Otto-von-Guericke-University Magdeburg
Faculty of Computer Science

Master Thesis

A comparison of Time Series Classification Algorithms based on their ability to learn on diminishing time series

Author:
Ismail, Wahba

Matriculation Number:
217526

Examiner and Supervisor:
Prof. Dr. Myra Spiliopoulou
Knowledge Management and Discovery Lab
Institute of Technical and Business Information Systems
Otto-von-Guericke-University Magdeburg

2nd Supervisor:
Name
Institute
University

day.month.year

Wahba, Ismail:
A comparison of Time Series Classification Algorithms based on their ability to learn in an Early Classification Context
Master Thesis, Otto-von-Guericke-University Magdeburg, year.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur eget porta erat. Morbi consectetur est vel gravida pretium. Suspendisse ut dui eu ante cursus gravida non sed sem. Nullam sapien tellus, commodo id velit id, eleifend volutpat quam. Phasellus mauris velit, dapibus finibus elementum vel, pulvinar non tellus. Nunc pellentesque pretium diam, quis maximus dolor faucibus id. Nunc convallis sodales ante, ut ullamcorper est egestas vitae. Nam sit amet enim ultrices, ultrices elit pulvinar, volutpat risus.

Acknowledgement

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur eget porta erat. Morbi consectetur est vel gravida pretium. Suspendisse ut dui eu ante cursus gravida non sed sem. Nullam sapien tellus, commodo id velit id, eleifend volutpat quam. Phasellus mauris velit, dapibus finibus elementum vel, pulvinar non tellus. Nunc pellentesque pretium diam, quis maximus dolor faucibus id. Nunc convallis sodales ante, ut ullamcorper est egestas vitae. Nam sit amet enim ultrices, ultrices elit pulvinar, volutpat risus.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BOSS	Bag of SFA Symbols
CNN	Convolutional Neural Networks
DFT	Discrete Fourier Transformation
DTW	Dynamic Time Warping
ECTS	Early Classification of Time Series
EE	Elastic Ensemble
ERP	Edit Distance with Real Penalty
eTSCA	Early Time Series Classification Algorithms
FS	Fast Shapelets
HM	Harmonic Mean
IG	Information Gain
i.i.d	Independent and Identically Distributed
KNN	K-Nearest Neighbor Algorithm
KNNED	K-Nearest Neighbor using Euclidean Distance
LCSS	Longest Common Subsequence
LOOCV	Leave One Out Cross Validation
LS	Learned Shapelets
MCB	Multiple Coefficient Binning
MPL	Mimimum Prediction Length
MSM	Move-Split-Merge
MTSC	Multivariate Time Series Classification
PF	Proximity Forest
SAX	Symbolic Aggregate Approximation
SFA	Symbolic Fourier Approximation
ST	Shapelet Transform
TSC	Time Series Classification
TSCA	Time Series Classification Algorithms
TSF	Time Series Forest
TWE	Time Warp Edit
WDTW	Weighted Dynamic Time Warping

Chapter 1

Introduction

Time Series Classification is a field of machine learning that has grabbed the attention of many reasearches in the last decade. Time series data exists, by nature, in numerous real scenarios; medical examination records of patients{reference}, signal processing{reference}, weather forecasting{reference} and astronomy{reference} are some of them.

Classification of time series data has been tackled with different objectives; the first is concerned with the accuracy of classification as well as space and time complexity. This objective is referred to simply as Time Series Classification (TSC). While the second objective adds the factor of earliness as a primary goal and is referred to as Early Time Series Classification (eTSC).

Numerous algorithms have been introduced to tackle the problem of Time Series Classification. According to the [Bag+17], these algorithms can be divided, based on their technique, into six groups.

Whole time series algorithms{reference} compare two time series, usually by employing an elastic distance measure between all data points of both time series. Phase dependent interval algorithms{reference} operate by extracting informative features from intervals of time series, they are more suitable for long and noisy data than whole time series algorithms. Phase independent interval algorithms{reference} are used when a class can be identified using a single or multiple patterns regardless of when they occur during the time series. Dictionary based algorithms{reference} consider the number of repetitions of patterns as a factor of classification and not just simple occurence of one. Ensembles{reference} combine the power of different algorithms, either of different or same core technique, then make the final classification decision based on voting. In addition to the previous algorithms, there are also deep learning time series algorithms which build classifiers using generative as well as discriminative models.

On the other hand, Early Time Series Classification algorithms are designed to deal with less data in order to achieve earliness of prediction, but of course this comes with a price of accuracy. Many of the ideas applied in TSC have also been applied in eTSC; including 1-NN with Mimimum Prediction Length (MPL){reference}, Phase independent intervals{reference}, generative classifiers{reference} and ensembles{reference}.

Both, Time Series Classification Algorithms (TSCA) and Early Time Series Classification Algorithms (eTSCA), have introduced well performing algorithms in terms of their respective performance measures. Their algorithms have been tested on pub-

licly available archives{reference}; to benchmark their performance on a diverse set of datasets with different characteristics.

According to [Bag+17], based on the "No free lunch theorem", no specific algorithm has proven to prevail over all others. This means that different problems with different datasets would require a choice between the algorithms based on how they perform on them, specially for non-public or non-experimented datasets. In this thesis, we tackle this idea; by offering a framework that runs state-of-the-art algorithms on the provided dataset and provides analyses about the performance of each algorithm.

Also due to their different objectives, TSCA and eTSCA have been dealt with as two different families. Which leaves studying the relationship between both algorithm families an open area for research. We study the relationship between TSCA and eTSCA, by extending TSCA to deal with earliness as a main objective and compare how they perform in an early time series classification problem context.

Goal of this Thesis

This master thesis had two main goals. The first goal was to create a testbed for comparing different algorithms on a non-public dataset. While the second one was to study the relationship between the two families of algorithms; TSCAs and eTSCAs.

The first goal was motivated by [Bag+17], one of the most comprehensive review papers in the time series field. With it's release, Bagnall et. al has set the foundation methodology for accurately benchmarking the performance of TSCAs for the ,at that time, currently existing and for algorithms that will be developed in the future. In their experiment, they have used 85 datasets publicly available from UCR and UEA, the biggest two data archives. Our goal was to offer a testbed, which can be used on private datasets. It runs state of the art algorithms, then provides analysis about their classification performance. The provided analysis can help, based on empirical evidence, choose the best fitting algorithm in accordance with the problem at hand.

As for the second goal, we extended the study of relationship between TSCAs and eTSCAs. Both families offer a wide variety of algorithms, but have different objectives and thus have different approaches in their learning processes. TSCAs focus primarily on the accuracy of the classification. In order to achieve this goal, full utilization of the whole time series data is done to achieve the highest possible accurate results. While eTSCAs objective tries to maximize both accuracy and earliness together, which is hard to attain because of the contradicting nature between both{reference}. This is why eTSCAs try to learn with as least possible data points as possible while maintaining classification accuracy. This study investigated the ability of TSCAs to perform in a simulated early classification context. TSCAs were trained on shortened training data, while keeping record of models' accuracy measure in comparison to a baseline utilizing complete training data points.

Structure of the Thesis

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur eget porta erat. Morbi consectetur est vel gravida pretium. Suspendisse ut dui eu ante cursus gravida non sed sem. Nullam sapien tellus, commodo id velit id, eleifend volutpat quam. Phasellus mauris velit, dapibus finibus elementum vel, pulvinar non tellus. Nunc pellentesque pretium diam, quis maximus dolor faucibus id. Nunc convallis sodales ante, ut ullamcorper est egestas vitae. Nam sit amet enim ultrices, ultrices elit pulvinar, volutpat risus.

Chapter 2

Related Work

The main contribution of this thesis is to provide a framework that compares TSCAs in an early classification context. The idea of creating frameworks to standardize the process of comparison between algorithms is not new to the time series data domain. It was actually needed due to the increasing interest in time series data and the vast amount of newly introduced algorithms. In this chapter we will briefly discuss 3 other frameworks that had been developed to compare TSCA on univariate and multivariate data sets.

2.1 The great time series classification bake off

One of the most famous studies in comparing TSCAs is [Bag+17] and which has inspired some of the work done in this thesis. At the time when this paper was published, the TSC domain had already witnessed hundreds of algorithms being proposed, each claiming superior performance. There was no clear structure, at the time, for experimenting and benchmarking algorithms' performance, which pushed for having a more concrete structure to be followed in succeeding research. The primary goal of the experiment was to assess the average accuracy of the different classifiers on a broad set of univariate data sets. The Scalability and Efficiency of the classifiers were considered as secondary goals.

2.1.1 Inclusion Criteria

The experiment collected 18 different classifiers to be compared with 2 baseline classifiers on the UCR data archive (discussed in 4.1.1). Algorithms were selected based on three criteria; an algorithm had to be published in a high impact conference or journal, used some data sets from the UCR archive to assess performance and the availability of the code.

In order to better describe the various techniques applied by the algorithms and what differences characterize each of them, they formulated a taxonomy which groups algorithms based on the features they try to find to discriminate between different classes. These groups were; whole series similarity, phase dependent intervals, phase independent intervals, dictionary based and ensemble algorithms. We followed the same taxonomy for our experiment as well.

2.1.2 Experimental Design

The framework ran 100 resampling folds for each of the data sets for each classifier, totalling 8500 (85 data sets × 100 resamples) experiments for each classifier. The first run used the default train/test split provided with the data set, while the remaining resamples were stratified to preserve the class distribution in the default split. Using the same parameters as the relvent published papers, the experiment limited the number of parameter values for each classifier to a maximum of 100. They used different number of cross validation folds based on the type of algorithm. LOOCV was used for algorithms utilizing distance based measures, while for the other classifiers a new model was created for each parameter set. Resulting in a total of 850,000 model per classifier.

2.2 Deep learning for time series classification: a review

Another paper that focused on comparison of deep learning algorithms for TSC was [Faw+19a]. Deep learning algorithms had already proved competency in many application fields including image classification, speech recognition and natural language processing [He+16; SK16; KSH12; Gua+19] and was gaining more popularity in TSC problems [Zhe+14; Zhe+16; Zha+17]. The main goal was to benchmark the performance of deep learning algorithms on TSC problems; as they were less studied than the other algorithms and provide an open source framework for deep learning on TSC.

2.2.1 Inclusion Criteria

The framework included 9 discriminative model end-to-end deep learning algorithms developed to work on TSC problems. They started by grouping deep learning algorithms into two main groups; generative and discriminative approaches. Generative models involve an unsupervised step which tries to learn a quality representation of the time series, then this new representation is fed to another classifier in a learning phase. But this family of algorithms was excluded due to it's complexity and incompetence compared to the other group [LGI17; Bag+17]. As for the discriminative models, algorithms that needed preprocessing of features prior to learning were also excluded; to avoid the bias of hand crafted features. The remaining discriminative approaches were; Multi Layer Perceptron (MLP), Fully Convolutional Neural Network (FCN), Residual Network (ResNet), Encoder, Multi-scale Convolutional Neural Network (MCNN), Time Le-Net, Multi Channel Deep Convolutional Neural Network (MCDCNN), Time Convolutional Neural Network (Time-CNN) and Time Warping Invariant Echo State Network (TWIESN).

These 9 algorithms were to be compared to each other and to 7 other classifiers. A group consisting of the best 4, out of the 18, classifiers that were included in the experiment by [Bag+17] which are; Elastic Ensemble (EE), Bag Of SFA Symbols (BOSS), Shapelet Transform (ST), Collective of Transformation-based Ensemble(COTE). They have also included Hierarchical Vote Collective of Transformation-Based Ensembles (HIVE-COTE), which is an extension of COTE using hierarchical voting and two extra classifiers. In addition to Proximity Forest, an ensemble based on the same concept of Random Forests but using class examplars instead of feature split values.

Finally, the classic 1-NN classifier utilizing Dynamic Time warping (DTW) elastic distance measure with warping window set through cross-validation on training data set. Many of these classifiers we cover in Chapter 3

2.2.2 Experimental Design

The experiment covered both univariate data sets and multivariate data sets. The univariate data sets were represented by the same 85 z-normalized data sets from the UCR archive as [Bag+17]. As for the multivariate data sets, 12 data sets from the archive by Mustafa Baydogan[1] were used. Due to the existence of instances with unequal lengths in the multivariate archive, they used linear interpolation, suggested by [RK05], such that all instances inside the same data set are adjusted to the length of the longest instance. Z-normalization was not applied to any of the Baydogan archive.

In order to avoid bias from the initial weights assigned to the classifiers, for each of the data sets for each classifier, the framework used 10 runs for training and then considered the mean accuracy of all runs together. The original train/test split was used in all 10 runs, but the initial weights were randomized. The framework applied optimization for the hyperparameters of the deep learning algorithms, but not for the other classifiers. With the exception of TWIESN, the number of epochs used during optimization ranged between 100 and 5000. A model checkpoint procedure was involved. This meant that after training a model for 1000 epochs, the model which attains the least error on the validation data set is chosen for evaluation. All the models were initialized randomly using the methodology from [GB10] and were optimized using variants of stochastic gradient descent; like Adam [KB14] and AdaDelta [Zei12]. For FCN, MLP, and ResNet if the training loss had not improved for 50 consecutive epochs, then learning rate was decreased by 0.5 to a minimum of 0.0001.

2.3 The great multivariate time series classification bake off

A recent study was carried out by [Rui+20] focusing on comparing multivariate TSCAs. With previous research paying more attention to univariate classifiers and univariate problems, multivariate time series classification (MTSC) recieved less attention. Which meant that the MTSC domain is now in the same position that univariate TSC had been in some years ago and there is a need for a benchmarking framework to guide upcoming research on how to compare multivariate TSCAs to already existing ones.

MTSC is simply the classifiation of time series data collected, where multiple features are collected at each time point and a single label is assigned. MTSC is more challenging than univariate TSC because the discriminative features can be in the interaction between the multiple dimensions and not only based on the interdependency of consecutive values in one dimension. The framework inlcuded two techniques that algorithms follow to deal with multivariate data; either dedicated multivariate TSCAs which are, by design, able to handle multiple dimensions, or adapted univariate classifiers which can handle multivariate time series instances. The adapted univariate classifiers technique is based on two pillars; assuming independence between the dimensions of the data and fitting separate univariate classifiers to each dimension

[1] http://www.mustafabaydogan.com/multivariate-time-series-discretization-forclassification.html

then ensemble their results. We used this technique in our experiment for handling multivariate data sets.

2.3.1 Inclusion Criteria

Although univariate and multivariate TSC problems differ in their complexity, but the algorithms for both can be grouped, based on the technique they apply, in the same way. These are; distance based similarity, phase dependent intervals, phase independent intervals (shapelets), dictionary based and deep learning algorithms.

