $P_A$,$I_A$ = ElementWiseMin($P_A$, $I_A$,D,idx)
   5. end
   6. return $P_A$,$I_A$
```

Figure 11: Outline of the STAMP algorithm

Recalculating the full matrix profile every time a data point is added would take $O(n^2\log(n))$ time, which would make it difficult to use on online data streams. Because of this, the authors of [20] also introduce an algorithm for incrementally updating the matrix profile called STAMP Incremental (STAMPI). In the STAMPI algorithm, a new data point is appended to the original time series before the distance between the resulting new subsequence and the rest of the all-subsequence set **A**'s subsequences is calculated. Next, the minimum distance in **D**, and the corresponding index, is appended to the last matrix profile. The time complexity of incrementing the matrix profile is just $O(n\log(n))$ compared to $O(n^2\log(n))$ for STAMP. The STAMPI algorithm is outlined in Figure 12.

Algorithm name: Scalable Time-series Anytime Matrix Profile Incremental (STAMPI)

```
STAMPI(T_A, P_A, I_A)
Inputs:
    $T_A$ – Original time series
    t – new data point t
    $P_A$, $I_A$ – The matrix profile and the associated matrix profile index

Outputs:
    $P_{A,new}$,$I_{A,new}$ – The incrementally updated matrix profile and the associated matrix profile index

7.  $T_{A,new} = [T_A,t]$
8.  S = last subsequence in $T_{A,new}$, idx = index of S in $T_{A,new}$
9.  D = MASS(S,$T_A$)
10. $P_A$, I = ElementWiseMin($P_A$, $I_A$,D,idx)
11. $P_{A,last}$,$I_{A,last}$ = findMin(D)
12. $P_{A,new} = [P_A, p_{A,last}]$, $I_{A,new} = [I_A, I_{A,last}]$
13. return $P_{A,new}$,$I_{A,new}$
```

Figure 12: Outline of the STAMPI algorithm

2.3 Fast Low-cost Unipotent Semantic Segmentation

Fast Low-cost Unipotent Semantic Segmentation (FLUSS) [12] is a segmentation algorithm that builds upon the matrix profile. This algorithm utilizes the matrix profile indices $\mathbf{I_A}$, given by the STAMP algorithm, to segment time series data. The basic concept is quite intuitive and straightforward.

The matrix profile index can be seen as arcs or arrows that point to the most similar subsequence in the time series. In time-series data gathered from a person wearing an activity sensor and performing the activities walking and running, one would expect that most of the walking subsequences would point to other walking subsequences, and most running subsequences would point to other running sequences.

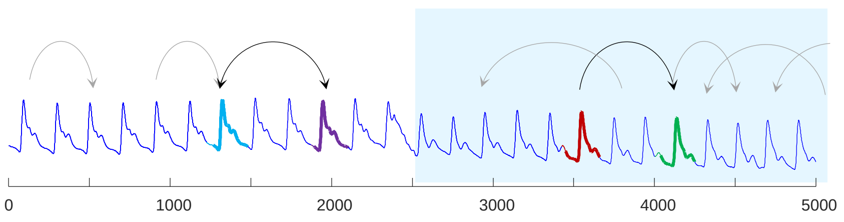

Figure 13: Visualising the arc-crossings. From [12]

If one for an index in the time series **T** counts the number of arcs crossing "over" that index, one would have a low arc-count in the areas where the time series is changing and a high arc-count in areas with a clear homogenous pattern. If these arc-crossings are counted for each index, the end result is the Arc Curve (AC).

The Arc Curve(AC) for a time series T of length n is itself a time series of also length n containing non-negative integer values. The i^{th} index in the AC specifies how many nearest neighbor arcs from the Matrix profile index spatially cross over location I [12]

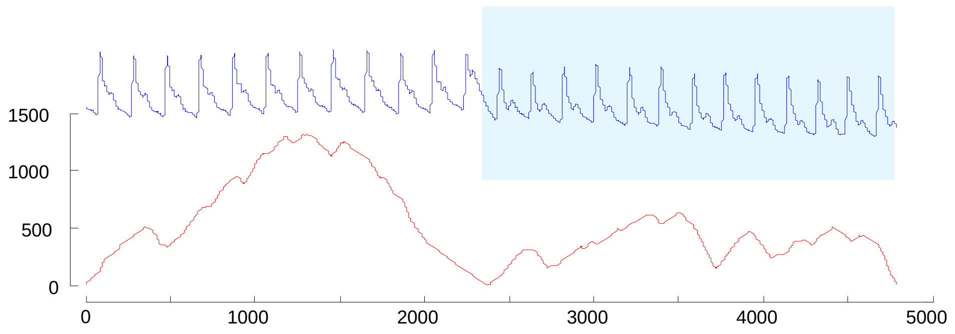

Figure 14: The top plots shows the ABP of a reclining male. At time 2400 he was rotated to a standing position. The bottom plot shows the corresponding AC. From [12]

Figure 14 shows that the AC is close to zero at the transition between the segments. However, the arc count is also low at the end and beginning of the time series due to fewer arcs that can cross the edges' indices. To compensate for this, the authors divide the AC with an inverted parabola with a height equal to half the length of the time series. This parabola is called the idealized arc curve (IAC) and is what the AC would look like for a time series with no structure, where all arc curves would just point to random locations. The empirical and theoretical IAC is depicted in the top plot in Figure 13. Dividing the AC with the IAC will normalize the time series between 0 and 1 and solve the edge effects, and the result is the

Corrected Arc Curve (CAC). A min function is also included to ensure that the CAC is between 0 and 1, even in the unlikely event that AC > IAC.

$$CAC = \min\left(\frac{CAC}{IAC}, 1\right)$$

Equation 2: The Corrected Arc Curve

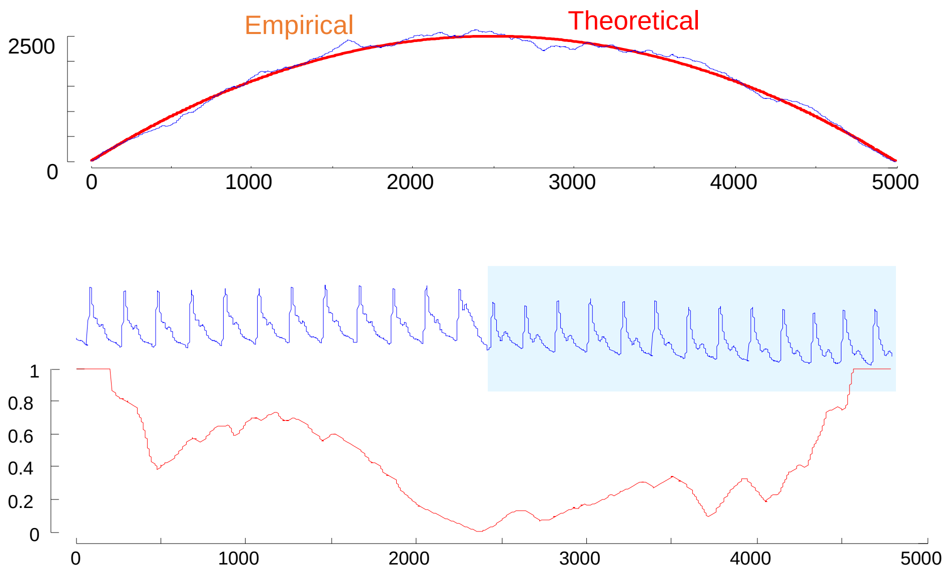

Figure 15: The top plot shows the epirical vs theoretical IAC. The bootom plot shows the corrected arc curve (CAC) and here we can that the arc curve is close to 0 when there is a change point. From [12]

2.3.1 Region Extraction Algorithm

As known from the previous chapter, a low number of arc crossings indicate a changepoint, but a method for extracting the changepoint locations has not yet been described. There are many ways to do this; a simple and straightforward approach is described in the paper [12]. The extraction algorithm proposed is called Regime Extracting Algorithm (REA), and searches for the k lowest "valley" points in the CAC. To extract these "valley" points trivial minimums have to be avoided. If **x** is the lowest point on the CAC, **x**-1 and **x**+1 is likely the second and third-lowest point. To avoid all the changepoints ending up in one "valley"(ref Figure 17), they set an exclusion zone around the already detected "valley" points. The algorithm is outlined in Figure 16.

Algorithm name: Regime Extracting Algorithm (REA)

REA (CAC,num_regimes,L)

Inputs:
CAC – a Corrected Arc Curv
numRegimes – number of regime changes

Outputs:
locRegimes – the location of the regimes

```
1.  locRegimes = empty array of length numRegimes
2.  for i=1 : numRegimes
3.      locRegimes(i) = indexOf(min(CAC))
4.      Set exclusion zone of 5×L          // To prevent matches to "self"
5.  end
6.  return locRegimes
```

Figure 16: Outline of the REA algorithm

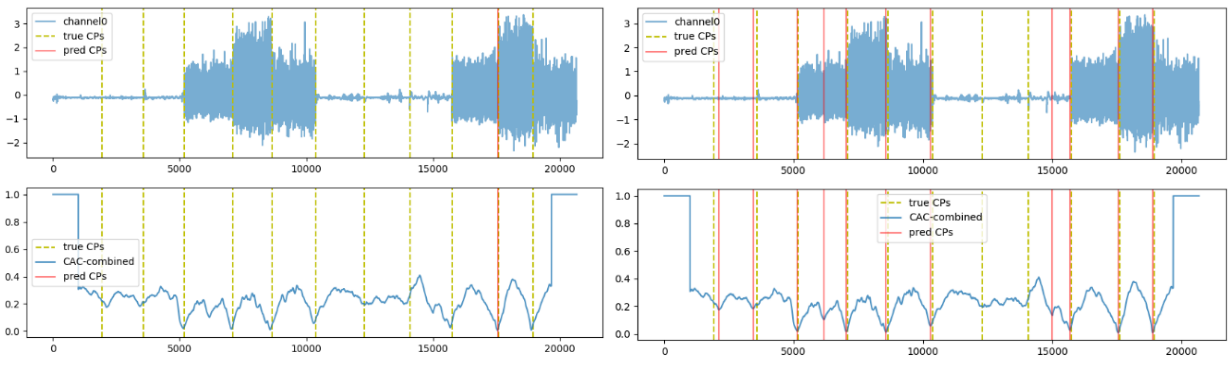

Figure 17: The top plots show channel 0 of subject 14 from the dataset "A Public Domain Dataset for Human Activity Recognition Using Smartphones" [22]. The bottom plots show the CAC produced by the FLUSS algorithm. The two left plots show the predicted changepoint when not using an exclusion zone, while the right plots show the predicted change points when using an exclusion zone. When there is no exclusion zone, all changepoints end up in the same "valley".

2.3.2 Fast Low-cost Online Semantic Segmentation

The FLUSS algorithm works well for batch data. It will, however, struggle to handle streaming data. Adding a point to the time series and calculating the nearest neighbor to the new subsequence will only use O(nlog(n)) time using the MASS algorithm, but maintaining the CAC also require an ejection of the leftmost point (earliest data point) in the matrix profile when adding a new one. This operation takes O(n^2) time because the whole matrix profile needs to be updated since every subsequence could point to the ejected point.

An online version of FLUSS called *Fast Low-cost Online Semantic Segmentation* (FLOSS) solves this issue by only use arcs pointing to the right. If all arcs point to the right, no arc can point to the leftmost point; ejecting it would then take O(1) time. Maintaining the one-directional CAC_{1D} would then take a total of O(nlogn). The IAC will also be more skewed to the right since the rightmost part of the CAC_{1D} will have a higher chance of arc crossings, as shown in Figure 18. Using the one-directional CAC helps accomplish online streaming functionality but will come at the cost of a slight loss in performance.

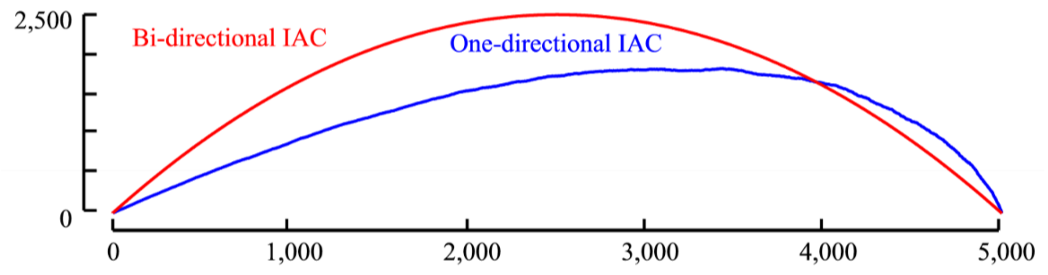

Figure 18: The Bi directional IAC vs. the one-directional IAC (IAC_{1D}). From [12]

2.3.3 Locality – Temporal constraint

In some time-series datasets, the activities or events are repeated multiple times with other segments in between. In these cases, the CAC would not be a good indicator of a regime change. This is because the arcs would have practically the same likelihood of pointing to

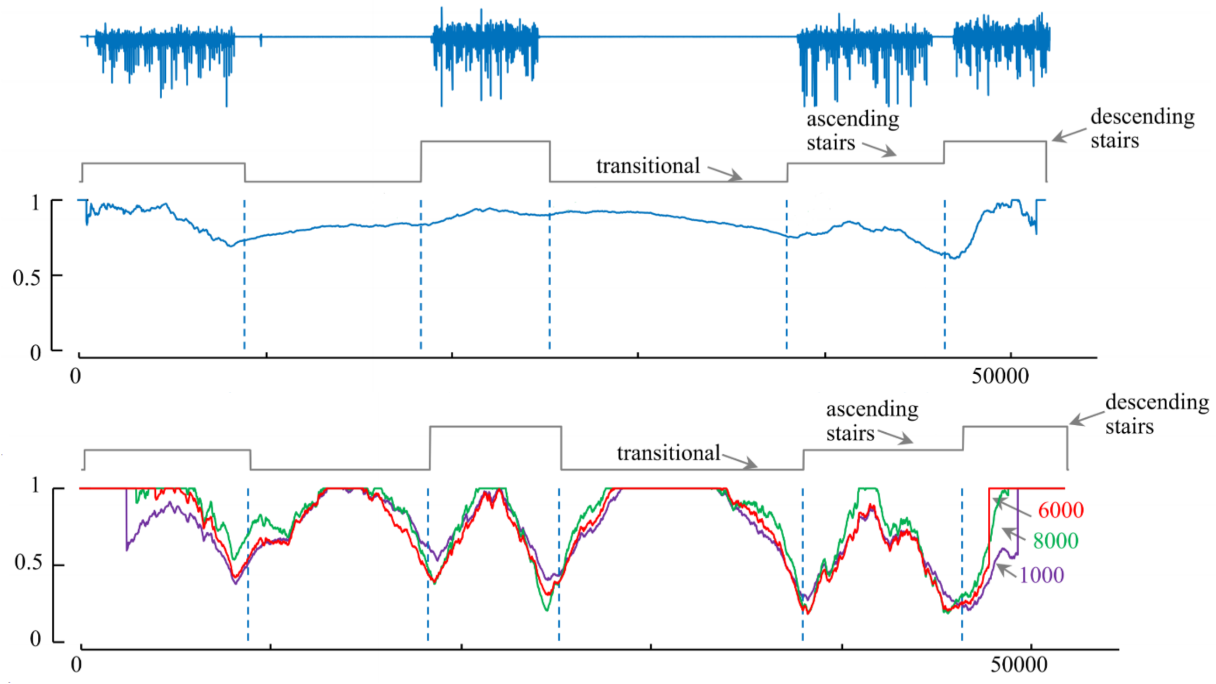

Figure 19: Top plot shows the accelerometer data of a who from PAMAP dataset. Person acending staris, descending stairs and another segment of ascending stairs again with trasistional activities which includes some noise data in between. Middle plot shows the CAC when not including a temporal constraint. While the bottom plot shows the CAC produced when usetting the temporal constraint to 1000, 6000 and 8000. From [12]

subsequences in a similar segment as pointing to subsequences in the same segment. The middle plot in Figure 19 shows that the CAC cannot locate some of the visually obvious change points because of the repeated activity “ascending stairs”. The solution proposed is to use a temporal constraint (TC) that limits how far in time arcs can point and will ensure that each index in the CAC is only based on a local area around that index. An added benefit is that the computing time will be reduced, with the only disadvantage being that an extra parameter must be specified. The authors claim that the temporal constraint can be set to circa maximum expected segment length. Therefore, setting this parameter requires some domain knowledge about the distribution of changepoints in time. However, as shown in the bottom plot in Figure 19, this parameter is not very sensitive to changes as setting TC to 1000 yields nearly the same results as setting it to 8000 for that particular time series. For both FLUSS and FLOSS, one need to normalize the arc curve (AC) using the ideal arc curve (IAC), which in this case would be a uniform distribution of height equal to the TC size as each index in the CAC can maximally be crossed by the total number of arcs inside the TC.

2.3.4 Multidimensional time series data

Many datasets contain data from different sensor types and sensor locations. When localizing change points, it is usually beneficial to use multiple sensors as these often contain more information about the underlying process. A straightforward way to do this is to calculate the CACs for each channel separately and take the mean over the CAC's to produce a single combined CAC. As the CACs are normalized between zero and one, this also works for data from sensors with different sampling rates. The authors of [12] presented an example of this using a dataset where a person performed the activities "basketball forward dribble", "walk with wild legs", and "normal walk". The person was wearing two sensors: one on the arm and one on the foot. Figure 20 shows that using data from only the hand or foot is not enough to differentiate between the three activities. Combining the two CACs, by taking the mean, makes it possible to segment all three activities.

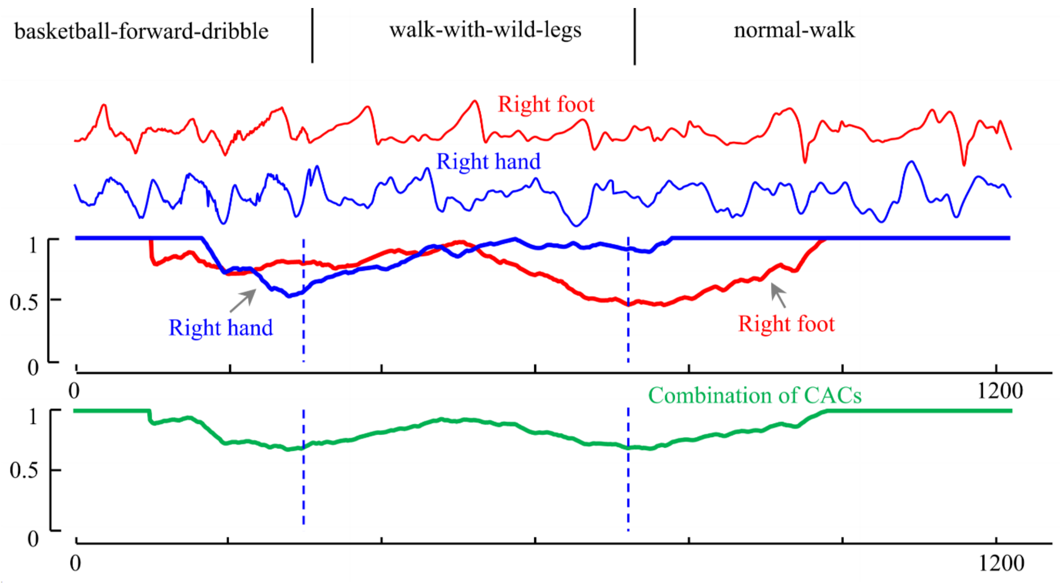

Figure 20: Example of multidimensional time series segmentation of three activities: "basketball forward dribble", "walk with wild legs" and "normal walk". Combining the two CACs, by taking the mean, makes it possible to segment all three activities. From [12]

There will, however, arise some problems when dealing with data containing many channels, as some of them might not contain information that is helpful for segmenting out activities, or they might be heavily correlated. When taking the average over all the channels, these "not so useful” channels can drown out the useful channels as they are all equally weighted when taking the average. The authors acknowledge this problem and recommend that when dealing with high dimensional data, one should search for or manually find the most useful subset of channels and remove the rest. The following section, about representation learning, will include methods that could help solve this problem by removing redundant and correlated information between channels by automatically extracting lower-dimensional encodings of the original multidimensional data.

2.4 Representation Learning

Representation learning, or feature learning, is a central concept in machine learning. Real-world data is often unstructured and high-dimensional. Thus, most tasks require us to design features that describe the data before doing tasks like classification. The problem with this is that one needs to hand pick these features and that the features selected will usually be sub-optimal. In supervised learning, one successful way of doing representation learning is by using deep neural networks. These networks contain layers that act as different abstraction levels and helps the model extract the most valuable features for the given classification task. Deep models like convolutional neural networks learn to detect low-level features, like edges in the first layer(s) and higher-level features, like facial features, in the “deeper” layers. The output from the final layer is a feature representation of the original input. The main benefit of these models is the capability to learn the representation in conjunction with the final decision boundary, which means one do not have to design the features manually.

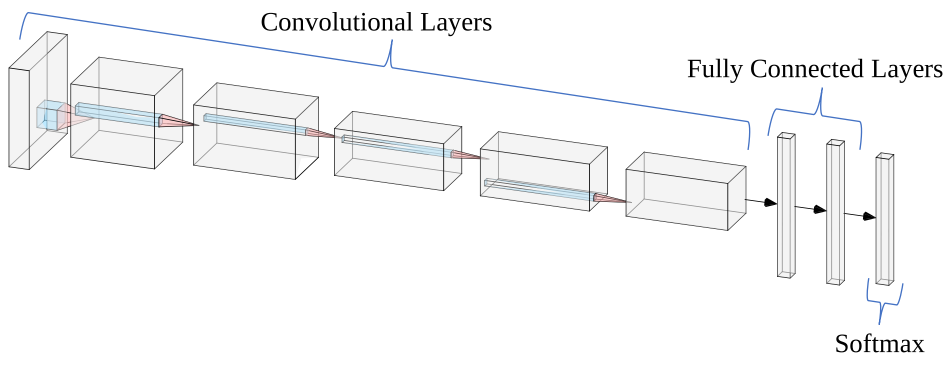

Figure 21: Alexnet [23]. Example of a deep convolutional neural network. The earlier layers will learn to detect low-level features, while the later layers will combine these lower-level features into high-level features.

There are also unsupervised methods for finding feature representations from “raw data” using dimensionality reduction. Principal Component Analysis (PCA) is a common technique for reducing the dimensionality of feature spaces in an unsupervised manner. Another method for reducing the dimensionality of the data is autoencoders. Autoencoders are neural networks that consist of an encoder and decoder network (example in Figure 22). In training, the networks are coupled together and try to recreate the input by minimizing the reconstruction error. The encoder network usually encodes the data into a smaller representation with some information loss, often referred to as a latent representation, while the decoder will try to recreate the original input data from this encoded representation. The training of the whole network is done like a regular feed-forward-network, through backpropagation. Minimizing the error between the input and output will result in the latent representation being a low-dimensional representation of the original input data. There are numerous ways of implementing autoencoders depending on the data type and use case. The next section will examine how representation learning techniques like autoencoders have been applied to changepoint detection in previous research.

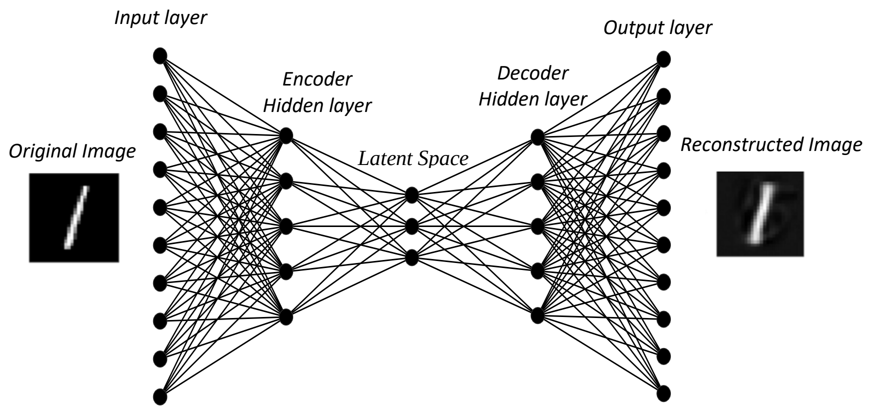

Figure 22: Example of a basic autoencoder architecture. The image outputted by the network closely resembles the input image. The latent layer in the middle of the network would here serve as a good low dimensional encoding of the input image.

2.4.1 Representation learning for change point detection

One popular supervised method for segmenting data is the well-known convolutional segmentation network U-net [24]. Even though this network has been mostly utilized on image data, there are examples of U-net-inspired architectures that work well on time series data [19] [20]. Since U-net requires to be trained on labeled data and this thesis is focused on unsupervised segmentation methods, further details on U-net will not be provided here.

There has been some research focused on applying representation learning to typical change point detection problems. Autoencoders are, for example, often used for anomaly detection [27]–[29]. The most common approach is to train the autoencoder on a dataset with none to relatively few anomalies. When doing inference, one can use the reconstruction error as a measure of how likely the current data is to contain an anomaly. The reconstruction error will be small when the input data is normal and high when there is an anomaly. This is because the model has learned to represent the normal data better than data with anomalies. While this approach successfully finds changepoints that are anomalies, it is not straightforward to use this method for segmentation because the different segments are a natural part of the data. An autoencoder trained on all the data would also have a low reconstruction error for all the different segments, and the reconstruction error would thus not be a good indicator of these changepoints.

It would be reasonable to think that measuring the distance between vectors of low-level representations of the data would benefit semantic segmentation. This method is often used in natural language processing to represent the words' semantic similarity. The idea is that words with a similar context should be represented by word vectors/embeddings close to each other. One of the most popular models for making these word embeddings is called Word2Vec [30]. Here, a neural net is used to map each word into a compact representation called a vector. The mapping tries to ensure that words with similar semantic meaning will be close to each other in the vector space. The article "Representation learning for unsupervised heterogeneous multivariate time series segmentation and its application" [31] tries to segment out driving patterns based on collected sensor data from driving vehicles. Here they use the word2vec model to make an encoded representation of time series data consisting of both categorical and numerical data in varying scales. They use this model because there is no universal distance measure that works for similarity search and segmentation between multivariate data. By applying this word embedding method, the data are transformed into continuous vectors, allowing them to be segmented using conventional distance metrics such as Euclidian distance. This article shows that representation learning holds great potential for the segmentation of multidimensional time series data collected from multiple sources.

