# Invariant Features For Time-Series Classification

A thesis submitted for the degree of

*Doctor of Natural Science (Dr. rer. nat.)*

by

**Josif Grabocka**

Department of Computer Science

Information Systems and Machine Learning Lab (ISMLL)

University of Hildesheim, Germany


Summer 2015

To Linda and My Family

# Acknowledgements

This thesis would not have been successful without the immense help of several individuals, to whom I sincerely feel indebted.

The tireless assistance of my supervisor Prof. Dr. Dr. Lars Schmidt-Thieme was instrumental in reforming my views on artificial intelligence and machine learning; literally achieving a quantum leap in moving from heuristic-based techniques towards principally-optimized methods.

Prof. Dr. Alexandros Nanopoulos, on the other hand, patiently taught me the basics of research and dissemination. His dynamic assistance in the early days of my doctorate studies had a powerful influence in my overall formation.

My colleagues at the Information Systems and Machine Learning Lab (ISMLL) are the perfect companions for exploring and discussing new ideas. Above all, Dr. Lucas Drumond, Martin Wistuba and Nicolas Schilling were extremely helpful in merging efforts towards fruitful research collaborations.

A key ingredient in my achievements was the funding provided by the Seventh Framework Programme of the European Commission, through project REDUCTION (#288254).

Finally, the largest gratitude goes to Linda, whose partnership transforms every day of my life to a wonderful moment. I am not able to verbally formulate the proper appreciation she deserves for changing my life. Last, but not least, I am deeply thankful to the unconditional assistance that my family has always granted me.

# Abstract

Time series represent the most widely spread type of data, occurring in a myriad of application domains, ranging from physiological sensors up to astronomical light intensities. The classification of time-series is one of the most prominent challenges, which utilizes a recorded set of expert-labeled time-series, in order to automatically predict the label of future series without the need of an expert.

The patterns of time-series are often shifted in time, have different scales, contain arbitrarily repeating patterns and exhibit local distortions/noise. In other cases, the differences among classes are attributed to small local segments, rather than the global structure. For those reasons, values corresponding to a particular time-stamp have different semantics on different time-series. We call this phenomena as intra-class variations. The lion's share of this thesis is composed of presenting new methods that can accurately classify time-series instances, by handling variations.

The answer towards resolving the bottlenecks of intra-class variations relies on not using the time-series values as direct features. Instead, the approach of this thesis is to extract a set of features that, on one hand, represent all the variations of the data and, on the other hand, can boost classification accuracy. In other words, this thesis proposes a list of methods that addresses diverse aspects of intra-class variations.

The first proposed approach is to generate new training instances, by transforming the support vectors of an SVM. The second approach decomposes time-series through a segment-wise convolutional factorization. The strategy involves learning a set of patterns and weights, whose product can approximate each sub-sequence of the time series. However, the main contribution of the thesis is the third approach, called shapelet learning, which utilizes the training labels during the learning process, i.e. the process is supervised. Since the features are learned on the training labels, there is a higher tendency of performing strongly in terms of predicting the testing labels. In addition, we

present a fast alternative method for shapelet discovery. Our strategy is to prune segment candidates using a two step approach. First of all, we prune candidates based on their similarity towards previously considered candidates. Secondly, non-similar (hence diverse) candidates are selected only if the features they produce improve the classification results. The last two chapters of the thesis describes two methods that extract features from datasets having special characteristics. More concretely, we propose a classification method suited for series having missing values, as well as a method that extract features from time series having repetitive patterns.

In particular, the proposed shapelet learning approach is highly successful in terms of classification accuracy. Experiments against 13 rival methods in 28 real-life datasets demonstrate that shapelet learning is the most accurate method in the realm of time-series classification. In addition, our fast shapelet discovery technique is able to classify MB-scale datasets in matters of seconds and GB-scale datasets in matters of minutes.

# Contents

# List of Figures

# List of Tables

## LIST OF TABLES

# Notation

This thesis will adhere to the following notational conventions:

| | |
|---|---|
| $N \in \mathbb{N}$ | The number of time-series in a dataset |
| $Q \in \mathbb{N}$ | The length of time-series |
| $T \in \mathbb{R}^{N \times Q}$ | Time-series data |
| $C \in \mathbb{N}$ | Number of time series labels |
| $Y \in \{1, \ldots, C\}^N$ | Labels of the time series |
| $L \in \mathbb{N}$ | Length of sliding window sub-sequences |
| $\delta \in \mathbb{N}$ | Sliding window incremental threshold |
| $M \in \mathbb{N}$ | Number of sliding window sub-sequences |
| $S \in \mathbb{R}^{N \times M \times L}$ | Segments of the time series |
| $K \in \mathbb{N}$ | Number of patterns |
| $P \in \mathbb{R}^{K \times L}$ | Patterns |
| $\eta \in \mathbb{R}$ | Learning Rate |
| $I \in \mathbb{N}$ | Number of Iterations |
| $F \in \mathbb{R}^{N \times K}$ | New time-series representation |
| $\lambda_P \in \mathbb{R}$ | Regularization hyper-parameter |
| $W \in \mathbb{R}^{K+1}$ | Classification Weights |
| $\sigma : \mathbb{R} \to \mathbb{R}$ | Sigmoid Logistic Function |
| $\hat{Y} \in \mathbb{R}^{N \times C}$ | Estimated series targets |
| $\mathcal{L} : (\mathbb{N} \times \mathbb{R}) \to \mathbb{R}$ | Loss function |
| $\mathcal{O} : \left(\mathbb{R}^{N \times Q} \times \mathbb{R}^{K+1}\right) \to \mathbb{R}$ | Objective function |

In order to improve the clarity of notation, a brief clarification of the main terms is expanded as follows:

- **Time-series:** A time-series is an ordered sequence of point values. In a dataset of $N$ series instances, where each series has $Q$ points, we denote the series dataset as $T \in \mathbb{R}^{N \times Q}$.
- **Sliding Window Segment:** A sliding window content of size $L \in \mathbb{N}$, is a series subsequence starting at a position $j \in \{1, \ldots, Q-$

$L\}$ of a series $i$ of dataset T, and is denoted as $S_{i,j} \in \mathbb{R}^L$, $S_{i,j} := (T_{i,j}, T_{i,j+1}, \ldots, T_{i,j+L-1})$.

- **All Dataset Segments:** The starting position of each sliding window segment is incremented by an offset $\delta \in \{1, \ldots, L\}$, therefore the maximum number of segments per series is defined as $M := \frac{Q-L}{\delta}$. All the segments of a time-series datasets are denoted as $S \in \mathbb{R}^{N \times M \times L}$.
- **Patterns:** Some methods presented in this thesis mine for $K$-many latent patterns, each having the same size as one segment, i.e $L$. So, the latent patterns are denoted as $P \in \mathbb{R}^{K \times L}$.

The aforementioned notation is a global convention for the whole thesis. In addition, each chapter introduces a local set of symbols whose lifespan is limited to the scope of the particular chapter. The local symbol definitions eclipse the global notation, wherever declared.

# Abbreviation

| | |
|---|---|
| BoW | Bag of Words |
| CID | Complexity Invariant Distance |
| DTW | Dynamic Time Warping |
| DTW-NN | Nearest Neighbor with Dynamic Time Warping |
| ED | Euclidean Distance |
| FS | Fast Shapelets |
| HMM | Hidden Markov Models |
| INFA | Invariant Factorization |
| LTS | Learning Time-Series Shapelets |
| LS | Logical Shapelets |
| MCR | Misclassification Rate |
| MDS | Multi-dimensional Scaling |
| MF | Matrix Factorization |
| MLS | Moving Least Squares |
| NN | Nearest Neighbors Classifier |
| PAA | Piecewise Aggregate Approximation |
| PCA | Principal Component Analysis |
| SAX | Symbolic Aggregate Approximation |
| SD | Scalable Shapelet Discovery |
| SMF | Supervised Matrix Factorization |
| SVM | Support Vector Machines |
| TSC | Time-series classification |
| VSV | Virtual Support Vectors |

## LIST OF TABLES

# Chapter 1

# Introduction

## Contents



## 1.1 Motivation

Time-series data are ordered sequences of real values and *arguably* constitute the most important type of data in the world. Virtually all the records in any industrial context have a time-stamp and naturally represent time-series measurements. The concrete applications involving time-series data are abundant, ranging from physiological sensors, astronomical light intensities, up to financial and economical recordings.

The study of time series gave birth to a series of data mining challenges. To mention a few, searching is a task aiming at finding similar occurrences of a query

pattern. In order to speed-up the searching time, researchers emphasized the need for time-series indexing methods [42]. Other challenging problems demand finding meaningful time-series similarity measures [42], as well as clustering the series. Yet, another historic problem with roots in econometrics necessitates the prediction of future values of a time-series [49]. In this thesis we deal with none of the above, instead our focus lies solely on *time-series classification* (TSC). Classification refers to the process of learning from expert-labeled time-series data, in order to automatically predict the labels of future time series. For instance, let us assume we possess a set of recorded electrocardiogram time series, some belonging to healthy patients and others to patients with heart complications. In this example, time-series classification would learn from the historic electrocardiogram series that include the label "healthy/non-healthy" provided by an expert cardiologist. The ultimate aim is to be able to accurately predict the label of a future electrocardiogram series, without the need of a cardiologist. For more details, Chapter 2 includes a formal description of the problem definition.