The framework collected 16 classifiers from these different groups and compared them on the 30 data sets from the UEA multivaraite data archive (discussed in 4.1.2). The included classifiers had to meet 2 simple criteria; code availability and runnable code. The classifiers encompassed 3 variations of the classic DTW which represented the baseline; DTW using Independent Warping (DTW_{I}) which calculates DTW distance for each dimension separately then sums all distances, DTW using Dependent Warping (DTW_{D}) which simultaneously calculates the DTW distance across all dimensions for each time point and DTW using Adaptive Warping (DTW_{A}) which adapts to each instance by calculating both DTW_{I} and DTW_{D} and then chooses between them based on a threshold score. The multivariate classifiers were; Generalized random shapelet forest (gRFS), WEASEL+MUSE, Canonical interval forest (CIF), Random Convolutional Kernel Transform (ROCKET), Multiple Representation Sequence Learner (MrSEQL), Residual network (ResNet), InceptionTime, Time series attentional prototype network (TapNet). The adapted univariate classifiers were; the ensemble HIVE-COTE and each of it's single components; Shapelet Transform (ST), Time Series Forest (TSF), Contractable Bag of Symbolic Fourier Approximation Symbols (CBOSS) and Random Interval Spectral Ensemble (RISE). Two specialized toolkits in time series machine learning were used for the experiment. One of them is python based, while the other is java based. Most of the algorithms are implemented in both, but few exist only in one. More information is available in [Rui+20].

2.3.2 Experimental Design

The framework ran 30 resamples for each data set for each classifier to calculate performance metrics, the first run using the default train/test split of the data set, while the remaining 29 runs using stratified to maintain the class distribution of the original split. In contrast to the previous experiments where only accuracy was measured, this experiment had other measures beside which are; area under the ROC, balanced accuracy, F1, negative log likelihood, Matthew's correlation coefficient, recall/sensitivity, precision and specificity. Still accuracy was the primary performance measure used in comparisons; due to it's easier interpretation. The classifiers were initialized using the default structure and hyperparameters in their original published papers. Apart from the classifiers that have internal tuning processes, no other external tunings were applied. Even for the window size of DTW, which is a was proved to have a small but significant improvement of performance for univariate data sets [RK05]. Using a side experiment, where an untuned DTW was compared to a naive implementation of DTW with window sizes between 0% and 100% on 21 of the data sets. The experiment had proved that the untuned DTW was better on 14 data sets, but with no significant difference between both. Which meant that tunning was not needed as far as the experiment is concerned.

The data sets of the UEA archive are not normalized and were presented to the classifiers without any preprocessing. This might had caused a disadvantage for WEASEL+MUSE, which doesn't have an internal normalization process, against other classifiers like, ROCKET; gRSF; TapNet; InceptionTime; ResNet; CBOSS; STC and CIF, which do. The decision of excluding data normalization was based on another side experiment that they carried out. A comparison was done sing the three DTW variants, once on normalized data and once on non-normalized data. The experiment had proved that data normalization had no significant effect on the performance of the classifiers, but overall the performance of DTW declined with normalization. The same experiment was repeated for HIVE-COTE and it's components and the same result was observed. This meant that data normalization was also not considered important for the scope of the experiment.

2.4 Evaluation Criteria

All three frameworks follow the same methodology, introduced by [Dem06], for comparing their classifiers over multiple data sets. When using multiple resamples over data sets, the score of a classifier on a given data set is calculated as the average of it's accuracy over all resamples. The methodology recommends using a Friedman ranking test to refute the null hypothesis; that there is no difference between the ranks of the classifiers. After refuting the null hypothesis and applying ranks to each classifier, a pairwise post-hoc analysis is done according to the recommendation of [BCM16]. Which applies a Wilcoxon signed rank test using a Holm's alpha correction forming cliques of classifiers. Each clique represents a group of classifiers where there is no significance between their pairwise comparisons. Results are described by a critical difference diagram, where the average rank of each classifier is noted and cliques are expressed by a thick line. We use the same code for drawing our critical difference diagrams from [Faw+19a]

Chapter 3

Classification Algorithms

This Chapter discusses the definitions and background of the topics mentioned in this thesis. We discuss the nature of time series data, the two problems of time series classification (TSC) and early time series classification (eTSC) then present the different techniques encompassed by them. We will discuss TSCAs in more details as they are the main focus of our framework, but a review of the problem of early classification and a review of it's algorithms is also included; as it motivates the context in which the framework operates.

3.1 Time Series Data

3.1.1 Definitions

A time series is a finite sequence of ordered observations, either based on time or another aspect [AML19; Bag+17]. The existence of the time component makes time series an abundant form of data that covers various domains like; medicine, finance, engineering and biology [LTB18]. A time series dataset is a collection of time series instances.

Definition 1

$$\textit{A set } T \textit{ of } n \textit{ time series instances, } T = \{T_1, T_2, \ldots, T_n\}.$$

Each of the time series instances T_i consists of a sequence of observations.

Definition 2

$$\textit{A time series } T_i\textit{, of length } L \textit{ is represented as } T_i = [t_1, t_2, \ldots t_L].$$

Time series data can come in different forms. It is important to comprehend what different forms the data can take and what implicit assumptions they convey; to be able to choose the suitable algorithms and tools to deal with it.

The first form is when the observations of instances capture a singular value, this is referred to as univariate time series.

Definition 3 *Univariate time series T_i, of length L is represented as $T_i = [t_1, t_2, \ldots t_L]$. With t_j as a real valued number.*

While the other form is when multiple measurments are captured by the observations. According to [Lön+19], it is essential to differentiate between the two ways multiple time series can be generated; panel data and multivariate time series data.

If more than one variable is being observed during a single experiment, with each variable representing a different measurement; this is called multivariate time series.

Definition 4 *Multivariate time series T_i of length L is represented as $T_i = [t_1, t_2, \dots t_L]$. With t_j having M dimensions, each is a univariate time series.*

While panel data is when the same kind of measurments is collected from independent instances; like different patients or diverse industrial processes.

For panel data, it is possible to assume that the different instances are i.i.d, but this assumption doesn't hold for observation of a single instance. The same goes for multivariate time series, individual univariate observations are assumed to be statistically dependant.

3.1.2 Nature of Time Series Data

Having discussed the dependency assumptions in time the different forms of time series data. It is this dependency that makes time series data challenging for conventional machine learning algorithms, which are used for tabular and cross-sectional data. Tabular and cross-sectional data assume observations to be independent and identically distributed (i.i.d) [Lön+19].

If we were to tabularize time series data; convert it into a tabular form by considering each observation as an individual feature. Then it would be possible to apply conventional machine learning algorithms, under the implicit modelling assumption that observations are not ordered. This means that if the order of the features was changed, still the model result will not change. This assumption can work for some problems, but it doesn't have to work for all problems.

3.2 Time Series Classification

Time series classification is a subtype of the general classification problem, which considers the the unique property of dependency between adjacent features of instances [BB17b]. The main goal of time series classification is to learn a function f, which given a training dataset $T = \{T_1, T_2, \dots, T_n\}$ of time series instances along with their corresponding class labels $Y = \{y_1, y_2, \dots y_n\}$ where $y_i \in \{1, 2, \dots C\}$, can predict class labels for unseen instances [Den+13].

Time series classification has been studied with different objectives, some papers focused on attaining the highest accuracy of classification as the main goal [Kat16; JJO11; BB17a; LTB18; SL17b; Faw+20], while other papers focused on attaining lower time complexity [RK04; Bag+17; TPW20; Pet+16; SL17a].

In this master thesis, we are more interested in assessing the results in terms of accuracy than time complexity. We define accuracy like [SL20]; as the percentage of correctly classified instances for a given dataset D, either being a training or testing dataset.

Definition 5

$$Accuracy = \frac{number\ of\ correct\ classifications}{|D|}$$

3.3 Time Series Classification Algorithms

This chapter will introduce different types of TSCAs and discuss the various techniques that each type apply. There are multiple ways to divide TSCAs in order to better understand them. In this thesis we follow the grouping defined by [Bag+17].

3.3.1 Whole Time Series Algorithms

Whole time series similarity algorithms, also called distance-based algorithms, are methods that compare pairs of time series instances. An unlabeled time series instance is given the class label of the nearest instance in a training data set [Kat16]. There are two main techniques for carrying out the comparison; either by comparing vector representations of the time series instances, or by combining a defined distance function with a classifier, KNN being the most common one [LTB18]. Whole time series algorithms are best suited for problems where the unique features can exist anywhere along the whole time series[Bag+17].

One of the simplest forms of whole time series is 1-NN with Euclidean Distance [FRM94], yet it can suprisingly attain high accuracy compared to other distance measures [XPK10]. But Nearest Neighbor with Euclidean Distance (KNNED) is an easy to beat baseline, due to it's sensitivity for distortion and inability to handle time series of unequal lengths [XPK10; Kat16; LTB18]. This lead many of the researchers to focus on defining more advanced and elastic distance measures that can compensate for misalignment between time series [AML19]. The standard and most common baseline classifier utilizing elastic distance measures is 1-NN with Dynamic Time Warping (DTW) [Bag+17]. In contrast to the idea that more powerful machine learning algorithms will be able to defeat the simple KNN and an elastic measure, DTW has proved to be a very tough opponent to other algorithms and other elastic distance measures as well [Kat16; LB15; Wan+13]. But there were also other distance metrics that have been introduced in literature, these inlcude extensions of DTW on one hand like; Weighted Dynamic Time Warping (WDTW) which penalizes large warpings based on distance [JJO11] and Derivative Dynamic Time Warping (DDTW) [KP01; GŁ13] which uses derivatives of sequences as features rather than raw data to avoid singularities. On the other hand, Edit Distance with Real Penalty (ERP) [CN04], Time Warp Edit (TWE) [Mar08], Longest Common Subsequence (LCSS) [DGM97] and Move-Split-Merge (MSM) [SAD12] are all alternatives for distance measures, yet multiple experiments have considered DTW to be relatively unbeatable [Bag+17; AML19; BB17a]. To the extend of our knowledge, the most powerful whole time series classifiers are Elastic Ensemble (EE) [LB15] and Proximity Forest (PF) [Luc+19].

Nearest Neighbor with ED

The Euclidean distance is a remarkably simple technique to calculate the distance between time series instances. Given two instances $T_1 = [t1_1,t1_2,...t1_n]$ and $T_2 = [t2_1,t2_2,...t2_n]$, the euclidean distance between them can be determined as:

$$ED(T_1, T_2) = \sqrt{\sum_{i=1}^{n}(t1_i - t2_i)^2} \tag{3.1}$$

Euclidean distnace has been preferred to other classifiers due to it's space and time efficiency, but it suffers from two main shortcomings [BRT13; JJO11; Kat16]. The

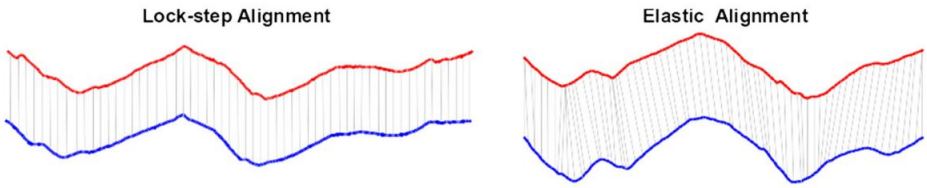

Figure 3.1: Mapping of Euclidean distance (lock-step measure) versus mapping of DTW distance (elastic measure) [AML19].

first one is that it cannot handle comparisons between time series of different lengths. While the second one is it's sensitivity to minor discrepancies between time series; it would calculate large distance values for small shiftings or misalignments. Although other metrics have been introduced to overcome the drawbacks of euclidean distance, experimental proof showed that this is only the case for small datasets, but for larger datasets the accuracy of other elastic measures converge with euclidean distance [Hil+14; Din+08; Bag+12].

Nearest Neighbor with DTW

Dynamic Time Warping was a very strong baseline for time series classification for a long time [AML19; Bag+17]. It was first introduced as a technique to recognize spoken words that can deal with misallignments between time series that Euclidean Distance couldn't handle [TPW20].

To calculate the distance between two time series instances $T_1 = [t1_1, t1_2, ...t1_n]$ and $T_2 = [t2_1, t2_2, ...t2_n]$; a distance matrix $M(T_1, T_2)$, of size n × n, is calculated for T_1 and T_2. With $M_{i,j}(t1_i, t2_j)$ representing the distance between $t1_i \in T_1$ and $t2_j \in T_2$. The goal of DTW is to find an optimal path that minimizes the cumulative distance between points of T_1 and T_2.

A candidate path $P = [p_1, p_2, ...p_p]$ is to be found by traversing M. For a path to be valid it must conform to some conditions:

- $p_1 = (t1_1, t2_1)$
- $p_p = (t1_n, t2_n)$
- for all i <m :
  - $0 \leq t1_{i+1} - t1_i \leq 1$
  - $0 \leq t2_{i+1} - t2_i \leq 1$

Finding an optimal path under DTW can be computationally expensive with complexity of $O(n^2)$ for a time series of length n [SL17a; Pet+16]. Consequently it is usual to use a contraint with the path; to prevent comparison of points outside a certain window [TPW20]; like the famous Sakoe-Chiba Band [SC78], Itakura Parallelogram [Ita75] and Ratanamahatana-Keogh Band [RK04]. Typically DTW can make use of it's warping window to handle distortion in time series, but still it is vulnerable to cases where the differene in length between instances length is larger than the warping window [Tan+19].

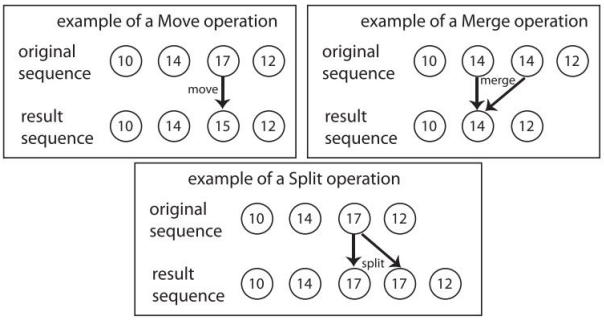

Figure 3.2: Examples of the Move, Split, Merge operations [SAD12].

Nearest Neighbor with MSM Distance

Move-Split-Merge is distance measure that was first introduced in [SAD12]. The main purpose of introducing MSM was to combine certain characteristics within one distance measure. These are; robustness to misalignments between time series instances, being an actual metric unlike other distance measures like DTW and LCSS, assure translation invariance and achieve quadratic run-time [LB15].

The way MSM works is pretty much like other edit distance methods; it determines the similarity between two instances through the usage of a set of operations to transform one of them to the other. These operations, as the name indicates, are; move, split and merge [Bag+17].

Move is the substitution of one single time point of a series with another. Split divides a single time point of a series into two consecutive time points holding the same value as the originial time point. Merge can be seen as the opposite of Split, it combines two consecutive time points holding the same value into one time point.

Each of the previously mentioned operations is associated with a cost. The cost of a move is equal to the absolute difference between the old and the new value of the time point. The costs of split and merge are equal and they are set to a constant to satisfy the symmetry property of metricity [SAD12; TPW20].