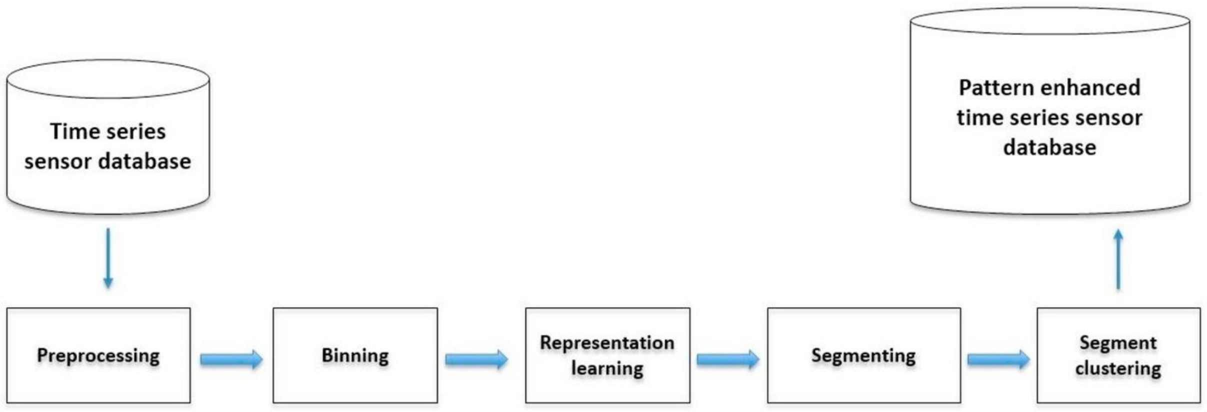
Time series sensor database
Pattern enhanced time series sensor database
Preprocessing
Binning
Representation learning
Segmenting
Segment clustering

Figure 23: Block diagram showing the modules used in the article "Representation learning for unsupervised heterogeneous multivariate time series segmentation and its application" [31]. It shows representation learning being used to combine channel data from different sources before segmentation. In the article, they used the word embedding model word2vec for the representation learning module.

There are also other ways to use representation learning on multidimensional data. One example is to use PCA to project the multidimensional data onto the principal components as they did in the article "A PCA based Change Detection Framework for multidimensional data streams" [32]. Multichannel data can often have some correlated channels that do not yield much useful information for segmentation tasks. Say one has a dataset where the data is collected from two sensors at each arm and one sensor on the leg. For many tasks, the sensors on the arms would be more correlated with each other than the leg sensor. If the segmentation model does not capture this correlation, the leg's sensor data can easily be "drowned out". Using representation learning techniques like PCA, these changes in channel correlations can be captured. Another benefit is that one can get a more general and compact representation of the time series data by only projecting the data to the n first principal components. As stated before, using the latent representation found by autoencoders have many of the same benefits as PCA. The paper “Time Series Segmentation through Automatic Feature Learning” [12] uses an autoencoder to do automatic feature representation/feature learning before segmenting the time series. As the method of segmenting time series data based on extracted latent representation will be central for this thesis, the next section will give a more detailed overview of the methods used in the paper.

2.4.2 Time Series Segmentation through Automatic Feature Learning

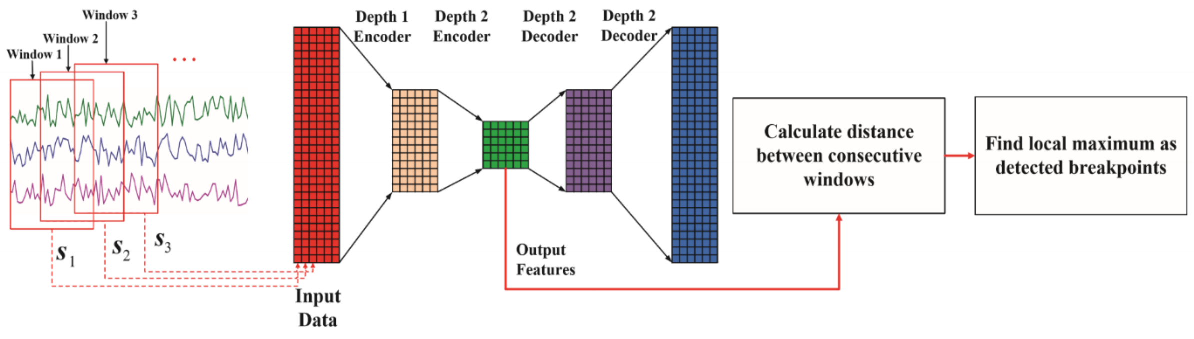

Figure 24: Model depiction from the paper “Time Series Segmentation through Automatic Feature Learning” [12]

The input to the autoencoder is the time series partitioned into subsequences of length $\mathbf{N_w}$. These subsequences might have some overlap, as shown in Figure 24. The channels of the subsequences are concatenated into a single one-dimensional vector. Thus, the input size for time step **t** will be the subsequence length $\mathbf{N_w}$ multiplied by the number of channels $\mathbf{N_C}$. The encoder $\mathbf{E_{ec}}$ will encode the input at time step $\mathbf{s_t}$ to a feature representation $\mathbf{f_t}$, while the decoder $\mathbf{D_{ec}}$ maps the feature representation back to the original input space. The parameters of the autoencoder consist of weights **W** and bias $\mathbf{b_e}$, and the decoder consists of weights **W’** and bias $\mathbf{b_d}$. The autoencoder used in the article uses tied weights, which means that the weights in the decoder are the transposed weights from the encoder (**W’=W$^\mathbf{T}$**). The activation function used is sigmoid.

$$f = E_{nc}(s) = sigmoid(W \times s + b_e)$$

Equation 3: Equation for encoded representation *f*

$$\tilde{s} = D_{ec}(f) = sigmoid(W' \times s + b_e)$$

Equation 4: Equation for reconstructed input $\tilde{s}$

The objective of the autoencoder is, as usual, to minimize the reconstruction error between the input $\mathbf{s_t}$ and the reconstructed input $\tilde{\mathbf{s}}$. The weights and biases are initialized randomly and are updated using standard gradient descent.

After training, the autoencoder is used to detect what the authors have defined as breakpoints, which are “human-specified changepoints”. The algorithm used for finding these breakpoints is outlined below.

```
Algorithm name: Breakpoint Segmentation
1. for each subseqence $s_t$:
2.     Extract features $\mathbf{f_t}$ according to Equation 3
3.     Compute distances $Dist_t$ between consecutive feature vectors according to Equation 5
4.     if $Dist_t$ is a local-maximal distance:
5.         Classify t as breakpoint
```

Figure 25: Outline of the Breakpoint Segmentation algorithm

$$Dist_t = \frac{\|f_t - f_{t-1}\|_2}{\sqrt{\|f_t\|_2 \times \|f_{t-1}\|_2}}$$

Equation 5: Formula for calculating the distance between the current and previous feature at time t

The numerator in the distance metric (Equation 5) is the Euclidean distance between features at consecutive times, and the denominator is a normalization term. The breakpoint segmentation algorithm will compute the distances at each time step, resulting in a distance curve. A high distance relative to the local area will indicate a change in the underlying process and is thus classified as a breakpoint.

A sensitivity analysis is also provided to give practical guidelines for choosing hyperparameters. They used three datasets from different real-world domains (EEG eye state data set [33], UCI human activity recognition data set [34], and the DCASE2016 sound data set [35]) for the sensitivity analysis. They found three main factors that could impact the performance: subsequence size, model depth, and feature size. As one would expect, the optimal subsequence size varied between the datasets. The best window size for the EEG data was 25, 400 for the UCI data, and 20000 for the DCASE data. The experiments found that the best performance on all three datasets was achieved using a model with two hidden layers. The feature size, the ratio between the feature representation and the input (dim($\mathbf{f_t}$)/dim($\mathbf{s_t}$)), had a negligible effect on the performance. The feature sizes tested were in the range between $0.05\times\mathbf{s_t}$ to $0.5\times\mathbf{s_t}$.

2.5 Evaluation Metrics

The evaluation methods are essential to get a fair comparison between CPD algorithms. Many papers written about changepoint detection do not contain concrete evaluation metrics but rely solely on visual inspection[4], making it hard to compare algorithms objectively. Visual inspection can also be quite time-consuming to do when ranking multiple algorithms on a wide array of data from different domains. A fair evaluation metric is thus crucial for evaluating CDP algorithms.

The evaluation criteria must be based on what type of output the CPD/segmentation algorithms yield. If the CPD model is a binary classifier (decide for each time step if a changepoint is present) or change point detection is done with varying levels of precision (decide if changepoint occurs within a time interval), it is possible to use standard machine learning metrics. What follows are examples of evaluation techniques that can be used with these types of models.

	Classified as non-CP	**Classified as CP**
True Non-CP	True negative (TN)	False positive (FP)
True CP	False negative (FN)	True positive (TP)

Accuracy is the ratio of correctly classified data points to the total number of data points.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Equation 6

Sensitivity/Recall is the portion of the positive class (will in this case be True CP) that was classified correctly.

$$Sensitivity/Recall = \frac{TP}{TP + FN}$$

Equation 7

Precision is the ratio of true positive (True CP) data points to total data points classified as CP.

$$Precision = \frac{TP}{TP + FP}$$

Equation 8

The **F1 score** is a balanced metric that combines precision and recall by calculating the ratio of weighted importance of precision and recall.

$$F_1 = (1 + \beta)^2 * \frac{Precision * Recall}{\beta^2 * Precision + Recall}$$

Equation 9: β is positive real factor that is chosen such that Recall is β times more important than precision

Another balanced metric often used in machine learning is the **Matthews correlation coefficient.**

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Equation 10

When the class distribution is highly skewed, some of these metrics will give very little information about the actual performance of the classifier. If a negative sample is 100 times more likely to appear than a positive one, the model's accuracy would be nearly 100% by classifying every sample as negative. This scenario is often the case when doing changepoint detection because the number of changepoint is usually much smaller than the number of non-changepoints. Looking at the confusion matrix and using metrics that are class balanced like the F1 score will give a better impression of the actual performance of the algorithm. What metrics to choose will also be dependent on the real-world use-case of the algorithm as different misclassifications could have different costs. If one were to make a CPD system that detects lethal anomalies in ECG data recorded from patients' hearts and alerts a human expert for proper examination, missing a true changepoint would have much more dire consequences than detecting a changepoint where there is none. In this case, one would probably prioritize a high sensitivity over a high precision.

Plotting the true-positive rate (TPR/sensitivity) against the false-positive rate (FPR/precision) provides an intuition about the trade-off between the two metrics. This type of plot are referred to as a Receiver Operating Characteristics (ROC) curve. The optimal algorithm would have a high TPR while still maintaining a low FPR, which means that the ROC curve for a good CPD algorithm should be pushed towards the left upper corner. A more concrete metric is AUC (Area Under the Curve), which is, as the name implies, the area under the ROC curve. The closer the ROC curve is to the top-left corner, the closer the AUC is to 1. In "Time series segmentation through Automatic feature learning" [36] (detailed in Section 2.4.2), they used ROC curves to find the best model depth for their autoencoder. Figure 26 depicts the ROC curves for different model depths found by running the algorithm on the UCI dataset [33].

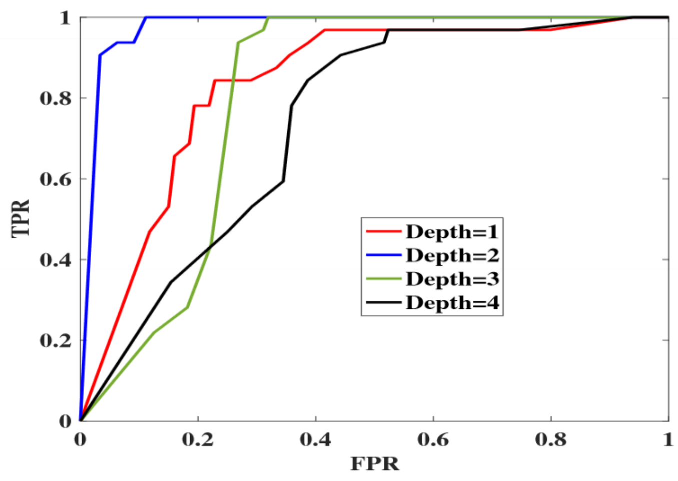

Figure 26: Example from the article "Time series segmentation through Automatic feature learning"[36] (detailed in Section 2.4.2) where they made a ROC curve to help evaluate the best model depth for an autoencoder based on the performance on the UCI dataset [33]

Often in time series changepoint detection and segmentation, the changepoints' exact location is not as important as the time difference between the predicted and actual changepoint. In these cases, metrics that utilize the distance to the true changepoint locations instead of the class labels are more telling of the performance. The most commonly used metrics in this context are the Mean Square Error (MSE) and the Mean Absolute Error (MAE).

MAE calculates the distance in time by finding the absolute difference between each ground truth CP and the nearest predicted CP before averaging over the number of true changepoints.

$$MAE = \frac{\sum_{i=1}^{N_{GT}} |CP_{pred} - CP_{actual}|}{N_{GT}}$$

Equation 11: Where N_GT is the number of ground truth changepoints

MSE is similar to MAE except that it uses the squared difference instead of the absolute difference. Because the error measure will increase quadratically, MSE will penalize outlier CPs more than MAE.

$$MSE = \frac{\sum_{i=1}^{N_{GT}} (CP_{pred} - CP_{actual})^2}{N_{GT}}$$

Equation 12:Where N_{GT} is the number of ground truth changepoints

There are also other similar metrics like Mean absolute error (MSA), Mean signed difference (MSD), Root mean square error (RMSE), and Normalized root mean square error (NRMSE).

Prediction ratio is the number of predicted CPs that corresponds to ground truth CPs divided by the ground truth number of CPs. The optimal prediction ratio will be 1. If the model outputs more than 1 the model overpredicts and if outputs less the model underpredicts.

$$Prediction\ ratio = \frac{N_{pred}}{N_{GT}}$$

Equation 13: Where N_{pred} is the number of predicted changepoints and N_{GT} is the number of ground truth changepoints.

If the model underpredicts the distance error metrics MSE or MAE will usually be worse, but if the model overpredicts, they will increase their score using these metrics. For example, if the model predicts all data points to be changepoints, both the MSE and MAE would be 0, and this does not represent the actual performance. To get a single metric that captures both the importance of predicting the right amount of changepoints and the distance from ground truth changepoints, the article [37] introduces a new measure they call **prediction loss**. This metric combines MSE and the prediction ratio into one measure. Both a prediction ratio bigger and smaller than one would increase the total prediction loss. The CPD algorithms need to have a good prediction ratio and MSE score to get a good score on this metric.

$$Prediction\ loss = \left|1 - \frac{N_{pred}}{N_{GT}}\right| \times MSE$$

Equation 14: Where N_{pred} is the number of predicted changepoints and N_{GT} is the number of ground truth changepoints.

A scoring metric that is less sensitive to outliers can be made by changing the MSE score to MAE. The metric will in this thesis be referred to as **prediction loss MAE.**

$$Prediction\ loss\ MAE = \left|1 - \frac{N_{pred}}{N_{GT}}\right| \times MAE$$

Equation 15: Where N_{pred} is the number of predicted changepoints and N_{GT} is the number of ground truth changepoints.

In the article [12], they introduce a scoring metric similar to MAE, referred to as **Regime Score.** Like MSE, this metric finds the absolute difference between each predicted CP and the nearest ground truth CP before summing them. This sum is then divided by the number of true segments and the length of the time series resulting in a normalized score between zero and one.

$$Regime\ score = \frac{\sum_{i=1}^{N_{GT}} |CP_{pred} - CP_{actual}|}{N_{GT} * n}$$

Equation 16: Where N_{GT} is the number of ground truth changepoints, and *n* is the length of the time series

2.6 Spectral density estimation

In spectral density estimation, one wants to automatically determine the frequency content of a signal by estimating the power spectral density. The power spectral density of a wide sense stationary process is defined as the Fourier transform of the autocorrelation of a signal.

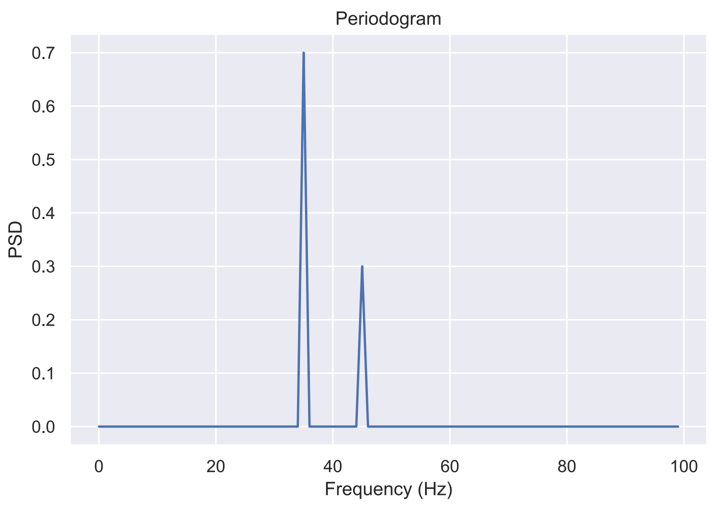

Figure 27: A power spectrum of two sinusoids. From the periodogram it is easy to see that the sinusoids have frequencies around 35Hz and 45Hz

One application of spectral density estimation is period estimation. A distinct high peak in the power spectrum at frequency ***f*** is a good indication of a periodic signal with a period equal to ***1/f***. As sampled time series data does not have an infinite number of samples, and the data often contain some noise, the exact power spectrum for a given process cannot be found directly, which means one has to estimate it.

2.6.1 Periodograms

The most straightforward method of estimating this is the periodogram. It is close to the power spectra definition since it is defined as the Fourier transform of the autocorrelation of the sampled time series.

$$\mathcal{P}_{xx}(\omega) = \sum_{k=-N+1}^{N-1} R_{xx}(k)e^{-j\omega k}$$

Equation 17: Formula for calculating the periodogram of a time series x

$$R_{xx}(k) = \begin{cases} \frac{1}{N}\sum_{n=0}^{N-1-k} x(n+k)\overline{x}(n), & k \geq 0 \\ \overline{R}_{xx}(-k), & k < 0 \end{cases}$$

Equation 18: Formula for calculating the autocorrelation of time series x

This estimator is both biased and suffers from high variance. The bias comes from "indirectly" using a rectangular window when doing the Fourier transform, which results in that the expected value of the periodogram is the true power spectrum convolved with a squared sinc function.

2.6.2 Windowed Periodograms

One way to reduce the bias is to use **windowed periodograms** ($\mathcal{P}_{xx}(\omega)^{w}$). This method reduces the bias by applying a window function to the time series before the Fourier transform. Figure 28 shows the different window functions in both the time and frequency domain, and Figure 29 shows the effect of using the windows on a signal composed of two sinusoids with added white Gaussian noise. When applying the window functions, the ripples around the main lobe are reduced, leading to a less biased estimate. Doing this comes at the cost of making it harder to separate close frequencies in the power spectral density estimate.

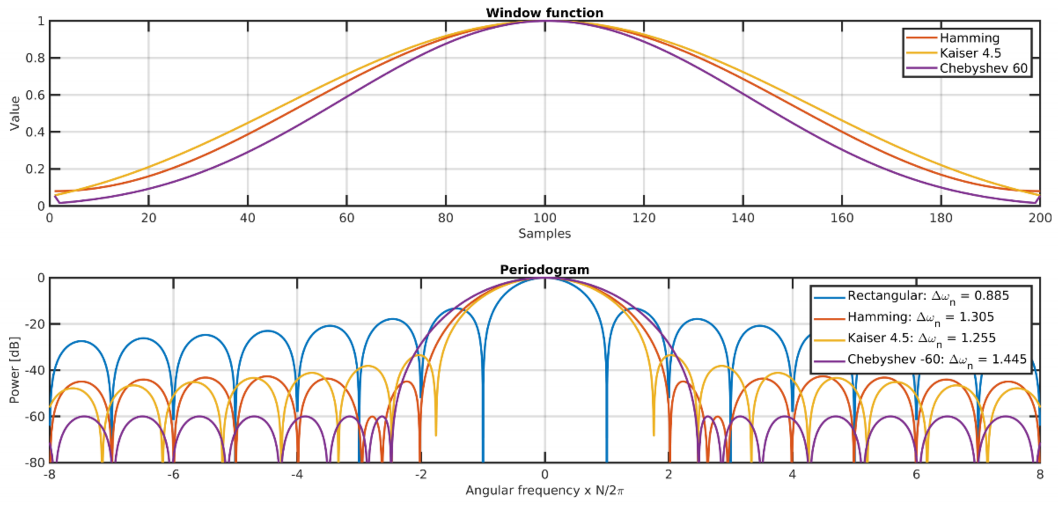

Figure 28: Window functions in time and frequency domain. From lecture "Spectral Estimation I" in INF4480, slide 40

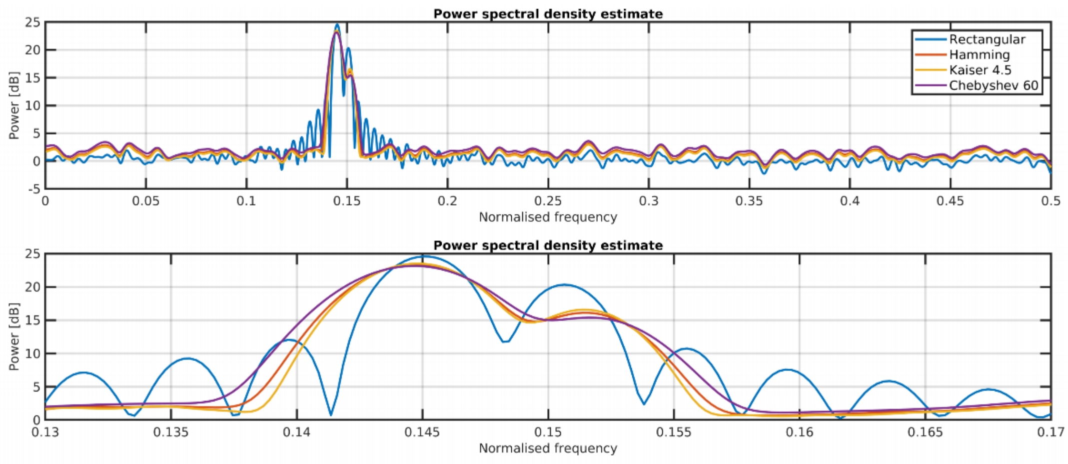

Figure 29: Two close sinusoids with frequencies at 0.145 and 0.150 with added white Gaussian noise. It shows the effect of using different window functions. From lecture "Spectral Estimation I" in INF4480 ,slide 41.

2.6.3 Welch's method

A common way to reduce variance in statistics is to average over multiple realizations of a process. It is possible to make multiple realizations of a time series by splitting it up into smaller parts. After doing this, one can estimate the PSD for each part before averaging over the resulting PSDs to reduce the variance of the PSD-estimator.

$$P_{xx}(\omega) = \frac{1}{K}\sum_{i=0}^{K-1} P_{xx}^{m,w}(\omega)$$

Equation 19: Equation for Welch's method for estimating the power spectra. $P_{xx}^{m,w}(\omega)$ is the the i'th windowed periodogram

Welch's method[38] tries to reduce both the bias and variance when estimating the power spectrum density by averaging over short windowed periodograms. Doing this comes at the cost of a lower spectral resolution because each PSD estimate uses fewer samples than would have been the case when using the whole time series. The main idea is to divide the time series into K segments with size L and offset each segment with D datapoints, calculate the windowed periodograms, and average over the resulting periodograms (see Equation 19). The partitioning of the time series is depicted in Figure 27.

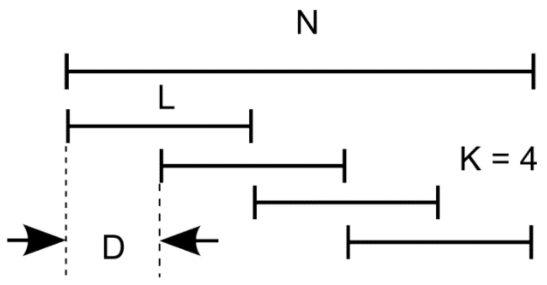

Figure 30: Visual depiction of how the time series is divided into sub-parts when using the Welch Method

3 Methods

This chapter is divided into three main sections based on the algorithm types: *data scalers* (3.1) which scale the data to values that are easier to process, *CPD algorithms* (3.2) that output a likelihood for changepoints occurring at each time step, and *change point extractor algorithms* (3.2) that extracts the location of the change points using the output from the CPD algorithms. The predicted segments are the areas between the predicted changepoints. Notice that the CPD algorithms, as defined here, do not output the location of changepoints but the likelihood of there being a changepoint at each time step. Table 1 shows an overview of the implemented algorithms. The underlined algorithms in Table 1 are new methods developed during this thesis, while the others are existing methods detailed in the background chapter. The following sections will go through the implementations of the various algorithms and describe the new methods and modifications to existing approaches.

Algorithms	Full Name	Section
Data Scalers		**3.1**
NoScaler	-	3.1.1
StandardScaler	-	3.1.2
MinMaxScaler	-	3.1.3
RobustScaler	-	3.1.4
CPD algorithms		**3.2**
<u>LS-USS</u>	Latent Space Unsupervised Sematic Segmentation	3.2.2
<u>LS-USS online</u>	Latent Space Unsupervised Sematic Segmentation online	3.2.2
FLUSS	Fast Low-cost Unipotent Semantic Segmentation	3.2.3
FLOSS	Fast Low-cost Online Semantic Segmentation	3.2.3
TSSTAFL	Named after paper: “Time Series Segmentation through Automatic Feature Learning”	3.2.4
<u>FLUSS-Welch</u>	-	3.2.5
CP extraction algorithms		**3.3**
REA	Region Extraction Algorithm	3.3.1
<u>LREA</u>	Local Region Extraction Algorithm	3.3.2
<u>LTEA</u>	Local Threshold Extraction Algorithm	3.3.3

Table 1: Overview of the implemneted algorithms. The <u>underlined</u> algorithms are new methods devloped during this thesis.

3.1 Data Scalers

Four data scaling techniques are applied to data for comparing the segmentation algorithms on the different datasets. These methods are implemented in scikit-learn [41] and are widely used for machine learning and data mining tasks. The scaling is done independently over all channels, and the statistics used for scaling the data are extracted from the training data. What follows is an overview of the different scaling methods as they are implemented in scikit-learn.

3.1.1 NoScaler

NoScaler does not perform any scaling

$$x_{scaled} = x$$

Equation 20

3.1.2 StandardScaler

StandardScaler scales the channels to zero mean and unit variance by subtracting the mean and dividing by the standard deviation.

$$x_{scaled} = \frac{x - \mu}{\sigma}$$

Equation 21

This method will alter the relative distances between samples.

3.1.3 MinMaxScaler

MinMaxScaler scales the channels to range between 0 and 1 based on the minimum and maximum values of the data.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Equation 22

The relative distance between samples will stay the same, but the method is not very sensitive to outliers.

3.1.4 RobustScaler

RobustScaler uses statistics that are sensitive to outliers. It removes the median and scales the data to IQR (Interquartile range). The IQR is the range between the 1st and 3rd quartile.