Time-series classification differs from the standard supervised learning problems, which have a fixed set of attributes. Stated otherwise, treating a series having $Q$ points as a $Q$-dimensional feature vector yields sub-optimal prediction results [53]. Time-series patterns are often shifted in time, have different scales, contain arbitrarily repeating patterns and exhibit local distortions/noise. In other cases, the differences among classes are attributed to small local segments, rather than the global structure. For those reasons, values corresponding to a particular time-stamp have different semantics on different time-series. We call this phenomena as **intra-class variations**. The lion's share of this thesis is composed of presenting new methods that can accurately classify time-series instances, by handling variations. While intra-class variations are not a strong characteristics of time-series data, they are present in other types of data as well, such as audio or image.

## 1.2 Time Series have Intra-class Variations

Figure 1.1 illustrates one type of intra-class variations. Instances from the TwoLeadECG dataset are plotted using the Multidimensional Scaling (MDS) display technique [13], by using the Euclidean distance between each pair of series. As can be seen in Figure 1.1.a, all the time series belong to two categories/classes, orange (denoted by A) and green (denoted by B). It is interesting to point out that several instances from any of the classes are closer to instances of the other class, rather than instances of the same class. In order to emphasize this point, we

highlighted four instances $A_1, A_2, B_1, B_2$. Even though they belong to different classes, $A_1$ is closer to $B_1$ and $A_2$ is closer to $B_2$.

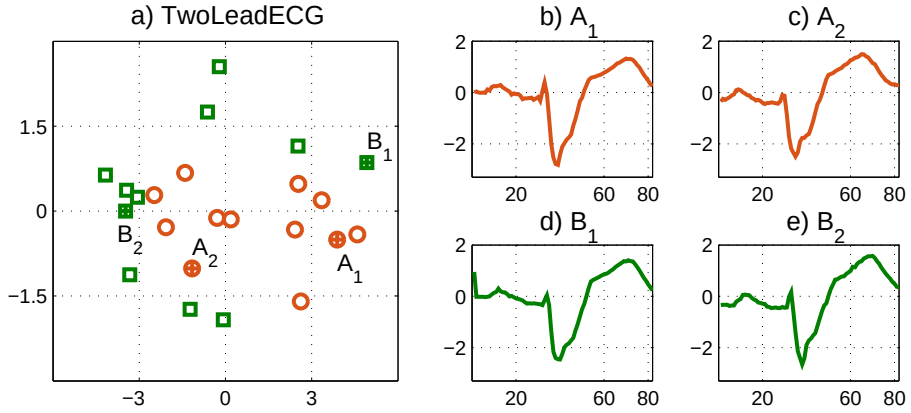



Figure 1.1: a) MDS of the Euclidean distances among series from the TwoLeadECG dataset; Instances belong to two classes: class A shown in orange and class B in green, b-e) Time series belonging to the points selected in a).

The TwoLeadECG dataset, shown in Figure 1.1, includes electrocardiogram recordings, while the classes represent measurements from "inpatients" (who require admission to the hospital) and "outpatients" (whose treatment can be performed outside the hospital, i.e. judged less critical) [67, 85]. The time-series of both types of patients look very similar in plots Figure 1.1.b-e, a behavior that is caused by the similar working mechanism of the hearts in different people. Whilst indistinguishable by the visual inspection of a layman, there is a reason why those series belong to different classes and only cardiologists can explain the underlying causes. From the perspective of this thesis, we are interested in mining the data and explore hidden causes of class differences. In that way, our algorithms that can predict the labels of electrocardiograms automatically, without the need of a cardiologist.

Figure 1.1 had the objective to illustrate just one type of intra-class variation. In the presented example, the differences among classes did not lie within the overall structure of the series, but on minor sub-sequences. Not surprisingly, a method (detailed in Chapter 6) which aims at learning discriminative series patterns can achieve a classification accuracy of 99.74% on this dataset [54, 55]. Further discussions on intra-class variations are covered in Chapters 2 and 4.

## 1.3 Contribution: Invariant Features

The existence of intra-class variations in time-series datasets limits the usage of off-the-shelf classifiers, such as Support Vector Machines (SVMs) [32]. Treating time-series instances as feature vectors suffers from a detrimental deficiency. Unfortunately, the feature vector representation completely ignores the order of series values, i.e. for a standard classifier the order of attributes is irrelevant. As a result, classifiers that excel in general tasks, such as SVMs, fail to impress in the domain of time-series (Chapter 4). Another drawback of directly using time-series values feature is that series have different lengths, while most classifiers operate under the assumption that the number of features across instances are fixed.

Efficient classification of time-series data requires the extraction of features that maximize the prediction accuracy. Since using the original values cannot address the intra-class variations (Chapter 4), we need to extract a new set of useful predictors from time series. Those features should include the richness of variations, and as a result provide a better discrimination/separation of the target variable. In this thesis we provide a series of approaches (listed below) that extract features which significantly improve the classification of time series, compared to the existing state-of-the-art.

### 1.3.1 Training Set Augmentation (Chapter 4)

As mentioned previously, standard classifiers which rely on the original series values, suffer from the inability to capture intra-class variations. In other words, due to those large variations instances from one class are located very close, in Euclidean distance terms, to instances of other classes. Such specimen instances are caused due to patterns being shifted in time or locally distorted. Nevertheless, the instances exhibiting intra-class variations are not noise as any properly-regularized classifier would consider. The first attempt to heal this deficiency is to add different types of intra-class variations as new training instances. Chapter 4 will cover in depth a novel method that adds new variations of existing instances to the training set. The technique presented therein operates over the support vector instances of SVMs and creates variations of support vectors using series transformations. The newly created series are added to the training set and the classifier is re-learned. Due to the addition of instance variations, the hyperplane of the SVMs is altered to include the regions where intra-class variations are located. As a result, the prediction accuracy of standard SVMs can be significantly improved.

### 1.3.2 Time-series Factorization (Chapter 5)

The other approach, proposed in Chapter 5 of this thesis, extracts invariant features by factorizing the time series. The presented approach elegantly addresses one of the most prominent types of intra-class variations "the shifts in time". With respect to accurate classification it is often more important to know *if* a pattern is present in a series, rather than *where exactly* it occurs. In that context, we developed a special factorization approach for time-series, which decomposes each local segment as a convolution of a set of learned patterns and membership weights. The new representation are the sums of the membership weights to the learned patterns, where high weights denote presences of patterns independent of their exact locations. Therefore, our method is able to extract features that are invariant to the shifts of patterns within time-series. In order to learn the factorization, we applied a coordinate descent optimization which in turns updates the latent patterns and the weights of segments to those patterns. One of the advantages of time-series factorization is the ability to trivially operate in both inductive or transductive modes. An SVM model learned using the factorized representation achieves significantly better prediction accuracy compared to both a standard SVM and also several state-of-the-art TSC methods.

### 1.3.3 Learning Time-Series Shapelets (Chapter 6)

The factorization approach of Chapter 5 is an important step forwards in tackling intra-class variations, especially given its transductive ability. However, such a factorization approach is still an unsupervised technique, and consequently has a weakness too. Features that are extracted using unsupervised methods have no guarantee of improving the classification accuracy, because they are not constructed based on the target variable of the training set. This argument does not mean that all methods that extract unsupervised features are not useful, because in several domains unsupervised features yield competitive results (e.g. bag-of-words models). Nevertheless, from a theoretic perspective, features that are learned over the target labels of the training set have a higher chance of being accurate on the testing instances. In the realm of time-series, there exists a technique that extracts invariant features in a supervised manner. Those features are called *shapelets* and currently represent the most accurate predictors in classifying time-series. Chapter 6 explains a novel method which can learn a set of patterns (called shapelets) whose occurrence can accurately classify series. In contrast to

existing shapelet discoveries, we propose a method that can learn shapelets directly over the regularized training loss. As a result, our method yields impressive results that are the current state-of-the-art in time-series classification.

### 1.3.4 Fast Shapelets Discovery (Chapter 7)

The novel technique of Chapter 6 achieves impressive classification accuracies over a large collection of time-series datasets. However, the gradient calculations of its learning algorithm iterate over all the series segments and are computationally demanding. Therefore, if time is critical, new approaches that can extract discriminative series features are needed. In Chapter 7 we present a method that greatly reduces the time required to discover shapelets. The proposed trick prunes the time-series segments by checking the similarity among candidates. We randomly iterate over the segments of a dataset and accept candidates only if they are not similar to previously accepted candidates. In case candidates are deemed novel, then we check whether their features improve the classification accuracy of the already accepted candidates. As a result over 99% of segments can be pruned. The proposed method can classify MB-scale datasets in a matter of seconds, and a GB-scale multivariate time-series dataset in a matter of minutes.