Proximity Forest

Proximity forest was developed by [Luc+19]. It was introduced as an addition to scalable time series classification, offering a more scalable and accurate classifier than EE [TPW20]. On one side, EE was an accurate classifier being one the state of the art algorithms and the best among distnace based algorithms, as it combines 11 NN-algorithms each using a different elastic measure. But on the other hand, EE's training process was very slow as it scales quadratically with the training size of the data set [LB15; Bag+17]. This goes back to the leave-one-out-cross-validation (LOOCV) used to optimize the parameters for each used metric [Shi+20].

Proximity Forest wanted to achieve two main goals. The first was to offer an adaptable algorithm that can scale with huge data sets consisting of millions of time series instances. Beating EE, by orders of maginute, and other state of the art algorithms in terms of training and testing run time complexity. While the other goal was

to develop a competitive algorithm on the UCR data sets archive without the need to sacrifice accuracy for scalability as is the case with BOSS-VS [Luc+19].

Capitalizing on the previous research that has been put in developing specialized time series distance measures and inspired by the existing EE [Faw+20; Faw+19b]. Proximity forests combine the the eleven elastic distances from EE along with a tree-based algorithms to form an ensemble of decision trees. The reason behind using tree-based algorithms lies in the divide-and-conquer strategy that they adopt, which makes them scalable for large data sets. Also a stochastic process is used for the selection of distance measures and their hyper-parameters, which usually hinders the performance of other algorithms, like KNN, that need to learn the hyper-parameters of the utilized distance measure for each data set before using it [Luc+19]. Proximity forests can scale sublinearly with training data set size, but quadratically with the length of the time series [Shi+20].

Proximity forests are based on a similar concept as Random Forests [Bre01], another tree-based ensemble, which learns only on a subset of the available features for building tree nodes. This process insinuates in a factor of variability between the trees that form the ensemble but each with a low bias. The collective classification accuracy of the ensemble then tends to provide better results than any if it's single classifiers [Luc+19].

The building unit of a proximity forest is called the proximity tree. A proximity tree and a decision tree are similar on all aspects, but they differ in the tests they apply in internal nodes. A conventional decision tree builds it's nodes using attributes. When an instance is being tested, it is compared to the value of the attribute and then follows the branch to which it conforms.

Unlike conventional decision trees, that use feature values for their nodes, proximity trees build their nodes using randomly selected examplars. When an instance to be tested, an elastic distance measure is calculated and then it follows the branch of the nearest examplar.

An internal node in the tree holds two attributes; *measure* and *branches*. As noted in [Luc+19], a measure is function $object \times object \rightarrow \mathbb{R}$. Proximity Forest uses the same 11 distance measures used by EE; Euclidean distance (ED) Dynamic time warping using the full window (DTW); Dynamic time warping with a restricted warping window (DTW-R); Weighted dynamic time warping (WDTW); Derivative dynamic time warping using the full window (DDTW); Derivative dynamic time warping with a restricted warping window (DDTW-R); Weighted derivative dynamic time warping (WDDTW); Longest common subsequence (LCSS); Edit distance with real penalty (ERP); Time warp edit distance (TWE); and, Move-Split-Merge (MSM). Proximity Forest saves a lot of the computational cost by replacing parameter searches with random sampling [Faw+20; Faw+19a]. While branches is a vector of the possible branches to follow, each branch holding two attributes; *examplar* and *subtree*. *examplar* is a time series instance to which a query instance is compared, and *subtree* refers to the tree an instance should follow in case it is closest to a specific examplar.

If all time series in a specific node share the same class, then a leaf node is created and the value of the class label is assigned to the *class* attribute of this node. During classification, if a query instance is to reach this node, it is directly labeled with the value of it's *class* attribute.

When a query time series is to be classified, it starts at the root node of a proximity tree. The distance between the query and each of the randomly selected examplars is calculated, using the randomly selected distance measure at the node. Then the query travels down the branch of the nearest examplar. This processes is repeated, passing

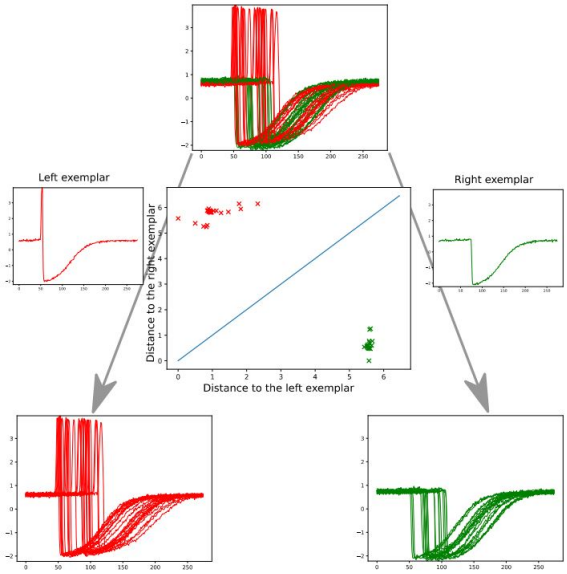

Figure 3.3: Visual depiction of the root node for the 'Trace' dataset (simplified to 2 classes). The top chart represents the data at the root node (one colour per class) while the data at the bottom left and right represent the data once split by the tree. The two time series in the middle left and right are the exemplars on which the tree is splitting. The scatter plot at the center represents the distance of each time series at the root node with respect to the left and right exemplars (resp. x- and y-axes) (Color figure online) [Luc+19].

through the internal nodes of the tree till the query reaches a leaf node, where it is assigned the class label of that node. This whole process is then repeated for all the trees constituting the forest. The final classification of the forest is made by majority voting between it's trees.

3.3.2 Phase Dependent intervals Algorithms

Phase dependent algorithms is a group of algorithms that extract temporal features from intervals of time series. These temporal features help with the interpretibility of the model; as they give insights about the temporal characteristics of the data [BR16], unlike whole time series algorithms that base their decisions solely on the similarities between instances. Another advantage of phase dependent algorithms is that they can also handle distortions and misalignments of time series data [Den+13].

According to [Bag+17], phase dependent algorithms are best used for problems where discrimanatory information from intervals exist, this would be the case with long timer series instances and which might include areas of noise that can easily deceive classifiers. Like the case with the SmallKitchenAppliances dataset, in which the usage of three classes; a kettle, a microwave and toaster is recorded every 2 minutes for one day. Not only the pattern of usage is discrimanatory in such case, but also the

time of usage.

Typically using interval features requires a two phase process; first by extracting the temporal features and then training a classifier using the extracted features [Den+13]. There are n(n-1)/2 possible intervals,for a time series of length n[Bag+17]. There is also a wide variety of features, also called literals, to extract for each interval. These cover simple statistical measures as well as local and global temporal features [SK16; RA04; Den+13]. This introduces one of the main challenges for phase dependent algorithms, that is which intervals to consider for the feature extraction step. Which [RA04] proposed a solution for by only considering intervals with lengths equal to powers of two [Bag+17].

Time Series Forest

Time Series Forest (TSF) is an algorithm that was introduced in 2013 by [Den+13]. They motivated their model with two main criteria; contributing to interpretable time series classification through the use of simple statistical temporal features and reaching this goal by creating an efficient and effective classifier.

TSF considers three types of interval features; mean, standard deviation and slope. If we were to consider an interval with starting point t_1 and with ending point t_2. Let v_i be the value at a specific point t_i. Then the three features can be denoted as:

$$mean(t_1, t_2) = \frac{\sum_{i=t_1}^{t_2} v_\mathrm{i}}{t_2 - t_1 + 1} \tag{3.2}$$

$$std(t_1, t_2) = \begin{cases} \frac{\sum_{i=t_1}^{t_2} (v_\mathrm{i} - mean(t_1, t_2))^2}{t_2 - t_1} & \text{if } \mathrm{t}_1 < \mathrm{t}_2 \\ 0 & \text{if } \mathrm{t}_1 = \mathrm{t}_2 \end{cases} \tag{3.3}$$

$$slope(t_1, t_2) = \begin{cases} m & \text{if } \mathrm{t}_1 < \mathrm{t}_2 \\ 0 & \text{if } \mathrm{t}_1 = \mathrm{t}_2 \end{cases} \tag{3.4}$$

Where m denotes the slope of the least squares regression line for the training dataset.

For building the trees, TSF introduced a new splitting criteria at the tree nodes, which they called the Entrace. A combination of Entropy and distance; to break the ties between features of equal entropy gain by preferring splits that have the furthest distance to the nearest instance. They also use a specific number of evaluation points rather than checking all split points for the highest information gain. In their experiment [Bag+17] found these two criteria to have negative effect on accuracy.

As mentioned earlier the feature space for creating interval features is huge, TSF adopts the same random sampling technique that Random Forests use reducing the feature space from $O(M^2)$ to only $O(M)$, by considering only $O(\sqrt{M})$ random interval sizes and $O(\sqrt{M})$ random starting points at each tree node [Den+13]. The final classification of a testing instance is done using majority voting of all time series trees created.

3.3.3 Phase Independent intervals Algorithms

Phase independent shapelets, or just shapelets as less formally known, are subseries which are ultimately distinctive of classes regardless of their place on the time series [SL17a; Bag+17]. They were first introduced in [YK09] as an alternative for KNN approaches; to overcome their shortcomings.

Shapelets reduce the space and time complexity needed by KNN, because they are formed from subsequences which are shorter than the original time series. Needing only one shapelet at classification time, they form a compressed format of the classification problem [BB17a; YK09; MKY11]. While KNN classify based on comparison to other instances, shapelets provide insight about the unique features of classes and thus more interpretable results of how the classification was carried out. Finally, shapelets are best suited for problems where a certain pattern can differentiate instances which is harder to detect when comparing whole series [Bag+17; BB17b].

The original shapelet algorithm enumerated all possible shapelets and embeded the best ones, based on information gain assessment, in a decision tree. Together with a calculated distance threshold, the shapelets and the threshold are used together as splitting criteria [LTB18; Sch15]. There have been many attempts to speed up the process of shapelets discovery, by determining good shapelets in a faster manner. Two of them are; Fast Shapelets (FS) [RK13] and Learned Shapelets (LS) [Gra+14]. FS applied discretization through Symbolic Aggregate Approximation (SAX) to reduce the length of time series, while LS tried to learn the shapelets [Shi+20]. Later on the idea of tranforming time series data to an alternative space was adopted in [Hil+14], the transformed data consists of distances to the best k shapelets, then classification is done using an ensemble of eight classifiers.

Learned Shapelets

Learned Shapelets (LS) was proposed in [Gra+14] as a new prospective for approaching time series shapelets. Instead of searching for shapelets through enumeration of all candidates, LS learns K near-to-optimal shapelets that can linearly separate instances through a stochastic gradient objective [LTB18; Bos18]. The found shapelets need not to be a subsequence of one of the training examples [Bag+17; SL17a].

LS follows a two steps technique. In the begining LS looks for a set of shapelets from the training data set using two parameters; L controls the length of shapelets searched, while R controls the scaling of subsequences. Then these shapelets are clustered using a K-Means algorithm and instances are represented in a new K-dimensional format where the values of the features represent the minimum distance between the instance and one of the shapelets.

For the second step, using the new features representation, LS can start learning class probabilities for instances by considering a logistic regression model for each of the classes and optimizing a regularized logistic loss function. The regularized loss function updates the shapelets and the weights of features. This process keeps iteratively going untill either the model converges or the maximum number of iterations is reached.[Bos18]. In summary, the main objective of the aglorithm is to learn collectively the optimal shapelets and the weights linear hyper-plane that minimizes the objective function [Bag+17; Gra+14].

Shapelet Transform

The first Shapelet Transform (ST) was introduced in [Hil+14]. While the original algorithm embeded shapelets discovery in decision trees and assesed candidates through enumeration and the use Information Gain (IG) at each node, ST proposed a different way that saved repeating the brute force multiple times [Bos18]. ST seggregated the shapelets discovery process from the classifier. This seggregation opened the door for choosing classifiers freely and considering more accurate classifiers than decision trees

[Bag+17; LB15]. Also [Hil+14] experimented with other shapelet assessment metrics like Kruskal-Wallis, F-stat and Mood's median to find out that F-stat attained higher accuracies than the other three and than IG [Bos18].

ST follows a three step procedure. In the beginning, a data transformation phase is carried out by utilizing a single-scan algorithm and extracting K best shapelets from the training data set where K represents a cutoff threshold for the maximum number of shapelets to extract without affecting the quality of the shapelets extracted. Then a reduction process is done by clustering the shapelets together untill they reach a user defined number. Finally, The clustered shapelets are then used to transform the original datasest, by representing instances in terms of their disatances to each one of the extracted shapelets. They experimented with different classifiers other than decision trees, these are; C4.5 tree, 1-NN, naive Bayes, Bayesian network, Random Forest, Rotation Forest, and support vector machine, for which decision trees proved to be the worst among all, while support vector machine proved to be the best [Hil+14].

ST was then extended again by [BB17b], the intuition behind it was that the previously used assessment technique couldn't hanlde multi-class problems [BB17b]. Instead of assessing shapelets that discriminate between all classes, they accomodated a one-vs-all technique so that shapelets are assessed on their ability to separate one class to all other classes. They also introduced a balancing technique to represent each of the classes with the same number of shapelets [Bag+17]. For the classification, a combination of tree based, kernel based and probabilistic classifiers were used in an ensemble on the transformed data set [Shi+20; LTB18]. Each of the classifiers was given a weight based on it's training accuracy and the final classification used weighted voting [BB17b]. Although ST has proved to be a competent accurate classifier, it suffers from high training-time complexity [Shi+20].

3.3.4 Dictionary Based Algorithms

Here should be intro for Dictionary

BOSS

Bag of SFA Symbols (BOSS) is a dictionary based algorithm that was introduced by [Sch15] in 2015. Like the previous dictionary based classifiers, Bag Of Patterns (BOP) and Symbolic Aggregate approXimation in Vector Space Model (SAXVSM), BOSS also used a windowing technique to extract patterns from the data and transform them into words, but it had other significant differences to them [Bag+17]. BOSS was concerned with the issue of dealing with noisy data which, at that time, received little attention; due to the common practice of either using raw data directly or handling noise in a preprocessing stage. The goal was to introduce an algorithm faster than it's rivals of the same group, robust for existence of noise in the data and competitive with existing TSCA.

The BOSS model is divided into several steps. In the beginning, it passes a window of size w over the time series to extract substructures. Each of the extracted windows is then normalized to obtain amplitude invariance, while obtaining offset invariance by subtracting the mean value for each window is data set specific and can be decided upon based on a parameter. Then the substructures are quantized using SFA transformations which transforms the data into unordered words; this helps further reduce the noise in the data by making it phase shift invariant and makes it possible to apply string matching algorithms on the data. Since some data sets might happen

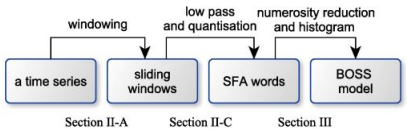

Figure 3.4: The BOSS workflow [Sch15]

to have long constant signals, this will lead to SFA transformations of their windows to produce the same words multiple times and cause higher weights to be assigned for them. BOSS applies a numerosity reduction technique adopted from [Lin+07; LKL12] that ignores multiple sequential occurences of the same word. Finally time series instances can be compared for differences, using the noise reduced substructures using a customized distance measure inspired by Euclidean distance. We will, briefly, discuss the main components that are used for each of the previously mentioned steps.