$$x_{scaled} = \frac{x - x_{median}}{x_{Q3} - x_{Q1}}$$

Equation 23

3.2 Change point detection algorithms

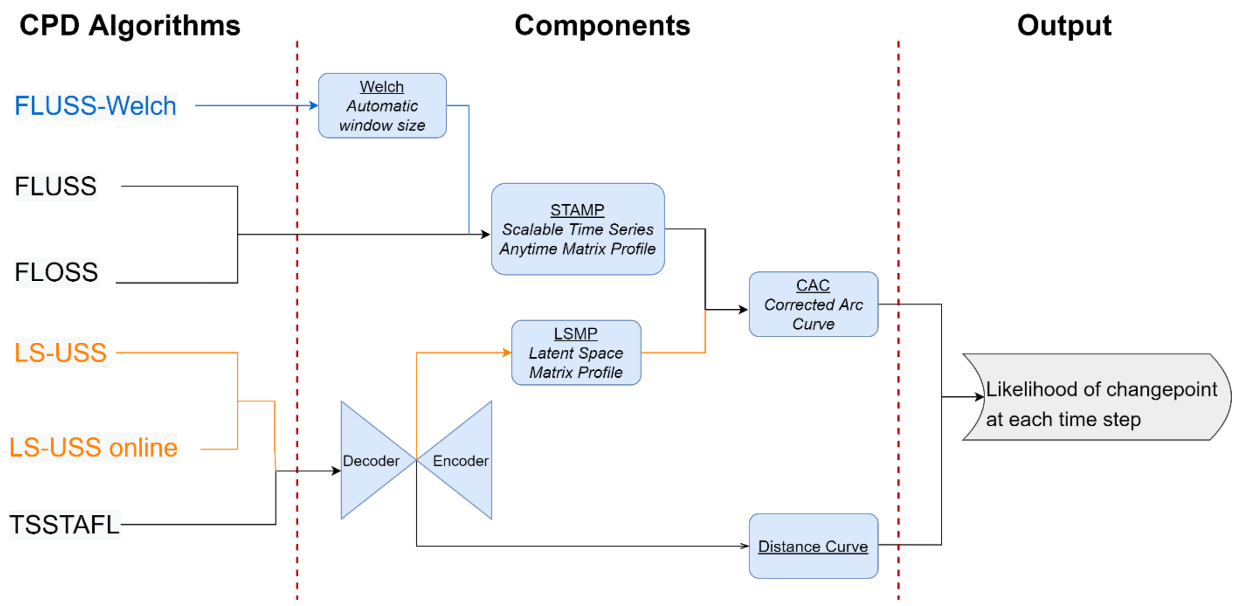
CPD Algorithms
Components
Output
FLUSS-Welch
FLUSS
FLOSS
LS-USS
LS-USS online
TSSTAFL
Welch Automatic window size
STAMP Scalable Time Series Anytime Matrix Profile
LSMP Latent Space Matrix Profile
CAC Corrected Arc Curve
Decoder
Encoder
Distance Curve
Likelihood of changepoint at each time step

Figure 31: Flow chart depicting the different model components used for each of the CPD algorithms. The colored algorithms are part of this thesis contribution.

The CPD algorithms in this thesis receive scaled time-series data as input and output a measure of the likelihood of a changepoint occurring at each time step. Figure 31 depicts the different CPD algorithms implemented and their components (marked in blue). The arrows pointing from the CPD algorithms and through the components represent the order in which the components are applied. As many of the components are shared between the different algorithms, the next subsection will go through the implementation of all the main components used, namely STAMP, the autoencoder, LSMP, CAC, the Distance Curve, and Welch. After this, the implementation of all the CPD algorithms (which utilize the components) is presented.

3.2.1 Components

3.2.1.1 STAMP implementation

STAMP (2.2.2) is the algorithm used for calculating the matrix profile. To implement STAMP, a python library called stumpy [39] is used. This library contains implementations of various ways to calculate exact or approximate matrix profiles depending on the hardware and time available. The specific stumpy methods used were based on the methods mstump and gpu-stump. The mstump method is for parallel computation of the multi-dimensional matrix profile and works well on multicore CPUs. As the name suggests, gpu-stump is a highly parallelized method for computing the matrix profile using GPUs.

As neither of these methods supported using a temporal constraint (TC), this was added by modifying the methods to only calculate the distance profile D over a local window in the MASS algorithm (2.2.1). The online version only has access to previous data. The equations used are shown in Equation 24.

$$D[i] = \sqrt{2m\left(1 - \frac{QT[i] - m\mu_Q M_T[i]}{m\sigma_Q \Sigma_T[i]}\right)}$$

$$i \in \begin{cases} [argmax(0, Q_{idx} - TC)\,, argmin(N, Q_{idx} + TC)], & if\ offline \\ [argmax(0, Q_{idx} - TC)\,, Q_{idx}], & if\ online \end{cases}$$

Equation 24: Equation for calculating a temporarily constrained distance profile

In addition to returning the matrix profile $\boldsymbol{P}$ and matrix profile index $\boldsymbol{I}$, the methods return the left and right matrix profile indices $\boldsymbol{I_R}$ $\boldsymbol{I_L}$, which are the indices to the nearest neighbors (smallest Euclidean distance) to the right and left.

3.2.1.2 Autoencoder Implementations

Two versions of the autoencoder have been implemented; the fully connected model from the paper “Time Series Segmentation through Automatic Feature Learning” [37] (described in section 2.4.2) and a convolutional model. The latter is implemented to test if modeling the input's temporal characteristics can help find a good latent representation.

3.2.1.2.1 Fully Connected Model

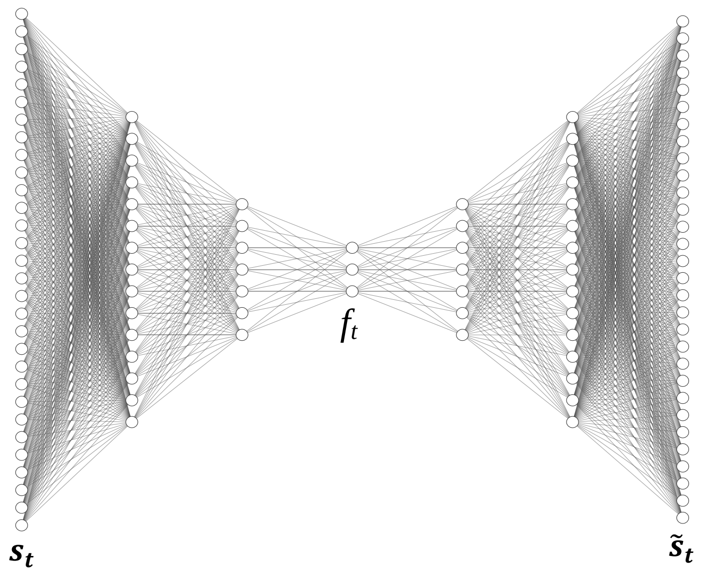

Figure 32: Fully Connected model for an input size of 30. Inspired by the paper “Time Series Segmentation through Automatic Feature Learning” [37].

In the original paper, the fully connected model is implemented using two hidden layers, transposed weights for the decoder, and the sigmoid activation function. The loss function is the mean square error, and the optimization function used to update the weights and biases is Adam [40]. The feature size, the ratio between the feature representation and the input (dim($\mathbf{f_t}$)/dim($\mathbf{s_t}$)), is set to 0.1. Figure 32 shows the model architecture for an input size of 30.

3.2.1.2.2 Convolutional Model

The input to the fully connected model is made by concatenating the $\mathbf{N_C}$ channels of the subsequence to a one-dimensional vector. However, in the convolutional model, the multichannel subsequence is processed as is, which means the input shape is [**N_C, N_W**]. The two first layers of the encoder are one-dimensional convolution layers, both with stride=2, and the convolution is done over the time axis. Next, the feature map is flattened to one dimension and passed to two fully connected layers, which outputs the latent feature $\boldsymbol{f_t}$. Then the latent feature is sent through the decoder to reconstruct the original input. In the decoder, the latent feature is first passed through transposed fully connected layers and reshaped back into 4×**N_c** channels before being passed through two one dimensional transposed convolutional layers that scale it back to the same shape as the original multidimensional subsequence. Transposed convolution does not guarantee to recover an earlier feature map but can be considered a learnable up-sampling operation that makes it possible to recover an earlier feature map's shape. Figure 33 shows a visualization of two-dimensional transposed convolution.

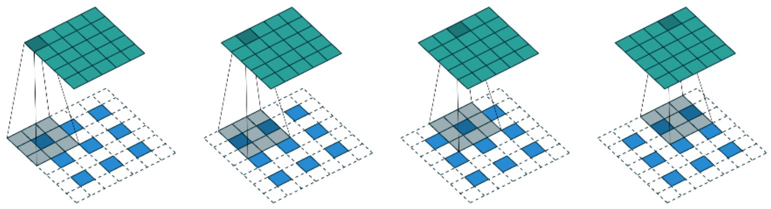

Figure 33:: The transposed of convolving a 3x3 kernel over a 5x5 input padded with a 1x1 border using two 2x2 strides. This examples shows 2D convolution, but the same principles apply to 1D convolution—example from the journal article [41].

All the layers, except the last layer, use the ReLU activation function. The last layer must use a linear activation function for it to be able to reconstruct the original input. The model is trained in the same way as the fully connected model using mean square error as loss function and Adam as optimizer. The feature size, which is now defined as the ratio (dim(**f_t**)/ **N_C**×**N_W**), is set to 1/6. For a more detailed overview of the model, have a look at Table 2.

Table 2: Overview of the convolutional model from input to output.

Layer	Type	Input Channels	Output Channels	Kernel size	Stride	Padding	Activation
Hidden Layer 1 Encoder	Convolution	N_c	2×N_c	3	2	1	ReLU
Hidden Layer 2 Encoder	Convolution	2×N_c	4×N_c	3	2	1	ReLU
Reshape Encoder	Reshaping	4×N_c	1	-	-	-	-
Hidden Layer 3 Encoder	Fully Connected	1	1	-	-	-	ReLU
Hidden Layer 4 Encoder	Fully Connected	1	1	-	-	-	ReLU
Hidden Layer 1 Decoder	Transposed Hidden Layer 4 Encoder	1	1	-	-	-	ReLU
Hidden Layer 2 Decoder	Transposed Hidden Layer 3 Encoder	1	1	-	-	-	ReLU
Reshape Decoder	Reshaping	1	4×N_c	-	-	-	-

Hidden Layer 3 Decoder	Transposed Convolution	4×N_c	2×N_c	3	1	1	ReLU
Hidden Layer 4 Decoder	Transposed Convolution	2×N_c	N_c	3	2	1	Linear

Figure 34 shows the latent representation and reconstruction of two subsequences from the EMG dataset. As one can see, the reconstruction is a bit less detailed than the original subsequence but contains the same general patterns. Remember that the goal here is not to get the best possible reconstruction of the subsequence but rather to find a good encoded representation of the subsequence for segmentation.

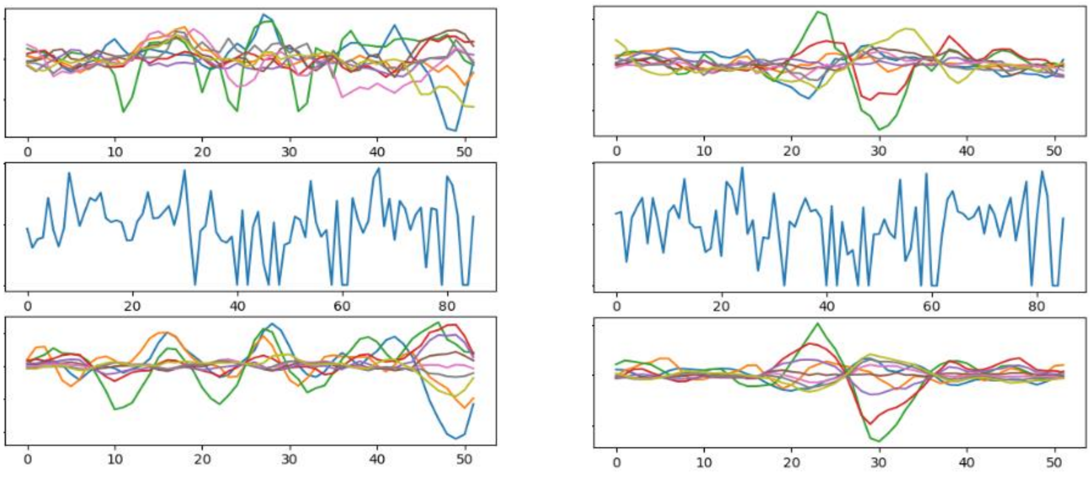

Figure 34: The top plots show all ten channels of two subsequences of size 52 taken from the Long-Term 3DC dataset [42]. The middle plots show the latent representation of the subsequences. The bottom plots show the reconstructed subsequence. The autoencoder used is the convolutional model.

3.2.1.3 Latent Space Matrix Profile (LSMP)

The latent space matrix profile is, as the name implies, a latent representation of the matrix profile. To produce the Latent Space Matrix Profile, the latent representation of the all-subsequence set **A** is needed. The all-subsequences set **A** of a time series **T** is an ordered set of all possible subsequences of **T** obtained by sliding a window of length **m** across **T**. The latent all subsequence set **F** is defined as the latent representations of the all-subsequence set **A**. Each latent representation $\boldsymbol{f}$ in **F** is found by feeding each subsequence **s** in **A** through a pretrained autoencoder ($f = E_{nc}(s)$). Remember that the autoencoder will represent multidimensional input as a one-dimensional feature in latent space.

The latent space matrix profile is defined by the same logic used for calculating the regular matrix profile in Section 2.2. With the only difference being the use of latent representation instead of subsequences. The latent space matrix profile $\mathbf{P_F}$ is thus defined as the vector of Euclidian distances between all the latent features in **F** with the nearest non-trivial neighbor in **F** itself. The corresponding latent space matrix profile index $\mathbf{I_A}$ is defined as the vector containing the indices of the nearest non-trivial neighbor in **F** for every latent representation in **F**.

In section 2.2, it was mentioned that calculating the matrix profile directly using convolution in the time domain was impractical due to high time and space complexity. The solution to this was SlidingDotProduct (outlined in Figure 9), which uses FFT to reduce the computation time substantially. Doing this is only possible if the all-subsequence set can be made using a sliding window on the time series. This is true for the all-subsequence set **A** but not for the latent set **F**, which means it is not possible to use the SlidingDotProduct method when calculating the latent space matrix profile $\mathbf{P_F}$. The following paragraphs will show that it is possible to work around this by using highly parallelized computation and exploiting the temporal constraint utilized when making the CAC.

The formula for the Euclidian distance between the vectors x and y is given by:

$$d(x, y) = \sqrt{(x - y)^2}$$

Equation 25

If the formula is expanded, this can be calculated by taking the dot product of the vectors themselves and the dot product between the vectors.

$$\sqrt{(x - y)^2} = \sqrt{x^2 + y^2 - 2xy}$$

Equation 26

By exploiting this, and that each distance calculation can be done independently, it is straightforward to vectorize the problem and run the calculations in a parallelized fashion. The GPU is perfect for this, as it is highly specialized for parallel processing of floating-point operations, making it great for doing simple math operations on vectorized data. The implementation in this thesis is based on the cdist function in the machine learning framework PyTorch [43]. This function can calculate the Euclidian distance between pairs of vectors and supports GPUs.

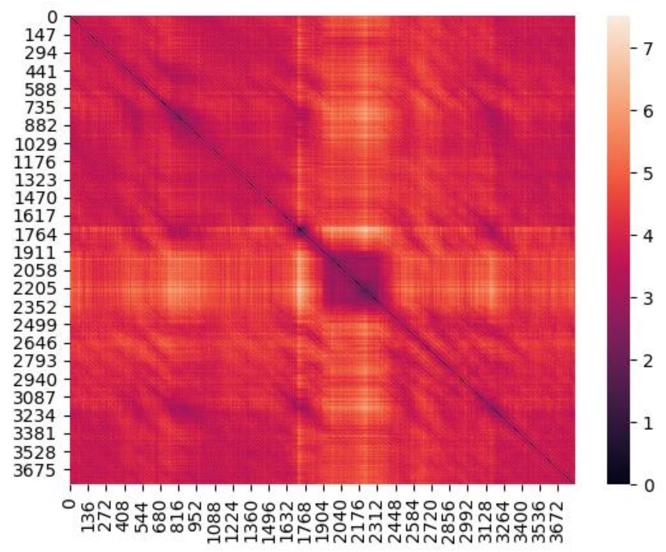

Figure 35: Example of the full distance matrix used to calculate matrix profile $\mathbf{P_{FF}}$ from a time series of length 4000. Calculating the matrix profile for this length is not that time-consuming on a modern GPU,

but handling larger time series would get difficult as the distance matrix would require exponentially more space and calculation time.

As seen in section 2.3.1, doing time series segmentation based on the matrix profile and CAC only benefits from finding arc crossings inside the current and next segment. If the time series includes repeated similar segments, the CAC might even be rendered useless. Therefore, the authors introduced the temporal constraint, which also radically decreases the computation time. A TC can be applied to the matrix profile $\mathbf{P_F}$, which means that only the distances between the latent features located inside the temporal constraint need to be calculated. Applying the temporal constraint results in a linear time and space complexity.

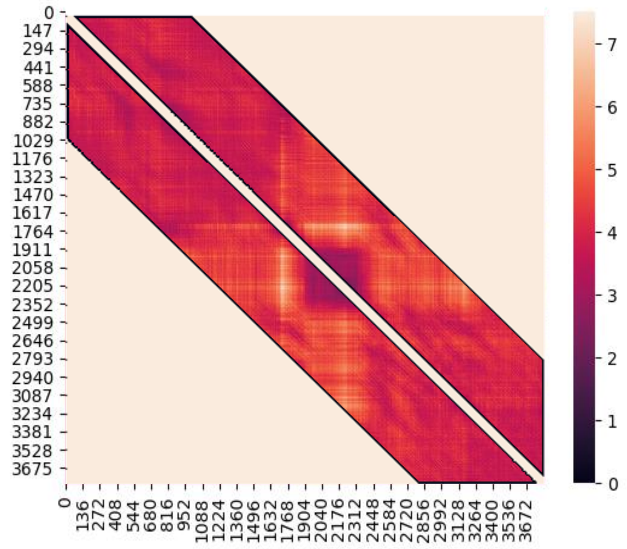

Figure 36: Results of only calculating the distances inside a temporal constraint TC of 800. To exclude trivial matches, the line along the diagonal is also not calculated.

The Euclidian distances are not normalized like in the STAMP algorithm. This is because the CAC only needs the matrix profile indices $\mathbf{I_F}$ and not the similarity score provided by the matrix profile (the index to the shortest distance is the same whether the distance is normalized or not). To find the matrix profile $\mathbf{P_F}$ and the matrix profile indices $\mathbf{I_F}$ the distance matrix is collapsed by saving the minimum distance and the corresponding index at each time step.

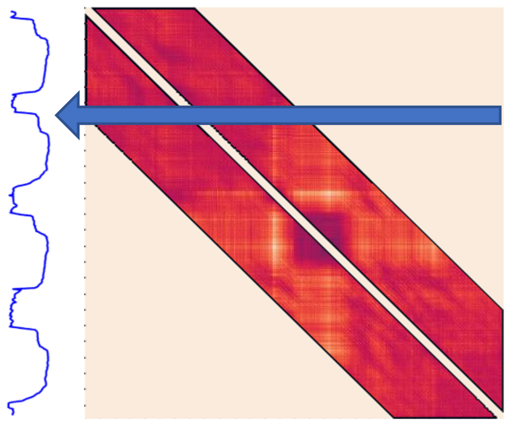
Figure 37: Shows the collapse of the distance matrix into the matrix profile. This is done by extracting the minimum value of each row. (P.S The matrix profile shown here is just for visualization and does not represent the actual matrix profile)

For long time-series data, a memory problem might arise because the algorithm saves 2*TC datapoints per time step to make the temporally constrained distance profile. However, using the TC makes it possible to collapse the distance profile before processing the whole time series. The algorithm for doing this is outlined in Figure 38.

Algorithm name: Batched Collapse

```
1. Calculate the distance matrix D until a specified amount of time steps $t_{lim}$
2. Collapse the distance matrix to get the matrix profile P
3. Keep P, Delete D
4. While $t_{lim}$ < lenght(T):
    a. $t_{lim\_prev} = t_{lim}$
    b. $t_{lim} += t_{lim}$
    c. Calculate the new distance matrix from $t_{lim\_prev}$-(TC*2-1) to $t_{lim}$
    d. Collapse the new distance matrix $D_{new}$ to get the matrix profile $P_{new}$
    e. Where P and $P_{new}$ overlaps in time -> Keep min(P,$P_{new}$), Delete
       max(P,$P_{new}$),
    f. P = $P_{new}$
    g. Keep P, Delete D
```

Figure 38: Outline of the algorithm Batched Collapse

As seen in the Batched collapse algorithm, collapsing the matrix profile before processing the entire time series comes at the cost of recalculating the last *(TC*2-1)* timesteps of the matrix profile. This recalculation is necessary because the first time-steps in the new matrix profile can point back to the old one. Figure 39 and Figure 40 give a visual intuition about why this is necessary.

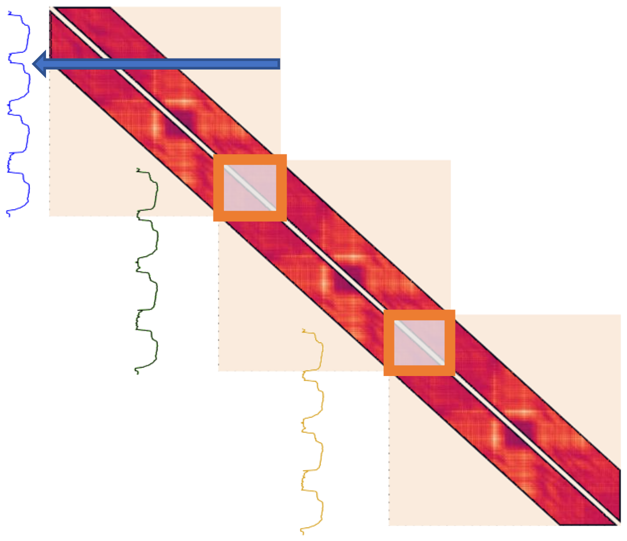

Figure 39: Shows the algorithm batched collapse on a time series of length 10400. The calculation is divided into three batches to reduce memory requirements for long-time-series data. Each batch produces one matrix profile. The orange squares show the area that must be recalculated when doing a "batch collapse".

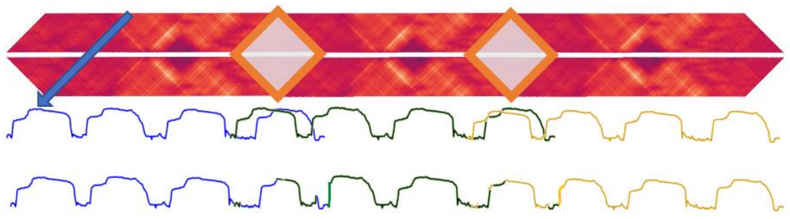

Figure 40: Top figure shows the temporarily constrained matrix profile of a time series of length 10400. The second figure the three batched matrix profiles calculated. The bottom figure shows the full matrix profile after taking the minimum value in the overlapping area.

The "batch collapse" could also be used to make the algorithm ε real-time by accumulating a batch of data before adding it to the end of the matrix profile. Doing this will entail a tradeoff between batch size and calculation time because every time a batch is added, *TC*2-1* timestep must be recalculated.

In section 2.3.1, FLOSS was made to work online by only using the right-pointing arcs. This trick can be borrowed to make the LSMP updateable in real-time as well. If one only looks for the closest distance forward in time, no subsequence can have a nearest-neighbor in the previous matrix profile. An example of this is shown in Figure 41. Doing this makes it possible to increment the matrix profile without having to do extra calculations. As with FLOSS, using only forward-pointing arcs will usually yield a slight decrease in performance at detecting change points compared to using both forward and backward-pointing arcs.

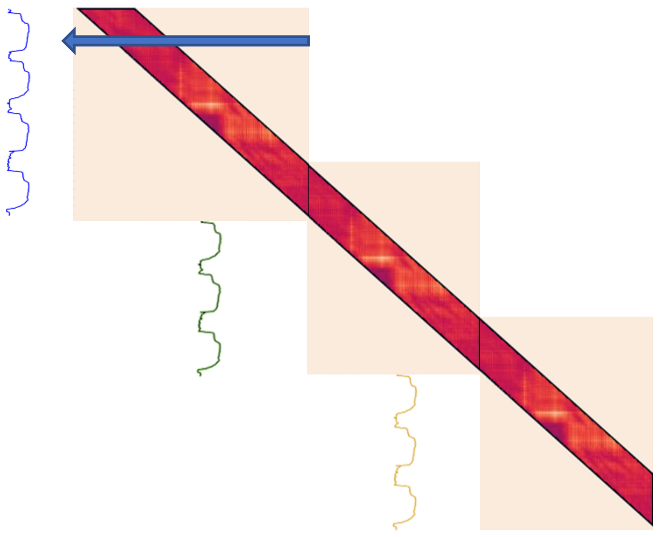

Figure 41: Shows the online version of LSMP on a time series of length 10400. No recalculation is needed when updating the distance matrix

3.2.1.4 CAC Implementation

The CAC, detailed in section 2.3, is found by counting the number of arc crossings over each index using the matrix profile index. In this thesis, a method from the stumpy API called _*cac* is used to produce the CAC. Support for multidimensional time-series data by averaging over the CACs calculated from each channel is added to the original implementation.

3.2.1.5 Distance Curve

The distance curve used in TSSTAFL is made from measuring the Euclidean distance between consecutive features and is implemented according to Equation 5 in section 2.4.2.

For the CAC, low values indicate changepoints. For the distance curve, however, high values indicate changepoints. Because of this, a modification to the distance curve has been made that allows for using the extraction algorithms interchangeably between the different CPD algorithms used in this thesis:

$$distance\ curve_{new} = 1 - distance\ curve_{original}$$

Equation 27

This modification will flip the distance curve upside down, and the minimum value will now indicate a change point. Since the distance curve is already normalized between zero and one, the maximum value will now be one, and the minimum value will be zero. For simplicity, both the distance curve and the corrected arc curve will hereafter be referred to as CAC.