### 1.3.5 Repetitive Time Series (Chapter 8)

The methods presented in the sections above cover techniques which tackle intra-class variations on very general time-series datasets. However, there are datasets which exhibit special properties and differ from the standard types of time series. Chapter 8 exposes a different method that target datasets having repetitive patterns.

Time series from a vast range of domains, such as physiological sensors, contain repetitive patterns which occur at arbitrary locations and frequencies. Such data types require methods that aggregate counts of patterns and make a natural habitat for bag-of-words models. Chapter 8 presents a new technique that collects histograms of local patterns for repetitive time-series. Our approach approximates local segments via polynomial functions and then converts the polynomials to symbolic strings. Occurrence counts of those symbolic words provide useful predictors for classification purposes. In addition, our method has a linear runtime in terms of the data size and scales to large datasets. Empirical results over real-life datasets evidences the success of the presented approach in comparison to other classifiers.

### 1.3.6 The Types of Data The Thesis Focuses On

In this thesis we focus primarily on univariate time-series data, which are ordered measurements of values across one-dimension. Therefore, our focus delineates from audio data and/or images, which can be perceived as multi-dimensional signal data. Nevertheless, we address thoroughly the task of classifying time-series using a wide range of univariate real-life datasets, to be specified in Section 2.11.

## 1.4 Published Works

Every chapter of this thesis is based on a separate work, as enlisted below.

- **Chapter 4** is based on:

  *Josif Grabocka*, Alexandros Nanopoulos, Lars Schmidt-Thieme (2012): **Invariant Time-Series Classification**, in Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML'12), LNCS, Volume 7524, 2012, pp 725-740

- **Chapter 5** is based on:

  *Josif Grabocka*, Lars Schmidt-Thieme (2014): **Invariant Time-Series Factorization**, Journal of Data Mining and Knowledge Discovery, Volume 28, Issue 5-6, pp 1455-1479.

- **Chapter 6** is based on:

  *Josif Grabocka*, Nicolas Schilling, Martin Wistuba, Lars Schmidt-Thieme (2014): **Learning Time-Series Shapelets**, in Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp 392-401.

- **Chapter 7** is based on:

  *Josif Grabocka*, Martin Wistuba, Lars Schmidt-Thieme (2015): **Scalable Discovery of Time-Series Shapelets**, CoRR, abs/1503.03238, (currently under review)

- **Chapter 8** is based on:

  *Josif Grabocka*, Martin Wistuba, Lars Schmidt-Thieme (2014): **Scalable Classification of Repetitive Time Series Through Frequencies of Local Polynomials**, IEEE Transactions on Knowledge and Data Engineering, Volume 27, Issue 6, pp 1683-1695.

Furthermore, Chapter 5 is motivated by my related research on data factorizations:

- *Josif Grabocka*, Lars Schmidt-Thieme (2014): **Learning Through Nonlinearly Supervised Dimensionality Reduction**, Springer Transactions on Large-Scale Data- and Knowledge-Centered Systems, LNCS, Volume 8970, pp 74-96.
- *Josif Grabocka*, Erind Bedalli, Lars Schmidt-Thieme (2014): **Supervised Nonlinear Factorizations Excel In Semi-supervised Regression**, in Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2014, LNCS, Volume 8443, 2014, pp 188-199.
- *Josif Grabocka*, Lucas Drumond, Lars Schmidt-Thieme (2013): **Supervised Dimensionality Reduction Via Nonlinear Target Estimation**, in Proceedings of the 15th International Conference on Data Warehousing and Knowledge Discovery, DaWaK 2013, LNCS, Volume 8057, pp 172-183.
- *Josif Grabocka*, Erind Bedalli, Lars Schmidt-Thieme (2012): **Efficient Classification of Long Time Series**, in Proceedings of ICT Innovations Conference 2012, Advances in Intelligent Systems and Computing, Volume 207, pp 47-57.

Overall, my contribution in terms of published works sums up to 9 papers; 8 of which published in peer-reviewed journals and conference proceedings, while 1 is currently under review. In this thesis I often use the word "WE" as a sign of courtesy towards the co-authors of my papers. However, I would like to point out that I am the first-author and the major contributor of all the aforementioned papers, which constitute the backbone of this thesis.

# Chapter 2

# Problem Definition

## Contents



## 2.1 Chapter Notations

This chapter will adhere to the notational conventions of Table 2.1.

| Symbol | Explanation |
|---|---|
| $D_{\mathrm{P}} \in \mathbb{N}$ | Number of model parameters |
| $\theta \in \mathbb{R}^{D_P}$ | Model parameters |
| $D_{\mathrm{HP}} \in \mathbb{N}$ | Number of model Hyper-parameters |
| $\gamma \in \mathbb{R}^{D_{HP}}$ | Model hyper-parameters |
| $\mathcal{M}(T_{:}, \theta, \gamma) : \left(\mathbb{R}^{Q} \times \mathbb{R}^{D^{\mathrm{P}}} \times \mathbb{R}^{D^{\mathrm{HP}}}\right) \rightarrow \mathbb{R}^{C}$ | Prediction Model |

Table 2.1: Notational conventions of Chapter 2

## 2.2 Description of the Task

In this chapter the problem of time-series classification will be formally stated. While the presented information will cover the problem description, it is not an introduction to the field of machine learning. In order to fully understand the forthcoming concepts, one needs to have acquired a general knowledge of machine learning, which is commonly found in relevant textbooks [15, 88].

The time-series classification task aims at predicting the label of a time-series instance by making usage of a recorded set of instances. The scenario involves an expert labeling the historic measurements, while the ultimate objective is to make use of the past expert-labeled recordings, in order to predict the label of future instances without the need of the expert. This process is called supervised learning and is the key focus of this thesis.

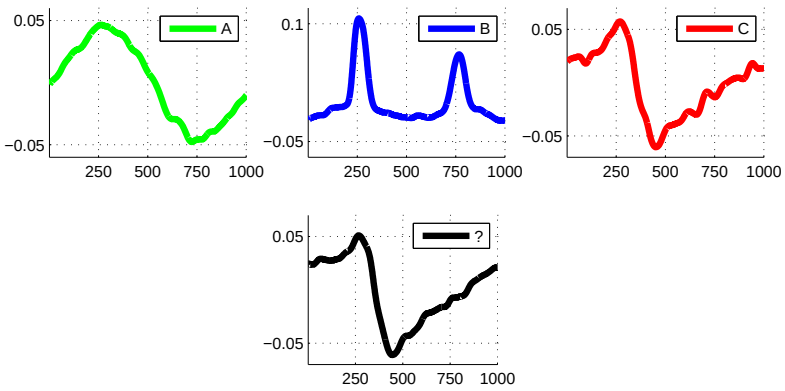



Figure 2.1: An illustration of the time-series classification task using instances from the *StarLightCurves* dataset, with the objective of predicting the category of a new series (shown in black).

Let us explain the concept with the assistance of a real-life example. Figure 2.1 represents photometric time series data from the astrophysics domain, where X-axis is the time stamp and the Y-axis the arriving light intensity emitted by a star [97]. An expert astronomer has examined the properties of the stars and provided a category for each of the three starts in the upper plots, whose time series are recorded. Examples of star categories are *Cepheid*, *Eclipsing Binaries* and *RR Lyrae* [97]. Therefore we can collect a set of star time series, denoted $T$, and a set of labels/categories denoted $Y$. In this context, the classification task is to learn from the recorded time-series and the expert-provided labels ($T$ and $Y$), in order to automatically predict the categories/labels of future recordings without the need of an astrologist. For example, in Figure 2.1 we provide three example series that an astrologist believes to belong to three different star categories, shortly named $\{A, B, C\}$. Given a newly recorded star light series (in black), can we predict its category by looking into the previously recorded and expert-labeled measurements? As can be trivially deduced, it is possible to foresee that the newly recorded series in black is similar to the red series and belongs to category C. This thesis focuses on describing innovative methods that accurately classify time series having diverse properties and arising in various domains.

Rephrasing the problem definition, assume we are given a training set of instances, denoted as $T^{\text{Train}} \in \mathbb{R}^{N^{\text{Train}} \times Q}$, and their expert-labeled targets $Y^{\text{Train}} \in \mathbb{N}^{N^{\text{Train}}}$. The problem relies on accurately predicting the unknown target variable $Y^{\text{Test}} \in \mathbb{N}^{N^{\text{Test}}}$ of a given set of testing series $T^{\text{Test}} \in \mathbb{R}^{N^{\text{Test}} \times Q}$. Generally we assume that the testing set is generated using the same probability distribution as the training set.

## 2.3 Setup and Data Description

A classification method is composed of a list of ingredients. First of all there is a prediction model (denoted by $\mathcal{M}$) that takes as inputs (i) a time-series, (ii) a set of parameters and (iii) hyper-parameters, and outputs a prediction for the label of the inputted time-series. The success of the prediction model is measured using a loss function (denoted by $\mathcal{L}$), that measures how (in)accurately can the model and its parameters predict the historic data for which expert labels are present. In that way, the expert labels are considered to be the *ground truth* and serve as a test bed to both measure the success of a model and modify its parameters in order to match the experts' predictions. A prediction model has both a set of parameters (denoted $\theta$) and a set of hyper-parameters (denoted $\gamma$). Roughly

speaking, parameters describe the structure of a probabilistic prediction model, while hyper-parameters are prior beliefs on the distribution of parameters [15, 88]. Those aspects will be covered in depth in the forthcoming parts of this section.