To extract the substructures from a given time series instance $T = [t_1, t_2, ...t_{\mathrm{n}}]$, a windowing function is used to split it into fixed sized windows $S_{\mathrm{i;w}} = (t_{\mathrm{i}}, , ...t_{\mathrm{i+w\text{-}1}})$, each of them is of a size w. The total number of windows that can be created for a time series of length n is $n - w + 1$, and each consecutive windows overlap on $w - 1$ points. In order to achieve offset and amplitude invariance, each window is z-normalized by subtracting the mean from it's original values and then dividing the difference by the standard deviation.

$$windows(T, w) = \{S_{1;\mathrm{w}}, S_{2;\mathrm{w}}, ..., S_{\mathrm{n\text{-}w+1;w}}\} \tag{3.5}$$

After running the windowing function, the real values of the time points inside the windows are transformed into words using Symbolic Fourier Approximation (SFA) [SH12]. SFA is an alternative way of representing time series data, that instead of using real values, uses a sequence of symbols which are referred to as SFA words based on a predefined alphabet of specific length. SFA accomplishes two things; low pass filtering through removal of noisy components from the data and string representation which allows for the use of string matching algorithms.

For SFA to achieve it's goals, two main operations have to be carried out; approximation and quantization. Approximation is the process of representing a specific signal of length n using another signal of lower length l. This is achieved using Discrete Fourier Transformation (DFT); a transformation technique which is applied to a singal represented by a sequence of equally spaces values into a sequence of coefficients of complex sinusoid waves ordered by their frequencies [LLY17]. The higher coefficients in a DFT refer to data with with rapid changes, which can be considered as noise in the signal and thus ignored. So considering only the first $\mathrm{l} \ll \mathrm{n}$ coefficients acts as a low pass filter for noise producing a smoother signal.

Quantization also helps reduce noise by splitting the frequency domain into equi-depth bins, then maps each of the Fourier real and imaginary coefficients into a bin. BOSS utilizes Multiple Coefficient Binning (MCB), an adaptive technique that minimizes the loss of information which is a side effect of quantization. After the approximation step, a matrix is is built from the Fourier transformations of the training data set using only the first $\frac{l}{2}$ coefficients, including both the real and imaginary values for each coefficient. Then using an alphabet Σ of size c, MCB creates for each

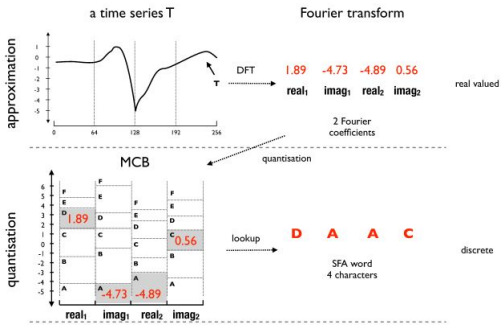

Figure 3.5: A time series is (a) approximated (low pass filtered) using DFT and (b) quantised using MCB resulting in the SFA word DAAC [Sch15]

columm of the matrix c+1 break points using equi-depth binning. During classification, to acquire the SFA word for a Fourier transformed time series, a lookup is done on the precomputed MCB bins and a word is assigned if the value falls within it's bin's boundaries.

After the transformation of instances to SFA words, BOSS uses a customized distnace measure referred to as BOSS distance; to measure the similarity between the transformed instances. BOSS distance is a variation of Euclidean distance, which compares instances based on the histograms formed from their transformed forms. The appearence of the same SFA words in both instances, is considered to be a notion of similarity. While the absence of SFA words might be caused by one of two reasons; the absence of some substructures from either instances, or due to the presence of noise which disfigures the time series. Instances under BOSS distance are compared based on shared SFA words only, thus excluding words of frequencies equal to 0. BOSS distance as noted by [Sch15] is:

Given two time series instances T_1 and T_2 and their corresponding BOSS histograms $B_1 : \Sigma^l \to \mathbb{N}$ and $B_2 : \Sigma^l \to \mathbb{N}$, where l is the word length and Σ is an alphabet of size c. Their BOSS distance is:

$$D(T_1, T_2) = dist(B_1, B_2) \tag{3.6}$$

where

$$dist(B_1, B_2) = \sum_{a \epsilon B_1; B_1(a) > 0} [B_1(a) - B_2(a)]^2 \tag{3.7}$$

Each single BOSS classifier utilizes 1-NN approach along with it's BOSS model. The reason behind using 1-NN is that it is a simple technique that doesn't add to the parameters of the model, but rather proved to be a robust one. When classifying a query isntance, BOSS searches for the nearest neighbor in a sample of candidates by chossing the closest instance based on the BOSS distance.

Finally, BOSS Ensemble is introduced as an ensemble of multiple BOSS classifiers. While a fixed window length is used over time series instances for BOSS classifier, BOSS Ensemble considers representing each time series instance by ensembling multiple BOSS classifiers, each of a different window length. When the training data is fitted, BOSS Ensemble acquires a group of scores for each of the different window

lengths. To classify a query instance, BOSS Ensemble acquires the best accuracy score from the score sets returned during training. Then considers all window lengths that obtained accuracies within a factor of the best accuracy score. Each of the considered windows predicts a class label for the query instance using 1-NN. Majority voting is applied and the most dominant class label is assigned.

WEASEL

After the success that BOSS achieved, it became one of the baseline classifiers of TSC. Other reasearch has either included it as a building component of their ensembles [LTB18; Bag+15], or have used it, as a baseline for comparison of their algorithm's performance [Faw+20; Shi+20; Luc+19] Later on, the same team that developed BOSS introduced a newer algorithm, Word ExtrAction for time SEries cLassification (WEASEL) [SL17a]. A dictionary based classifier which is very similar to BOSS and can be thought of as an extension to it, with more focus on scalability [MVB19]. WEASEL motivated their work with the absence of scalable classifiers, at the time, that can deal with big data sets; as the existing were either not scalable enough or not accurate enough. Despite being a non-ensemble classifier, WEASEL can compete with powerful ensembles in terms of accuracy without the need for long prediction times, but this comes with the cost of resource expensive training [MVB19].

Like the other dictionary based approaches, WEASEL uses a sliding window over the data instances to extract substructures. These substructures are then discretized per window to extract features and finally a machine learning classifier is learned over the transformed features. But WEASEL differentiates itself from the other algorithms in the way it constructs and filters the features. We will discuss the new approaches that WEASEL uses for extracting discriminative features, building a model to deal with multiple occurences of words and variable windows' lengths and selecting features to reduce runtime and exclude irrelevant features.

In the beginning, WEASEL uses multiple sliding windows of different lengths; to extract substructures. While keeping track of their order; in order to use them later on as bi-gram features. This replaces the process of building multiple models then choosing the best one, with building only one model which can learn from the concatenated high-dimensional feature vector. Then each of the substructures is normalized and a DFT is applied. Instead of filtering out the higher Fourier coefficients, WEASEL applies an ANalysis Of VAriance (ANOVA) F-test to keep real and imaginary Fourier values that best separate instances from different classes. Then the kept coefficients are discretized into words, based on alphabet of size c, using binning boundaries. Instead of using equi-depth binning, WEASEL applies an information gain based binning technique; which further more helps separating instances of classes. In the end, a histogram is built using all the windows lengths and the extracted features, unigrams and bi-grams. Irrelevant features are then filtered out using a Chi-squared test. The final highly discriminative feature vector produced is then used to learn a logistic regression classifier.

WEASEL, like BOSS, transforms extracted windows from a time series into words using an alphabet of size c. They identify two main drawbacks for SFA, and introduce a new supervised symbolic representation technique which is based on SFA but overcomes the drawbacks. The first drawback of SFA is that it acts like a low pass filter and excludes the high frequency components from the Fourier transformation, which might discard important features in some scenarios. The other drawback, is that SFA defines the boundaries of bins during quantization independent of the class labels;

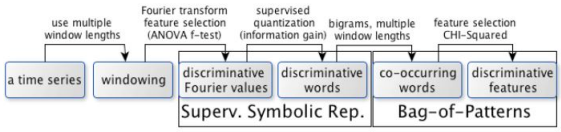

Figure 3.6: WEASEL Pipeline: Feature extraction using a novel supervised symbolic representation, and a novel bag-of-patterns model [SL17a]

which might cause SFA words of equal frequencies to appear unnecessarily in multiple classes. In order to over come the drawbacks of SFA, WEASEL follows two steps; discriminative approximation using ANOVA F-test and discriminative quantization using information gain.

As mentioned earlier, approximation involves representing a time series of length n with a shorter, yet informative, representation of length l by applying a Fourier transformation. WEASEL aims at keeping the best class separating l, real and imaginary, Fourier coefficients by applying a one-way ANOVA F-test. The test verifies the hypothesis that the means of two or more distributions/groups differ from each other [Low14]. This can be tested by comparing two variances; the variance between groups and the variance within the groups. Using the notations in [SL17a], $meansquarebetween$ (MS_{B}) and $meansquarewithin$ (MS_{W}) respectively. The F-values is then calculated as :

$$F = \frac{MS_{\mathrm{B}}}{MS_{\mathrm{W}}} \tag{3.8}$$

Then l coefficients with the highest F-values are kept; as these represent features which have big differences between the classes (high MS_{B}) and small differences within the same class (low MS_{W}).

After that, discretization is carried out to set the split thresholds for each of the extracted Fourier value. Discretization involves a binning process, where each Fourier value is divided into a number of bins. Each bin is represented by an upper and a lower boundary, and assigned a letter from an alphabet c. Previous quantization techniques used equi-depth or equi-width binning, but these techniques are solely based on values and ignore the distribution of classes. WEASEL applies a binning technique based on IG; assuring that for each partition the majority of the values would belong to the same class.

The result of the feature extraction phase is a high dimensional feature vector with a dimensionality of $\mathcal{O}(min(Nn^2, c^l)$, where c is an alphabet, l is the length of a word, n is the total number of instances and n is the length of the time series. Since WEASEL uses bi-grams and $\mathcal{O}(n)$ window lengths, the dimensionality of the feature space increases to $\mathcal{O}(min(Nn^2, c^{2l} \cdot n)$. This enormous feature space is then reduced using a Chi-squared (χ^2) test. Chi-squared tests is a statistical test used to determine if there is a significant difference between the recognised frequencies and the expected frequencies of of a features within a group. This implies that features which have high (χ^2) values are statistically frequent in certain classes. By comparing the (χ^2) values of features to a threshold, all features with scores lower than the threshold can be excluded from the faeture space. This usually reduced the feature space by 30% - 70% and helped train the logistic regresion classifier in a timely manner.

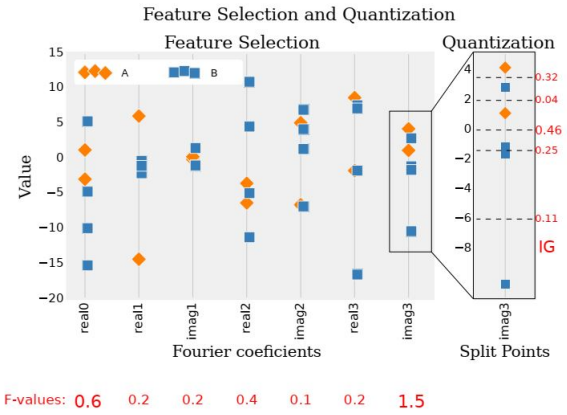

Figure 3.7: On the left: Distribution of Fourier coefficients for a sample dataset. The high F-values on imag3 and real0 (red text at bottom) should be selected to best separate the samples from class labels 'A' and 'B'. On the right: Zoom in on imag3. Information gain splitting is applied to find the best bins for the subsequent quantization. High information gain (IG) indicates pure (good) split points [SL17a]

3.3.5 Deep Learning Algorithms

Deep learning is a long established group of machine learning algorithms which has proved competence in many areas [LBH15] and which have encouraged the introduction of deep learning algorithms for time series classification [WYO17]. Deep learning is appealing for investigating time series data; due to the role that the time dimension play as a structure for the data and because deep learning algorithms can scale linearly with training size [Shi+20].

The general framework of deep learning neural networks is described by [Faw+19a] as an architecture of L layers, each one of them is considered an input domain representation. Each of the layers consists of small computing units referred to as neurons. Neurons compute elements for the output of each layer. A layer l_{i} applies an activation function to it's input then passes the output to the next layer $l_{\mathrm{i}+1}$. The behavior of activation functions in each layer is controlled by a set of parameters θ_{i} and are referred to by the term weights. The weights are assigned to the links between the inputs and outputs of the network's layers. A neural network carries out a series of computations to predict the class label for a given input x, these calculation are noted as:

$$f_{\mathrm{L}}(\theta_{\mathrm{L}}, x) = f_{\mathrm{L\text{-}1}}(\theta_{\mathrm{L\text{-}1}}, f_{\mathrm{L\text{-}2}}(\theta_{\mathrm{L\text{-}2}}, \ldots, f_1(\theta_1, x))) \tag{3.9}$$

Where f_{i} represents the activation function applied at layer l_{i}.

In the training phase of a neural network, the weights are randomly initialized. Then a forward pass of the input x is done through the model and an output vector is computed, in which every component of the vector represent a class probability. Using a cost function, the model's predicion loss is calculated and weights of the network are updated, in a backward pass process, using a gradient descent. By continuous interation of forward passes and weight updates by backward passes, the neural network

learns the best weights to minimize the cost function. For the prediction of unseen instances, or inference phase, the input data is passed through the network in a forward pass. Then using the class probabilities of the output vector, the class with the highest probability is assigned.

Many of the deep learning research on time series focused on the use of variants of Convolutional Neural Networks (CNN). [Faw+19b] CNN model is based on the idea of learning convolutional filters that can accurately represent the data. Using these filters, CNNs can learn the hierarchical structure of the data while incorporating translation invariance[LMT16]. Multi-Scale Neural Networks (MCNN) [CCC16] and LeNet [LMT16] were among the first experiments of using CNN in time series classification. MCNN was composed of three stages; transformation stage, local convolution stage and full convolution stage. The transformation stage aimed at applying a low pass filter to exclude noise and to capture different temporal patterns. Local convolution is a stage where different scale features are extracted and local features of the time series are learned. In the final stage, the full convolution stage, all the features are concatenated and the final prediction is made. On the other hand, LeNet consists of 2 convolutional layers, after each one a max pooling is carried out to do sub-sampling. For the first layer, 5 filters are applied and the max pooling size is 2. While for the second there are 20 filters and max pooling size is 4. In [WYO17] MultiLayer Perceptrons (MLP), Fully Convolutional Networks (FCN) and the Residual Networks (ResNet) were tested on 44 datasets from the UCR time series archive, where FCN was not significantly different from state of the art and MCNN was not significantly different from COTE and BOSS [SL17a].