3.2.1.6 Welch Implementation

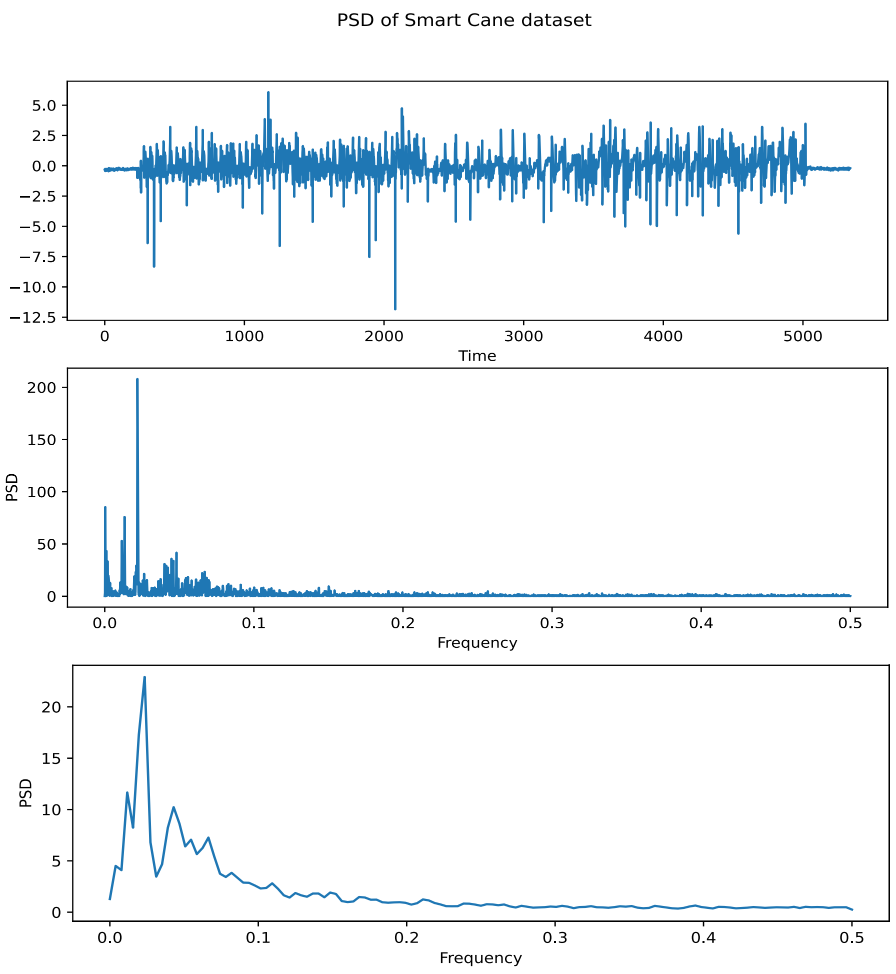

Figure 42: Top figure shows the original "Smart Cane" time series from the UCR dataset. The middle figure shows the periodogram of the time series. The bottom figure shows the periodogram created using Welch's method

As known from section 2.3, the authors of the article "Domain agnostic online semantic segmentation for multi-dimensional time series" [12] presented a segmentation algorithm FLUSS that only requires one parameter: the subsequence length. They also state that this parameter can be set to roughly a period. In [12], they found this subsequence length based on visual inspection. If the signal is relatively periodic over time, there should be a way to estimate this parameter automatically. Section 2.6 showed that one way to find the periodicity of a signal is by searching for the highest peak in the signal's power spectrum. A high peak in the power spectrum at frequency ***f*** is a good indication of a periodic signal with a period equal to ***1/f***. There are multiple ways to estimate the signal's power spectral density, depending on how much variance or bias can be tolerated. Since the FLUSS and FLOSS algorithms are not that sensitive to minor variations in subsequence length, one does not have to pinpoint the signal's exact periodicity, which makes using welch optimal since it trades some of the spectral resolution for reduced variance and bias. One can see this clearly in Figure 42, where the standard periodogram has high spectral resolution and more variance, while the modified periodogram from welch has a lower spectral resolution but less variance.

For the implementation of Welch's method, the open-source Python library SciPy [44] is used with the following hyperparameters: the length of each segment (L) is set to 256, the offset of each segment (D) is set to 128, and the window function used is Hann.

3.2.2 Latent Space Unsupervised Semantic Segmentation

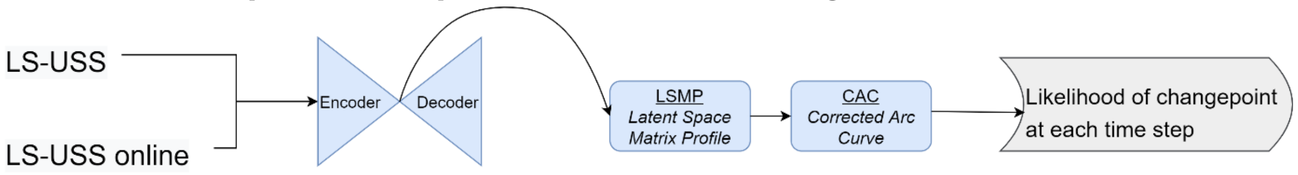
LS-USS
LS-USS online
Encoder
Decoder
LSMP
Latent Space Matrix Profile
CAC
Corrected Arc Curve
Likelihood of changepoint at each time step

Figure 43: Overview of the LS-USS and LS-USS online algorithms.

As seen in the background, representation learning has proven beneficial for segmenting high-dimensional data, partly because it allows for filtering out correlated or redundant information from the time series data. Combining representation learning with the basic building blocks used in FLUSS could resolve some of the problems with high-dimensional times series data (see section 2.3.4) and allow for better utilization of data from multiple sources. This is the thought process behind developing the segmentation algorithm Latent Space Unsupervised Sematic Segmentation (LS-USS).

All the primary components used to build LS-USS have been detailed in 3.2.1. As mentioned earlier, it utilizes components from both FLUSS and TSSTAFL. LS-USS uses an autoencoder to encode the multidimensional all subsequences set A into the one-dimensional latent all subsequence set F. After this, the latent space matrix profile (LSMP) $\mathbf{P_F}$ and the corresponding LSMP index $\mathbf{I_F}$ is constructed. The indices $\mathbf{I_F}$ are then used to make the CAC, which is the graph that contains the number of arc crossings at each time step. Few arc crossings indicate a high likelihood of changepoint at that time step, and a high number of arc crossings indicate a low likelihood for a changepoint.

LS-USS online is similar to LS-USS, except that it uses the version of the LSMP that works online by only considering right-pointing arcs. Doing this makes it possible to update $\mathbf{I_F}$ without recomputing the distance matrix for the last *(TC*2-1)* timesteps. As mentioned, the regular LSMP can also be ε real-time by accumulating a batch of data before adding it to the end of the $\mathbf{P_F}$, making the regular LS-USS ε real-time. An overview of the components used to make the two CPD algorithms are shown in Figure 43.

3.2.3 FLUSS and FLOSS Implementation

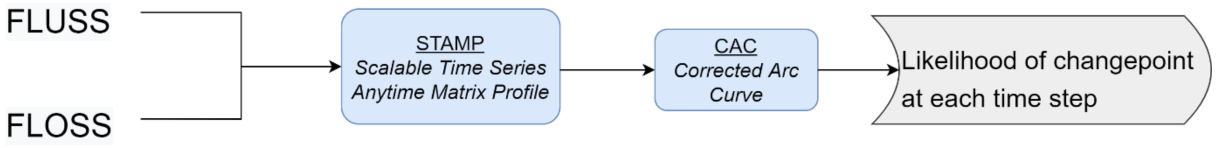
FLUSS
FLOSS
STAMP
Scalable Time Series Anytime Matrix Profile
CAC
Corrected Arc Curve
Likelihood of changepoint at each time step

Figure 44: Overview of the FLUSS and FLOSS algorithms.

Section 2.3 went through the main components used in the FLUSS and FLOSS (2.3.2) algorithms. First, the STAMP algorithm produces the matrix profile before the CAC is found by counting the number of arc crossings over each index using the matrix profile index. If there are few arc crossings over an index, it is more likely to be a changepoint than if there are many arc crossings.

The difference between FLOSS and FLUSS is that the former only uses the right-pointing arcs to enable online functionality.

3.2.4 TSSTAFL Implementation

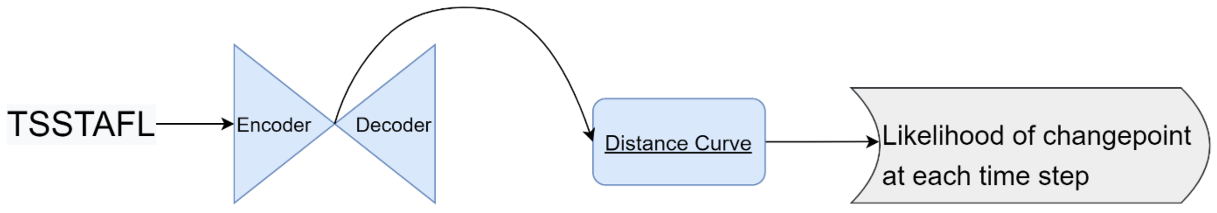

Figure 45: Overview of the TSSTAFL algorithm.

Another algorithm detailed in the background (section 2.4.2) is from the paper “Time Series Segmentation through Automatic Feature Learning”[37]. As the authors did not name the method described in that paper, it will be referred to as TSSTAFL going forward. This algorithm compares the distance between consecutive subsequences represented in latent space using an autoencoder. The output from TSSTAFL is a distance curve (slightly modified in section 3.2.1.5), where local minima values on this curve indicate a high likelihood for changepoint.

3.2.5 FLUSS-Welch

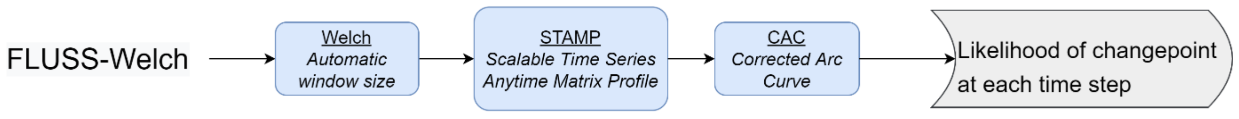
FLUSS-Welch
Welch
Automatic window size
STAMP
Scalable Time Series Anytime Matrix Profile
CAC
Corrected Arc Curve
Likelihood of changepoint at each time step

Figure 46: Overview of the FLUSS-Welch algorithm.

FLUSS-Welch uses the Welch method to estimate the periodicity of the signal. This estimate is used as the subsequence length in STAMP, which produces the matrix profile. After this, the CAC is found by counting the number of arc crossings over each index using the matrix profile index. If there are few arc crossings over an index, it is more likely to be a changepoint than if there are many arc crossings. The difference between this and regular FLUSS is that the subsequence size used in STAMP does not have to be manually specified when using FLUSS-Welch.

3.3 Change Point Extraction Algorithms

Changepoint extractors locate changepoints using the output from the CPD algorithms. The predicted segments are the areas between the predicted change points. The extractors in this thesis are used to find the changepoint based on the CAC.

The first section details the implementation of REA (2.3.1) used in [12]. The two sections after that will go through the extraction algorithms: LREA and LTEA. The former extracts changepoints based on local statistics, and the latter is a changepoint extractor that can be used with online data streams.

3.3.1 Regime Extraction Algorithm Implementation

The *Regime Extraction Algorithm* (REA) was presented in section 2.3.1. It searches for the k lowest “valley” points in the CAC and utilizes an exclusion zone around each valley point to avoid that all changepoints end up in the same place. In this thesis, it is implemented using the method *_rea* from the stumpy API [39].

3.3.2 Local Regime Extractor Algorithm

REA was presented as a simple algorithm for extracting segments based on the k lowest "valley" points of the CAC. There are some problems with this algorithm for long time-series data, however. In the datasets used in this thesis, the changepoints are usually distributed relatively evenly in time, which means that when extracting changepoints, one cares more about local minimums than about the global minimums. The REA algorithm will always pick the global k lowest "valley" points on the CAC instead of locating where the CAC is at its lowest compared to the local area around it. To combat this, a method that scales the CAC depending on the local statistics is introduced. The method scales each point in the CAC to zero mean and unit variance based on the local mean and standard deviation calculated over a rolling window (as shown in Equation 28). Figure 47 shows the effect of scaling the CAC. After the scaling is done, the same procedure used in the REA algorithm extracts the lowest k “valley points” from the scaled CAC.

$$CAC_{scaled} = \frac{CAC - \mu_{rolling}}{\sigma_{rolling}}$$

Equation 28: Where $\mu_{rolling}$ and $\sigma_{rolling}$ are time series that contain the rolling mean and rolling standard deviation of the CAC

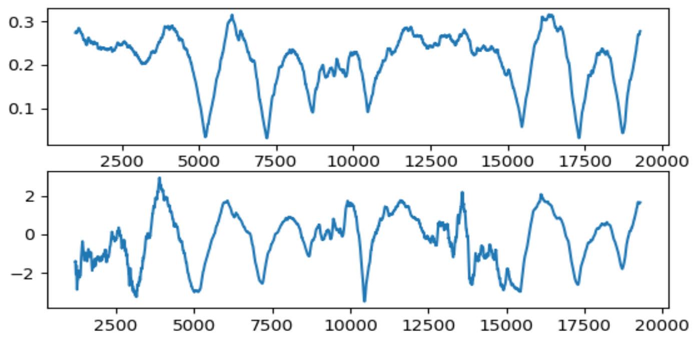

Figure 47: The top plot shows the original CAC, while the bottom plot shows the scaled CAC.

The downside with this algorithm is that it assumes that the changepoints are somewhat evenly distributed, but it should yield good results if the size of the rolling window is large enough to capture the local statistics of the CAC. If one increases the window size, the extraction will be increasingly global and more like the original REA. If one reduces the window size, the extraction will be more local. This thesis does not include any analysis on the optimal local window size, but the algorithm usually yields good results when it is set to the average segment length. The LREA algorithm is outlined in Figure 48.

Algorithm name: Local Regime Extracting Algorithm (LREA)

LREA (CAC,num_regimes)

Inputs:
CAC – a Corrected Arc Curve or Distance Curve
numRegimes – number of regime changes

Outputs:
locRegimes – the location of the change points

```
1.  CAC_scaled = empty array with same length as CAC
2.  avg_seg_length = length(CAC)/(numRegimes)
3.  for i=1 : length(CAC)
4.      local_window = CAC[i-( avg_seg_length/2): i+(avg_seg_length/2)]
5.      local_mean = mean(local_widow)  //Mean over a local window
6.      local_std = std(local_window)  // standard deviation over a local window
7.      CAC_scaled[i] = (CAC[i]-local_mean)/local_std  // Scales to zero mean and unit variance
8.  locRegimes = empty array of length numRegimes-1
9.  for i=1 : numRegimes
10.     locRegimes(i) = indexOf(min(CAC_scaled))
11.     Set exclusion zone         // To prevent matches to "self"
12. end
13. return locRegimes
```

Figure 48: Outline of the LREA algorithm

3.3.3 Local Threshold Extraction Algorithm

For comparisons between the offline CPD algorithms in this thesis, the algorithms REA and LREA are used. These algorithms require information about the number of segments, which is often not available. LREA and REA work great for comparing the CPD algorithms but have limited use in real-world segmentation tasks. To address this, the Local Threshold Extraction Algorithm (LTEA) is presented. As the name suggests, this algorithm works by scaling the CAC before doing threshold-based change point extraction.

The same CAC scaling used in the LREA algorithm is also utilized in LTEA. When the algorithm is used on online streaming data, the rolling statistics are only based on previous data points. Since the number of changepoints/segments is not included, it is more challenging to set the local window size. Setting this parameter requires some domain knowledge, but so does setting the temporal constraint. In this thesis, the local window size is set to 2xTC.

In LTEA, the scaled CAC-values over a given threshold are disregarded by being set to the value one. As the scaled CAC is standardized to zero mean and unit variance, setting the threshold is relatively trivial. Usually, the threshold can be set to around minus one standard deviation. To get better performance, one can find a good threshold by visualizing the scaled

CAC for a small amount of data. If some labeled data are available, one can also do a hyperparameter search to find a suitable threshold.

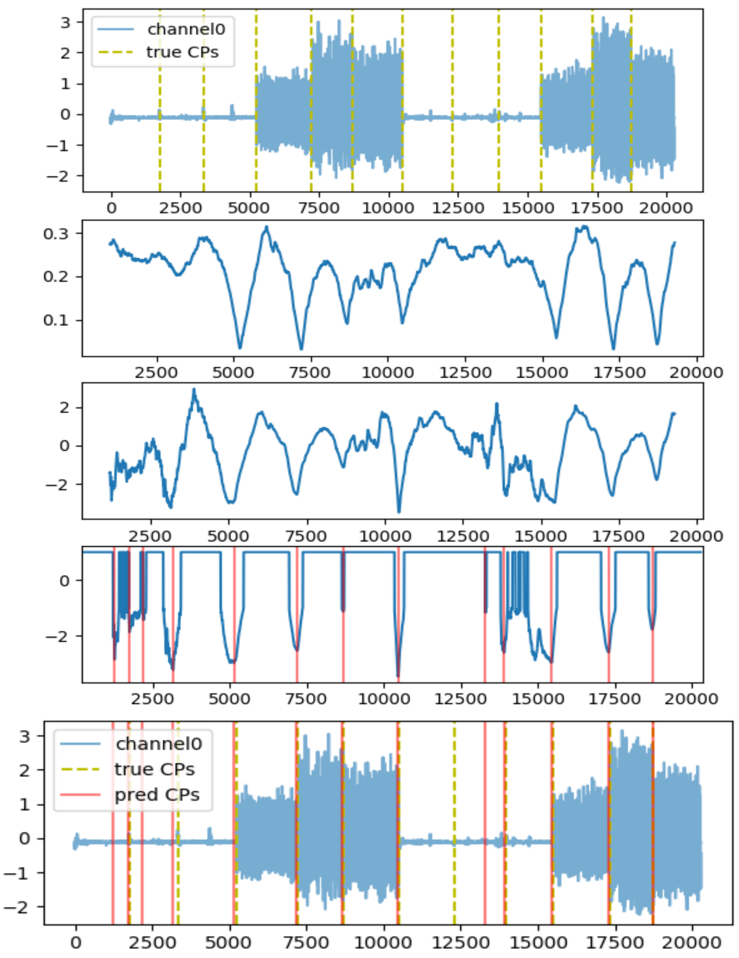

Figure 49: The top plot show channel0 and the true CP locations of a subject in the UCI dataset. The second plot show the associated CAC. The third plot show the scaled CAC. The fourth plot show the thresholded CAC with the predicted changepoints in each "valley". The last plot shows the predicted changepoints in conjuction with channel0 and the true locations.

After thresholding the CAC, one gets some clear valleys separated by intervals of "ones", as can be seen in the 4th plot in Figure 49. For each valley, the index of the minimum value in the valley is set to be a changepoint location. An exclusion zone like the one used in LREA and REA is also utilized to avoid trivial matches caused by valleys close in time. The algorithm is outlined in Figure 50.

Algorithm name: Local Threshold Extracting Algorithm (LTEA)

LREA (CAC,local_window_size=Threshold=-1)

Inputs:

CAC – a Corrected Arc Curve or Distance Curve
local_window_size – The size of the window used to nomalize the CAC
Threshold = -1 – Threshold for where to search for change points. Defaults to -1

Outputs:

locRegimes – the location of the change points

```
1.  CAC_scaled = empty array with same length as CAC
2.  avg_seg_length = length(CAC)/(numRegimes)
3.  for i=1 : length(CAC)
4.     local_window = CAC[i-(avg_seg_length) : i+(avg_seg_length)]
5.     local_mean = mean(local_widow)  //Mean over a local window
6.     local_std = std(local_window)  // standard deviation over a local window
7.     CAC_scaled[i] = (CAC[i]-local_mean)/local_std  // Scales to zero mean and unit variance
8.     If CAC_scaled[i] > -1
9.        CAC_scaler[i] = 1
10. CPs = empty array of length numRegimes-1
11. for i=1 : n_valleys
12.    CPs[i] = indexOf(min(valley))
13. end
14. return CPs
```

Figure 50: Outline of the LTEA algorithm

4 Datasets

This chapter will present the datasets used to compare and test the algorithms implemented in the methods chapter. Table 3 presents an overview of the datasets utilized and their features. The abbreviations will be used to refer to the datasets when doing experiments, and the leftmost column shows which sections contain experiments involving the datasets.

The original training data in the datasets Expressive motion with dancers and Long-Term 3DC Dataset is used to construct two artificial datasets. The following sections will detail how each dataset was made and how it is used in this thesis.

Original dataset names	**Abbreviations used in this thesis**	**Containing ground truth changepoint locations**	**Containing human specified subsequence sizes**	**Sections involving dataset**
UCI Human Activity Recognition Using Smartphones Dataset	UCI	✓	✗	5.1.1.1
Expressive motion with dancers	Dance artificial	✓	✗	5.1.1.2
	Dance	✓	✗	5.1.1.3
Long-Term 3DC Dataset	EMG artificial	✓	✗	5.1.1.4
	EMG	✓	✗	5.1.1.5
Depresjon	Depresjon	✗	✗	5.2
UCR Time Series Semantic Segmentation Archive	UCR	✓	✓	5.2.2

Table 3: Overview of the different datasets and corresponding abbreviations and features. The original training data in the datasets Expressive motion with dancers and Long-Term 3DC Dataset is used to construct artificial datasets.

4.1 UCI Human Activity Recognition Using Smartphones Dataset

The *UCI Human Activity Recognition Using Smartphones Dataset* [22] contains 30 volunteers between the ages of 19 and 48 who perform six activities wearing a smartphone on the waist. The activities are walking, walking upstairs, walking downstairs, sitting, standing, and laying down. The collected data came from the 3-axial accelerometer and 3-axial gyroscope located in a Samsung Galaxy S2 and were sampled at 50Hz. The raw acceleration data signals have three main components: body movement, gravity, and noise. The sensor data was filtered using a median filter and a 3rd order low pass Butterworth filter with a corner frequency of 20 Hz for noise removal. A low pass Butterworth filter, with a corner frequency of 0.3 Hz, is used to separate the body movement and gravity signals. The reason for choosing to use a 0.3 Hz cut-off frequency is that gravitational forces are assumed only to have low-frequency components. This thesis will use the data from the gyroscope, accelerometer, and the filtered body movement from the accelerometer. As all these signals are sampled in all three spatial dimensions, the dataset contains nine channels in total. Nine subjects constitute the training set, five subjects are used as a validation set, while the remaining 16 subjects are picked for the test set.

Channels	**Time Signals**
Channel0	Body accelerometer x
Channel1	Body accelerometer y
Channel2	Body accelerometer z
Channel3	Body gyroscope x
Channel4	Body gyroscope y
Channel5	Body gyroscope z
Channel6	Total accelerometer x
Channel7	Total accelerometer y
Channel8	Total accelerometer z

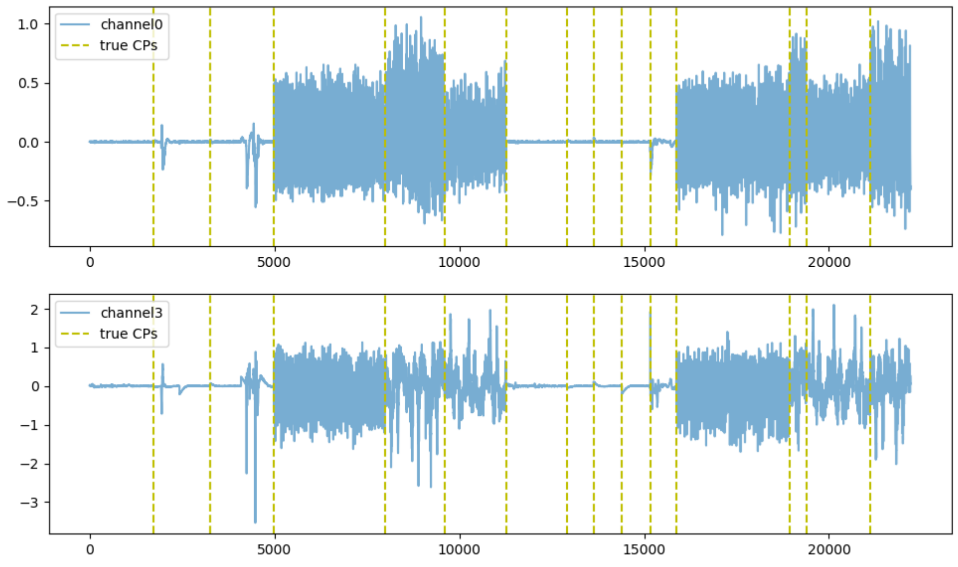

Figure 51: Channel zero and three in a time series from the UCI dataset

4.2 Expressive Motion With Dancers

The dataset *Expressive motion with dancers*[45] is made in collaboration with professional dancers performing different dance moves that correspond to three emotional states. The dancers were wearing two Myo armbands, one on the calf and one on the forearm. The Myo is a low-cost eight-channel consumer-grade, dry-electrode EMG armband that also integrates a nine degree of freedom IMU (inertial measurement unit). Data from the two IMU sensors are represented by yaw, pitch roll coordinates and are sampled at 50Hz. For the experiments conducted in this thesis, only the IMU channels will be utilized.

	Pitch	Yaw	Roll
Arm	channel0	channel1	channel2
Leg	channel3	channel4	channel5

The dancer was instructed to develop three choreographic moods differentiated by the emotion they subjectively represented for the performer. To generate training data, the dancer repeated each sequence of moods for around 20 seconds. In addition to this, the dancers also created dance performances using the same lexicon of sequences used when building the training set. The dataset contains data from 27 participants. Here, the training data is used as the training set, while the dance performances from 9 participants are used as the validation set, and the remaining 18 are used for testing.

An additional artificial dataset is made from the original training data. The new training set includes raw data from 12 participants, while the validation and test sets have been made by randomly concatenating ten to seventeen choreographic moods from remaining six and nine participants, respectively. The validation set consists of 20 artificial time series', while the test set consists of 50.

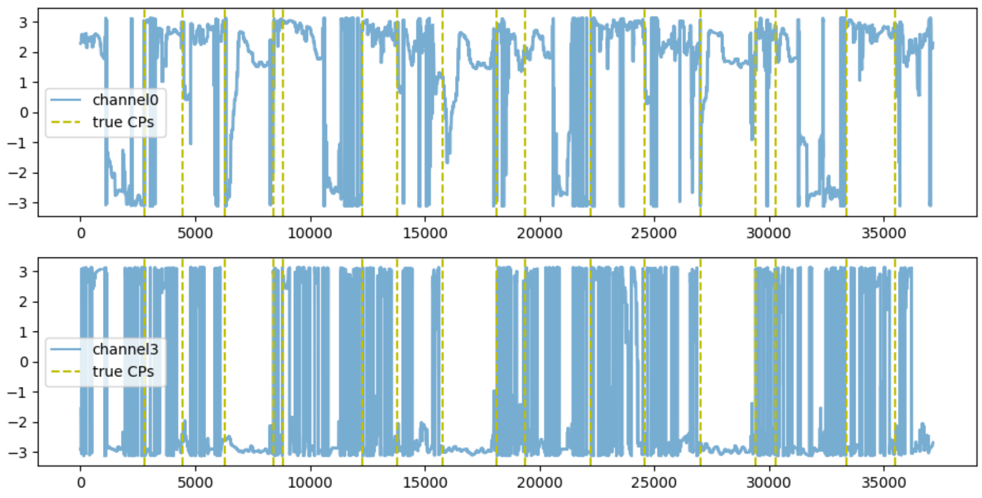

Figure 52: Channel zero and three in a time series from the Dance dataset

4.3 Long-Term 3DC Dataset

The dataset *Long-Term 3DC* Dataset was initially used in the article "Virtual reality to study the gap between offline and real-time EMG-based gesture recognition" [46] and is made publicly available on [42]. This dataset contains data from 20 able-bodied participants between 18 and 34 years old, making eleven hand gestures (see Figure 54). Each of the participants participated in three to four recording sessions in a period of 14 days. The EMG armband used to make this dataset is the 3DC Armband[47]. The 3DC is a ten-channel, dry electrode 3D printed EMG band with a sampling rate of 1000 Hz. The armband also includes a 9-axis Magnetic, Angular Rate, and Gravity (MARG) sensor, but only 10 EMG channels are used in this thesis. The dataset is divided into training sessions and evaluation sessions. In the training session, the participants were asked to hold each of the 11 gestures for 5 seconds, repeated four times by each participant on every recording day. For the evaluation session, the participants were asked to do a total of 42 gestures in one continuous session. The requested gestures were selected at random every 5 seconds. Each participant recorded a minimum of 6 evaluation sessions. The original paper [46] contains a more detailed description of the making of this dataset.