As previously mentioned, the data arrives in terms of a set of $N$-many recorded time series of length $Q \in \mathbb{N}$ and denoted as $T \in \mathbb{R}^{N \times Q}$. In addition, the expert-decided labels of those time-series are denoted as $Y \in \mathbb{N}^N$. However, this dataset has to be divided into three disjoint sets that are named $\{\text{Train}, \text{Validation}, \text{Test}\}$. The training set serves to learn the optimal model parameters, given a set of hyper-parameters. It contains of $N^{\text{Train}}$ series, denoted as $T^{\text{Train}} \in \mathbb{R}^{N^{\text{Train}} \times Q}$, and labels $Y^{\text{Train}} \in \mathbb{N}^{N^{\text{Train}}}$. On the other hand, the validation set having $N^{\text{Validation}}$ series helps learning the best values for the hyper-parameters and is denoted by $T^{\text{Validation}} \in \mathbb{R}^{N^{\text{Validation}} \times Q}$ and labels $Y^{\text{Validation}} \in \mathbb{N}^{N^{\text{Validation}}}$. Finally the test set having $N^{\text{Test}}$ series is used to compute the accuracy of the model with its best parameters and hyper-parameters and is denoted by $T^{\text{Test}} \in \mathbb{R}^{N^{\text{Test}} \times Q}$ and labels $Y^{\text{Test}} \in \mathbb{N}^{N^{\text{Test}}}$. It is worth noting that $N = N^{\text{Train}} + N^{\text{Validation}} + N^{\text{Test}}$, $T = \left\{T^{\text{Train}} \cup T^{\text{Validation}} \cup T^{\text{Test}}\right\}$ and $Y = \left\{Y^{\text{Train}} \cup Y^{\text{Validation}} \cup Y^{\text{Test}}\right\}$. **Furthermore, it is important to realize that the scope of the symbols including the terms 'train, validation and test' are valid only within this chapter**. In the forthcoming chapters, we are going to improve the readability of symbols by relaxing the notation and assuming $T, Y$ refer only to the training set.

## 2.4 Prediction Model for Classification

A prediction model, also known as a classifier, is a function that takes the $i$-th series instance $T_i \in \mathbb{R}^Q$ as an input, together with a set of $D_{\text{P}}$-many parameters (denoted $\theta \in \mathbb{R}^{D_{\text{P}}}$) and a set of $D_{\text{HP}}$-many hyper-parameters (denoted $\gamma \in \mathbb{R}^{D_{\text{HP}}}$). The utilitarian mission of a prediction model is to output estimated labels denoted as $\hat{Y}_i \in \mathbb{R}^C$. Technically speaking, the prediction model will output an estimate for each of the $C$ classes. The formalism of a prediction model and its range is given in Equation 2.1.

$$\hat{Y}_i := \mathcal{M}(T_i, \theta, \gamma) \; : \; \left(\mathbb{R}^Q \times \mathbb{R}^{D_{\text{P}}} \times \mathbb{R}^{D_{\text{HP}}}\right) \to \mathbb{R}^C, \;\; i = 1, \ldots, N \tag{2.1}$$

## 2.5 Loss Function

A loss function (denoted by $\mathcal{L}$) measures how accurate are the label estimates $\hat{Y}$ of a predictions model, when compared to the ground truth labels $Y$ of the experts. The formalization of a loss function, also known as the *empirical risk* is provided in Equation 2.2.

$$\mathcal{L}\left(Y_i, \hat{Y}_i\right) = \mathcal{L}\left(Y_i, \mathcal{M}\left(T_i, \theta, \gamma\right)\right), \quad i = 1, \ldots, N \tag{2.2}$$

When dealing with a classification task, the target labels are a categorical/nominal variable. In the simplest case of two categories, the problem is referred to as a binary classification. There exist a couple of standard loss functions for binary classification, such as the Hinge loss and the Logistic loss, which are shown in Equation 2.3.

$$\mathcal{L}\left(Y_i, \hat{Y}_i\right) = \begin{cases} \max\left(0, 1 - Y_i \hat{Y}_i\right) & \text{Hinge, for } Y_i \in \{-1, 1\} \\ -Y_i \ln\left(\sigma(\hat{Y}_i)\right) - (1 - Y_i) \ln\left(1 - \sigma(\hat{Y}_i)\right) & \text{Logistic, for } Y_i \in \{0, 1\} \end{cases} \tag{2.3}$$

For multi-categorical targets, a standard approach is to use a series of one-vs-all prediction models. The procedure involves creating as many prediction models as there are categories in the target variable. For each particular class, there is one model for a binary target distinction by merging instances of all other classes as a new class.

## 2.6 Best Model Parameters

A statistical model with a set of parameters is called a *parametric model*. Assuming we face a model $\mathcal{M}$ with parameters $\theta^* \in \mathbb{R}^{D_\mathrm{P}}$, the main challenge is to find the best values of a model's parameter. The correct approach in terms of learning the parameters is to minimize a regularized loss function, as the ones in the previous section.

$$\theta_\gamma := \underset{\theta^* \in \mathbb{R}^{D_\mathrm{P}}}{\operatorname{argmin}} \sum_{i=1}^{N^{\mathrm{Train}}} \mathcal{L}(Y_i^{\mathrm{Train}}, \mathcal{M}(T_i^{\mathrm{Train}}, \theta^*, \gamma)) + \mathcal{R}\left(\theta^*, \gamma\right) \tag{2.4}$$

The optimization relies on minimizing the regularized *training* loss, i.e. by predicting close label values as the ground truth $Y$. Given fixed hyper-parameter

values $\gamma$, the best model parameters for are denoted as $\theta_\gamma$. Those optimal parameters are learned among a pool of parameter values $\theta^* \in \mathbb{R}^{D_\text{P}}$, and are set to the candidate parameters that achieve the lowest possible value of Equation 2.4. The regularization term $\mathcal{R}(\theta^*, \gamma)$ imposes constraints on the parameters in order to avoid over-fitting to the training set [15, 88]. Typical regularization mechanisms are the Tikhonov (a.k.a $L_2$) and $L_1$ regularization penalty terms [15].

## 2.7 Learning Method

A learning method or otherwise known as a learning algorithm, is a sequence of operations that finds the best parameters $\theta$ of a model $\mathcal{M}$, by minimizing the training loss of Equation 2.4.

**Algorithm 1**: **LearnModelParameters**: Learn the best parameters of a model

**Data**: Time-series data $T^\text{Train} \in \mathbb{R}^{N^\text{Train} \times Q*}$, Labels $Y^\text{Train} \in \mathbb{N}^\text{Train}$, Hyper-parameters $\gamma \in \mathbb{R}^{D_\text{HP}}$, Learning rate $\eta \in \mathbb{R}$, Number of iterations $I \in \mathbb{N}$

**Result**: Learned model parameters $\theta \in \mathbb{R}^{D_\text{P}}$

1 Initialize model parameters: $\theta \sim \mathcal{N}(-\epsilon, +\epsilon)^{D_\text{P}}$;

2 **for** $1, \ldots, I$ **do**

3 $\quad \theta \leftarrow \theta - \eta \left( \left( \sum_{i=1}^{N^\text{Train}} \frac{\partial \mathcal{L}\left(Y_i^\text{Train}, \mathcal{M}(T_i^\text{Train}, \theta, \gamma)\right)}{\partial \theta} \right) + \frac{\partial \mathcal{R}(\theta, \gamma)}{\partial \theta} \right)$;

4 **end**

5 **return** $\theta$

A depiction of one of the most popular optimization methods called *gradient descent* is described in Algorithm 1. First of all, the parameter values are randomly initialized to some Gaussian noise (line 1). Afterwards, the parameters $\theta^\mathcal{M}$ are updated in the negative direction of the first gradient (line 3), in order to minimize the regularized loss function of Equation 2.4. The updates are carried on by a magnitude specified through a hyper-parameter known as the learning rate (denoted by $\eta$), while the learning procedure is repeated according to a hyper-parameter known as the number of iterations (denoted by $I$). It is worth noting that the output of Algorithm 1 is the local optimum solution of Equation 2.4, stated otherwise $\theta_\gamma := \text{LearnModelParameters}\left(T^\text{Train}, Y^\text{Train}, \gamma, \eta, I\right)$. Whilst formally $\eta$ and $I$ can be merged inside the hyper-parameters vector $\gamma$, here they are shown separately for the ease of understanding the learning method.