A recent more comprehensive review of deep learning algorithms was done by [Faw+19a]. In which an experiment between nine deep learning algorithms was done. These included the classical MCNN and LeNet, in addition to the three algorithms from the previously mentioned experiment; MLP, FCV and ResNet. The other algorithms included were Encoder [SPK18], Multi Channel Deep Convolutional Neural Network (MCDCNN) [Zhe+14; Zhe+16], Time Convolutional Neural Network [Zha+17] and Time Warping Invariant Echo State Network (TWIESN) [TH16]. For more details about the algorithms and their structures, please refer to [Faw+19a].

Finally, the newest deep learning algorithm introduced for time series classification is InceptionTime [Faw+20], which is the newest state of the art and which is an ensemble of of deep convolutional neural networks. It can attain high accuracy scores while maintaining scalability. We will discuss in more details the InceptionTime algorithm in the next section.

InceptionTime

InceptionTime [Faw+20] is a recent deep learning algorithm for TSC problems, which is able to achieve high accuracy score [Rui+20]. Motivated by the increasing interest in deep learning algorithms in the TSC domain along with the need for scalable algorithms that can deal with large data sets, either in number of instances or in length of the time series, and scale to them. InceptionTime is inspired by AlexNet [KSH12], an algorithm that was considered a breakthrough for deep learning algorithms [Alo+18], and wanted to achieve the same but for the domain of TSC.

InceptionTime is based on the idea that the combination of deep CNN with residual connections, like in ResNet, can attain higher classification performance [Faw+19a] Since CNN has proved competency with image classification, there seemed a potential opportunity to be able to use deeper CNN for time series; since time series data is

mainly structured on only one dimension which is time, while images have two spacial dimensions. This opened the door for using more complex models for TSC problems which would be computationally challenging to use for images. The building blocks of an InceptionTime network are called Inception modules, these are were introduced by [Sze+15] and evolved later on to Inceptionv4 [Sze+17]. The InceptionTime classifier is an ensemble of 5 InceptionTime networks each initialized with random weights and are assigned equal weights for the final prediction. We will discuss in more details the structure of an InceptionTime network using the notation mentioned in [Faw+20].

InceptionTime's structure is similar to that of a ResNet, but instead of using three residual blocks, InceptionTime uses only two. Each of the residual blocks in composed of three Inception modules instead of the traditional fully convolutional network. Like residual networks, a linear skip connection exists between the two residual blocks of the network, passing the input from one block to be concatenated with the input of the other; this helps passing information from earlier layers of the network to deeper layers and thus mitigating the vanishing gradient issue [He+16]. After the residual blocks, a Global Average Pooling (GAP) layer exists where the multivariate time series output is averaged over the time dimension. The final component of the network is a conventional fully-connected softmax layer with a count of neurons similar to the count of output classes.

Inside the inception module, there are two main components; the bottleneck layer and the variable length filters. Assuming that the input in a multivariate time series data of M dimensions. The job of the bottleneck layer is to transform the data from having M dimensions, into a multivariate data set having m dimensions, where $m \ll M$. This is done by passing a group of sliding filters m with length 1 and a stride of size 1. Which will substantially reduce the dimensionality of the time series data and consequently will also decrease the model's comlexity making it more robust for overfitting problems on data sets of small sizes. The other benfit of including the bottleneck layer, is that it allows utilizing longer filters on the data than the original ResNet using approximately the same number of parameters; due to the lower number of dimensions that the filters will have to deal with.

The output of the bottleneck layer is then passed for a set of variable length filters, the second component of the network, of length l where $l \in 10, 20, 40$. In addition to the filters, a parallel MaxPooling operation is carried out followed by a bottleneck layer; to make the model robust to small data noises. The final multivariate output is then formed by concatenating the output from each filter based convolution along with the output of the MaxPooling operation. This whole process is executed for each Inception module in the network.

In the end, an InceptionTime network is able to learn the underlying hierarchical structures of a time series by stacking multiple Inception modules and learning from the different filter sizes, which had been learned during training, inside them.

The final InceptionTime classifier is an ensemble of 5 InceptionTime networks. InceptionTime networks accuracy scores showed high variances, a problem which have been discussed by [SW17] and was found in ResNet networks as well [Faw+19b]. This happens due to the random initialization of the networks and due to the stochastic approach used for optimization. InceptionTime classifier follows an ensembling technique for neural networks to handle TSC [Faw+19b] in order to leverage the high variability; adhering to the idea that combining multiple networks would yield better results than one classifier. During the classification of an instance, InceptionTime combines the logistic output of the 5 networks and assigns an equal weight to each of them. This

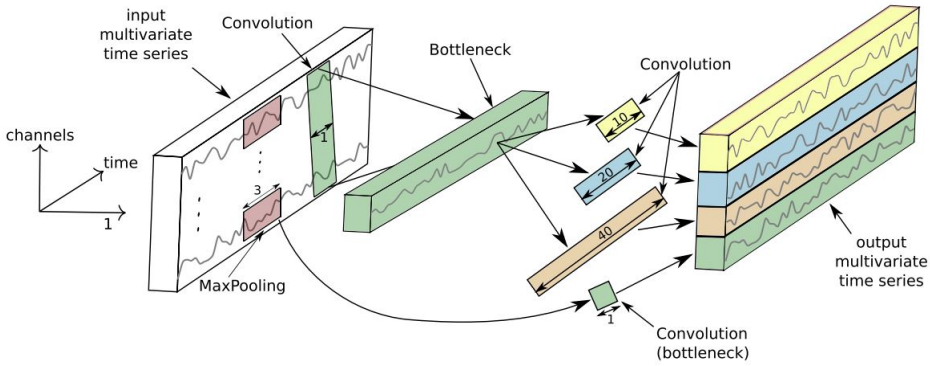

Figure 3.8: Inside Inception module for time series classification. For simplicity a bottleneck layer of size m = 1 is used [Faw+20]

can be denoted by the equation

$$\hat{y}_{\mathrm{i,c}} = \frac{1}{n}\sum_{j=1}^{n} \sigma_{\mathrm{c}}(x_{\mathrm{i}}, \theta_{\mathrm{i}}) | \forall_{\mathrm{c}} \epsilon [1, C] \tag{3.10}$$

Where $\hat{y}_{\mathrm{i,c}}$ is the class probability for instance x_{i} as belonging to the class c, which is the averaged logistic output σ_{c} over the randomly initialized models n.

3.4 Early Time Series Classification

On another side, Early Time Series Classification (eTSC) is also a classification problem which considers the temporal nature of data, but with a focus on slightly different objectives. and had been used in multiple domains.

eTSC's main objective is to learn a model which can classify unseen instances as early as possible, while maintaining a competitive accuracy compared to a model that uses full length data or to a user defined threshold[XPP09]. Which is a very challenging objective; due to the, naturally, contradicting nature of earliness and accuracy. In general, the more data is made available for the model to learn the better accuracy it can attain [Mor+19; TM16; XPP12; Mor+17b]. This is why eTSC is considered as a problem of optimizing multiple objectives.

We have already defined accuracy for TSC in section 3.2. For earliness, we follow the definition mentioned by [SL20]; as the mean number of data points s after which a label is assigned.

Definition 6

$$Earliness = \frac{\sum_{Ti \in D} \frac{s}{len(T_i)}}{|D|}$$

As well as the objective measure, Harmonic mean (HM), mentioned by [GO12; SL20], which combines both accuracy and earliness in one equation. For the scope of our framework, we consider HM as a weighted average between accuracy and earliness.

Definition 7 $F_\beta = (1+\beta^2)\frac{accuracy(1 - earliness)}{\beta^2(1 - accuracy) + earliness}$

The value of β can be used to give higher importance to one of the aspects over the other, but we use equal weights for both.

eTSC is needed in situations in which waiting for more data to arrive can be costly or when making late decisions can cause unfavorable results [Mor+17a; Par+13; Lin+15]. This is why eTSC has been applied in various domains like early medical diagnosis [GM01; GO12], avoiding issues in network traffic flow [Ber+06], human activity recognition [YD19; Gup+20] and early prediction of stock crisis [GRO14].

3.4.1 Early Time Series Classifiation Algorithms

ECTS

The idea of eTSC evolved over time. In the beginning, similar concepts to eTSC existed where the main goal was to classify prefixes of data, but these techniques lacked the reasoning of how to provide reliable classification results[Lv+19; SK16]. One of the earliest papers that formally defined eTSC as a tradeoff between earliness and accuracy was the paper by [XPP09], proposing a technique which is named after the problem itself; Early Time Series Classification (ECTS). The main idea was to examine the stability of of Nearest Neighbor approaches, to keep correctly classifying data instances, in the space of full length series and the subspaces created from subsequences of the training data set [Lv+19; Mor+17a]. ECTS utilizes the concept of reverse nearest neighbors (RNN) to learn for each training instance the Minimum Prediction Length (MPL). MPL represents the earliest point in time for which a training instance can be used to classify a query instance without loss of accuracy. The definition mentioned by [XPP09] defines MPL as:

Definition 8 *In a training data set T with full-length L, for a time series $t \epsilon T$, the minimum prediction length (MPL for short) of t, $MPL(t) = k$ if for any $l (k \leq l \leq L)$, (1) $RNN^l(t) = RNN^L(t) \neq \varnothing$ and (2) $RNN^{k-1}(t) \neq RNN^L(t)$. Specifically, if $RNN^L(t) = \varnothing$, then $MPL(t) = L$.*

During classification MPL is calculated for each instance (in the same study a relaxed variant is introduced using clusters of instances). As for the classification, as the data points of the testing instance start to arrive, the classification method tries to find the nearest instance which has reached it's MPL. If there was no suh instance found, then more data points are needed till a trusted classification can be done. ECTS was criticized [He+15; GRO14; Xin+11] because it is based on nearest neighbor approaches; which only provide classification results based on distance without deriving patterns from the data. This means that data users cannot attain insights about the problem using the results.

Shapelet Based Algorithms

A group of methods [Xin+11; GO12; GRO14; He+15] focused on the interpretability of the results obtained by eTSCAs, motivated by domains where not only correct classification is important, but also getting insights and understanding what contributed to the final result. One of these domains would be the medical field, where undesrtanding the reason for a patient's disease is very important than just linking the patient to previously seen patients [Xin+11]. Shapelets provided a good solution for these methods. Since shapelets, as previously mentioned, are defined as unique subsequences which can uniquely identify classes. Early Distinctive Shapelet Classification (EDSC)

was a framework consisting of 2 steps; feature extraction and feature seletion. In the feature extraction step, all local shapelets are extracted and a distance threshold is learned. EDSC developed a technique which they called the best-matching-distance (BMD), which consideres a shapelet to be distinctive, by comparing it to all instances in the training data set and considering target classes distribution. If the majority of instances closer to the shapelet belong to a target class, it is considered to be distinctive of this class. In the feature selection step, EDSC selects a smaller set of features based on three main criteria; earliness, frequency and distinctivness. EDSC was later extended in two different directions.

In [GO12] EDSC was extended in the aspect of data dimensionality; as the original EDSC only convered univariate data sets. The extension covered; adapting the feature extraction to discover multivariate shapelets along with calculating distance thresholds for each dimension and using a distance threshold based on information gain, adapting the feature selection step, by modifying the equations calculating the notions of earliness, frequency and distinctivness to multiple dimensions. In [GRO14] EDSC was extended in the aspect of data uncertainty. The main idea was that shapelets use a distance threshold when being compared to instances, if the distance between the shapelet and the instance is less than the threshold, the instance is given the label of the shapelet. On the other hand, since the threshold represents a range, a shapelet is more certain of the class label given to an instance which has a distance closer to 0 than the class label given to an instance falling on the limits of the threshold. Motivated by the interest of uncertainty in the medical field, the goal was to provide a notion of uncertainty along with the interpretability of shapelets. The proposed Modified EDSC with Uncertainty estimates (MEDSC-U) calculated uncertainty as (1 - confidence), where confidence is a numeric value represting two aspects; confidence that the distance between the shapelet and the instance is less than the threshold and confidence that the shapelet can correctly classify the instance. This confidence notion could then be generalized for classes represented by multiple shapelets. The issue of confidence for eTSC had been discussed in other papers like [Par+13] and [Lv+19].

ECDIRE

The paper by [Mor+17b] propsed a new framework; Early Classification of Time Series based on Discriminating Classes Over Time (ECDIRE). There were two goals that ECDIRE wanted to achieve [Mor+17a]. The main goal of the framework was to track the accuracy evolution for a group of probabilistic classifiers across time; to identify safe time points after which predictions can be made. This would allow for avoiding unnecessary checking at each time point. The other goal was to offer an outliers discarding mechanism using a reliability condition. ECDIRE consisted of three training steps. In the first step the framework analyzes the data set and tries to identify the safe points at which classes can be differentiated from each other. This is done by defining time points as percentages of the full length and training, at each time point, a probabilistic classifier. The early points are then defined as those which maintain an accuracy of a desired percentage from that obtained on the complete data [SK16]. In the second step, a reliability threshold is determined using the class probabilities of the assigned class and the second highest class; this helps avoid uncertain classifications or very soon time points. In the last step, the final probabilistic classifiers are trained using the early time points identified in the first step. As a backup plan, if the last early time point did not coincide with the full length of the time series, ECDIRE acts like other methods and builds classifiers on

each time point.

TEASER

One of the latest classifiers introduced is the Two-tier Early and Accurate Series classifiER (TEASER) [SL20]. TEASER's main focus was challenging the logic of previous classifiers on determining the point of time at which a classification can be assigned to a time series instance. Previous methods would try to learn from the training data set a fixed prefix needed to be seen before it can start classification; like MPL in ECTS and EDSC, or to learn a safe point like in ECDIRE. These techniques would require the classifiers to always wait till they reach these fixed points of time before they can assign a label. Whereas TEASER was designed to give a classification once confident of it's decision, and not to depend on any fixed time points nor assume that all data instances were recorded at the same starting point. TEASER models the problem of eTSC with a two tier solution. In the first tier, a slave classifier is regularly checking the incoming data and assigning class probabilities. These probabilities are then assessed in the second tier by a master classifier, which either accepts the classification from the slave, or rejects it and decides to wait for more data to arrive. The classifiers are paired through out the whole process, such that for each slave classifier there is a master classifier that reads it's output and assesses it's results. TEASER starts off by defining the number of classifier pairs that will be neeeded, this is calculated by dividing the length of the longest data instance in the training data set by the window length which is a user defined parameter. Which is noted as $S = [n_{max}/w]$, where n_{max} is the length of the longest instance and w is the window length. During training, slave classifiers are trained on subsequences s_i, or snapshots, of the time series where the length of $s_i = i.w$. These snapshots are z-normalized based on the values existing only in them and not on values from the full length data as recommended by [Mor+17a]. The slave classifiers would then calculate three features; the class label with the highest probability $c(s_i)$, the vector of class probabilities for all available classes $P(s_i)$, and the difference between the class probability of the highest two classes denoted as Δd. These three features are then passed to the paired master classifier, which trains a model over them using a one-class Support Vector Machine (oc-SVM). A parameter that plays an important role in this process is v, it is the number of consecutive times the same class label is given to an instance. Once the same label is given v times for consecutive snapshots, the master classifier is confident of this class label.