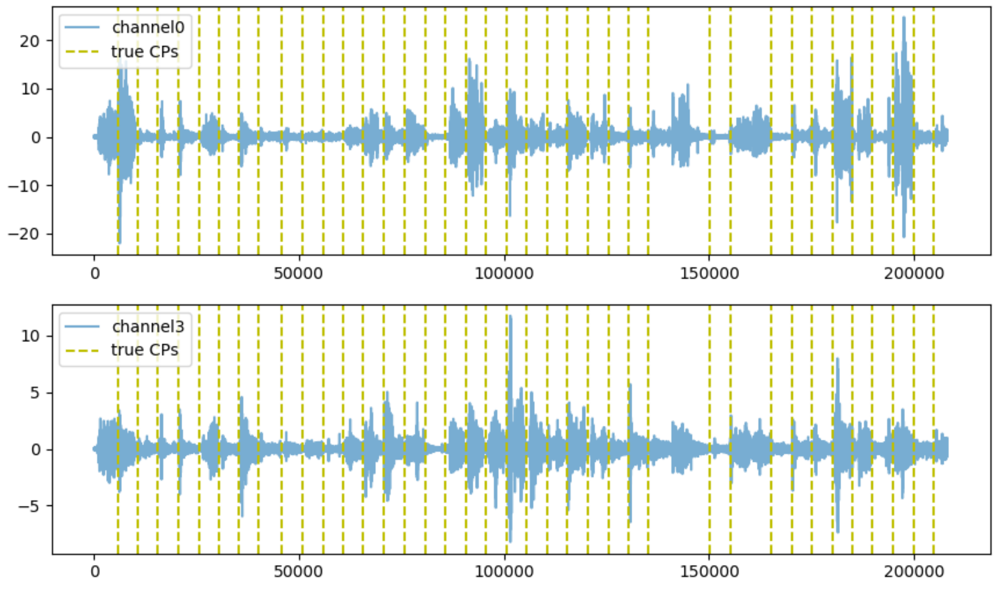

Figure 53: Channel zero and three in a time series from the EMG dataset

For this thesis, the evaluation session will be used as a unique dataset. Here, the first evaluation of six participants is used as a training set, the first evaluation of five participants is used as a validation set, and the remaining evaluation runs for every participant is used as a test set.

From the training sessions, an additional artificial dataset is made. The artificial training set consists of training sessions recorded from 10 participants. As with the artificial dance dataset, the validation and test sets are made by randomly concatenating ten to seventeen gestures. The validation set consists of 15 time series made by gestures from 4 participants, while the test sets include 30 time series made by gestures from 8 participants.

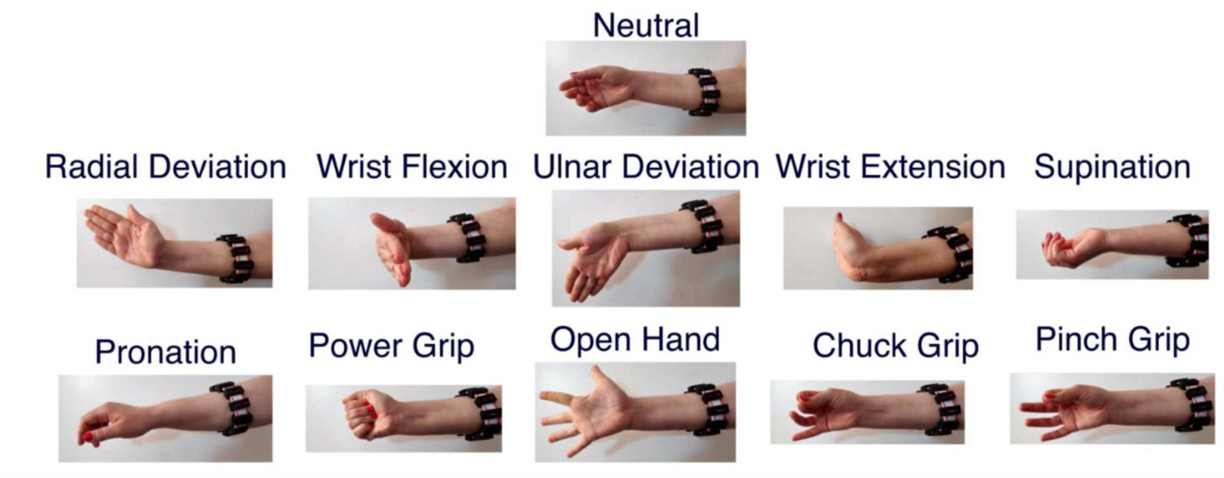

Figure 54: Gestures used in the Long-Term 3DC Dataset **dataset. From [46]**

4.4 Depresjon

The datasets presented thus far are helpful for comparing and testing the different segmentation algorithms objectively because they contain ground truth labels. The *Depresjon* [48] dataset, however, does not contain changepoints, so it will not be used to compare the segmentation performance between the different algorithms. Instead, it will be used to test a practical application of the unsupervised segmentation algorithm LS-USS. This dataset contains motor activity recordings from 23 unipolar and bipolar depressed patients in addition to 32 healthy controls. The Montegomery-Asberg Depression Rating Scale (MADRS) is used for rating the severity of the ongoing depression, and all the depressed patients had a MADRS-score above 10. The motor activity was monitored with an actigraph watch (Actiwatch, Cambridge Neurotechnology Ltd, England, model AW4) worn on the right wrist. The sampling rate used is 32Hz, and the movement has to be above 0.05 g to be recorded. The data points in the dataset are the total activity counts recorded over one-minute intervals. Table 4 shows the first 10 minutes of data for one participant in the Depresjon dataset.

Table 4: The first 10 minutes of data for one participant in the Depresjon dataset

	timestamp	date	Activity Count
0	2006-01-30 12:30:00	2006-01-30	7
1	2006-01-30 12:31:00	2006-01-30	79
2	2006-01-30 12:32:00	2006-01-30	4
3	2006-01-30 12:33:00	2006-01-30	3
4	2006-01-30 12:34:00	2006-01-30	4
5	2006-01-30 12:35:00	2006-01-30	4
6	2006-01-30 12:36:00	2006-01-30	94
7	2006-01-30 12:37:00	2006-01-30	4
8	2006-01-30 12:38:00	2006-01-30	10
9	2006-01-30 12:39:00	2006-01-30	3
10	2006-01-30 12:40:00	2006-01-30	9

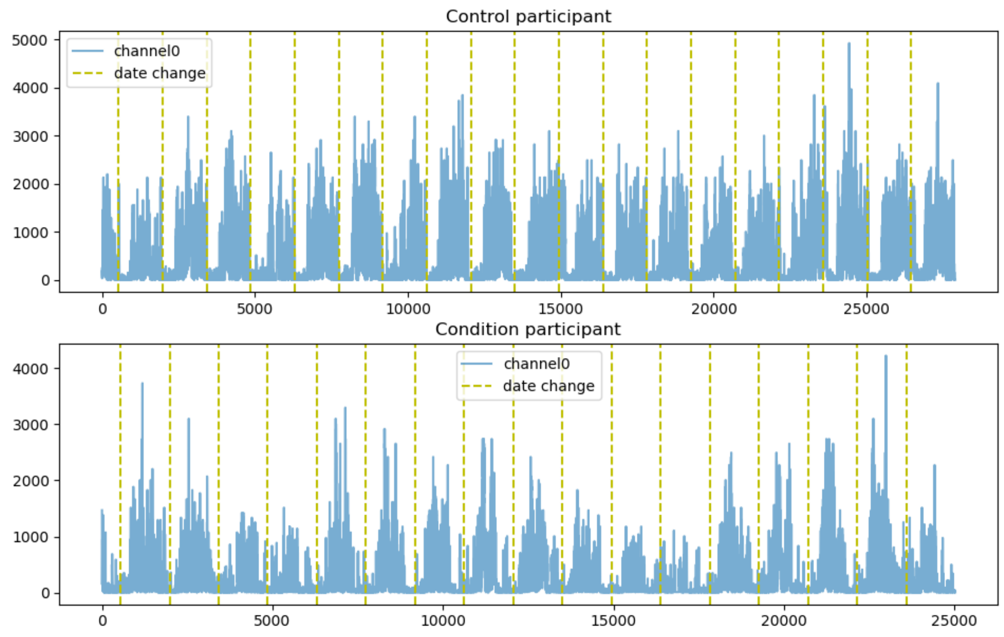

Figure 55: Examples of control and condition participants from the Depresjon dataset.

4.5 UCR Time Series Semantic Segmentation Archive

UCR Time Series Semantic Segmentation Archive[2] was used to showcase the FLUSS and FLOSS segmentation algorithms[12]. This dataset contains many short datasets from different domains. Many of these datasets are not that difficult to segment independently, but they are useful for testing how domain agnostic a segmentation algorithm is. To get a sense of how diverse the data is, take a look at Table 5 and Figure 56.

This dataset includes human specified subsequence sizes, which makes it possible to test the performance of FLUSS-Welch against regular FLUSS utilizing human specified subsequence sizes.

Table 5: Descriptions from a presentation on article website [2]

Dataset-name	**# Channels**	**# CPs**	**Short Description**
Cane	1	1	Data from two different users of a smart can concatenated together
DutchFactory	1	1	This dataset is one year of electrical power demand in a Dutch facility city
EEGRat	2	1	Contains 5 sec of a two-channel EEG recording at left and right frontal cortex of male adult WAG/Rij rats. Signals were referenced to an electrode placed at the cerebelum
Fetal2013	1	2	Concatenated three AECG1 leads from separate fetal recordings, a64, a68, a57
GrandMalSeizures	1	1	Tonic-clonic seizures of a subject recorded with a scalp right central (C4) electrode
GrandMalSeizures2	1	1	Tonic-clonic seizures of a subject recorded with a scalp right central (C4) electrode
GreatBarbet	2	2	Three birds calls, all from the same species, Great Barbet, and concatenated approximately one minute snippets of their songs.
InsesctEPG 1,2,3,4	1	1	Four datasets containing EPG recordings of feeding states in the insect Asian citrus psyllid
NogunGun	1	1	The time series is the y-axis position of an actors hand as she points her finger about 20 times, followed by pointing a gun about 30 times.
PigInternalBleeding	3	1	The data has been collected from a cohort of 52 pigs subjected to induced slow bleeding. Each animal has been sedated, instrumented and bleed with a pump at a rate of 20mL/min. From data of each pig, two 30 second long segments of data were randomly sampled: one from the period before and one approximately 2 minutes into the bleeding.
Powerdemand	1	1	This dataset is 320 days of electrical power demand in an Italian city, beginning in mid august. At time point 4500, the remaining data was flipped left to right.
PulsusParadoxus	3	1	This dataset records the onset of Pulsus Paradoxus on a patient. The dataset records the volunteers SpO2(one channel) and ECG(two channels)
RoboticDogActivityX	1	1	x-axis acceleration of the Sony RoboDog first walking on cement and then "playing"
RoboticDogActivityY	1	1	x-axis acceleration of the Sony RoboDog first walking on cement and then "playing"
RoboticDogActivityY2	1	1	x-axis acceleration of the Sony RoboDog first walking on cement before walking on carpet
SimpleSyntetic	1	2	Dataset created by these two lines of matlabocde: >> a= sin(0:0.05:400) + randn(size(sin(0:0.05:400)))/30; >> a(3000:5000)=abs(a(3000:5000));
SuddenCardiacDeath	2	1	Data is at 50Hz. Patient is Female, 82 undergoing Heart failure. For the first 65 seconds the patient a very irregular beat, then there is a sustained ventricular tachyarrhythmia. Contains data from two different ECG-channels.
SuddenCardiacDeath1	1	2	Patient is Male, 75 undergoing Cardiac surgery. For the first 125 seconds the patient has Bundle branch block beats, then there is a burst of Premature ventricular contractions lasting for about 28 seconds before the patient settles back to normal heartbeats
TiltABP	1	1	A subject was lying on tilt table with foot support. At timestep 25,000 the tilt table was rapidly rotated to the stand-up position. This dataset records the volunteers ABP.
TiltECG	1	1	A subject was lying on tilt table with foot support. At timestep 25,000 the tilt table was rapidly rotated to the stand-up position.This dataset records the volunteers ECG
WalkJogRun	2	2	A person walking, jogging and running with a rotationsensor on the left lower arm and on the left calf

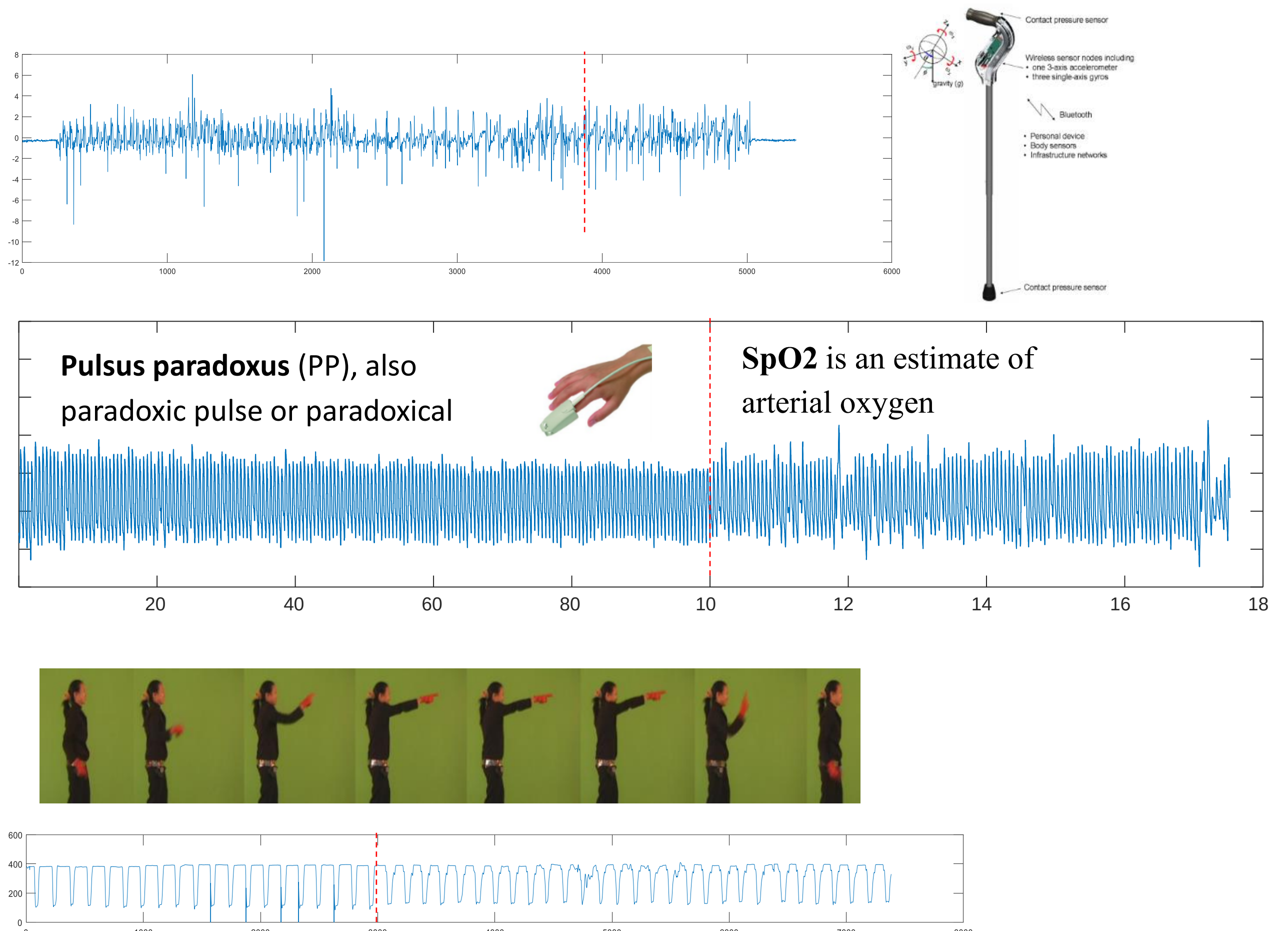

Figure 56: Tree sub-datasets from the UCR Time Series Semantic Segmentation Archive[3] . The first plot shows the Cane dataset. The second plot shows the PulsusParadoxus dataset. The last plot shows the nogunGun dataset, and the images above show how it was made.

5 Experiments

This chapter includes three sections containing different types of experiments:

The first section will include an in-depth comparison between the different segmentation algorithms used in the thesis (except FLUSS-Welch). The comparisons are performed on five datasets containing long multi-dimensional time-series data.

The second section will demonstrate a practical application of unsupervised segmentation algorithms, the last section in the experiments chapter shows the utility of LS-USS for finding natural segments on the Depresjon dataset. Statistical features are extracted from the found segments and used for the classification of depressed vs. non-depressed participants. The results are then compared to the classification performance achieved when using fixed segment sizes.

The third section in this chapter is dedicated to the parameter-free segmentation algorithm FLUSS-Welch obtained by estimating the subsequence size using Welch’s method. The experiments are performed on the UCR datasets where FLUSS-Welch’s performance is compared to using the recommended subsequence lengths included with each of the datasets.

Each of the sections is split into two subsection: experiments and discussion. The experiment sections present the results, while the discussion sections contain an analysis of the results

5.1 Comparisons Done on Long Multidimensional Data

The comparisons in this section are made on five datasets: UCI, Dance Artificial, Dance, EMG artificial, and EMG. The datasets are split into test, validation, and training sets as specified in the dataset section (2.5) in the methods chapter. The mean distance between the changepoints from the training set is calculated for use as the local window size for scaling the CAC when using the LREA and LTEA extractors. The training set is also utilized for training the autoencoders. As the autoencoders require both training and validation sets, 80% of the training set is used as training examples, while the remaining 20% is used to validate the autoencoder. The validation portion of the dataset is used to find the optimal hyperparameters of the segmentation algorithms. After the best performing model configurations on the validation set are found, the test set is used to make the final comparisons between the different segmentation algorithms.

Offline Models	Datascalers	N_W		TC		Step-size		Autoencoder
				EMG datasets	Others			
LS-USS - REA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		Fully Connected Convolutional
LS-USS - LREA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		Fully Connected Convolutional
FLUSS - REA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		-
FLUSS - LREA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		-
TSSTAFL - LREA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	25 50 100 150	200 250 350 500	Fully Connected

Table 6: Overview of the online algorithms used and their hyperparameters. N_w=Subseqence size, TC=Temporal Constraint, step-size=distance in time between each subsequence used (decides how much each subsequence overlap)

Online Models	Datascalers	N_W		TC		Step-size		Autoencoder	Threshold	
				EMG datasets	Others					
LS-USS - LTEA (Ɛ-real time)	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		Fully Connected Convolutional	-0.5 -1.0 -1.5	-2.0 -2.5 -3.0
LS-USS online - LTEA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		Fully Connected Convolutional	-0.5 -1.0 -1.5	-2.0 -2.5 -3.0
FLUSS - LTEA (Ɛ-real time)	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		-	-0.5 -1.0 -1.5	-2.0 -2.5 -3.0
FLOSS - LTEA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	-		-	-0.5 -1.0 -1.5	-2.0 -2.5 -3.0
TSSTAFL - LTEA	NoScaler StandardScaler RobustScaler MinMaxScaler	50 100 150 200	300 400 500	1000 1500 2000	800 1200 1600	25 50 100 150	200 250 350 500	Fully Connected	-0.5 -1.0 -1.5	-2.0 -2.5 -3.0

Table 7: Overview of the online algorithms used and the hyperparameters. N_w=Subseqence size, TC=Temporal Constraint, step-size=distance in time between each subsequence used (decides how much each subsequence overlap)

The segmentation algorithms are divided into two categories: online and offline. Table 6 and Table 7 present the different segmentation algorithms and their associated hyperparameters. The hyperparameter optimization is done using grid search. The step size is used in TSSTAFL and decides the amount of overlap between subsequences. If the step size is set to one, all subsequences in the all-subsequences-set A will be used to construct the distance curve, and if the step size is set to 25, every 25th subsequence will be used. The threshold parameter is the threshold used in the extraction algorithm LTEA. TSSTAFL only uses the local extractors LREA and LTEA because the original paper states that only local maxima should be classified as a breakpoint. Even if they are not fully online, LS-USS and FLUSS are also included in the online category as they can be made to work Ɛ real-time when using a temporal constraint. Also, notice that the data scalers are part of the hyperparameter search. Due to time constraints, only certain ranges of the hyperparameters are considered for the different datasets based on prior knowledge about the scaling of the data.

The offline algorithms are evaluated using the Regime Score (Equation 16), while the online algorithms are evaluated using the Prediction Loss MAE (Equation 15). The Regime Score works well for evaluating segmentation algorithms that have access to the number of changepoints to predict, such as the offline algorithms used in this thesis. When the number of changepoints is not known in advance, the segmentation algorithms run the risk of under or over-predicting. The Prediction Loss (MAE) takes this into account by weighting the mean absolute error with the prediction ratio. Thus, the Prediction Loss is a fairer evaluation metric for the online algorithms as the number of changepoints to predict is not given beforehand. Note that the metrics used are not comparable and cannot be used to make comparisons between the online and offline algorithms.

5.1.1 Experiments

The following subsections will give an overview of the experiments done on each dataset. Each section starts with two tables presenting the different model configurations found on the validation set for the offline and online models. The leftmost column shows the mean Regime Score on the test set for the offline algorithm, while the mean Prediction Loss MAE is shown for the online ones. Then, two boxplots, one for the offline and one for the online models, show the spread in segmentation scores over the dataset, which is useful for looking into the consistency of each algorithm. Each section also includes a plot of one segmentation run using two of the algorithms developed in this thesis: the offline LS-USS LREA and the Ɛ real time algorithm LS-USS LTEA. These plots are included to get a sense of the models' real-world segmentation performance and give some context about what the segmentation scores represent.

5.1.1.1 UCI

Offline	Datascaler	N_W	TC	Step-size	Autoencoder	Regime Score Mean
TSSTAFL - LREA	RobustScaler	50	-	250	Fully Connected	0.01589
FLUSS - REA	RobustScaler	50	800	1	None	0.02824
FLUSS - LREA	RobustScaler	50	1200	1	None	0.00931
LS-USS - REA	MinMaxScaler	50	800	1	Fully Connected	0.00687
LS-USS - LREA	MinMaxScaler	50	800	1	Fully Connected	0.01041

Table 8: Model configurations found from doing hyperparameter search on the UCI dataset. The leftmost column shows the mean regime score from segmenting the test set. N_W-range = [50-150], Step-size-range=[25-250]

Online	Datascaler	N_W	TC	Step-size	Autoencoder	Threshold	Prediction Loss MAE Mean
TSSTAFL - LTEA	MinMaxScaler	100	-	100	Fully Connected	-2	51.99
FLOSS - LTEA	RobustScaler	150	1600	1	None	-1	87.36
FLUSS - LTEA	RobustScaler	50	1600	1	None	-1	47.27
LS-USS online - LTEA	RobustScaler	50	1200	1	Fully Connected	-0.5	114.82
LS-USS - LTEA	MinMaxScaler	150	800	1	Fully Connected	-1.5	45.54

Table 9: Model configurations found from doing hyperparameter search on the UCI dataset. The leftmost column shows the mean prediction loss MAE from segmenting the test set. N_W-range = [50-150], Step-size-range=[25-250]

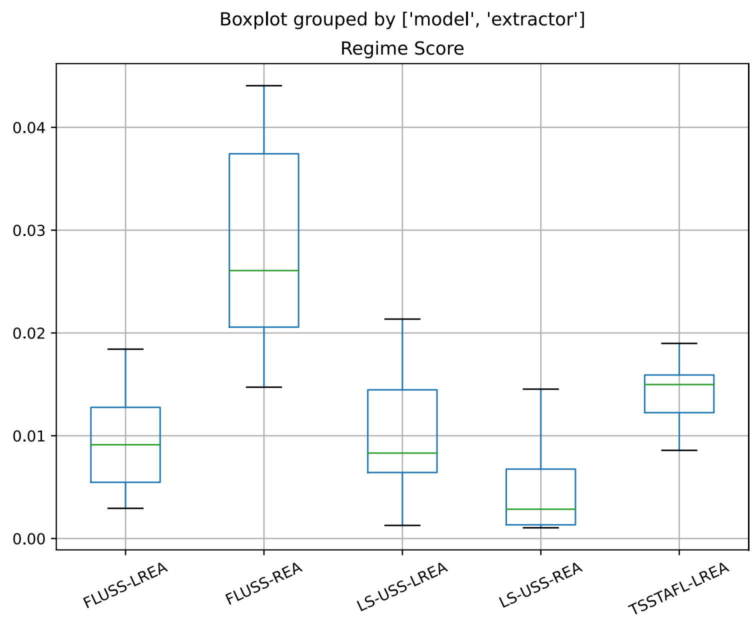

Figure 57: Boxplot showing the performance of the offline segmentation algorithms on the UCI dataset

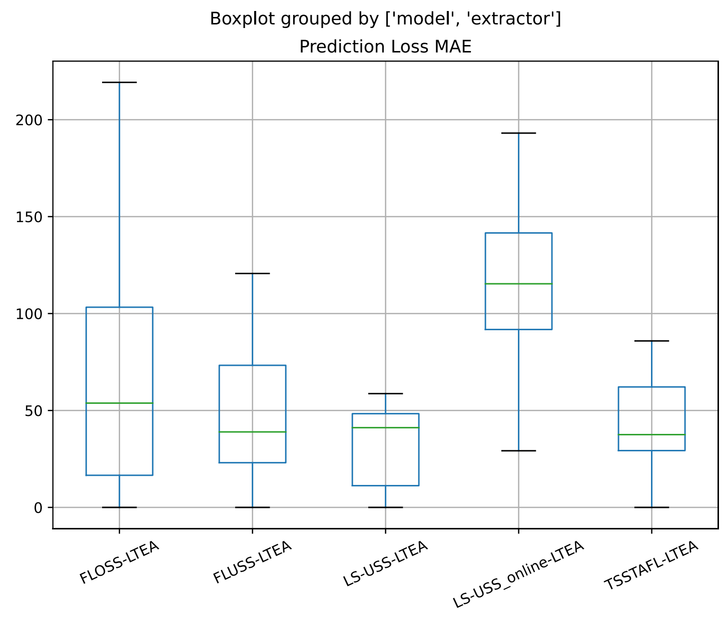

Figure 58: Boxplot showing the performance of the online segmentation algorithms on the UCI dataset

LS-USS REA is the best performing algorithm on the test set for the offline models, scoring higher than the local counterpart LS-USS LREA. Scaling the CAC with a local window equal to the mean segment length on the training set might not be optimal as the segment lengths on this dataset vary quite a lot, making it difficult to detect the smaller segments when using the local extractor LREA. Most of the algorithms perform similarly on the online dataset, with mean scores around 50, except for LS-USS online LTEA. RobustScaler and MinMaxscaler appear to be the best scaling methods for the UCI dataset. Also, notice that all the LS-USS algorithms utilizes the Fully Connected model on this dataset. The example plots below show that LS-USS LREA is nearly spot on the true changepoints. One can also see that although it does not have any information about the number of changepoints, LS-USS LTEA performs a nearly optimal segmentation as well.