## 2.8 Model Hyper-parameters

From a Bayesian inference perspective hyper-parameters are parameters of the prior distribution of the previously described parameters $\theta$, i.e. parameters of parameters. In a more general sense, hyper-parameters refer to a variety of model settings, such as regularization, learning rate, number of latent dimensions (for latent models), sliding window sizes for time-series segmentation. Those settings should be learned on an independent dataset, often known as the validation set [88].

$$\begin{aligned} \gamma^{\mathcal{M}} \quad &:= \quad \underset{\gamma^* \in \mathbb{R}^{D_{\text{HP}}}}{\text{argmin}} \quad \sum_{i=1}^{N^{\text{Validation}}} \mathcal{L}\left(Y_i^{\text{Validation}},\ \mathcal{M}\left(T_i^{\text{Validation}}, \theta_{\gamma^*}, \gamma^*\right)\right) \qquad (2.5) \\ \text{with} \quad \theta_{\gamma^*} &:= \underset{\theta^* \in \mathbb{R}^K}{\text{argmin}} \quad \sum_{i=1}^{N^{\text{Train}}} \mathcal{L}(Y_i^{\text{Train}},\ \mathcal{M}(T_i^{\text{Train}}, \theta^*, \gamma^*)) \ +\ \mathcal{R}\left(\theta^*, \gamma^*\right) \end{aligned}$$

Equation 2.5 formalizes the process of finding the best hyper-parameters values $\gamma^{\mathcal{M}} \in \mathbb{R}^{D_{\text{HP}}}$ for a model $\mathcal{M}$. The equation defines the best hyper-parameters values to be those which achieve the smallest loss value on the validation set, using the best model parameters learned on the training set. On the other hand, it is worth noting that the best parameters $\theta_{\gamma^*}$ are computed per each hyper-parameter configuration $\gamma^*$. A standard procedure in learning the hyper-parameters is to grid-search the values of $\gamma^{\mathcal{M}}$ in a grid of optional values. For every hyper-parameter combination from the grid, first the parameters are learned on train and then the loss on validation is measured, while the final winning combination is the one achieving the smallest validation loss.

## 2.9 Model Accuracy

Once the parameters and the hyper-parameters of the model are optimized, we need to report the accuracy of a prediction model on the test dataset. Equation 2.6 formalizes the accuracy of a model $\mathcal{M}$ using its best hyper-parameters $\gamma^{\mathcal{M}}$ and parameters $\theta_{\gamma^{\mathcal{M}}}$.

$$\text{Accuracy}^{\mathcal{M}} \quad := \quad \frac{1}{N^{\text{Test}}} \sum_{i=1}^{N^{\text{Test}}} \mathcal{L}^{\text{True}}\left(Y_i^{\text{Test}},\ \mathcal{M}\left(T_i^{\text{Test}}, \theta_{\gamma^{\mathcal{M}}}, \gamma^{\mathcal{M}}\right)\right) \qquad (2.6)$$

The accuracy is the *average* **true** loss function over the test dataset series. The reason for *averaging* is based on the need for error (or accuracy) rates as a percentage over the total number of instances. However we do not need to average the training and validation losses, because the term $\frac{1}{N^*}$ is a constant term that does not influence the optimal values of the parameters $\theta$ or the hyper-parameters $\gamma$.

$$\mathcal{L}^{\text{True}}(Y_i^{\text{Test}}, \mathcal{M}\left(T_i^{\text{Test}}, \theta_{\gamma^{\mathcal{M}}}, \gamma^{\mathcal{M}}\right)) = \begin{cases} 1, & Y_i^{\text{Test}} = \mathcal{M}\left(T_i^{\text{Test}}, \theta_{\gamma^{\mathcal{M}}}, \gamma^{\mathcal{M}}\right) \\ 0, & Y_i^{\text{Test}} \neq \mathcal{M}\left(T_i^{\text{Test}}, \theta_{\gamma^{\mathcal{M}}}, \gamma^{\mathcal{M}}\right) \end{cases} \quad (2.7)$$

The loss of Equation 2.6 is denoted as the **true** classification loss, in order to differentiate it from the losses of Equations 2.4 and 2.5. For instance, the classification rate of Equation 2.7 is the true loss that we care when reporting results, however we cannot optimize the parameters on the classification rate because it is not differentiable. Therefore, during parameter learning we can only differentiable proxies such as the hinge loss or logistic losses of Equation 2.3, that smoothly approximate the true loss of Equation 2.7. Sometimes, practitioners prefer the error rate, which is the average number of misclassifications, i.e. $\text{Error}^{\mathcal{M}} = 1 - \text{Accuracy}^{\mathcal{M}}$.

## 2.10 Cross Validation

When describing the data setup, the time series were divided into three disjoint sets, namely Train, Validation and Test. However, still one needs to decide how to divide the instances into those three sets. In case of a random split, there exists the risk that the prediction model is contaminated by the influence of the random division of instances. In order to measure the prediction variance induced by the splits, a procedure known as *cross-validation* is followed. As a first step, the data is split into a number of folds and then multiple prediction models are applied with each fold being the test instance once. More concretely, a split scenario for the 5-folds cross-validation case is shown below:

$$
\begin{aligned}
1^{\text{st}} \text{ fold}: \quad & T^{\text{Train}} := \left\{T^1 \cup T^2 \cup T^3\right\},\ T^{\text{Validation}} := \left\{T^4\right\},\ T^{\text{Test}} := \left\{T^5\right\} \\
2^{\text{nd}} \text{ fold}: \quad & T^{\text{Train}} := \left\{T^5 \cup T^1 \cup T^2\right\},\ T^{\text{Validation}} := \left\{T^3\right\},\ T^{\text{Test}} := \left\{T^4\right\} \\
3^{\text{rd}} \text{ fold}: \quad & T^{\text{Train}} := \left\{T^4 \cup T^5 \cup T^1\right\},\ T^{\text{Validation}} := \left\{T^2\right\},\ T^{\text{Test}} := \left\{T^3\right\} \\
4^{\text{th}} \text{ fold}: \quad & T^{\text{Train}} := \left\{T^3 \cup T^4 \cup T^5\right\},\ T^{\text{Validation}} := \left\{T^1\right\},\ T^{\text{Test}} := \left\{T^2\right\} \\
5^{\text{th}} \text{ fold}: \quad & T^{\text{Train}} := \left\{T^2 \cup T^3 \cup T^4\right\},\ T^{\text{Validation}} := \left\{T^5\right\},\ T^{\text{Test}} := \left\{T^1\right\}
\end{aligned}
$$

In the illustrated split, the data is divided into five non-empty disjoint sets, such that $T := T^1 \cup T^2 \cup T^3 \cup T^4 \cup T^5$. The same train, validation and test divisions are also done for the labels, however are omitted for brevity. The learning process will be repeated five times, once per each fold, and there would be five different accuracy scores, hence allowing to have a mean and a variance (confidence) for the accuracy.

## 2.11 Description of Used Datasets

The scope of this thesis is mostly on univariate time-series datasets. Most of the chapters will use the supervised datasets retrieved from the UCR collection of datasets [67]. Such a collection includes a vast range of datasets, 45 at the moment of retrieval, belonging to diverse domains such as energy consumption, human actions and health-related measurements. The time series across datasets have diverse lengths, ranging from 24 to 1882 measurements, and diverse number of instances, ranging from 56 to 9236 labeled instances. The target variable of the datasets is multi-categorical, ranging from 2 up 50 to different classes. Further details of the datasets, including access permission, can be found at the collections' website [67]. Throughout this thesis we are going to refer to these datasets as the *UCR collection*. The detailed description of the datasets will be provided together with each particular experimental context in the forthcoming chapters. In addition, Chapter 8 focuses on scalable classification of highly repetitive time-series, therefore we used a set of highly repetitive univariate datasets, to be detailed in Section 8.5.2.

## 2.12 What makes time-series classification non-trivial?

Before moving forwards with the explanation of the related literature in time-series classification, it is important to understand the difficulty of the task. For instance, a devil's advocate might question whether there is a trivial solution to classify time-series, for instance by considering each time-measurement as an attribute. In other words, each series having $Q$ points could be treated as an instance having $Q$ attributes. Could we use standard off-the-shelf models to classify time series, such as Support Vector Machines [32], or the Nearest Neighbor with Euclidean distance similarity metric?

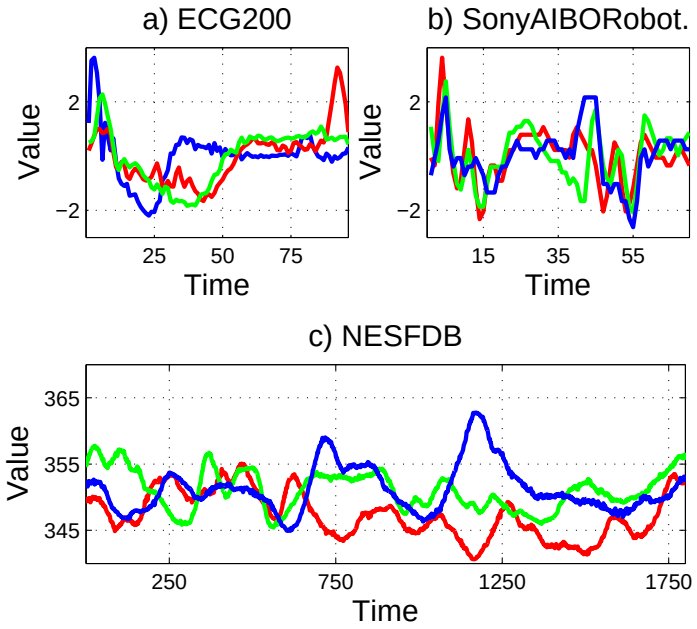



Figure 2.2: An illustration of intra-class variations in three time-series datasets; Sub-plots a) depicts instances from the ECG200 dataset, b) instances from the SonyAIBORobotSurface dataset and c) series from the NESFDB dataset. All the three time series in each plot belong to the same class within the dataset.