Chapter 4

Data Sets

In order to address our research questions, this thesis considered a group of data sets from the UCR data archive [Dau+18] and the UEA data archive [Bag+18]. This Chapter will describe the data sets that were used to conduct this empirical experiment and their general properties.

4.1 UCR/UEA Archives

The University of California Riverside (UCR) and the University of East Anglia (UEA) data archives have been used as references in literature for many experiments on time series problems [AML19; Faw+20; Bag+17; YD19; Rui+20; Faw+19a] and specially for benchmarking the performance of algorithms.

4.1.1 UCR Archive

The UCR archive previously had 85 univariate data sets but in 2018 was then expanded to reach 128 data sets. Despite sharing the property of being univariate, the data sets cover a variety of values for other aspects like; training data size [16, 8926]; testing data size [20, 16800]; number of classes [2, 60]; length of time series [15, 2844] and for some data sets length of instances vary within the data set; and the nature of the data being collected. The archive offers a single predefined train/test split for each of the data sets to facilitate reproducability of results. Finally, the data is z-normalized; to remove offset and scaling. Z-normalization is the transformation of the data to have a mean of zero and one unit of standard deviation.

4.1.2 UEA Archive

The other archive we used is the UEA archive for multivariate data sets. Like the UCR archive, was developed and expanded over time. It started as a small archive of 12 data sets collected by Mustafa Baydogan, then later on, in 2018, was expanded to reach 30 data sets as a collaboration between researchers of UEA and UCR. The data sets vary in their characteristics; training data size [12, 30000]; testing data size [15, 20000]; number of classes [2, 39]; length of time series [8, 17984]; number of dimensions [2, 1345] and six different different groups based on their application area. The archive

was processed so that the data instances have equal lengths, instances of missing data points were excluded and a single predefined train/test split is provided.

4.1.3 Used Data Sets

For the scope of our experiment, a total of 84 data sets were selected from both archives to be used within the framework. There were two main points of interest driving our selection criteria for data sets:

- The data was collected from real life scenarios, that implies that the time series data collected was measured from existing subjects.
- The data represents a real time series structure behind it and not converted from another data format; like images and videos.

Based on the previously mentioned criteria we excluded data sets that belonged to type 'Simulated'; to comply with our first criterion. For the second criterion, we excluded data sets that belonged to either Image or Motion type. We have also excluded data sets that contain instances of varying lengths; as we believe that time series data imputation is a whole other topic outside the scope of our thesis.

The remaining collection of data sets still represented a wide variety in terms of data criteria. We present the statistics about the remaining data sets all together in figure 4.1. Since the smallest data set (StandWalkJump) and the largest data set (InsectWingbeat) are the same as the UEA archive; training and testing data sizes are the same as previously mentioned ranges in the original archive, [12, 30000] and [15, 20000] respectively; number of classes covers [2, 52]; length of time series the range is [24, 3000], with Chinatown being the shortest and MotorImagery the longest; the number of dimensions for multivariate data sets still covered the whole range [2, 1345]; and the remaining 15 types are AUDIO, DEVICE, ECG, EEG, EOG, EPG, HAR, HEMODYNAMICS, MISC, OTHER, SENSOR, SOUND, SPECTRO, SPEECH and Traffic.

Although the data sets that we used covered a variety of properties, but within some properties there were huge imbalances in the distribution of values. Within grouping by training data set size, the group of data sets with sizes [501, 1000] was the smallest, with only 4 data sets which is $\frac{1}{3}$ of the next smallest group. Grouping by series length was the best, among the different groupings, in terms of imbalance. Data sets of lengths [0, 50] were the smallest group with a count of 9 compared to the largest group, length ≥ 1001, of 20 data sets. Number of classes was highly imbalanced; the binary classification group contained 31 data sets, which was more than double the second largest group [6, 10] of 15 data sets. Finally, there was an over representation of type SENSOR than any other type with a total of 19 data sets. While multiple other groups were represented by very few data sets like MISC, SOUND and SPEECH which only had 1 data set; or like EOG, EPG and Traffic 2 data sets.

4.2 Karolinska data set

data set	Train	Test	Length	Classes	Type
ACSF1	100	100	1460	10	DEVICE
Beef	30	30	470	5	SPECTRO
Car	60	60	577	4	SENSOR
Chinatown	20	345	24	2	Traffic
CinCECGtorso	40	1380	1639	4	ECG
Coffee	28	28	286	2	SPECTRO
Computers	250	250	720	2	DEVICE
DodgerLoopDay	78	80	288	7	SENSOR
DodgerLoopGame	20	138	288	2	SENSOR
DodgerLoopWeekend	20	138	288	2	SENSOR
Earthquakes	322	139	512	2	SENSOR
ECG200	100	100	96	2	ECG
ECG5000	500	4500	140	5	ECG
ECGFiveDays	23	861	136	2	ECG
ElectricDevices	8926	7711	96	7	DEVICE
EOGHorizontalSignal	362	362	1250	12	EOG
EOGVerticalSignal	362	362	1250	12	EOG
EthanolLevel	504	500	1751	4	SPECTRO
FordA	3601	1320	500	2	SENSOR
FordB	3636	810	500	2	SENSOR
FreezerRegularTrain	150	2850	301	2	SENSOR
FreezerSmallTrain	28	2850	301	2	SENSOR
Fungi	18	186	201	18	OTHER
Ham	109	105	431	2	SPECTRO
HouseTwenty	34	101	3000	2	DEVICE
InsectEPGRegularTrain	62	249	601	3	EPG
InsectEPGSmallTrain	17	249	601	3	EPG
InsectWingbeatSound	25000	25000	600	10	AUDIO
ItalyPowerDemand	67	1029	24	2	SENSOR
LargeKitchenAppliances	375	375	720	3	DEVICE
Lightning2	60	61	637	2	SENSOR
Lightning7	70	73	319	7	SENSOR
Meat	60	60	448	3	SPECTRO
MelbournePedestrian	1194	2439	24	10	TRAFFIC
MoteStrain	20	1252	84	2	SENSOR
NonInvasiveFetalECGThorax1	1800	1965	750	42	ECG
NonInvasiveFetalECGThorax2	1800	1965	750	42	ECG
OliveOil	30	30	570	4	SPECTRO
Phoneme	214	1896	1024	39	SOUND

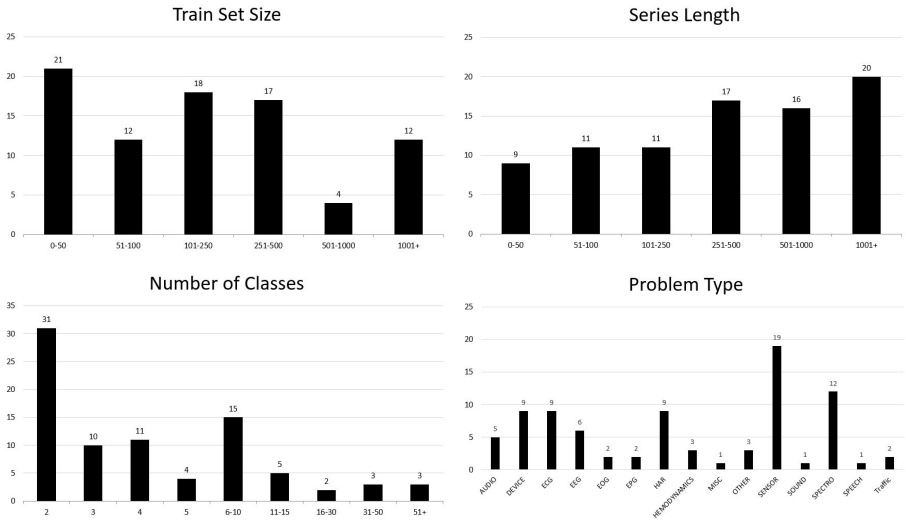

Figure 4.1: Metadata of the used data sets

Chapter 5

Framework and Experimental Design

This chapter describes our proposed framework which compares TSCAs in a context that is inspired by the problem of eTSC and the design of our experiment. We begin by defining what we mean by early classification context and how does our framework simulate it. Then we discuss the structure of the framework, the software tools that were used to build it and how to evaluate it. In the second section of the chapter we represent our experimental setup.

5.1 The Framework

As motivated earlier, eTSC is a field which is concerned with classification of time series data with earliness and accuracy as the main objectives. Dedicated eTSCAs focus on building models that can learn class labels of the data as early as possible while maintaining accuracy [Mor+17a]. Our framework, on the other hand, investigates the adaptation of the context in which conventional TSCA operate to learn a specific classification problem and the effect it has on the classifiers' performance.

5.1.1 Early Classification Context

We define our early classification context as the problem in which a TSCA is trained on labeled but incomplete time series data, then used to classify data instances of full length.

The motivation behind experimenting with such a context is driven by the medical field, where sometimes patients have to go through strenuous tests to be diagnosed. This is specially the case in audiological examinations; as patients are exposed to signals which pushs the limits of their hearing discomfort to be able to examine their problems. A classifier which can attain comparable reasults while only learning on a fraction of the whole time series data would shorten this process and thus relief patients from their discomfort.

To answer our first research question ***How can we adapt existing ts classification workflow for the early TS scenario ?***

We propose a framework that simulates an early classification context and runs a series of experiments using different TSCAs. It divides whole timeseries data instances into smaller subseries using a chopping algorithm, then starts multiple learning processes. In each learning process, the data is exposed incrementally to the classifier by revealing the next sub-series. These processess simulate an incremental learning scenario, where the classifier is not exposed to all the data points collected from the beginning to the end at once. However in each learning process it gets to know more information about the data instances than the previous process. In order to assess the performance of classifiers that were exposed to less data points, we use the performance of classifiers that were exposed to full length training data as a baseline. In the next subsection 5.1.2 we discuss the the details and the structure of the framework.

5.1.2 Structure and Toolkits

We implemented our framework in python language. For the implementation of the non-deep learning algorithms we used the open source libraires *sktimeV*.0.5.3 [Lön+19] and *pytsV*.0.11.0 [FJ20], while for the implementation of the deep learning algorithm InceptionTime we used the library *sktimedlV*.0.2.0 which provides an interface for the implementation provided in the study [Faw+20]. Finally, we used the library *TslearnV*.0.5.0.5 [Tav+20] as a mediator between the different model implementations; due to it's capacity to transform the raw data into any needed input format for all the other libraires.

The framework can be divided into several consecutive steps:

1. User input
2. Experiment preparation
3. Experiment Execution
4. Analysis report

User Input

In the user input step, the user provides the necessary information required by the framework to start it's experiments. These information provide the guidelines of the process which will be carried out and decide what steps should be done in the experiment preparation step. Table 5.2 shows a summary of each of the needed information, represented as a parameter for the framework.

We implemented the user input interface using *docopt* [1]; a library which offers command-line interfaces through the use of simple arguments and elements. It also allows to set default values for parameters which can be overriden if a value is passed. In order to start running the framework, one provides the necessary parameter values through a simple terminal command. Listing 5.1 shows a sample command for running our master python file *run.py* on the Computers data set using hyperparameters optimization, while applying a split into 10 chunks.

```
python run.py --etsc --dataset=Computers --cv --split=10
```

Listing 5.1: Sample command for providing user input to the framework

[1] http://docopt.org

Parameter	Data Type	Description
Dataset	String List	The name of the data set(s)
Splits	Integer	The number of splits to apply on the data set
Hyperparam	Boolean	Learn hyperparameters for the model or skip
NumIterations	Integer	Number of hyperparameters combinations to sample
ChunksToKeep	Integer List	The indexes of the chunks to use
FromBeg	Boolean	Reveal data chunks from beginning or end
ScoringFunction	String	The strategy for evaluating the model on test data

Table 5.2: Input parameters required for using the framework

Experiment Preparation

In the experiment preparation step, the framework starts preparing the different components of the experiment to be run. These components include data set management, models initialization, preparation of the early classification context and configuration of the training and testing processes.

Our framework's data sets handling module is configured to process data sets in either of two ways; automatic download from the UCR/UEA data archives, or manually provided data sets that comply with some specifications. Since the data loading module from *pyts* [FJ20] offers integrated tools with the UCR/UEA data archives for fetching and downloading data, we have configured our data module to extend *pyts* by transparently checking both data archives when provided with a data set name. The other option would be to provide a data set folder holding the name of the data set. Inside the folder there should exist one training data set file with the name "FolderName_TRAIN" and one testing data set file with the name "FolderName_TEST". Which is the case for most of the data sets from both archives. For multivariate data sets, our framework assumes that the training data set file, as well as the testing data set file, contains data of all dimensions. For maximum compatibility, our framework extends the data processing utilities from *sktime* [Lön+19] which can read data sets that comply with the guidelines provided by the UCR and the UEA archives. We consider "arff" format as the primary expected data type.

As mentioned earlier in section 2.3, there are two ways to work on multivariate time series data sets [Rui+20]. The first way is by using bespoke multivariate classifiers which can deal with the multiple dimensions available in the data. While the other is by using ensembles of univariate classifiers, each fit to a separate dimension under

the assumption of independence between the dimensions. Our framework applies the second technique for handling multivariate data sets.

In the experiment preparation step, the framework checks the data and determines the number of dimensions in consists of. If the data set is univariate then the model initalizes the default classifer implementations provided by the libraries. In case the data set was found to be multivariate, we initialize a special classifier type which extends the usage of univariate classifiers for multivariate problems. This special classifier is reffered to as $ColumnEnsembleClassifier$ in the *sktime* library implementation and as the $MultivariateClassifier$ in *pyts*.

There are two assumptions that we make at this step in our framework. First, we assume that to extend a specific univariate classifier to a problem, we should fit the same type of the classifer on all dimensions of the data. This is goes back to our goal, that we want to compare classifiers from different groups and not to create an ensemble which makes use of different classifiers applying different techniques, like HIVE-COTE does. Second, we assume that we always have to fit one classifier per dimension, which have proved to be not a feasible solution for high dimensional data sets. We will discuss this in more details in our results.

In preparation for the following steps in the framework, the experiment preparation step plays a role in configuring the training and the testing processes. For the training process, it is possible to either run using the default configuration of the classifiers from their libraries, or to learn it's hyperparameters through cross validation. We use a randomized search over the hyperparameters space of classifiers. The framework parameter $NumIterations$ represents the number of random samples to consider when optimizing the hyperparameters. If the space of the hyperparameters is less than $NumIterations$, then the space is exhausted by applying a grid search. We have considered 5-folds cross validation for all experiments.

For the testing process, the framework allows the usage of any chosen performance metric which is supported by *sklearn* library [Ped+11]. The chosen performance metric is utilized for the evaluation during both; the cross validation process and for calculating performance on the test data set. We consider "balanced accuracy" as our default performance metric; to account for problems where data sets have imbalanced classes distribution.