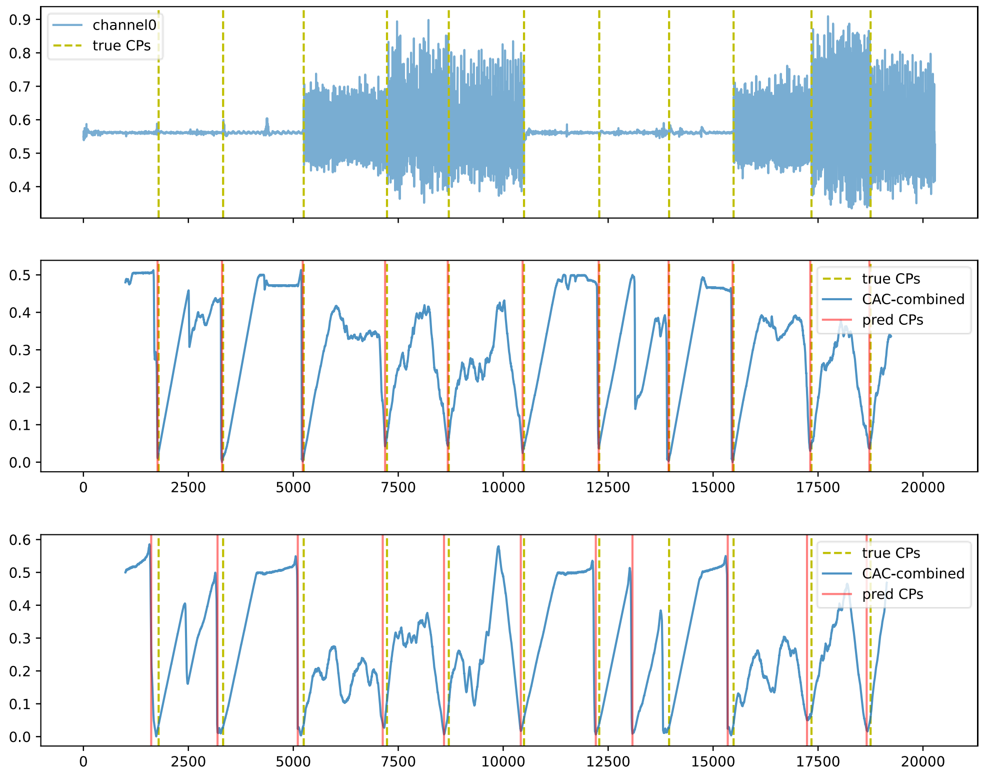

Figure 59: Top: Channel 0 of subject 4 in the UCI dataset. Middle: Segmentation using LS-USS – LREA-Bottom: Segmentation using LS-USS – LTEA.

5.1.1.2 Dance Artificial

Offline	Datascaler	N_W	TC	Step-size	Autoencoder	Regime Score Mean
TSSTAFL - LREA	MinMaxScaler	100	-	1	PaperModel2	0.03807
FLUSS - REA	RobustScaler	50	800	1	None	0.02608
FLUSS - LREA	StandardScaler	50	800	1	None	0.01735
LS-USS - REA	MinMaxScaler	100	800	1	ConvModel	0.0257
LS-USS - LREA	NoScaler	50	800	1	ConvModel	0.01803

Table 10: Model configurations found from doing hyperparameter search on the Dance Artificial dataset. The leftmost column shows the mean regime score from segmenting the test set. N_W-range = [50-400], Step-size-range=[25-250]

Online	Datascaler	N_W	TC	Step-size	Autoencoder	Threshold	Prediction Loss MAE Mean
TSSTAFL - LTEA	MinMaxScaler	50	-	1	PaperModel2	-0.5	109.04
FLOSS - LTEA	NoScaler	50	800	1	None	-0.5	55.06
FLUSS - LTEA	MinMaxScaler	100	800	1	None	-0.5	48.01
LS-USS online - LTEA	MinMaxScaler	50	800	1	ConvModel	-0.5	64.08
LS-USS - LTEA	MinMaxScaler	50	800	1	ConvModel	-0.5	27.15

Table 11: Model configurations found from doing hyperparameter search on the Dance Artificial dataset. The leftmost column shows the mean Prediction Loss MAE score from segmenting the test set. N_W-range = [50-400], Step-size-range=[25-250]

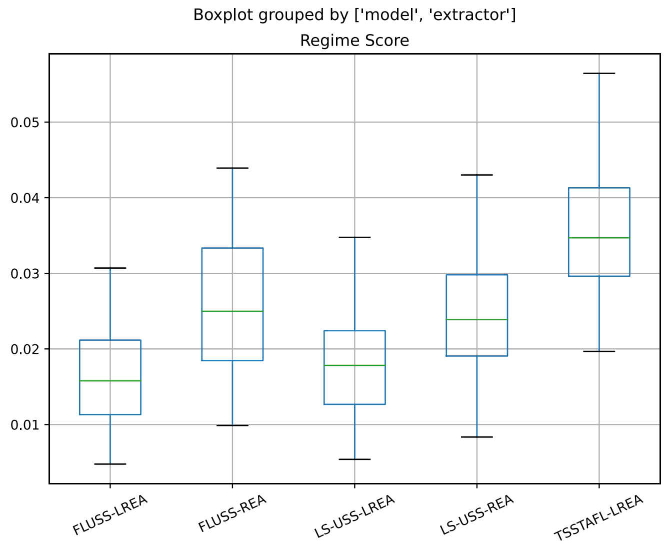

Figure 60: Boxplot showing the performance of the offline segmentation algorithms on the Dance Artificial dataset

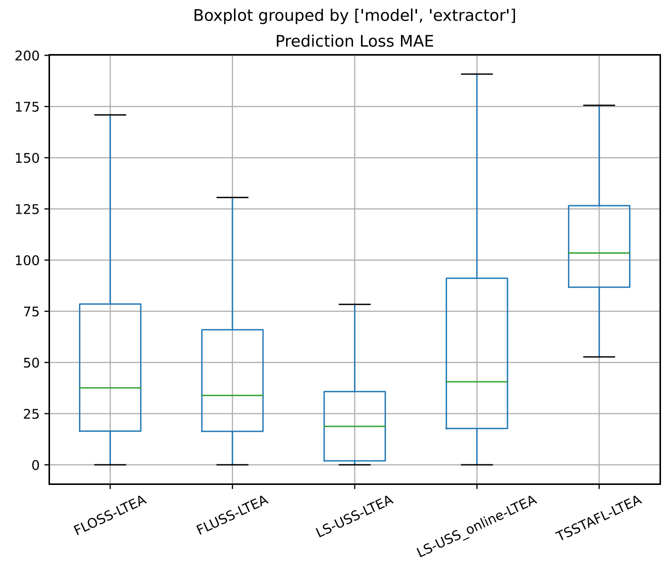

Figure 61: Boxplot showing the performance of the online segmentation algorithms on the Dance Artificial dataset

The best performing offline algorithm on the test set is here FLUSS LREA, closely followed by LS-LSS LREA. On this dataset, there seems to be a benefit in scaling the CAC with the local statistic as both the LS-USS and FLUSS scores higher using the local region extractor LREA. For the online models, the Ɛ real-time algorithm LS-USS LTEA has the best score. The rest of the online algorithms have similar scores except TSSTAFL, which is the worst-performing model on the test set. Also, notice that all the LS-USS-based algorithms ended up using the convolutional model on this dataset. The figure below shows that LS-USS yields similar segmentation results for both LREA(offline) and LTEA(online).

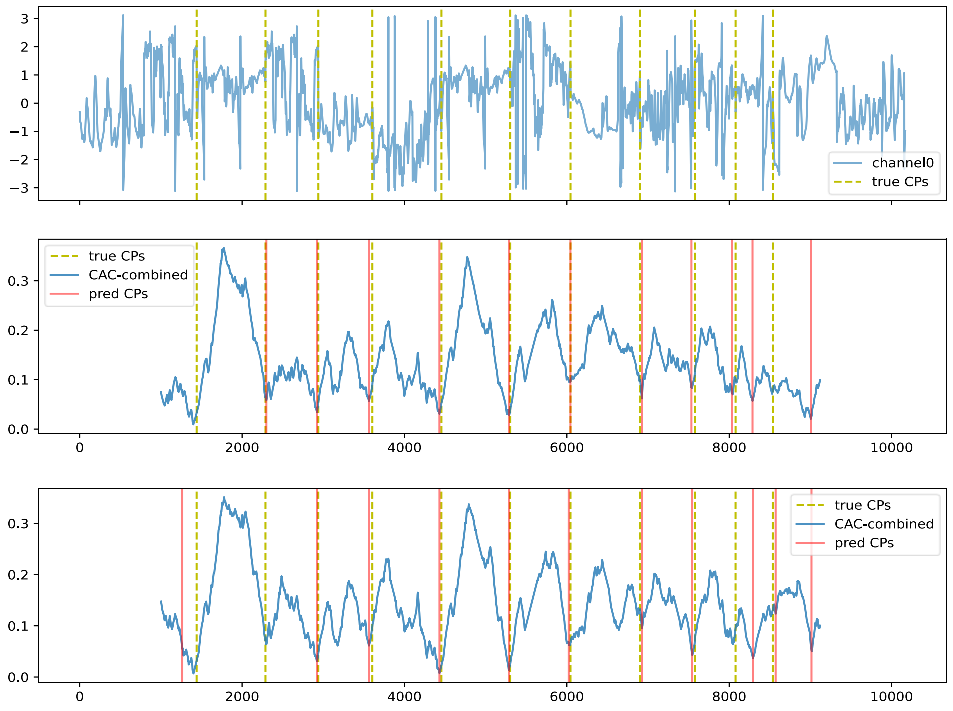

Figure 62: Top: Channel 0 of d41 in the Dance Artificial dataset. Middle: Segmentation using LS-USS – LREA- Bottom: Segmentation using LS-USS – LTEA.

5.1.1.3 Dance

Offline	Datascaler	N_W	TC	Step-size	Autoencoder	Regime Score Mean
TSSTAFL - LREA	MinMaxScaler	400	-	100	Fully Connected	0.03665
FLUSS - REA	RobustScaler	400	800	1	None	0.03802
FLUSS - LREA	RobustScaler	50	800	1	None	0.02969
LS-USS - REA	RobustScaler	400	1200	1	Fully Connected	0.05459
LS-USS - LREA	MinMaxScaler	400	1600	1	Fully Connected	0.029

Table 12: Model configurations found from doing hyperparameter search on the Dance dataset. The leftmost column shows the mean regime score from segmenting the test set. N_W-range = [50-400], Step-size-range=[25-250]

Online	Datascaler	N_W	TC	Step-size	Autoencoder	Threshold	Prediction Loss MAE Mean
TSSTAFL - LTEA	MinMaxScaler	125	-	200	Fully Connected	-1.5	289.2028
FLOSS - LTEA	RobustScaler	125	1200	1	None	-2	334.65
FLUSS - LTEA	RobustScaler	200	1200	1	None	-1.5	281.0156
LS-USS online - LTEA	MinMaxScaler	200	1600	1	Fully Connected	-1.5	204.9742
LS-USS - LTEA	RobustScaler	200	1200	1	Fully Connected	-2	415.32

Table 13: Model configurations found from doing hyperparameter search on the Dance dataset. The leftmost column shows the mean Prediction Loss MAE score from segmenting the test set. N_W-range = [50-400], Step-size-range=[25-250]

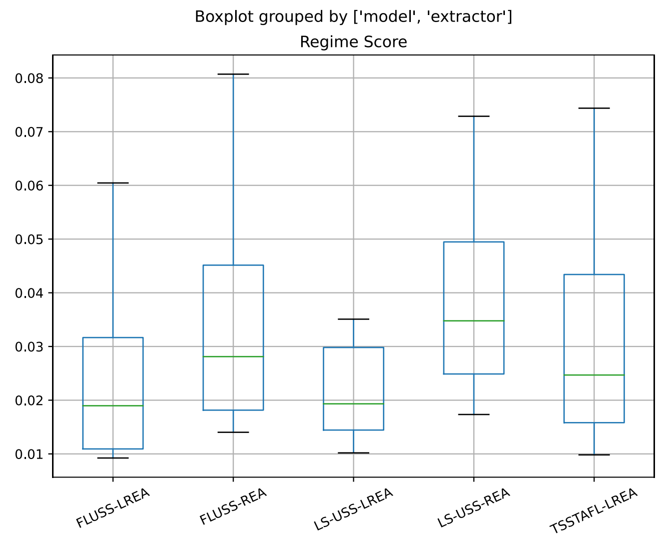

Figure 63: Boxplot showing the performance of the offline segmentation algorithms on the Dance dataset

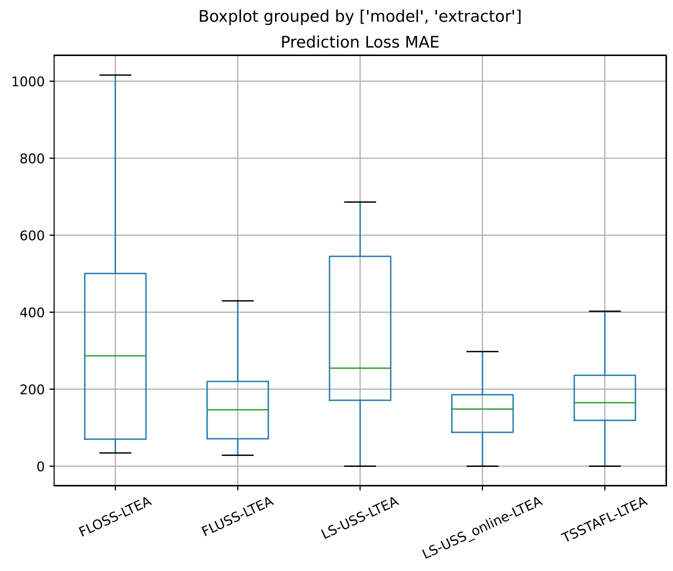

Figure 64: Boxplot showing the performance of the online segmentation algorithms on the Dance dataset

The algorithm with the best mean regime score on the Dance test set is LS-USS, closely followed by FLUSS LREA. One can also see that models utilizing the REA extractor have the worst performance. The fully connected model is the autoencoder used for both the online and offline models on this dataset. On the online dataset, every algorithm performs poorly, and the results seem a bit random. This could be caused by too much variation between the individual dance performances, which might lead to the best performing configurations in the hyperparameter search not generalizing well to the dance performances in the test set. The results seem to be inconsistent and less than optimal when inspecting the online models' segmentation visually as well. Figure 65 shows that the offline model LS-USS LREA yields a reasonably good segmentation, while the Ɛ real time LS-USS LTEA algorithm underpredicts.

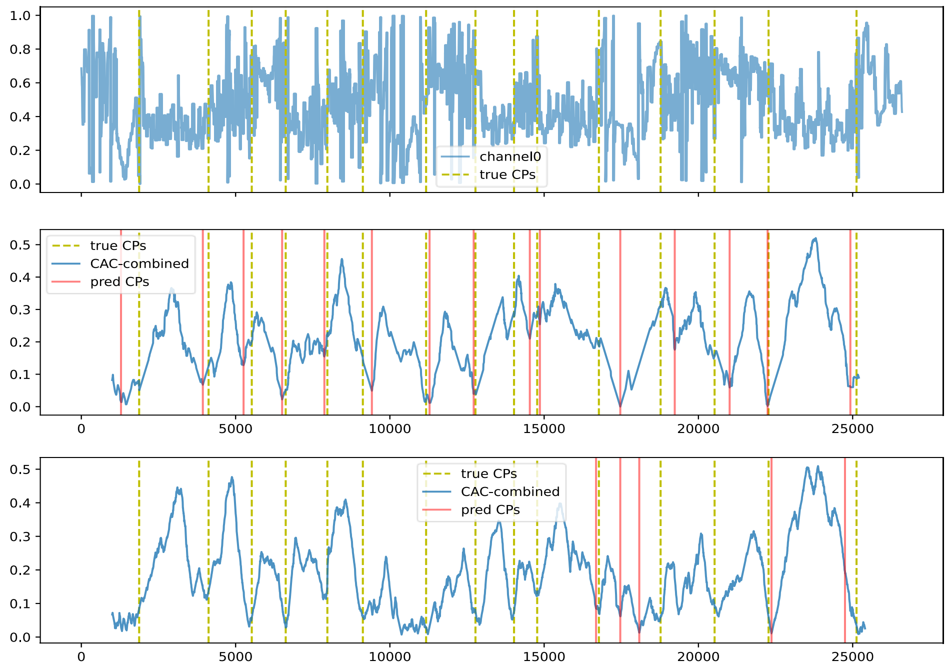

Figure 65: Top: Channel 0 of d24 in the Dance dataset. Middle: Segmentation using LS-USS – LREA- Bottom: Segmentation using LS-USS – LTEA.

5.1.1.4 EMG artificial

Offline	Datascaler	N_W	TC	Step-size	Autoencoder	Regime Score Mean
TSSTAFL - LREA	StandardScaler	500	-	150	Fully Connected	0.02536
FLUSS - REA	StandardScaler	50	2000	1	None	0.00797
FLUSS - LREA	StandardScaler	50	1500	1	None	0.00718
LS-USS - REA	StandardScaler	50	2000	1	ConvModel	0.00251
LS-USS - LREA	StandardScaler	50	1500	1	ConvModel	0.00214

Table 14: Model configurations found from doing hyperparameter search on the EMG Artificial dataset. The leftmost column shows the mean regime score from segmenting the test set. N_W-range = [50-500], Step-size-range=[25-500]

Online	Datascaler	N_W	TC	Step-size	Autoencoder	Threshold	Prediction Loss MAE Mean
TSSTAFL - LTEA	StandardScaler	300	-	150	Fully Connected	-0.5	97.54
FLOSS - LTEA	StandardScaler	100	2000	1	None	-0.5	86.44
FLUSS - LTEA	StandardScaler	50	1500	1	None	-1	26.73
LS-USS online - LTEA	StandardScaler	50	1500	1	ConvModel	-1	16.1
LS-USS - LTEA	StandardScaler	50	2000	1	ConvModel	-0.5	6.64

Table 15: Model configurations found from doing hyperparameter search on the EMG artificial dataset. The leftmost column shows the mean Prediction loss MAE from segmenting the test set. N_W-range = [50-500], Step-size-range=[25-500]

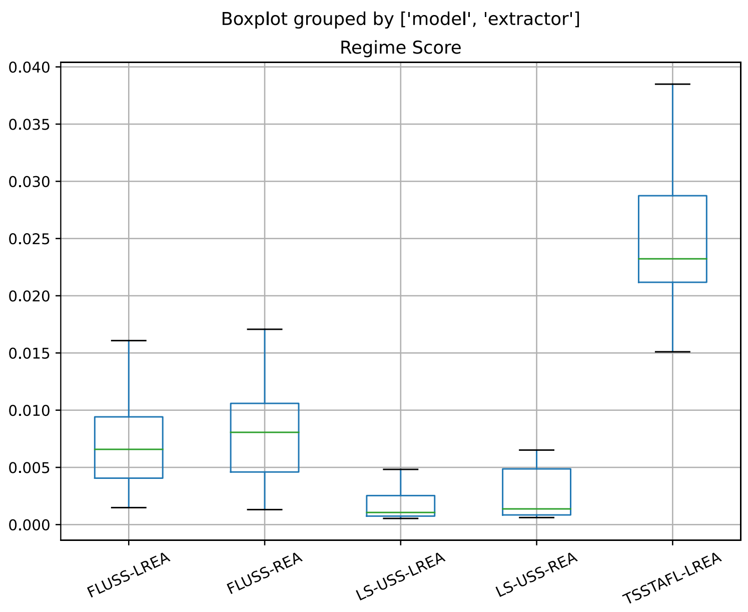

Figure 66: Boxplot showing the performance of the offline segmentation algorithms on the EMG artificial dataset

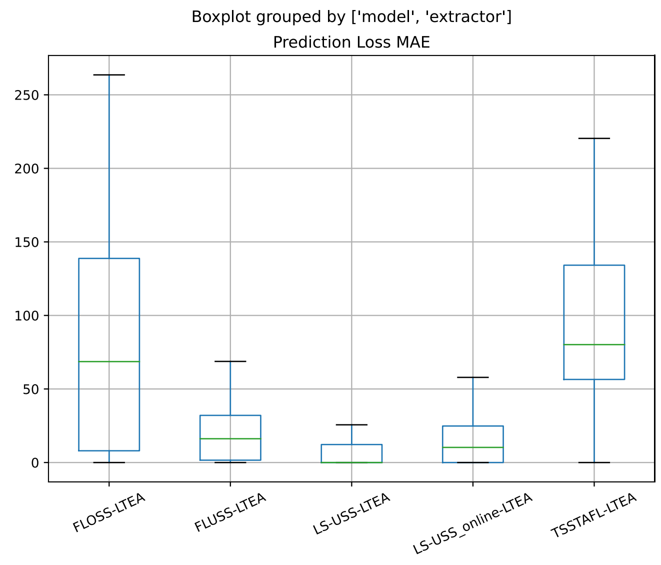

Figure 67: Boxplot showing the performance of the online segmentation algorithms on the EMG artificial dataset

The two best performing offline models on the test set are here LS-USS LREA and LS-USS REA. Both FLUSS and LS-USS get better scores using the local extractor LREA. The best performing online model is the Ɛ real time algorithm LS-USS LTEA, followed by LS-USS online LTEA. The Ɛ real-time algorithms perform better than the fully online versions of both the FLUSS and LS-USS algorithms. Scaling the data to zero mean and unit variance seems beneficial for segmentation on this dataset. The convolutional model seems to be the best autoencoder for LS-USS on this dataset.

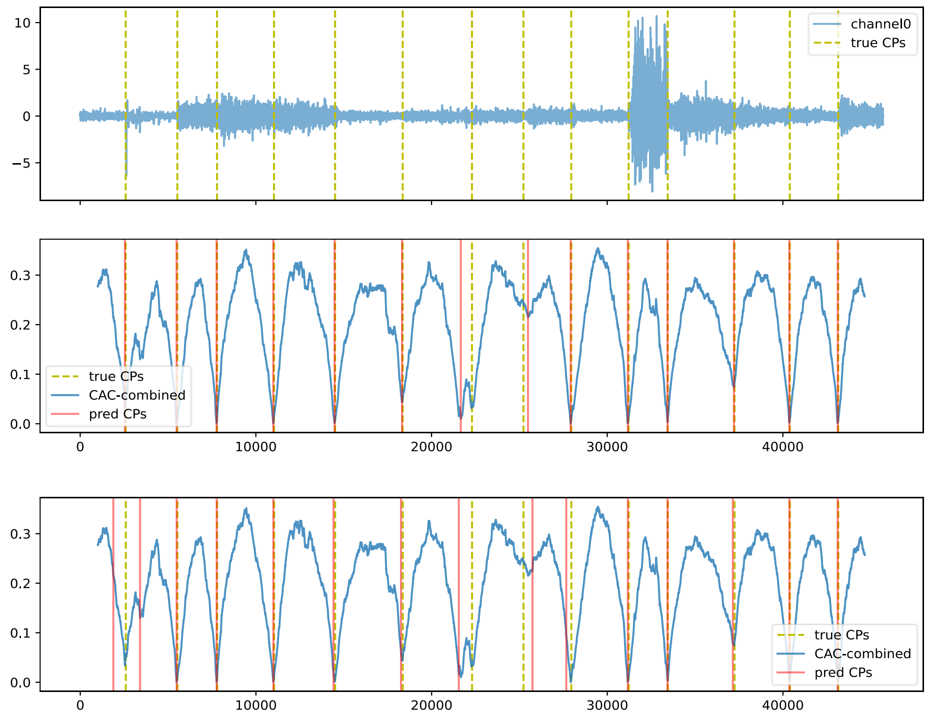

Figure 68: Top: Channel 0 of gesture19 in the EMG Artificial dataset. Middle: Segmentation using LS-USS – LREA- Bottom: Segmentation using LS-USS – LTEA.