Unfortunately, classifying time-series instance is a complex and challenging task because time series within one class exhibit intra-class variations. For instance Figure 2.2 illustrates instances of the same class from three different time-series datasets. Figure 2.2.a indicates that series within the same class have local shifts. In the red and green series there is an increasing trend around time 50,

however such a pattern occurs earlier for the blue series. This pattern-shifting phenomenon impedes the usage of standard classifiers that treat each time stamp as an attribute, because the same time-stamps on different series indicate different positions within the series patterns. In other datasets, even though time-series of the same class share local structures, they might contain local distortions and noise. Figure 2.2.b shows that while all series have segments with a common structure, they differ in multiple local segments, for instance between time-stamps 35 and 55. Such local distortions prohibit the usage of absolute time-stams as attributes, because local distortions represent noisy model predictors. The last illustration of Figure 2.2.c shows three series of the same class from the NESFDB dataset [48], which represents postural measurements (walking styles) of young vs old people. As can be seen, time-series of walking styles contain repetitions of local patterns (steps), rather than a single global pattern. The special challenges of time-series data that are described in this section should serve as an initial motivation for the reader. Due to the existence of intra-class variations, off-the-shelf classifiers like the SVms perform non-optimally in the realm of time-series classification [53]. Deeper analysis on intra-class variations and further challenging time-series aspects are covered in the forthcoming chapters of this thesis.

## 2.13 Difference to Time-series Forecasting

There is another famous field of research that is focused on time series forecasting (a.k.a. time series prediction). The domain of this research stream focuses on predicting the future values of a series, given a history of measured points [49]. For instance, assuming a long time series of $Q$ points denoted as $T = (T_1, T_2, \ldots, T_Q)$, the task is to find the next points $(T_{Q+1}, T_{Q+2}, \ldots)$. Research on time-series forecasting is quite old (ca. 35 years of research) and include veteran methods such as ARIMA [111], which captures seasonal and cyclical patterns from long series by applying an iterated moving average procedure. Interested readers on classical forecasting are invited to consult with [49]. On the other hand, machine learning techniques have also been adopted to operate on the forecasting scenario [72]. Recently, there is a growing trend to predict future values of a series using other collateral series [40]. For instance, a series measuring the number of people playing video games can help predicting the current and future unemployment rate [40]. Needless to say, the task of this thesis is different to time-series forecasting, because classification involves the prediction of a globally-associated series label, instead of the next series point.

# Chapter 3

# Related Work

## Contents

## 3.1 Chapter Notations

This chapter will adhere to the notational conventions of Table 3.1.

| Symbol | Explanation |
|---|---|
| $D \in \mathbb{N}$ | Number of latent dimensions |
| $U \in \mathbb{R}^{N \times D}$ | Latent instance-specific matrix |
| $V \in \mathbb{R}^{D \times Q}$ | Latent attribute-specific matrix |
| $W \in \mathbb{R}^{D+1}$ | Classification weights vector |
| $B_U \in \mathbb{R}^{N}$ | Bias of the instance-specific matrix |

Table 3.1: Notational conventions of Chapter 3

This chapter aims at offering the reader the preliminary knowledge needed to understand the forthcoming technical chapters. As such, it covers an introduction of related work in different aspects of time-series classification, data factorization and the classifiers used in the thesis. Regarding the literature on time-series classification, one can categorize the successful related work into *i) similarity measures*, *ii) invariant classifiers*, *iii) time-series representations* or factorization, and *iv) supervised feature extractions* (shapelets). The next sections will explain the related methods of each group of approaches.

## 3.2 Time-Series Similarity Measures

The time-series community has invested considerable efforts in understanding the notion of similarity among series. Time series patterns exhibit high degrees of intra and inter class variations, which are found in forms of noisy distortions, phase delays, frequency differences and signal scalings. Therefore, accurate metrics that compute the distance among two series play a crucial role in terms of classification accuracy. Euclidean distance, commonly known as the $L_2$ distance, is a fast metric which compares the distance values of every pair of points from two series. Despite being a fast metric of linear run-time complexity, the Euclidean distance is not directly designed to detect pattern variations. A popular dissimilarity measure called Dynamic Time Warping (DTW) overcomes the deficiencies of the Euclidean distance by detecting relative time indexes belonging to similar series regions. Nearest neighbor using DTW achieves competitive classification accuracy results and was historically regarded as a strong baseline [39], until was recently outperformed by the "Learning Shapelets" method [54]. Even

though DTW is slow in its original formulation with a quadratic run-time complexity, still recent techniques involving early pruning and lower bounding have adopted DTW for fast large scale search [94]. DTW is regarded as the most popular of those approaches [69], due to its success in overcoming deficiencies of the $L_2$ norm distance by aligning the time indexes of two series instances. In terms of classification, this similarity measure is typically plugged into a nearest neighbor classifier. In addition, DTW has been speed up using lower boundary heuristics [94].

Various techniques that have elaborated distance penalties for assessing the similarity between time series [26, 27], are inspired by the edit distance principle of strings which counts the number of atomic operations needed to convert a string into to another. In the context of time series the analogy is extended to the sum of necessary value changes needed for an alignment. Furthermore, the longest common subsequence of time series has also been used as an indication of similarity [112]. The motivation leading to the detection of the longest common subsequence of series, relies in the assumption that time series have a fingerprint segment which is the most determinant with respect to classification [112]. Detection of similarities in streaming time-series gave birth to methods that handle scaling and shifting in the temporal and amplitude aspects [30]. Moreover, similarities of sequential data have been measured using sparse spatial sample kernels [73]. A state of the art method called complexity-invariant distance metric (CID) introduces the total variation regularization for time-series. CID significantly improves the accuracy of the nearest neighbor classifier with Euclidean distance [9, 10].

### 3.2.1 Euclidean Distance and Dynamic Time Warping

In this section I will describe in more depth the most commonly used similarity measures for time series, the Euclidean distance and the Dynamic Time Warping [39]. Technically speaking, those two measures are dissimilarity measures (the greater the distance, the less similar). Euclidean distance (denoted as ED) simply compares the points among two series with respect to the absolute time-stamps of values. As shown in Equation 3.1, the Euclidean distance is the sum of pairwise points' distances. It is worth noting that the $q$-th point in one series is compared with the $q$-th point in the other series. Therefore, if the pattern is shifted in time, the absolute matching of time-stamps fails to detect variations. An intuition is provided in Figure 3.1.a), while the illustration details will be described in the forthcoming paragraphs.

$$\mathrm{ED}(T_i, T_j) = \sqrt{\sum_{q=1}^{Q} (T_{i,q} - T_{j,q})^2} \tag{3.1}$$

Despite the major effort spent in building accurate time series classifiers, a similarity technique called Dynamic Time Warping (DTW) has been reported to produce accurate results in combination with the Nearest Neighbor classifier [123]. DTW overcomes the drawback of other similarity measures because it can detect pattern variations, such as translations/shifts, scale and deformations. The metric builds an optimal alignment of points among two time series by dynamically constructing a progressive cost matrix. It computes the path of the minimal overall point pairs' distance [69]. Adding warping window size constraints have been reported to occasionally boost the classification [70].

Concretely, a warping path between two series $T_{i,:} = (T_{i,1}, ..., T_{i,Q})$ and $T_{j,:} = (T_{j,1}, ..., T_{j,Q})$, denoted as $\tau^{T_i,T_j}$ is defined as an alignment $\tau^{T_i,T_j} = (\tau_1^{T_i,T_j}, \tau_2^{T_i,T_j})$ between the indices of $T_{i,:}$ and $T_{j,:}$. The alignment starts and ends with extreme points,

$$\begin{aligned} P &= |\tau^{T_i,T_j}| \\ 1 = \tau^{T_i,T_j}(1)_1 &\leq ... \leq \tau^{T_i,T_j}(P)_1 = Q \\ 1 = \tau^{T_i,T_j}(1)_2 &\leq ... \leq \tau^{T_i,T_j}(P)_2 = Q \end{aligned}$$

while involving incremental alignment of adjacent pairs as:

$$\begin{pmatrix} \tau^{T_i,T_j}(t+1)_1 - \tau^{T_i,T_j}(t)_1 \\ \tau^{T_i,T_j}(t+1)_2 - \tau^{T_i,T_j}(t)_2 \end{pmatrix} \in \left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}, \quad t = 1, \ldots, P.$$

The overall distance of the points aligned by a warping path is computed as the sum of distances of each aligned pair. Such distance is called the Dynamic Time Warping distance [69].