Based on the assumption that all data instances are of the same length, we implement our early classification context using a time series chopping algorithm. The algorithm is provided with some time series data set T with instances of length L and three user defined parameters; s which decides the number of splits L should be split to, $FromBeg$ a flag which indicates whether the data chopping should occur from the beginning or the end of the time series T, and $ChunksToKeep$ which provides the user with the option to use only specific chunks of interest and exclude the processing for all other chunks. Algorithm 1 demonstrates how the chopping algorithm works.

The result of the algorithm is then used to create new copies of T; $[T_1, T_2, \ldots, T_s]$, each copy T_i containing all instances from T but revealing a subsequence of L which we call the chunk. A chunk is a sequence of p data points starting from the beginning or the end of the instances in T based on the value of the parameter $FromBeg$. Each data set copy T_{i+1} reveals one more chunk than the previous data set T_i and the data set T_s represents the set with instances of the full length L. It should be noted that our algorithm tries to have an equal value for p used for each chunk, but this is limited by the length of the time series and the provided number of splits. The resulting data sets are then passed to the training process, where each algorithm applies it's own technique to learn on the data set.

Algorithm 1 The Chopping Algorithm

function $GetSplitIndexes(T, s, ChunksToKeep)$
  $L \leftarrow ExtractLength(T)$
  $ChunksSizes \leftarrow GetChunkSizes(L, s)$
  $SplitIndexes \leftarrow CumSum(ChunksSizes)$
  if $KeepChunks$ is not Null **then**
    $FilteredSplitIndexes \leftarrow KeepChunks(SplitIndexes, ChunksToKeep)$
  else
    $FilteredSplitIndexes \leftarrow SplitIndexes$
  end if
  return $FilteredSplitIndexes$
end function

To better understand how the chopping algorithm works, consider the next example. Let's assume we have a data set T with instances of length $L = 100$, we set the value of s= 10 and $FromBeg$ = True. The algorithm starts by calculating the value of p for each of the required 10 chunks using the function 2.

Algorithm 2 Function to Get Chunks Sizes

function $GetChunkSizes(L, s)$
  $ChunksSizes \leftarrow [\,]$
  for $i \leftarrow 0$ `to` $s - 1$ **do**
    if $i < L \pmod s$ **then**
      $p = (L \div s) + 1$
    else
      $p = L \div s$
    end if
    $ChunksSizes.append(p)$
  end for
  return $ChunksSizes$
end function

The resulting list $ChunksSizes$ from the algorithm will contain the value 10 for each of the chunks ($ChunksSizes$ = [10, 10, 10, 10, 10, 10, 10, 10, 10, 10]); this is because the length of the time series is divisible by the number of splits provided, so it results in perfect splits of the data and thus chunks of equal sizes. If we assume the same scenario again but with a different s value like 8, the algorithm will try to provide as equal values of p as possible giving back a $ChunksSizes$ of values [13, 13, 13, 13, 12, 12, 12, 12].

After calculating the chunk sizes, the chopping algorithm translates $ChunksSizes$ into a list of indexes called $SplitIndexes$. Which is then used to create the different copies T_i by selecting subsequences. The values of $SplitIndexes$ are the indexes of the last time point that a specific chunk should read. We calculate these values by carrying out a cumulative sum over the values of $ChunksSizes$. For example, the list $ChunksSizes$ = [10, 10, 10, 10, 10, 10, 10, 10, 10, 10] is translated into the list $SplitIndexes$ = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]. This means that the

corresponding data set T_1 will contain all it's data instances represented by the subsequences from the 1^{st} time point till the 10^{th} time points, and the data set T_3 will contain subsequence from the 1^{st} time point till the 30^{th} time point. When we apply the same translation to the list $ChunksSizes$ = [13, 13, 13, 13, 12, 12, 12, 12], it gets translated into $SplitIndexes$ = [13, 26, 39, 52, 64, 76, 88, 100].

Finally, the algorithm filters out the list $SplitIndexes$, if the user provided a list of specific chunks to keep using the variable $ChunksToKeep$. The values passed through $ChunksToKeep$ corresponds to the indexes of the chunks the user is interested in, all the other chunks are excluded. Note that this process is different from reducing the number of splits; as the number of splits affects the sizes of the chunks being created. While $ChunksToKeep$ is used only to filter out the chunks after their sizes are already determined and translated into $SplitIndexes$. The space of possible values for $ChunksToKeep$ is between [1, s]. The logic of the function is demonstrated in function 3

Algorithm 3 Function to filter out Chunks

```
1: function KeepChunks(SplitIndexes, ChunksToKeep)
2:     FilteredSplitIndexes ← [ ]
3:     for i ϵ ChunksToKeep do
4:         FilteredSplitIndexes.append(SplitIndexes[i])
5:     end for
6:     return FilteredSplitIndexes
7: end function
```

If we consider the list $ChunksToKeep$ = [1, 3, 5] for our example $SplitIndexes$ = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]. The resulting filtered list will be $FilteredSplitIndexes$ = [10, 30, 50], which will result in only 3 copies of T; T_1, T_3 and T_5, each represented by it's respective subsequence length.

If we have a closer look at the parameter s, we can recognise that it contributes to the granularity of the earliness factor that we are calculating. The value of s decides the total number of chunks we will use, which subsequently decides also the ratio represented by each chunk to the total length of the time series T. That means the greater the value of s, the smaller the chunk sizes and consequently the more granular results we can test for the algorithms. If we consider our previous example, when we set the value of $s = 10$ then the length of each subsequence would be close to 10% of the total length. If we decided to change the value of $s = 20$, then each chunk will roughly represents 5% of the total length and so on. Increasing the number of splits to be used comes with the cost of time and resources, each extra split adds one extra run per algorithm per data set.

Experiment Execution

The experiment execution step is where the actual processing happens. After the preparation of the chunks for the data sets in the previous step, learning processes start and each of the implemented classifiers starts training on the prepared data sets. Each of these training processes is then followed by a testing process, where performance scores are calculated using the function provided by the parameter $ScoringFunction$ on testing data sets.

As mentioned in section 3.3, we follow the grouping criteria by [Bag+17] which arranges TSCAs into 5 main categories based on their techniques. We have included classifiers representing each of the 5 groups, in addition to a 6^{th} group which represents the family of deep learning time series algorithms. We tried, as far as the implemented libraries allowed, to represent each group with 2 classifiers; one that is a non-ensebmle and another which is an ensemble. A total of 10 classifiers are included in the framework covering all 6 groups, these can be broken down into the following:

- Distance based : 1NN using MSM distance and PForest
- Phase dependent algorithms : TSF
- Shapelets: LS and ST
- Dictionary based : WEASEL and BOSS
- Deep learning : InceptionTime
- Classical baselines: 1NN using euclidean distance and 1NN using DTW

Table 5.4 shows the configuration and hyperparameters for all the included classifiers. We used the same hyperparameter space like that of the original published papers as closely as possible. During training, the number of hyperparameter sampling is decided by the framework parameter $NumIterations$ and we fix the number of cross validation folds to 5.

After the training processes are finished, the classifiers are then shown the testing data sets to evaluate their performance. If the experiment is configured to do hyperparameter optimization, then the classifier version that attained the highest validation score, from the 5 cross validation folds, is used for calculating the performance score on the testing data set.

Analysis Report

The last step of our framework is the analysis report preparation. After the successful running of the experiments and calculating performance scores using the provided scoring function, we create a single report for each run experiment. We produce one report per classifier per data set. We refer here by data set to each data set that was finally produced by the chopping algorithm. These reports collect the essential data and statistics that form the basis of our results. Table 5.6 shows the structure of the output for the analysis report.

Most of these information are already available once the testing process is finished; because either they are metadata about the experiment or metrics calculated during training and testing. Metadata about the process is already known before the running of the process; like the $Classifier$, $Dataset$ and $Revealed\%$. While the processes metrics are $Traintime$, $Testtime$; which represent the total duration of the learning and testing processes, and the $Testscore$ which represents performance of the classifier.

There is only one metric which is calculated during the analysis report phase after all the other metrics are calculated; the harmonic mean. We previously defined the HM in chapter 3.4, it has been used by [SL20] as the objective function for evaluating their algorithm $TEASER$. The HM is a popular choice for calculating a score that combines 2 objectives at the same time, which is in our case earliness and performance like the other eTSC problems. Earliness is represented by the value of the parameter $Revealed\%$ which is a fraction of the full series length, e.g. 0.5 means 50% of the total length. On the other hand, performance is represented by the value of the parameter $Testscore$. We use HM as the means of evaluating classifiers in our experiment; due to

Classifier	**Parameters**
MSM	c = {0.01, 0.1, 1, 10, 100}
PForest	k = 100 r = 5
TSF	r = 500 splitting = {entropy, gini} maxfeatures = {sqrt, log2}
LS	λ_w = {0.01, 0.1, 1} L_{min} = {0.025, 0.075, 0.125, 0.175, 0.2} R = {1, 2, 3} K = {0.05, 0.15, 0.3} η = 0.01 maxIter = {2000, 5000, 10000}
ST	t = 60 mins n={500,100}
CBOSS	t = 60 mins k={50, 100, 250, 500} s=250 p=[0.5 - 1]
WEASEL	ANOVA = {True, False} bigrams ={True, False} binning = {equi-depth, equi-width, information-gain}
Inception	epochs = 1500 batch size = 64 η = 0.001 kernel sizes = {10, 20, 40}
DTW	full warping window

Table 5.4: Parameters and configuration for TSCAs

it's capacity to incorporate both objectives in one score. Which allows us to compare the trained classifiers in terms of a one score value without worrying about fixing the value of either the earliness or the performance.

5.2 Experiments

Our experiments were conducted on univariate data sets from the UCR archive as well as multivariate data sets from the UEA archive. The data sets were chosen based on the criteria discussed in the subsection 4.1.3. Each of the data archives offers a train and test split of the data which we have used unchanged for all the classifiers. All our

Item	Description
Classifier	The classifier name
Train time	The total training time (CPU Time)
Test time	The total testing time (CPU Time)
Test score	Performance score on testing data set
Params	List of hyperparameters used and their values
Revealed %	Percent of data points used for training from the total length
Harmonic Mean	Harmonic mean between test score and revealed %
Data set	The name of the data set

Table 5.6: Analysis report created for each experiment

experiments were run on a LINUX server with an AMD Ryzen 7 1700 Processor and 32GB RAM using Python 3.7.10.

We used the same experimental configuration through out all our experiments. We fixed the value of the parameter $Splits = 10$, that is for every data set the data would be revealed for the classifiers incrementally in batches of 10% from the total length. The chopping algorithm was set to always reveal data from the beginning of the time series. Due to time limitations, we restricted our experiments to run only on the 1^{st}, 2^{nd}, 3^{rd} and 10^{th} chunksof the data sets by setting $ChunksToKeep = \{1,2,3,10\}$. We included the 10^{th} chunk to represent the baseline performance of each classifier if shown the full length data. While the first 3 chunks were used to represent the classifiers in the early context scenarios. Hyperparameters optimization was always carried out on all data sets with a $NumIterations$ maximum sampling value of 50 iterations, unless proven unfeasible for any of the classifiers due to memory shortage or time constraint. In this case the experiment is repeated for all classifiers without hyperparameters optimization. The selected performance metric for our experiments was *BalancedAccuracy*. Balanced accuracy is a performance metric which can handle data sets with skewed class distributions by avoiding inflated performance estimates. To calculate balanced accuracy, the *Recall* value is computed for each class then averaged over the total number of classes. We have set all our experiments to use the same configuration for all classifiers on the same data set.

5.2.1 Excluded Classifiers

Some of the included classifiers in the framework were excluded from our experiments; either because they couldn't operate in the early classification context or because they couldn't attain comparable results to the published performances by previously published frameworks.

For example, KNNED and InceptionTime were excluded due to the nature of their techniques which couldn't handle our created context. Both algorithms are clearly able to learn on chopped training data sets. However once the testing phase is reached,

they would fail to classify instances of the testing data, showing errors related to mismatches between the expeccted length of instances and the provided length. Yet they would finish the last chunk where the data is provided in it's original full length. The reason why KNNED fails such scenario, is that it uses ED which is a point-wise comparison distance measure that cannot compare time series of unequal lengths [Tan+19]. On the other hand, InceptionTime is a deep learning model whose input layer architecture, represented by number of nodes, depends on the length of the input time series [Faw+19a]. Since we use full length instances for testing, this caused an overflow of input data than what the model structure was expecting. There have been literature discussing adapting TSCAs to unequal time series, but this is out of the scope of our experiments. For more details refer to [CCP09; Tan+19; Faw+19a]

Although KNNDTW has been a competent time series classifier for decades; thanks to the exploitation of the elastic distance measure DTW. We couldn't get either implementation of KNNDTW from *sktime* and *pyts* to work on the chopped data; due to errors in the data representation needed by lower level internal libraries. KNNDTW would have been able to operate in the early classification context if used with a full warping window; which would have been successful in handling extreme classification cases like classifying full length data even when learning on the 10% chunk data set.

Two classifiers were excluded because they attained significantly inferior results compared to the published scores; these are KNNMSM and LS. Specially on the InsectWingbeatSound data set, for which we attained a difference in performance of -45.71% for KNNMSM and -20.71% for LS than the results published by [Bag+17]

Our experiments proceeded with the remaining 5 classifiers

Chapter 6

New Framework

Chapter 7

Results

This chapter should discuss the Results

The baseline performance for each classifier is determined by it's performance on the full length data.

Chapter 8

Findings Discussion

This chapter should discuss the Findings

Chapter 9

Summary and Outlook

This chapter should discuss the Summary

Appendix

Bibliography

[Alo+18] Md Zahangir Alom et al. "The history began from alexnet: A comprehensive survey on deep learning approaches". In: *arXiv preprint arXiv:1803.01164* (2018).

[AML19] Amaia Abanda, Usue Mori, and Jose A Lozano. "A review on distance based time series classification". In: *Data Mining and Knowledge Discovery* 33.2 (2019), pp. 378–412.

[Bag+12] Anthony Bagnall et al. "Transformation based ensembles for time series classification". In: *Proceedings of the 2012 SIAM international conference on data mining*. SIAM. 2012, pp. 307–318.

[Bag+15] Anthony Bagnall et al. "Time-series classification with COTE: the collective of transformation-based ensembles". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (2015), pp. 2522–2535.

[Bag+17] Anthony Bagnall et al. "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery* 31.3 (2017), pp. 606–660.

[Bag+18] Anthony Bagnall et al. "The UEA multivariate time series classification archive, 2018". In: *arXiv preprint arXiv:1811.00075* (2018).

[BB17a] Aaron Bostrom and Anthony Bagnall. "A shapelet transform for multivariate time series classification". In: *arXiv preprint arXiv:1712.06428* (2017).