5.1.1.5 EMG

Table 16: Model configurations found from doing hyperparameter search on the EMG dataset. The

Offline	Datascaler	N_W	TC	Step-size	Autoencoder	Regime Score Mean
TSSTAFL - LREA	RobustScaler	300	-	200	Fully Connected	0.01206
FLUSS - REA	StandardScaler	50	1500	1	None	0.00768
FLUSS - LREA	StandardScaler	50	1500	1	None	0.00593
LS-USS - REA	NoScaler	100	1000	1	ConvModel	0.00746
LS-USS - LREA	RobustScaler	50	2000	1	ConvModel	0.00452

leftmost column shows the mean regime score from segmenting the test set. N_W-range = [50-500], Step-size-range=[25-500]

Online	Datascaler	N_W	TC	Step-size	Autoencoder	Threshold	Prediction Loss MAE Mean
TSSTAFL - LTEA	StandardScaler	500	-	500	Fully Connected	-1.5	388.8155
FLOSS - LTEA	StandardScaler	200	1000	1	None	-2	163.7127
FLUSS - LTEA	StandardScaler	50	1500	1	None	-2	251.408
LS-USS online - LTEA	RobustScaler	200	1000	1	Fully Connected	-2	164.6831
LS-USS - LTEA	StandardScaler	300	1000	1	Fully Connected	-2	121.9777

Table 17: Model configurations found from doing hyperparameter search on the EMG dataset. The leftmost column shows the mean Prediction loss MAE from segmenting the test set. N_W-range = [50-500], Step-size-range=[25-500]

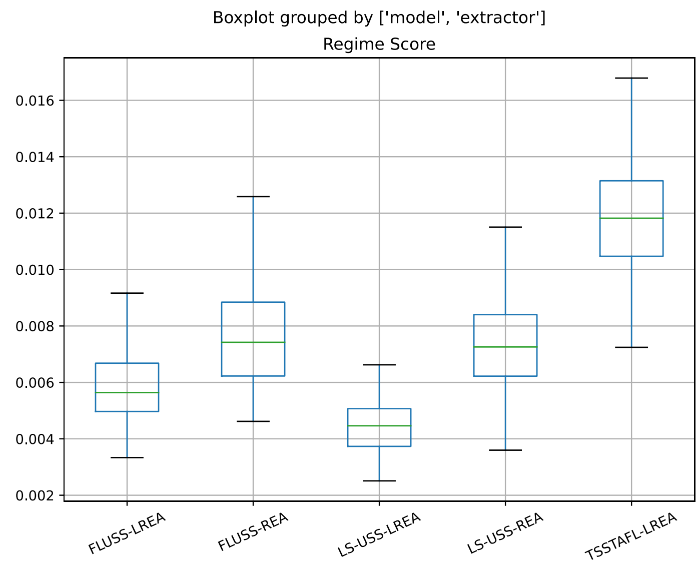

Figure 69: Boxplot showing the performance of the offline segmentation algorithms on the EMG dataset

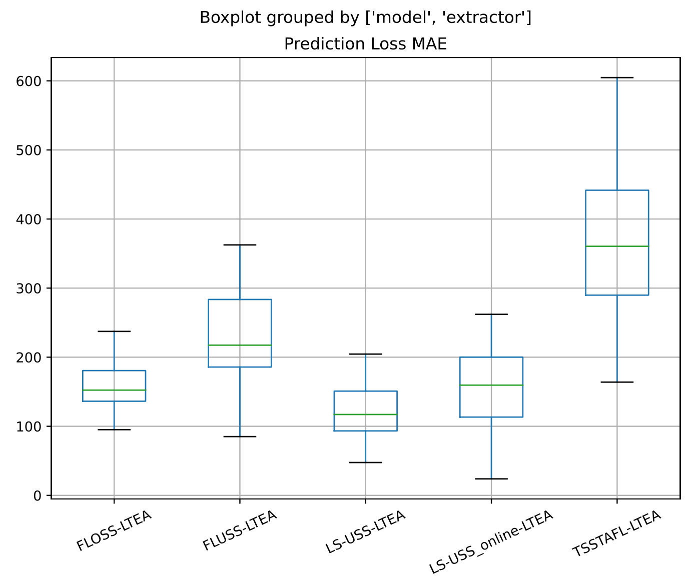

Figure 70: Boxplot showing the performance of the online segmentation algorithms on the EMG dataset

LS-USS LREA is the best performing online model on the test set. Both LS-USS and FLUSS get the best test score when using the LREA extraction algorithm. On the online dataset, the Ɛ real-time algorithm LS-USS LTEA performs the best. Nearly all the models use StandardScaler and the similar RobustScaler. The offline LS-USS models use the convolutional model, while the fully connected model is utilized for the online LS-USS models.

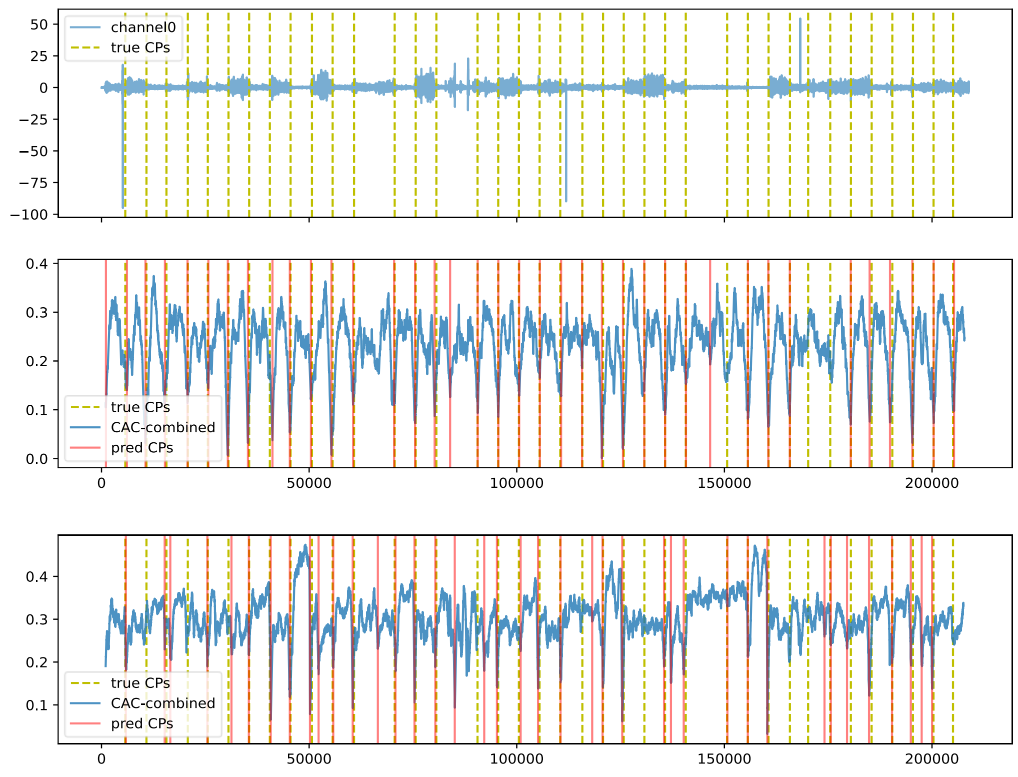

Figure 71: Top: Channel 0 of participant3_evaluation3 in the EMG dataset. Middle: Segmentation using LS-USS – LREA- Bottom: Segmentation using LS-USS – LTEA.

5.1.2 Discussion

This section will first go through some general observations about the hyperparameter search before making a more in-depth comparison between the different segmentation models.

5.1.2.1 General observations

Section 2.3.1 showed that the FLUSS and FLOSS algorithm was not very sensitive to the TC size. The same findings are observed here, as the model configurations often utilize the smaller TC choices, which might indicate that there is no benefit in including subsequences far away from the current time step when using the LS-USS or FLUSS/FLOSS models. Another observation is that the scaling used varies between the datasets, which is expected since the scaler is highly dependent on the original scaling and distribution of the time series data. An interesting result is that the autoencoder selected by the hyperparameter search for the LS-USS models (both offline and online) varies between the datasets; some use the less complex, fully connected model, while others use the convolutional model. As pointed to in previous research [36], using a smaller model with only two hidden layers might be beneficial for feature representation as it could lead to more general ways to represent the data. The drawback of simpler autoencoder models is that one risks losing some of the information needed for segment differentiation. Finding the optimal autoencoder is a complex problem as it involves a combination of factors such as layer types, feature-length, layer size, and the number of layers.

TSSTAFL usually has the lowest or among the lowest scores both for the online and offline version. As mentioned in section 2.4.2, TSSTAFL has been made to detect breakpoints, which the authors describe as human-specified changepoints. Even though most of the datasets included here use changepoints that can be segmented reasonably well by a human expert, some of the segmentation done might be outside the original scope of the TSSTAFL algorithm.

5.1.2.2 Statistical approach

A two-step procedure recommended in [49] is used to analyze the algorithms' performance. This involves using a Friedman test to rank the algorithms against each other. Then, the Holm's post-hoc test is applied using the highest-ranked algorithm as a comparison basis. The null hypothesis of the post hoc test is that the performance of the two models is the same. The null hypothesis is rejected when $p < 0.05$. Table 18 and

Table 19 show the mean test score, the Friedman rank, and the adjusted p-value on each of the datasets for the offline and online algorithms. The highest-ranked algorithm (used as the control model) on each dataset is shown in bold. The EMG dataset contains six evaluation recordings per participant, where five recordings from each participant are used to construct the test set. The statistical analysis is done on the average scores over each recording in the test set. This is done under the assumption that sufficient time has passed between the recordings so that the evaluations are iid (independent and identically distributed).

5.1.2.3 Offline Model Comparisons

	LS-USS - REA	LS-USS - LREA	FLUSS - REA	FLUSS - LREA	TSSTAFL - LREA
UCI					
Mean	**0.00687**	0.01041	0.02824	0.00931	0.01589
Friedman rank	**1.63**	2.57	4.80	2.60	3.40
H0 (Adjusted p-value)	**N/A**	1 (0.42388)	0 (0.00000)	1 (0.37627)	0 (0.00885)
Dance Artificial					
Mean	0.0257	0.01803	0.02608	**0.01735**	0.03807
Friedman rank	3.27	1.92	3.28	**1.88**	4.55
H0 (Adjusted p-value)	0 (0.00001)	1 (0.89934)	0 (0.00002)	**N/A**	0 (0.00000)
Dance					
Mean	0.05459	0.029	0.03802	**0.02969**	0.03665
Friedman rank	4.59	2.18	3.59	**1.59**	3.05
H0 (Adjusted p-value)	0 (0.00000)	1 (0.27808)	0 (0.00068)	**N/A**	0 (0.01339)
EMG Artificial					
Mean	0.00251	**0.00214**	0.00797	0.00718	0.02536
Friedman rank	1.93	**1.37**	3.53	3.17	5.00
H0 (Adjusted p-value)	1 (0.16512)	**N/A**	0 (0.00000)	0 (0.00002)	0 (0.00000)
EMG					
Mean	0.00746	**0.00452**	0.00768	0.00593	0.01206
Friedman rank	3.30	**1.00**	3.60	2.10	5.00
H0 (Adjusted p-value)	0 (0.00001)	**N/A**	0 (0.00000)	0 (0.02781)	0 (0.00000)

Table 18: Table showing the mean test score, the Friedman rank, and the adjusted p-value on each of the datasets for the offline algorithms. The highest-ranked algorithm (used as the control model) on each dataset is shown in bold. H0=1 means that the null hypothesis is accepted. H0=0 means that the null hypothesis is rejected.

LS-USS REA has the highest score on the test set but is not significantly better than LS-USS LREA and FLUSS LREA on the UCI dataset. On both the Dance and Dance artificial datasets, the lowest-ranked algorithm is FLUSS LREA which is significantly better than all algorithms except for the LS-USS LREA. On the EMG and EMG artificial datasets, the best performing model is LS-USS LREA, which is significantly better than all algorithms except for LS-USS REA on the artificial dataset. In conclusion, LS-USS LREA seems to be the most consistent performing offline segmentation algorithm over all the datasets. On every dataset except for the UCI dataset, the offline segmentation algorithms FLUSS and LS-USS got higher mean regime score when using the LREA extractor compared to using the REA extractor, which indicates that scaling the CAC based on the local statistics is useful (and often necessary) when dealing with longer time series data.

5.1.2.4 Online Model Comparisons

	LS-USS - LTEA (Ɛ-real time)	**LS-USS online - LTEA**	**FLUSS - LTEA (Ɛ-real time)**	**FLOSS - LTEA**	**TSSTAFL - LTEA**
UCI					
Mean	**45.54**	114.82	47.27	87.36	51.99
Friedman rank	**2.50**	4.20	2.56	3.12	2.60
H0 (Adjusted p-value)	**N/A**	0 (0.01294)	1	1 (0.81797)	1
Dance Artificial					
Mean	**27.15**	64.08	48.01	55.06	109.04
Friedman rank	**1.92**	3.17	2.77	2.80	4.34
H0 (Adjusted p-value)	**N/A**	0 (0.00023)	0 (0.01078)	0 (0.01078)	0 (0.00000)
Dance					
Mean	415.32	**204.9742**	281.0156	334.65	289.2028
Friedman rank	3.88	**2.18**	2.71	3.12	3.12
H0 (Adjusted p-value)	0 (0.00663)	**N/A**	1 (0.32897)	1 (0.24799)	1 (0.24799)
EMG Artificial					
Mean	**6.64**	16.1	26.73	86.44	97.54
Friedman rank	**1.82**	2.28	2.80	3.88	4.22
H0 (Adjusted p-value)	**N/A**	1 (0.25300)	0 (0.03202)	0 (0.00000)	0 (0.00000)
EMG					
Mean	**121.9777**	164.6831	251.408	163.7127	388.8155
Friedman rank	**1.50**	2.50	3.70	2.55	4.75
H0 (Adjusted p-value)	**N/A**	1 (0.07146)	0 (0.00003)	1 (0.07146)	0 (0.00000)

Table 19: Table showing the mean test score, the Friedman rank, and the adjusted p-value on each of the datasets for the online algorithms. The highest-ranked algorithm (used as the control model) on each dataset is shown in bold. H0=1 means that the null hypothesis is accepted. H0=0 means that the null hypothesis is rejected.

The highest-ranked algorithm on the UCI dataset is LS-USS LTEA, but the results are not significant. On the dance dataset, the LS-USS LTEA is statistically better than all the other models. For the Dance artificial dataset, the highest-ranked algorithm is LS-USS online, but the results on this dataset were bad in general, and the results might be somewhat random. LS-USS LTEA is also the highest-ranked algorithm on both EMG datasets and is significantly better than every algorithm except for LS-USS online on the artificial EMG dataset. The Ɛ-real time LS-USS LTEA model seems to be the most consistent performing online segmentation algorithm over all the datasets. Both Ɛ real time algorithms (LS-USS LTEA and FLUSS LTEA) achieve higher scores than their completely online counterparts (LS-USS online and FLOSS) on four out of five datasets, which is expected as the Ɛ real time models use arcs pointing both forward and backward in time, while the online models only use forward-pointing arcs. Which algorithm to use can be seen as a tradeoff between calculation time and segmentation performance.

5.1.2.5 Limitations

The segmentation outcomes for each of the algorithms are heavily dependent on the hyperparameter search. Therefore, it would be beneficial to do a more in-depth hyperparameter search that includes sampling the hyperparameters from predefined distributions instead of doing a grid search. Some of the datasets used are also a bit small for doing extensive hyperparameter search, in that the validation set does give a good indication of the real-world performance because of the limited size.

It would also be useful to do a hyperparameter sensitivity analysis for finding how sensitive LS-USS is to the choices of the different hyperparameters.

It is worth noting that LS-USS requires more hyperparameters to be specified than FLUSS due to the autoencoder. There might be an autoencoder implementation that works well for most time series data without needing extensive hyperparameter tuning, so more research into the effects of different autoencoder architectures might alleviate this drawback.

5.2 Improving Depression Detection Using Natural Segments

One of the papers written under the INTROMAT project is called **“Motor Activity Based Classification of Depression in Unipolar and Bipolar Patients”** [50]**.** This paper tried to classify non-depressed and depressed participants using various classifiers and oversampling techniques on the Depresjon dataset (Section 4.4). The features used for the classifier are statistical descriptors extracted from 24-hour fixed segments. There is evidence that depression leads to reduced daytime motor activity and increased nighttime activity compared to healthy individuals [11]. A fixed time window of 24 hours does not capture the difference in night and daytime activity, which might mean important information gets lost. Choosing the right segment size when doing time series classification can be difficult and will often have to be based on domain-specific assumptions. This is where an unsupervised segmentation algorithm can come in handy. Instead of dividing the time series into fixed segments, the segmentation algorithm finds natural segments based on the data itself. The hypothesis is that this will lead to better features, which will improve the classifier's performance.

To test this, a comparison will be made between using features extracted from fixed segment sizes and using features extracted from segments found by the unsupervised segmentation algorithm LS-USS.

5.2.1 Experiments

Since the goal is to compare the effects of different segmentation approaches, the comparisons between classification models are not the focus. Thus, only the best performing model from [50] will be implemented. The statistical features calculated from the segments are the mean activity, the standard deviation, and the number of samples with no activity

(activity level=0). The highest MCC- score(0.44) and F1 score(0.68) in [50] was achieved using the SMOTE oversampling technique in conjunction with the Random Forrest Classifier. The evaluation is done using Leave-One-User-Out validation.

For the oversampling, the implementation of SMOTE included in the python library imbalanced learn [51] is used, and for the Random Forest Classifier, the implementation in scikit-learn [52] with default parameters is utilized.

Each of the following experiments is performed 30 times because of the high variation in performance between runs. The best MCC score (for the model implemented based on [50]) was 0.47, and the best F1 score was 0.65. These metrics are similar to those found in the original paper [50], indicating that the implementation is correct.

5.2.1.1 Experiment 1

The experiment is done using LS-USS combined with the extraction algorithm LTEA instead of fixed 24-hour segments. To make it a fair comparison, the rest of the classification pipeline remains the same. To not overfit the data, best guess parameters are used for LS-USS and chosen extraction algorithm LTEA. The temporal constraint is set to 12 hours to ensure that arcs do not point to repeated activities on other days. Relatively big changes in the activity are more useful for this task since segmenting out every activity would be meaningless. Because of this, the subsequence size must be set quite large and is therefore set to 4 hours. The local window used in LTEA to normalize the CAC is set to 6 hours (this parameter is not very sensitive to changes). Figure 72 shows an example of the segmentation done by LS-USS on one of the participants. The segments make sense visually as well, as high activity periods often seem to be separated from low activity periods.

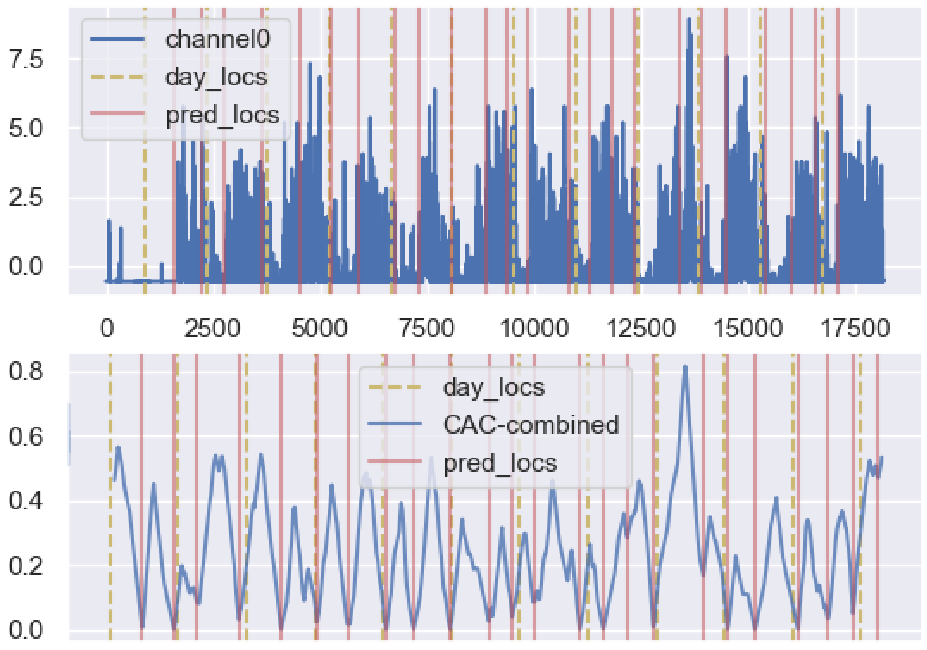

Figure 72: Top plot show 13 days of activity data from a participant. Bottom plot shows the CAC outputted by LS-USS. Notice that on this plot, the striped green lines do not represent the ground truth changepoints but a change of date.

As shown in Table 20, LS-USS outperforms the fixed segment size of 24 hours on all the performance metrics used. From the boxplots in Figure 73, one can see that the mean F1 and MCC scores using LS-USS segmentation are as high as the maximum of the fixed 24-hour segments.

	Specificity	Recall	Precision	Accuracy	F1	MCC
LS-USS	0.886	0.546	0.778	0.744	0.641	0.469
24 H fixed	0.771	0.507	0.618	0.662	0.556	0.292

Table 20: Classification results. The metrics shown are the mean scores over 30 runs

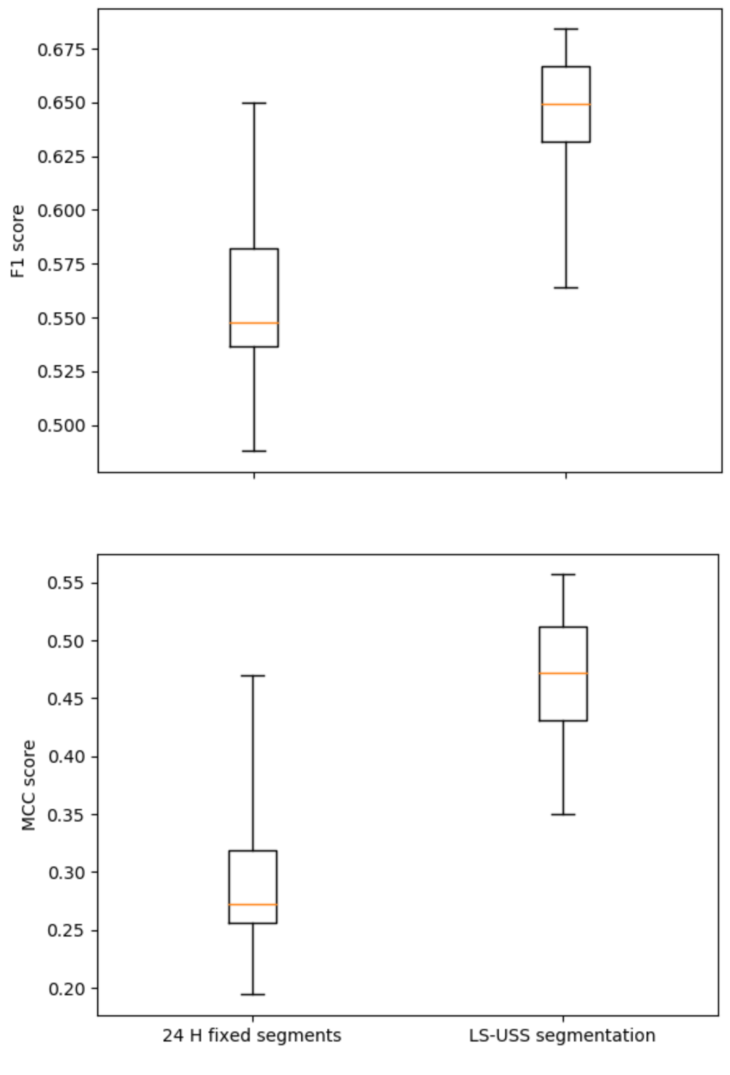

Figure 73: Boxplots of the F1 and MCC scores for 30 runs.

5.2.1.2 Experiment 2

As the ideal segmentation length for this classification problem might not be 24 hours, this experiment will compare LS-USS to using other fixed segment lengths. Here each whole hour fixed segment length between one and 24 hours is tested over 30 runs each. The results are shown in Figure 74.

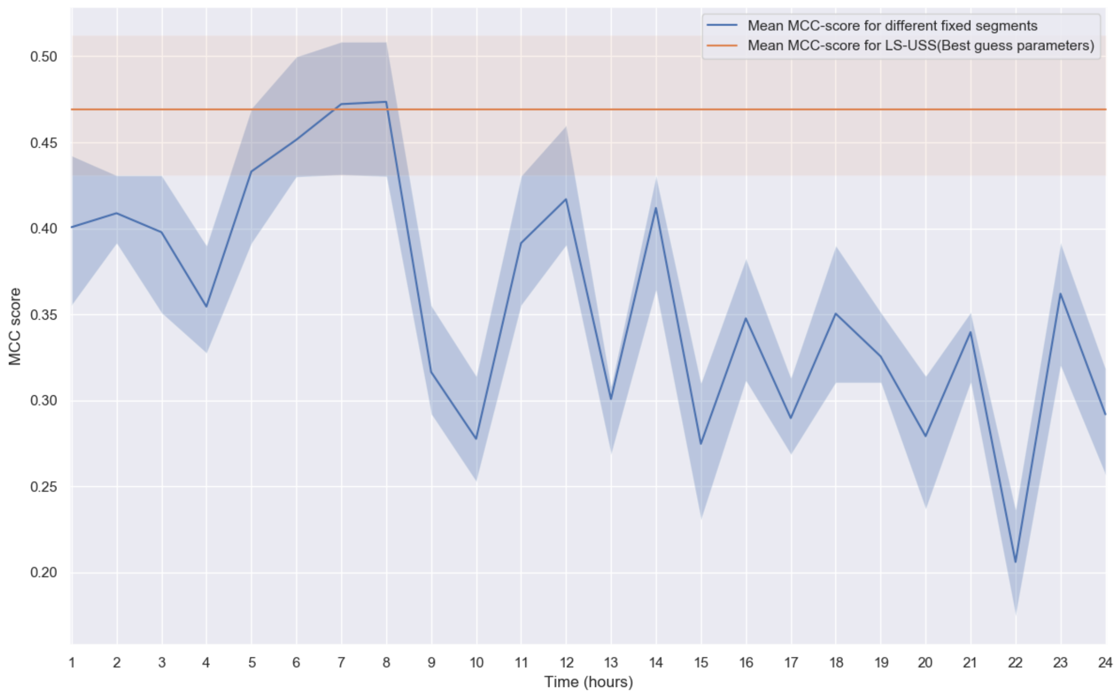

Figure 74: Performance of different fixed segment lengths (blue) vs performance of using LS-USS (orange). The MCC-scores are the mean over 30 runs. The light blue shows the IQR (Inter Quartile Range) of the fixed segments, while the light orange shows the IQR of the LS-USS segmentation.

5.2.2 Discussion

To test if the improvement seen when using LS-USS is statistically significant, the Wilcoxon signed-rank test is utilized[49]. The statistical test is done on the mean correct classification percentage when using LS-USS and fixed 24-hour segments for each participant over the 30 runs. The null hypothesis of the post hoc test is that the performance of the two models is the same. The null hypothesis is rejected when $p < 0.05$. Here, it was found that the LS-USS is significantly better than using fixed 24-hour segments.

The plot in Figure 74 clearly shows that 24-hour segments are not optimal. The best fixed segment length seems to be around 8 hours. Using LS-USS yields approximately the same performance as the optimal fixed segment length, which shows that LS-USS seems to find a near-optimal segmentation. There is also worth pointing out that when trying all the 24 different fixed segments, one also has a higher chance of overfitting to the data, so real-world performance when using the fixed segments might be worse.

5.2.2.1 Limitations

The parameters selected for the LS-USS algorithm in the experiments did require some subjective assumptions. It would be interesting to see how changing the different hyperparameters like the window size would affect the segmentation. In this case, the segmentation provided by LS-USS seemed to separate primarily based on daytime and nighttime activity.

This experiment was used to showcase the utility of unsupervised segmentation in the context of MHMS. As the main benefit of LS-USS is the way it handles multidimensional data, it would be preferable if the dataset included data from more sensors with a higher sampling rate. This proved difficult to get access to due to the sensitive nature of personal data from people suffering from mental health issues.

5.3 Automatic subsequence size using FLUSS-Welch

As mentioned in section 4.5, the UCR dataset includes recommended subsequence lengths for each dataset manually specified by a human expert. This experiment compares using the recommended subsequence length vs. using a subsequence size found using the spectral density estimation method Welch.

5.3.1 Experiment

The implementation of the FLUSS algorithm used in this experiment does not use any temporal constraint and utilizes the standard REA algorithm for extracting change points. The results are presented in Table 21.