$$\mathrm{DTW}(T_i, T_j) = \underset{\tau^{T_i,T_j}}{\operatorname{argmin}} \sum_{p=1}^{|\tau^{T_i,T_j}|} \left( T_{i,\tau^{T_i,T_j}(p)_1} - T_{j,\tau^{T_i,T_j}(p)_2} \right)^2$$

Dynamic Time Warping distance is practically computed by a dynamic algorithm, which is calculated via a cost matrix, denoted $W$. Each cell of the

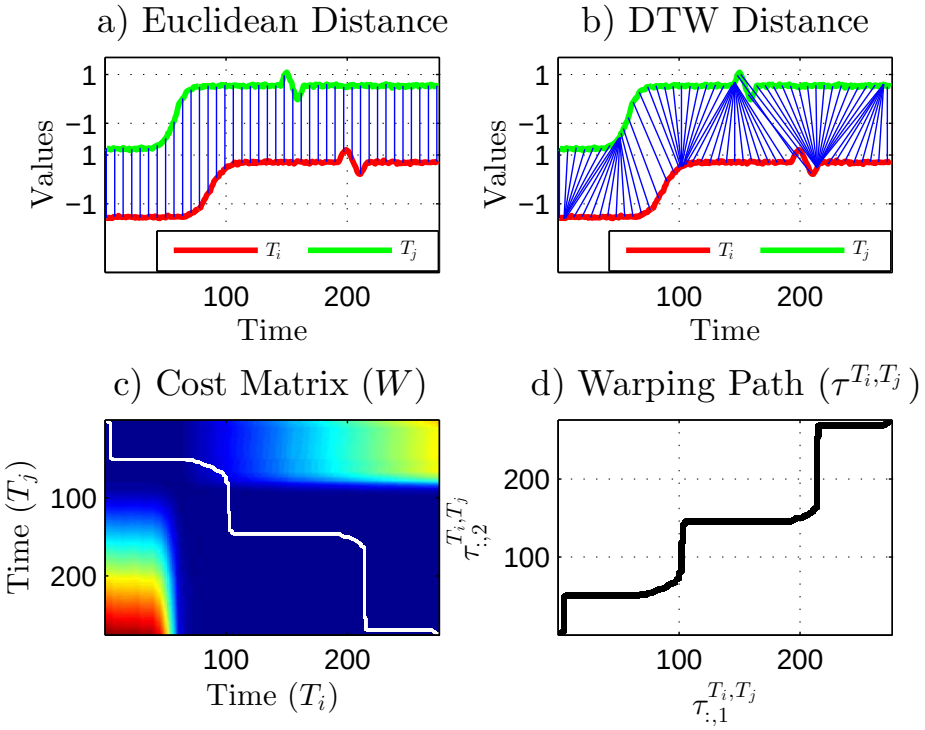



Figure 3.1: An illustration of DTW on two series from the Trace dataset, a) and b) Euclidean and DTW distances of two series $T_i$ in red and $T_j$ in green, c) Cost matrix, min values in blue and max values in red, d) Warping path alignment

cost matrix is computed as depicted in Equation 4.1. The computation of DTW according to Equation 4.1 has a runtime complexity of $\mathcal{O}(Q^2)$.

$$\begin{aligned} \text{DTW}(T_i, T_j) &= \mathbf{W}_{\mathbf{Q},\mathbf{Q}}, \text{ such that:} \\ \mathbf{W}_{1,1} &= (T_{i,1} - T_{j,1})^2 \\ \mathbf{W}_{a,b} &= (T_{i,a} - T_{j,b})^2 + \min(\mathbf{W}_{a-1,b}, \mathbf{W}_{a,b-1}, \mathbf{W}_{a-1,b-1}) \\ & \quad a = 2, \ldots, Q, b = 2, \ldots, Q \end{aligned} \tag{3.2}$$

An example illustration of the DTW distance among two series of the Trace datasets is shown in Figure 3.1. In Figure 3.1.a and Figure 3.1.b we illustrate the difference between the warping of the time indices of series in the cases of Euclidean and DTW distances. In this plot, the two series $T_i, T_j$ are shown above each other for illustration purposes and each has its own axis. As can be noticed, DTW overcomes the concerns with the shifted patterns, because the indices are aligned according to the warping path that yields the minimum overall distance.

The cost matrix (W) is depicted in Figure 3.1.c, while the warping path $\tau^{T_i,T_j}$ in Figure 3.1.d. For instance, both series have a sinusoidal pattern that is located around time 150 in $T_i$ (red) and around time 200 in $T_j$ (green). The warping path of Figure 3.1.d) shows that time indices around 200 of $T_i$ are matched to time indices around 150 of $T_j$.

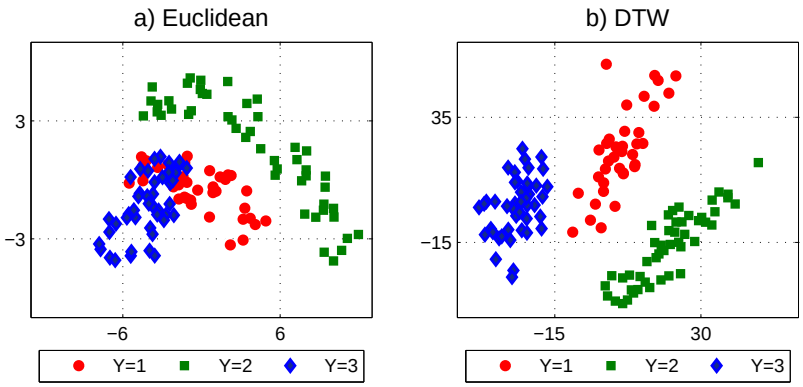



Figure 3.2: Multi-dimensional scaling: Euclidean vs. Dynamic Time Warping on CBF dataset; DTW allows similar series to be located nearby and therefore boosts classification accuracy

DTW is used as a dissimilarity measure for the nearest neighbor classifier and this combination has been shown to have higher classification accuracy than other dissimilarity measures [39, 116, 123]. The performance of DTW is attributed to the fact that similar instances are located closer in the DTW-space than the Euclidean space. This argument is not easy to see, because it is not trivial to plot the DTW-space. However, a demonstration of DTW can be achieved by reconstructing a 2-dimensional representation of a dataset given only the pair-wise diss-similarity matrix among series. For instance, the pair-wise distances among series for the DTW case are $X_{i,n} = DTW(T_i, T_n), i = 1, \ldots, N; n = 1, \ldots, N$. Multi-dimensional scaling [13] is a technique that can reconstruct the geometric positioning of points, given only the pairwise distance matrix (e.g $X$ in our case above). Figure 3.2 illustrates the reconstructed 2-dimensional representation of the CBF dataset using the pair-wise Euclidean and DTW distances of time-series. In contrast to Euclidean distance, DTW outputs close values for similar series of the same class, therefore intra-class series are clustered together. In this way, a classifier in the DTW-space, can achieve a higher prediction accuracy [65].

## 3.3 Time-Series Representations

In order to understand the regularities embedded inside time series, a large number of researchers have invested efforts into deriving and discovering time-series representations. Formally speaking, Equation 3.3 describes a conversion a series $T_i \in \mathbb{R}^Q$ into another representation named 'REP' and denoted by $T_i^{\text{REP}} \in \mathbb{R}^{Q'}$. It is worth noting that usually the dimensionality of the new representation has a lower dimension $Q' << Q$, achieving a dimensionality reduction.

$$T_i \in \mathbb{R}^Q \xrightarrow{\text{Represented as}} T_i^{\text{REP}} \in \mathbb{R}^{Q'}, \quad \forall i \in \{1, \ldots, N\} \tag{3.3}$$

The ultimate target of representation methods is to encapsulate the regularities of time-series patterns by omitting the intrinsic noise. Discrete Fourier transforms have attempted to represent repeating series structures as a sum of sinusoidal signals [44]. Similarly, wavelet transformations approximate a time-series via orthonormal representations in the form of wavelets [23]. However, such representations perform best under the assumption that series contain frequently repeating regularities and little noise which is not strictly the case in real-life applications. Singular Value Decomposition is a dimensionality reduction technique which has also been used to extract latent dimensionality information of a series [18], while supervised decomposition techniques have aimed at respecting the class segregation in the low-rank data [51].

In addition to those approaches, researchers have been also focused on preserving the original form of the time series without transforming them to different representations. Nevertheless, the large number of measurement points negatively influence the run-time of algorithms. Attempts to shorten time series by preserving their structure started by linearly averaging chunks of series points. Those chunks are converted to a single mean value and the concatenation of means create a short form known as a Piecewise Constant Approximation [68]. Another technique operates by converting the mean values into a symbolic form and is called Symbolic Aggregate Approximation, denoted shortly as SAX [75, 119]. Further elaborations of lower bounding techniques facilitate efficient indexing and searching [105], enabling large scale mining of time series [20]. Also least squares approximation of time series using linear lines [28] or orthogonal polynomials have been proposed for the representation of sliding windows [45]. Finally, a recent study has generated a supervised and codebook-based symbolic representation for multivariate time-series [11]. In comparison, Chapter 8 will present a novel

representation for repetitive time-series which based on the utilization of polynomial functions to fit sliding windows of a time series. Dimensionality reduction, as a special type of time-series representations, will be covered in an independent section right after the current one.