[BB17b] Aaron Bostrom and Anthony Bagnall. "Binary Shapelet Transform for Multiclass Time Series Classification". In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXII: Special Issue on Big Data Analytics and Knowledge Discovery*. Ed. by Abdelkader Hameurlain et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 24–46. ISBN: 978-3-662-55608-5. DOI: 10.1007/978-3-662-55608-5_2. URL: https://doi.org/10.1007/978-3-662-55608-5_2.

[BCM16] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. "Should we really use post-hoc tests based on mean-ranks?" In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 152–161.

[Ber+06] Laurent Bernaille et al. "Traffic classification on the fly". In: *ACM SIGCOMM Computer Communication Review* 36.2 (2006), pp. 23–26.

[Bos18] Aaron Bostrom. "Shapelet Transforms for Univariate and Multivariate Time Series Classification". PhD thesis. University of East Anglia, 2018.

[BR16] Mustafa Gokce Baydogan and George Runger. "Time series representation and similarity based on local autopatterns". In: *Data Mining and Knowledge Discovery* 30.2 (2016), pp. 476–509.

[Bre01] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[BRT13] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. "A bag-of-features framework to classify time series". In: *IEEE transactions on pattern analysis and machine intelligence* 35.11 (2013), pp. 2796–2802.

[CCC16] Zhicheng Cui, Wenlin Chen, and Yixin Chen. "Multi-scale convolutional neural networks for time series classification". In: *arXiv preprint arXiv:1603.06995* (2016).

[CCP09] Jorge Caiado, Nuno Crato, and Daniel Peña. "Comparison of times series with unequal length in the frequency domain". In: *Communications in Statistics—Simulation and Computation®* 38.3 (2009), pp. 527–540.

[CN04] Lei Chen and Raymond Ng. "On the marriage of lp-norms and edit distance". In: *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 2004, pp. 792–803.

[Dau+18] Hoang Anh Dau et al. *The UCR Time Series Classification Archive*. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/. Oct. 2018.

[Dem06] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *The Journal of Machine Learning Research* 7 (2006), pp. 1–30.

[Den+13] Houtao Deng et al. "A time series forest for classification and feature extraction". In: *Information Sciences* 239 (2013), pp. 142–153.

[DGM97] Gautam Das, Dimitrios Gunopulos, and Heikki Mannila. "Finding similar time series". In: *European Symposium on Principles of Data Mining and Knowledge Discovery*. Springer. 1997, pp. 88–100.

[Din+08] Hui Ding et al. "Querying and mining of time series data: experimental comparison of representations and distance measures". In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1542–1552.

[Faw+19a] Hassan Ismail Fawaz et al. "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery* 33.4 (2019), pp. 917–963.

[Faw+19b] Hassan Ismail Fawaz et al. "Deep neural network ensembles for time series classification". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–6.

[Faw+20] Hassan Ismail Fawaz et al. "Inceptiontime: Finding alexnet for time series classification". In: *Data Mining and Knowledge Discovery* 34.6 (2020), pp. 1936–1962.

[FJ20] Johann Faouzi and Hicham Janati. "pyts: A Python Package for Time Series Classification". In: *Journal of Machine Learning Research* 21.46 (2020), pp. 1–6. URL: http://jmlr.org/papers/v21/19-763.html.

[FRM94] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. "Fast subsequence matching in time-series databases". In: *Acm Sigmod Record* 23.2 (1994), pp. 419–429.

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[GŁ13] Tomasz Górecki and Maciej Łuczak. "Using derivatives in time series classification". In: *Data Mining and Knowledge Discovery* 26.2 (2013), pp. 310–331.

[GM01] M Pamela Griffin and J Randall Moorman. "Toward the early diagnosis of neonatal sepsis and sepsis-like illness using novel heart rate analysis". In: *Pediatrics* 107.1 (2001), pp. 97–104.

[GO12] Mohamed F Ghalwash and Zoran Obradovic. "Early classification of multivariate temporal observations by extraction of interpretable shapelets". In: *BMC bioinformatics* 13.1 (2012), p. 195.

[Gra+14] Josif Grabocka et al. "Learning time-series shapelets". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 392–401.

[GRO14] Mohamed F Ghalwash, Vladan Radosavljevic, and Zoran Obradovic. "Utilizing temporal patterns for estimating uncertainty in interpretable early decision making". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 402–411.

[Gua+19] Chaoyu Guan et al. "Towards a deep and unified understanding of deep neural models in nlp". In: *International conference on machine learning*. PMLR. 2019, pp. 2454–2463.

[Gup+20] Ashish Gupta et al. "A Fault-Tolerant Early Classification Approach for Human Activities using Multivariate Time Series". In: *IEEE Transactions on Mobile Computing* (2020).

[He+15] Guoliang He et al. "Early classification on multivariate time series". In: *Neurocomputing* 149 (2015), pp. 777–787.

[He+16] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[Hil+14] Jon Hills et al. "Classification of time series by shapelet transformation". In: *Data Mining and Knowledge Discovery* 28.4 (2014), pp. 851–881.

[Ita75] Fumitada Itakura. "Minimum prediction residual principle applied to speech recognition". In: *IEEE Transactions on acoustics, speech, and signal processing* 23.1 (1975), pp. 67–72.

[JJO11] Young-Seon Jeong, Myong K Jeong, and Olufemi A Omitaomu. "Weighted dynamic time warping for time series classification". In: *Pattern recognition* 44.9 (2011), pp. 2231–2240.

[Kat16] Rohit J Kate. "Using dynamic time warping distances as features for improved time series classification". In: *Data Mining and Knowledge Discovery* 30.2 (2016), pp. 283–312.

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[KP01] Eamonn J Keogh and Michael J Pazzani. "Derivative dynamic time warping". In: *Proceedings of the 2001 SIAM international conference on data mining*. SIAM. 2001, pp. 1–11.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[LB15] Jason Lines and Anthony Bagnall. "Time series classification with ensembles of elastic distance measures". In: *Data Mining and Knowledge Discovery* 29.3 (2015), pp. 565–592.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[LGI17] Thach Le Nguyen, Severin Gsponer, and Georgiana Ifrim. "Time series classification by sequence learning in all-subsequence space". In: *2017 IEEE 33rd international conference on data engineering (ICDE)*. IEEE. 2017, pp. 947–958.

[Lin+07] Jessica Lin et al. "Experiencing SAX: a novel symbolic representation of time series". In: *Data Mining and knowledge discovery* 15.2 (2007), pp. 107–144.

[Lin+15] Yu-Feng Lin et al. "Reliable early classification on multivariate time series with numerical and categorical attributes". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2015, pp. 199–211.

[LKL12] Jessica Lin, Rohan Khade, and Yuan Li. "Rotation-invariant similarity in time series using bag-of-patterns representation". In: *Journal of Intelligent Information Systems* 39.2 (2012), pp. 287–315.

[LLY17] Xin Liao, Kaide Li, and Jiaojiao Yin. "Separable data hiding in encrypted image based on compressive sensing and discrete fourier transform". In: *Multimedia Tools and Applications* 76.20 (2017), pp. 20739–20753.

[LMT16] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. "Data augmentation for time series classification using convolutional neural networks". In: *ECML/PKDD workshop on advanced analytics and learning on temporal data*. 2016.

[Lön+19] Markus Löning et al. "sktime: A unified interface for machine learning with time series". In: *arXiv preprint arXiv:1909.07872* (2019).

[Low14] Richard Lowry. "Concepts and applications of inferential statistics". In: (2014).

[LTB18] Jason Lines, Sarah Taylor, and Anthony Bagnall. "Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles". In: *ACM Transactions on Knowledge Discovery from Data* 12.5 (2018).

[Luc+19] Benjamin Lucas et al. "Proximity forest: an effective and scalable distance-based classifier for time series". In: *Data Mining and Knowledge Discovery* 33.3 (2019), pp. 607–635.

[Lv+19] Junwei Lv et al. "An effective confidence-based early classification of time series". In: *IEEE Access* 7 (2019), pp. 96113–96124.

[Mar08] Pierre-François Marteau. "Time warp edit distance with stiffness adjustment for time series matching". In: *IEEE transactions on pattern analysis and machine intelligence* 31.2 (2008), pp. 306–318.

[MKY11] Abdullah Mueen, Eamonn Keogh, and Neal Young. "Logical-shapelets: an expressive primitive for time series classification". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011, pp. 1154–1162.

[Mor+17a] Usue Mori et al. "Early classification of time series by simultaneously optimizing the accuracy and earliness". In: *IEEE transactions on neural networks and learning systems* 29.10 (2017), pp. 4569–4578.

[Mor+17b] Usue Mori et al. "Reliable early classification of time series based on discriminating the classes over time". In: *Data mining and knowledge discovery* 31.1 (2017), pp. 233–263.

[Mor+19] Usue Mori et al. "Early classification of time series using multi-objective optimization techniques". In: *Information Sciences* 492 (2019), pp. 204–218.

[MVB19] Matthew Middlehurst, William Vickers, and Anthony Bagnall. "Scalable dictionary classifiers for time series classification". In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2019, pp. 11–19.

[Par+13] Nathan Parrish et al. "Classifying with confidence from incomplete information". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 3561–3589.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Pet+16] François Petitjean et al. "Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm". In: *Knowledge and Information Systems* 47.1 (2016), pp. 1–26.

[RA04] Juan Jose Rodriguez and Carlos J Alonso. "Support vector machines of interval-based features for time series classification". In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer. 2004, pp. 244–257.

[RK04] Chotirat Ann Ratanamahatana and Eamonn Keogh. "Making time-series classification more accurate using learned constraints". In: *Proceedings of the 2004 SIAM international conference on data mining*. SIAM. 2004, pp. 11–22.

[RK05] Chotirat Ann Ratanamahatana and Eamonn Keogh. "Three myths about dynamic time warping data mining". In: *Proceedings of the 2005 SIAM international conference on data mining*. SIAM. 2005, pp. 506–510.

[RK13] Thanawin Rakthanmanon and Eamonn Keogh. "Fast shapelets: A scalable algorithm for discovering time series shapelets". In: *proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM. 2013, pp. 668–676.

[Rui+20] Alejandro Pasos Ruiz et al. "The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery* (2020), pp. 1–49.

[SAD12] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. "The move-split-merge metric for time series". In: *IEEE transactions on Knowledge and Data Engineering* 25.6 (2012), pp. 1425–1438.

[SC78] Hiroaki Sakoe and Seibi Chiba. "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE transactions on acoustics, speech, and signal processing* 26.1 (1978), pp. 43–49.

[Sch15] Patrick Schäfer. "The BOSS is concerned with time series classification in the presence of noise". In: *Data Mining and Knowledge Discovery* 29.6 (2015), pp. 1505–1530.

[SH12] Patrick Schäfer and Mikael Högqvist. "SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets". In: *Proceedings of the 15th international conference on extending database technology*. 2012, pp. 516–527.

[Shi+20] Ahmed Shifaz et al. "Ts-chief: A scalable and accurate forest algorithm for time series classification". In: *Data Mining and Knowledge Discovery* (2020), pp. 1–34.

[SK16] Tiago Santos and Roman Kern. "A Literature Survey of Early Time Series Classification and Deep Learning." In: *Sami@ iknow*. 2016.

[SL17a] Patrick Schäfer and Ulf Leser. "Fast and accurate time series classification with weasel". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, pp. 637–646.

[SL17b] Patrick Schäfer and Ulf Leser. "Multivariate time series classification with WEASEL+ MUSE". In: *arXiv preprint arXiv:1711.11343* (2017).

[SL20] Patrick Schäfer and Ulf Leser. "TEASER: early and accurate time series classification". In: *Data Mining and Knowledge Discovery* 34.5 (2020), pp. 1336–1362.

[SPK18] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. "Towards a Universal Neural Network Encoder for Time Series." In: *CCIA*. 2018, pp. 120–129.

[SW17] Simone Scardapane and Dianhui Wang. "Randomness in neural networks: an overview". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7.2 (2017), e1200.

[Sze+15] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[Sze+17] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[Tan+19] Chang Wei Tan et al. "Time series classification for varying length series". In: *arXiv preprint arXiv:1910.04341* (2019).

[Tav+20] Romain Tavenard et al. "Tslearn, A Machine Learning Toolkit for Time Series Data". In: *Journal of Machine Learning Research* 21.118 (2020), pp. 1–6. URL: http://jmlr.org/papers/v21/20-091.html.

[TH16] Pattreeya Tanisaro and Gunther Heidemann. "Time series classification using time warping invariant echo state networks". In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2016, pp. 831–836.

[TM16] Romain Tavenard and Simon Malinowski. "Cost-aware early classification of time series". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2016, pp. 632–647.

[TPW20] Chang Wei Tan, François Petitjean, and Geoffrey I Webb. "FastEE: Fast Ensembles of Elastic Distances for time series classification". In: *Data Mining and Knowledge Discovery* 34.1 (2020), pp. 231–272.

[Wan+13] Xiaoyue Wang et al. "Experimental comparison of representation methods and distance measures for time series data". In: *Data Mining and Knowledge Discovery* 26.2 (2013), pp. 275–309.

[WYO17] Zhiguang Wang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline". In: *2017 International joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 1578–1585.

[Xin+11] Zhengzheng Xing et al. "Extracting interpretable features for early classification on time series". In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM. 2011, pp. 247–258.

[XPK10] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. "A brief survey on sequence classification". In: *ACM Sigkdd Explorations Newsletter* 12.1 (2010), pp. 40–48.

[XPP09] Zhengzheng Xing, Jian Pei, and S Yu Philip. "Early prediction on time series: a nearest neighbor approach". In: *Twenty-First International Joint Conference on Artificial Intelligence*. Citeseer. 2009.

[XPP12] Zhengzheng Xing, Jian Pei, and S Yu Philip. "Early classification on time series". In: *Knowledge and information systems* 31.1 (2012), pp. 105–127.

[YD19] Omolbanin Yazdanbakhsh and Scott Dick. "Multivariate Time Series Classification using Dilated Convolutional Neural Network". In: *arXiv preprint arXiv:1905.01697* (2019).

[YK09] Lexiang Ye and Eamonn Keogh. "Time series shapelets: a new primitive for data mining". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2009, pp. 947–956.

[Zei12] Matthew D Zeiler. "Adadelta: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[Zha+17] Bendong Zhao et al. "Convolutional neural networks for time series classification". In: *Journal of Systems Engineering and Electronics* 28.1 (2017), pp. 162–169.

[Zhe+14] Yi Zheng et al. "Time series classification using multi-channels deep convolutional neural networks". In: *International conference on web-age information management*. Springer. 2014, pp. 298–310.

[Zhe+16] Yi Zheng et al. "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification". In: *Frontiers of Computer Science* 10.1 (2016), pp. 96–112.

9.1 Declaration of Authorship

I hereby declare that I have written this thesis "TITLE TITLE TITLE" without any help from others and without the use of documents and aids other than those stated above. Furthermore, I have mentioned all used sources and have cited them correctly according to the citation rules. Moreover, I confirm that the paper at hand was not submitted in this or similar form at another examination office, nor has it been published before.
Magdeburg, DATE, SIGNATURE