Dataset names	FLUSS with recommended subsequence size		FLUSS with automatic subsequence size	
	Regime Score	**Matches**	**Regime Score**	**Matches**
Cane	0.0155	1/1	**0.0137**	1/1
DutchFactory	0.0134	1/1	**0.0037**	1/1
EEGRat	0.1240	1/1	**0.0900**	1/1
Fetal2013	**0.0018**	2/2	0.1627	2/2
GrandMalSeizures	0.2395	1/1	**0.2102**	1/1
GrandMalSeizures2	0.1302	1/1	**0.1129**	1/1
GreatBarbet	0.0089	2/2	**0.0048**	2/2
InsectEPG1	**0.0022**	1/1	0.0098	1/1
InsectEPG2	**0.0189**	1/1	0.0200	1/1
InsectEPG3	0.1026	1/1	**0.0300**	1/1
InsectEPG4	**0.0016**	1/1	0.0309	1/1
NogunGun	**0.0028**	1/1	0.0153	1/1
PigInternalBleeding	**0.0037**	1/1	0.0619	1/1
Powerdemand	0.0073	1/1	**0.0027**	1/1
PulsusParadoxus	0.0035	1/1	**0.0009**	1/1
RoboticDogActivityX	0.0042	1/1	**0.0007**	1/1
RoboticDogActivityY	0.0035	1/1	**0.0006**	1/1
RoboticDogActivityY2	0.0082	1/1	**0.0029**	1/1
SimpleSynthetic	**0.0070**	2/2	0.1250	1/2
SuddenCardiacDeath	**0.0067**	1/1	0.0215	1/1
SuddenCardiacDeath1	0.0084	2/2	**0.0083**	2/2
TiltABP	0.0034	1/1	**0.0022**	1/1
TiltECG	**0.0101**	1/1	0.0129	1/1
WalkJogRun	**0.0306**	2/2	0.1530	1/2

Table 21: Scores on the UCR dataset using FLUSS with recommended window size vs. automatic window size. The best scoring algorithm on each sub-dataset is bolded out.

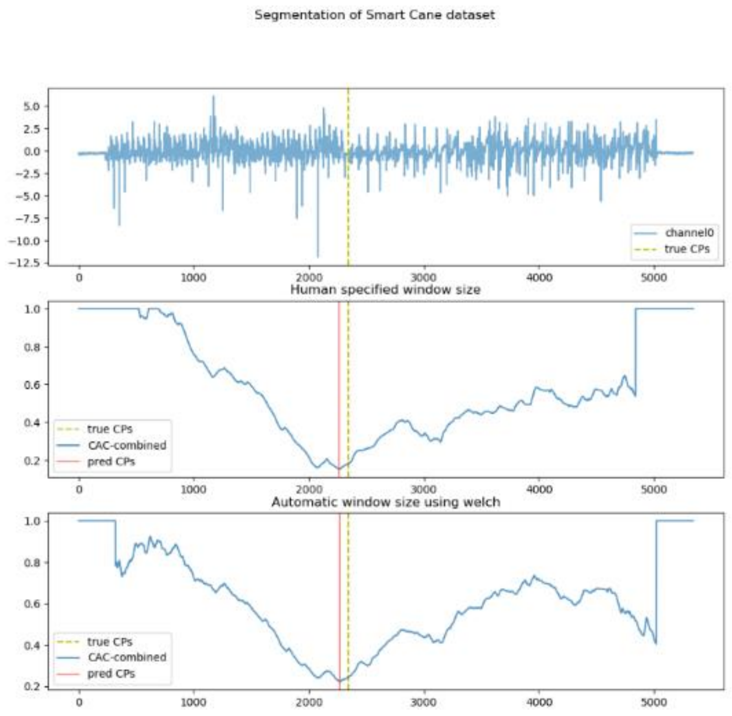

Figure 75: Segmentation done on the Smart Cane dataset.

5.3.2 Discussion

Using the automatic subsequence size produces the best performance on 14 of the 24 datasets. The Regime scores achieved by the two algorithms are not significantly different, with an adjusted p-value at 0.86, when using the Wilcoxon signed-rank test ($p < 0.05$). The results indicate that using the automatic subsequence size yields comparable results to the recommended subsequence size specified by a human expert. The main benefit of the FLUSS-Welch algorithm over regular FLUSS is that it is entirely parameter free.

5.3.2.1 Limitations

On some of the datasets containing more than one changepoint, FLUSS-Welch seems to be doing much worse. Because of this, a short inspection of some of the datasets where the algorithm seemed to fail was also done. Figure 76 and Figure 77 show that although the CACs look very similar for the automatic and recommended subsequence sizes, the predicted changepoint ends up in one “valley” when using the automatic window size on these datasets. These are not major failures and could be remedied by a larger exclusion zone.

I also tried applying FLUSS-Welch on the Dance and UCI datasets, but the resulting periodograms did not yield any useful information. This is most likely because those datasets do not have a clear periodicity or that the periodicity changes through the time series. Implementing a version of FLUSS-Welch that dynamically changes the subsequence length by estimating the periodogram based on a local window could help resolve this, with the added benefit of making FLUSS more robust to data where the periodicity of the time series changes drastically over time.

Welch’s method for automatically finding a good subsequence size could also be beneficial for LS-USS. Analysis of this is not provided in this thesis, but this could be interesting to look into in future research.

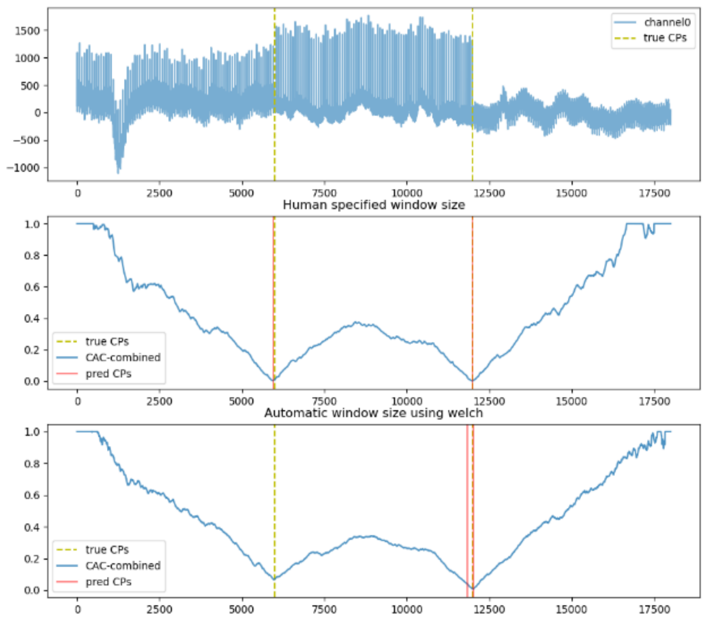

Figure 76: Segmentation done on the Fetal2013 dataset. The top plot shows channel0. The middle plot shows the segmentation result when using standard FLUSS. The Bottom plot shows the segmentation result when using FLUSS-Welch. The CACs are similar, but when using the FLUSS-Welch both changepoints ends up in the same “valley”

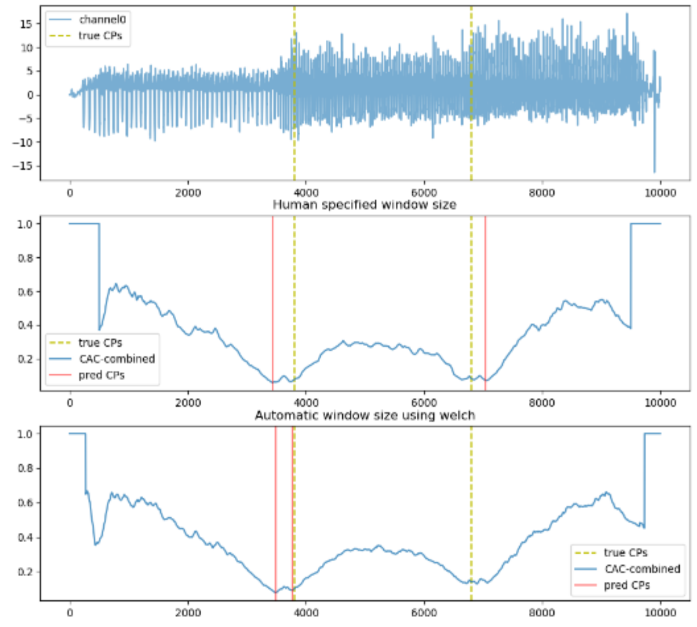

Figure 77: Segmentation done on the WalkJogRun dataset. The top plot shows channel0. The middle plot shows the segmentation result when using standard FLUSS. The Bottom plot shows the segmentation result when using FLUSS-Welch. The CACs are similar, but when using the FLUSS-Welch the both changepoints ends up in the same “valley”

6 Conclusion

This thesis intended to research and develop unsupervised segmentation algorithms with the main focus being on bio signal-based time series data. One contribution is the parameter-free segmenation algorithm FLUSS-Welch obtained by estimating the window size using Welch's method. The algorithm delivered promising results that are shown on a broad selection of datasets from the UCR Time Series Semantic Segmentation Archive[2].

Another important contribution is the segmentation algorithms LS-USS and LS-USS online. Through extensive testing conducted on both artificial and real-world datasets from various domains and sensors, it was found that LS-USS generally delivers on par or better segmentation scores compared to other state-of-the-art algorithms such as FLUSS/FLOSS and TSSTAFL. The LS-USS algorithms have many desirable properties. They can be implemented both online and in an "anytime" fashion. They are domain agnostic and do not need any extensive tuning of hyperparameters. They do not make any statistical assumption about the input data. The LSMP component used in LS-USS also shows that it is possible to calculate a temporarily constrained matrix profile from feature vectors by exploiting highly parallelized hardware. The same methods used for constructing the LSMP can be used on every Representation Learning algorithm that produces a feature vector, which presents an excellent potential for further research.

The extraction algorithms LREA and LTEA developed in the thesis also show potential. LREA usually outperformed the more global REA algorithm, which shows that scaling the CAC based on the local statistics is highly useful, especially for long time series data. The online extraction algorithm LTEA uses the same CAC scaling as LREA but instead of extracting the n lowest "valley" points, it utilizes a threshold. In contrast to REA and LREA, LTEA does not need any information on the number of change points to extract, making it applicable to more real-world segmentation problems.

To demonstrate a real-world application, the dataset Depresjon was used. This was to see if using natural segments, found by unsupervised segmentation, would increase the robustness of mood state detection (depression and control). Using the LS-USS CPD algorithm in conjunction with the LTEA extraction algorithm improved the previous best results significantly and proved to produce near-optimal segments for classifying depressed and non-depressed participants.

6.1 Future work

As seen in section 2.1, one of the most critical areas of future research in CPD algorithms is developing fast online or anytime algorithms. The LS-USS algorithms proposed in this thesis are fast enough for many time series segmentation applications, but for very time-sensitive tasks, there would be a clear benefit to making a tradeoff between computation time and detection performance. This can be implemented by decreasing the resolution of the latent space matrix profile (LSMP) by only calculating the similarity between every n^{th} subsequence. For example, by only calculating the LSMP for every 10^{th} subsequence in the latent all subsequence set **F**, the segmentation time can be reduced approximately ten times. The same concept could be used to make an anytime algorithm. As each similarity score in the LSMP can be calculated in parallel, doing it in temporal order is not necessary. Instead of calculating the matrix profile from start to end, it should be possible to make an algorithm that starts by calculating the matrix profile at a low resolution before increasing the resolution gradually. If the process is interrupted, the current best resolution LSMP will be used.

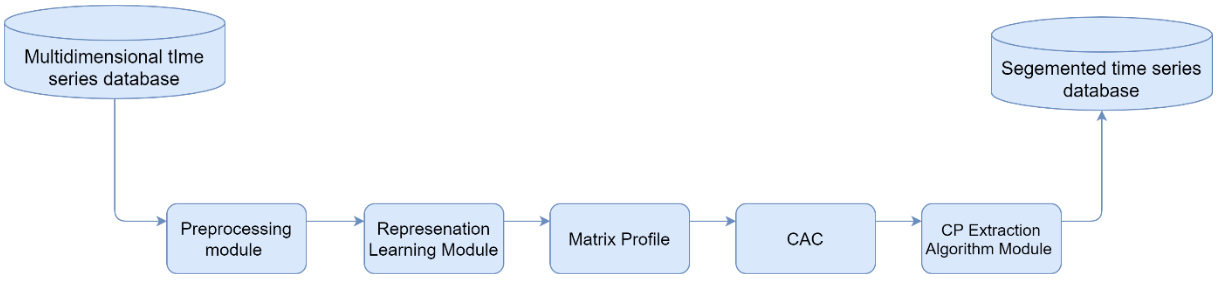
Multidimensional tIme series database
Preprocessing module
Represenation Learning Module
Matrix Profile
CAC
CP Extraction Algorithm Module
Segemented time series database

Figure 78: A proposed modular approach to unsupervised time series segmentation.

One of the main contributions of this thesis is to show the potential of representation learning in conjunction with constructing the matrix profile and CAC. To showcase future research possibilities, a modular approach to time series segmentation is presented in Figure 78, which makes it possible to isolate the different areas of research. The CAC and Matrix profile format must be static, but the method for calculating the matrix profile, either approximately or exact, can be investigated further. It would also be interesting to look at the effects of using other similarity metrics than Euclidian distance, for example cosine similarity, when constructing the matrix profile. As the CAC has the same format regardless of the other modules, it is possible to design general CP extraction algorithms that work for all representation learning techniques.

The representation learning module has great potential for future research. In this thesis, only two autoencoder implementations were implemented, but there are plenty of other model architectures to try. Two-dimensional convolutional models might, for example, better capture the temporal and inter-channel relations. The background section 2.4.1 also mentioned other representation learning methods such as PCA and NLP techniques such as word2vec. Another common autoencoder architecture used on time series data is LSTM autoencoders. More recently, transformer-based models have also been used to do representation learning on multivariate time series data [53].

The experiments conducted in this thesis have mainly been on multidimensional data where the channels are from similar sensors, similar sampling rates, and numerical data only. It would be interesting to use the model on time series data containing channels from a broader range of sources with both numerical and categorical variables. This could have great potential for applications in mental health monitoring systems (MHMS) since much of the data are gathered from varying sources and from both wearables and external sensors. The authors of the survey paper “Mental health monitoring with multimodal sensing and machine learning: A survey” [9] suggests that a robust MHMS system should use a combination of sensing types. Figure 79 shows the combinations of sensors used in the papers reviewed in the survey.

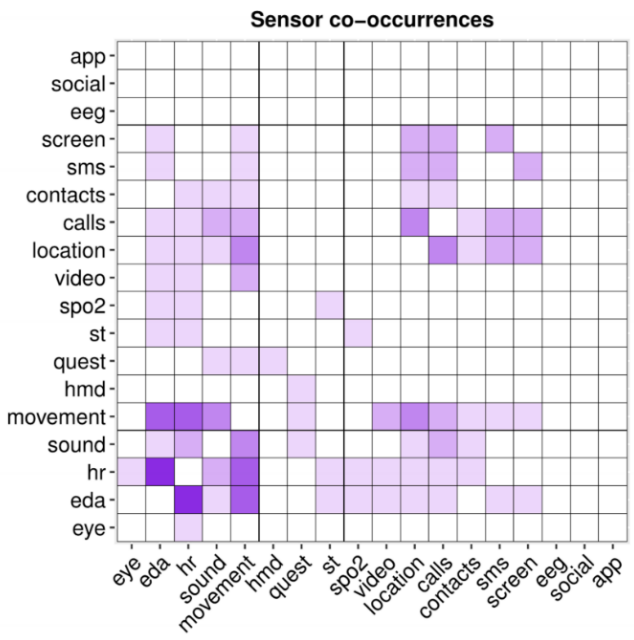

Figure 79: Sensor co-occurrences.app: application usage, social: social media, EEG: electroencephalogram, screen: smartphone screen state, contacts: phone contact information, calls: phone call logs, location: GPS and cell towers,spo2: blood oxygen saturation level.

Doing an in-depth analysis on how the number of channels affects the performance of LS-USS would also be beneficial. As the algorithms used could be implemented more efficiently, the thesis does not include any extensive comparisons on processing time for different time series data. This is also something to look more into in future research.

7 References

[1] A. Radulescu, Y. S. Shin, and Y. Niv, “Human Representation Learning,” *Annual Review of Neuroscience*, vol. 44, no. 1, p. null, 2021, doi: 10.1146/annurev-neuro-092920-120559.

[2] “Eamonn Keogh.” https://www.cs.ucr.edu/~eamonn/FLOSS/ (accessed Sep. 24, 2020).

[3] P. Yang, G. Dumont, and J. M. Ansermino, “Adaptive change detection in heart rate trend monitoring in anesthetized children,” *IEEE Trans Biomed Eng*, vol. 53, no. 11, pp. 2211–2219, Nov. 2006, doi: 10.1109/TBME.2006.877107.

[4] S. Aminikhanghahi and D. J. Cook, “A Survey of Methods for Time Series Change Point Detection,” *Knowl Inf Syst*, vol. 51, no. 2, pp. 339–367, May 2017, doi: 10.1007/s10115-016-0987-z.

[5] “About INTROMAT – Intromat.” https://intromat.no/about/ (accessed Jan. 27, 2020).

[6] H. A. Whiteford, A. J. Ferrari, L. Degenhardt, V. Feigin, and T. Vos, “The Global Burden of Mental, Neurological and Substance Use Disorders: An Analysis from the Global Burden of Disease Study 2010,” *PLoS One*, vol. 10, no. 2, Feb. 2015, doi: 10.1371/journal.pone.0116820.

[7] G. V. Polanczyk, G. A. Salum, L. S. Sugaya, A. Caye, and L. A. Rohde, “Annual Research Review: A meta-analysis of the worldwide prevalence of mental disorders in children and adolescents,” *Journal of Child Psychology and Psychiatry*, vol. 56, no. 3, pp. 345–365, 2015, doi: 10.1111/jcpp.12381.

[8] H. A. Whiteford *et al.*, “Global burden of disease attributable to mental and substance use disorders: findings from the Global Burden of Disease Study 2010,” *The Lancet*, vol. 382, no. 9904, pp. 1575–1586, Nov. 2013, doi: 10.1016/S0140-6736(13)61611-6.

[9] E. Garcia-Ceja, M. Riegler, T. Nordgreen, P. Jakobsen, K. J. Oedegaard, and J. Tørresen, “Mental health monitoring with multimodal sensing and machine learning: A survey,” *Pervasive and Mobile Computing*, vol. 51, pp. 1–26, Dec. 2018, doi: 10.1016/j.pmcj.2018.09.003.

[10] T. text provides general information S. assumes no liability for the information given being complete or correct D. to varying update cycles and S. C. D. M. up-to-D. D. T. R. in the Text, “Topic: Fitness & activity tracker,” *www.statista.com*. https://www.statista.com/topics/4393/fitness-and-activity-tracker/ (accessed Mar. 16, 2020).

[11] C. Burton, B. McKinstry, A. Szentagotai Tătar, A. Serrano-Blanco, C. Pagliari, and M. Wolters, “Activity monitoring in patients with depression: a systematic review,” *J Affect Disord*, vol. 145, no. 1, pp. 21–28, Feb. 2013, doi: 10.1016/j.jad.2012.07.001.

[12] S. Gharghabi *et al.*, “Domain agnostic online semantic segmentation for multi-dimensional time series,” *Data Min Knowl Disc*, vol. 33, no. 1, pp. 96–130, Jan. 2019, doi: 10.1007/s10618-018-0589-3.

[13] E. Garcia-Ceja *et al.*, “Depresjon: a motor activity database of depression episodes in unipolar and bipolar patients,” in *Proceedings of the 9th ACM Multimedia Systems Conference on - MMSys ’18*, Amsterdam, Netherlands, 2018, pp. 472–477. doi: 10.1145/3204949.3208125.

[14] Y. Zhu, C.-C. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh, “Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds,” in *2018 IEEE International Conference on Data Mining (ICDM)*, Singapore, Nov. 2018, pp. 837–846. doi: 10.1109/ICDM.2018.00099.

[15] M. Viberg, “Subspace Methods in System Identification,” *IFAC Proceedings Volumes*, vol. 27, no. 8, pp. 1–12, Jul. 1994, doi: 10.1016/S1474-6670(17)47689-0.

[16] S. Gharghabi *et al.*, “Domain agnostic online semantic segmentation for multi-dimensional time series,” *Data Mining and Knowledge Discovery*, vol. 33, pp. 1–35, Sep. 2018, doi: 10.1007/s10618-018-0589-3.

[17] E. Keogh and J. Lin, “Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research,” p. 20.

[18] E. Keogh and J. Lin, “Clustering of time-series subsequences is meaningless: implications for previous and future research,” *Knowl Inf Syst*, vol. 8, no. 2, pp. 154–177, Aug. 2005, doi: 10.1007/s10115-004-0172-7.

[19] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans, “Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data,” in *2011 IEEE 11th International Conference on Data Mining*, Dec. 2011, pp. 547–556. doi: 10.1109/ICDM.2011.146.

[20] C.-C. M. Yeh *et al.*, “Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets,” Dec. 2016, pp. 1317–1322. doi: 10.1109/ICDM.2016.0179.

[21] T. Rakthanmanon *et al.*, *Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping*, vol. 2012. 2012. doi: 10.1145/2339530.2339576.

[22] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, “A Public Domain Dataset for Human Activity Recognition Using Smartphones,” *Computational Intelligence*, p. 6, 2013.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[24] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *arXiv:1505.04597 [cs]*, May 2015, Accessed: Nov. 18, 2020. [Online]. Available: http://arxiv.org/abs/1505.04597

[25] M. Perslev, M. Jensen, S. Darkner, P. J. Jennum, and C. Igel, “U-Time: A Fully Convolutional Network for Time Series Segmentation Applied to Sleep Staging,” *Advances in Neural Information Processing Systems*, vol. 32, pp. 4415–4426, 2019.

[26] F. Isensee, P. F. Jaeger, P. M. Full, I. Wolf, S. Engelhardt, and K. H. Maier-Hein, “Automatic Cardiac Disease Assessment on cine-MRI via Time-Series Segmentation and Domain Specific Features,” in *Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges*, Cham, 2018, pp. 120–129. doi: 10.1007/978-3-319-75541-0_13.

[27] M. Sakurada and T. Yairi, “Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction,” in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, Gold Coast, Australia QLD, Australia, Dec. 2014, pp. 4–11. doi: 10.1145/2689746.2689747.

[28] C. Zhou and R. C. Paffenroth, “Anomaly Detection with Robust Deep Autoencoders,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, NS, Canada, Aug. 2017, pp. 665–674. doi: 10.1145/3097983.3098052.

[29] C. Zhang *et al.*, “A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data,” *AAAI*, vol. 33, no. 01, Art. no. 01, Jul. 2019, doi: 10.1609/aaai.v33i01.33011409.

[30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” p. 9.

[31] H. Kim *et al.*, “Representation learning for unsupervised heterogeneous multivariate time series segmentation and its application,” *Computers & Industrial Engineering*, vol. 130, pp. 272–281, Apr. 2019, doi: 10.1016/j.cie.2019.02.029.

[32] A. Qahtan, S. Wang, B. Alharbi, and X. Zhang, “A PCA-Based Change Detection Framework for Multidimensional Data Streams,” p. 10.

[33] “UCI Machine Learning Repository: EEG Eye State Data Set.” https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State# (accessed Nov. 20, 2020).

[34] “UCI Machine Learning Repository: Human Activity Recognition Using Smartphones Data Set.” https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones (accessed Sep. 25, 2020).

[35] A. Mesaros *et al.*, “Detection and Classification of Acoustic Scenes and Events: Outcome of the DCASE 2016 Challenge,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 2, pp. 379–393, Feb. 2018, doi: 10.1109/TASLP.2017.2778423.

[36] W.-H. Lee, J. Ortiz, B. Ko, and R. Lee, “Time Series Segmentation through Automatic Feature Learning,” *arXiv:1801.05394 [cs, stat]*, Jan. 2018, Accessed: Jun. 14, 2020. [Online]. Available: http://arxiv.org/abs/1801.05394

[37] W.-H. Lee, J. Ortiz, B. Ko, and R. Lee, “Time Series Segmentation through Automatic Feature Learning,” *arXiv:1801.05394 [cs, stat]*, Jan. 2018, Accessed: Jun. 14, 2020. [Online]. Available: http://arxiv.org/abs/1801.05394

[38] P. Welch, “The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms,” *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, Jun. 1967, doi: 10.1109/TAU.1967.1161901.

[39] S. Law, “STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining,” *JOSS*, vol. 4, no. 39, p. 1504, Jul. 2019, doi: 10.21105/joss.01504.

[40] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017, Accessed: Dec. 09, 2020. [Online]. Available: http://arxiv.org/abs/1412.6980

[41] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv:1603.07285 [cs, stat]*, Jan. 2018, Accessed: Dec. 09, 2020. [Online]. Available: http://arxiv.org/abs/1603.07285

[42] UlysseCoteAllard, *UlysseCoteAllard/LongTermEMG*. 2021. Accessed: May 16, 2021. [Online]. Available: https://github.com/UlysseCoteAllard/LongTermEMG

[43] “PyTorch.” https://www.pytorch.org (accessed Apr. 05, 2021).

[44] “SciPy — SciPy v1.5.3 Reference Guide.” https://docs.scipy.org/doc/scipy/reference/index.html (accessed Oct. 26, 2020).

[45] D. St-Onge, “Expressive motion with dancers.” IEEE, Apr. 22, 2018. Accessed: Sep. 28, 2020. [Online]. Available: https://ieee-dataport.org/documents/expressive-motion-dancers

[46] U. Côté-Allard *et al.*, “Virtual Reality to Study the Gap Between Offline and Real-Time EMG-based Gesture Recognition,” *arXiv:1912.09380 [cs, stat]*, Dec. 2019, Accessed: Aug. 31, 2020. [Online]. Available: http://arxiv.org/abs/1912.09380

[47] “(PDF) A Low-Cost, Wireless, 3-D-Printed Custom Armband for sEMG Hand Gesture Recognition,” *ResearchGate*. https://www.researchgate.net/publication/333988468_A_Low-Cost_Wireless_3-D-Printed_Custom_Armband_for_sEMG_Hand_Gesture_Recognition (accessed Sep. 29, 2020).

[48] E. Garcia-Ceja *et al.*, “Depresjon: a motor activity database of depression episodes in unipolar and bipolar patients,” in *Proceedings of the 9th ACM Multimedia Systems Conference on - MMSys ’18*, Amsterdam, Netherlands, 2018, pp. 472–477. doi: 10.1145/3204949.3208125.

[49] J. Demsˇar and J. Demsar, “Statistical Comparisons of Classifiers over Multiple Data Sets,” p. 30.

[50] E. Garcia-Ceja *et al.*, “Motor Activity Based Classification of Depression in Unipolar and Bipolar Patients,” in *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*, Karlstad, Jun. 2018, pp. 316–321. doi: 10.1109/CBMS.2018.00062.

[51] “Welcome to imbalanced-learn documentation! — imbalanced-learn 0.7.0 documentation.” https://imbalanced-learn.org/stable/ (accessed Nov. 16, 2020).

[52] “scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/ (accessed Oct. 26, 2020).

[53] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, “A Transformer-based Framework for Multivariate Time Series Representation Learning,” *arXiv:2010.02803 [cs]*, Dec. 2020, Accessed: Dec. 14, 2020. [Online]. Available: http://arxiv.org/abs/2010.02803