### 3.3.1 Fourier and Wavelet Transformation

Fourier transformations and the wavelet transformation are historic representation techniques for signals. The discrete Fourier transform represents a series as a sum of complex sinusoidal functions, while the new representation are the coefficients of the summation [44]. Wavelet transformations, on the other hand, approximate a series through a summation of orthonormal wavelet functions [23]. While Fourier and Wavelet transformations have been applied to time-series data quite early [23, 44], they are not superior to less complex representations with respect to several tasks, such as indexing and searching [90].

### 3.3.2 PAA and SAX

Piecewise Aggregate Approximation (PAA) is a simple, yet effective technique to reduce the dimensionality of time-series. The mechanism is a moving-average type of aggregation, where the time series are divided into segments and the average value of each segment is stored [22]. Equation 3.4 formalizes how a time series $T_i \in \mathbb{R}^Q$ is converted to a PAA representation $T_i^{\text{PAA}} \in \mathbb{R}^{\frac{Q}{L}}$. The configuration divides the series into non-overlapping segments of length $L \in \mathbb{N}$ and computes the average of each segment, therefore the new series length is $\frac{Q}{L}$. In addition, PAA will be used in Chapter 7 as a fast and locally-sensitive dimensionality reduction.

$$T_{i,m}^{\text{PAA}} := \frac{1}{L} \sum_{l=1}^{L} T_{i,l+(m-1)L}, \quad i = 1, \ldots, N;\ m = 1, \ldots, \frac{Q}{L} \tag{3.4}$$

A follow-up representation based on PAA proposes to convert the averaged segment values of Z-normalized series to symbolic forms [76]. The approach is named Symbolic Aggregate Approximation (SAX) and relies on using breakpoints to divide the range of series values into bins and assign symbols to each bin. The division of a real-valued range into bins is conducted according to a Gaussian distribution of values. For example, Z-normalized values having mean 0 and standard deviation of 1 can be assigned to one of the following three bins $a =$

$(-\infty, -0.43], b = (-0.43, 0.43], c = (0.43, \infty)$. In that way a hypothetical series $T_i = \{-0.7, 0, 0.6, -0.1, -0.5\}$ is transformed to the string 'abcba'. The number of bins is also referred to as the alphabet size [90]. For the sake of completeness we are briefly formalizing the SAX representation in the following lines. Assume an alphabet having $w$-many symbols is denoted by $(s_1, s_2, \ldots, s_w)$. Each symbol will be represented by a range of values that constitute a bin. Practically the bin is delimited by two breakpoint values, the lower $\psi_{s_*} \in \mathbb{R}$ and the higher value $\phi_{s_*} \in \mathbb{R}$, satisfying $\psi_{s_i} = \phi_{s_{i-1}}, i = 2, \ldots, w$. Every PAA value that falls within the breakpoints is assigned to that bin's symbol. The formalization of SAX is presented in Equation 3.5.

$$T_{i,m}^{\text{SAX}} := s_v \text{ if } \psi_{s_v} \leq T_{i,m}^{\text{PAA}} < \phi_{s_v};\ i = 1, \ldots, N;\ m = 1, \ldots, \frac{Q}{L}, v \in \{1, \ldots, w\} \quad (3.5)$$

The conversion of real-valued time series into symbolic sequences motivated the research community to import string-based methods from related domains, such as bioinformatics [90]. An illustration of the SAX representation on a time series from the SwedishLeaf dataset is provided in Figure 3.3. In the scope of this thesis, SAX is used as the local representation for a bag-of-words baselines in Chapter 8.

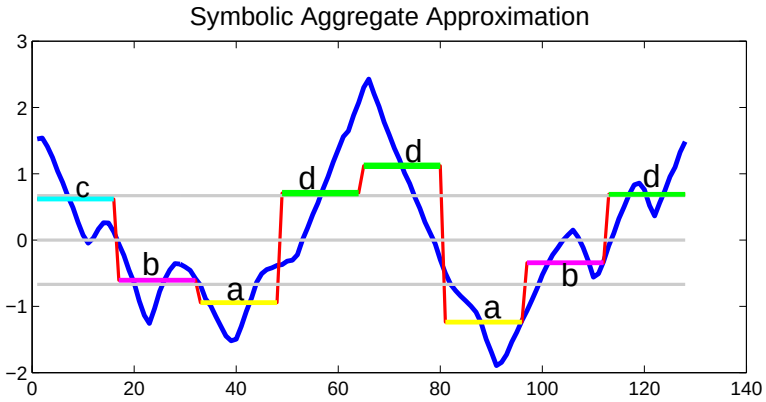



Figure 3.3: Symbolic Aggregate Approximation of an instance from the SwedishLeaf dataset. The PAA average values are located into four bins $\{a, b, c, d\}$ separated by gray horizontal lines. The SAX output is the string 'cbaddabd'.

### 3.3.3 Bag-of-Words

A recent direction of research has drawn attention on the need to segment the time series into local patterns and measure the frequencies of patterns as classification features. For instance, frequencies of time-series motifs have been fed into standard classifiers [17]. One similar bag-of-words approach has also been applied to long bio-medical data [115]. Other approaches have utilized bag of words approaches using string representations of series segments [77, 78] (detailed below in Section 3.3.3.1), as well as supervised bag of features (detailed below in Section 3.3.3.2). Furthermore, a method that extracts bags of local polynomial words will be presented in Chapter 8.

#### 3.3.3.1 Bag of SAX Words

One way of building bag of words is by converting all series' segments to strings [78]. Those symbolic words are produced by the aforementioned piecewise constant approximation technique called SAX [75], while the frequencies of the SAX words are used ultimately for classification [77, 78]. This representation will be referred throughout the thesis using the acronyms BoW (bag of words) and/or BSAX (bag of SAX words) interchangeably.

As a first step, every segment of length $L \in \mathbb{N}$, denoted by $S_{i,j} = T_{i,j}, \ldots, T_{i,j+L-1}$, is converted to its SAX representation $S_{i,j}^{\text{SAX}}$ using the transformation of Equation 3.5. A dictionary of all the SAX words of a time-series datasets is defined as the set $\mathcal{D}$ of Equation 3.6. it is worth noting that a set notation omits duplicates and stores only the unique SAX words.

$$\mathcal{D} := \bigcup_{i=1}^{N} \bigcup_{j=1}^{M} S_{i,j}^{\text{SAX}} \tag{3.6}$$

With a slight abuse of formalism, assume $D$ to be accessible as a list of $|\mathcal{D}|$ unique words, where the $z$-th word of the dictionary is indexable as $\mathcal{D}_z$. Finally the new representation of each time-series $T_i \in \mathbb{R}^Q$ will be a vector $T_i^{\text{BoW}} \in \mathbb{N}^{|\mathcal{D}|}$, created according to Equation 3.7. From a semantic perspective, the $z$-th element of the new representation $T_i^{\text{BoW}}$ counts how often does the $z$-th dictionary word occur in the $i$-th series.

$$T_{i,z}^{\text{BoW}} := \left| \left\{ j \in \{1, \ldots, M\} \mid S_{i,j}^{\text{SAX}} = \mathcal{D}_z \right\} \right|;\ i = 1, \ldots, N, z = 1, \ldots, |\mathcal{D}| \tag{3.7}$$

#### 3.3.3.2 Supervised Bag-of-Features (TSBF)

Moreover, a bag-of-patterns study proposes to extract series segments of various lengths and positions, in addition to generating a supervised codebook of those patterns [12]. A random forest classifier has been trained over the extracted features. That study demonstrates considerable improvements over classical baselines in terms of prediction accuracy [12]. In brief terms, the method (denoted as TSBF in the thesis) can be summarized in the following steps:

1. Extract local features (i.e. mean, variance, linear regression coefficients, etc ...) from each segment of the time-series;
2. Create a supervised codebook of the segments' features using a random forest,
3. Aggregate the codebook features of each series into a global bag-of-features representation;
4. Classify the global bag-of-features representation using a random forest classifier;

## 3.4 Time-series Shapelets

Currently, the most prominent state of the art technique for extracting time-series features is called shapelets mining. Shapelets were first proposed by [126] as time-series segments that maximally predict the target variable. All possible segments were considered to be potential candidates, while the minimum distances of a candidate to all training series were used as a predictor feature for ranking the information gain accuracy of that candidate on the target variable. Other accuracy metrics have been proposed for evaluating the prediction accuracy of a shapelet candidate such as F-Stats [80], Kruskall-Wallis or Mood's median [62]. In addition, the minimum distance of a set of shapelets to time series can be perceived as a kind of data transformation [80], while standard classifiers have achieved high accuracy over the shapelet-transformed representation [62].

An illustration to shapelets is presented in Figure 3.4. The figure includes plots from the Gun Point dataset, where three series of each class are displayed. The purpose of the illustration is to underscore the fact that local segments can discriminate a class from another. In the current example, both classes have the same overall structure, but differ only in the highlighted segments.

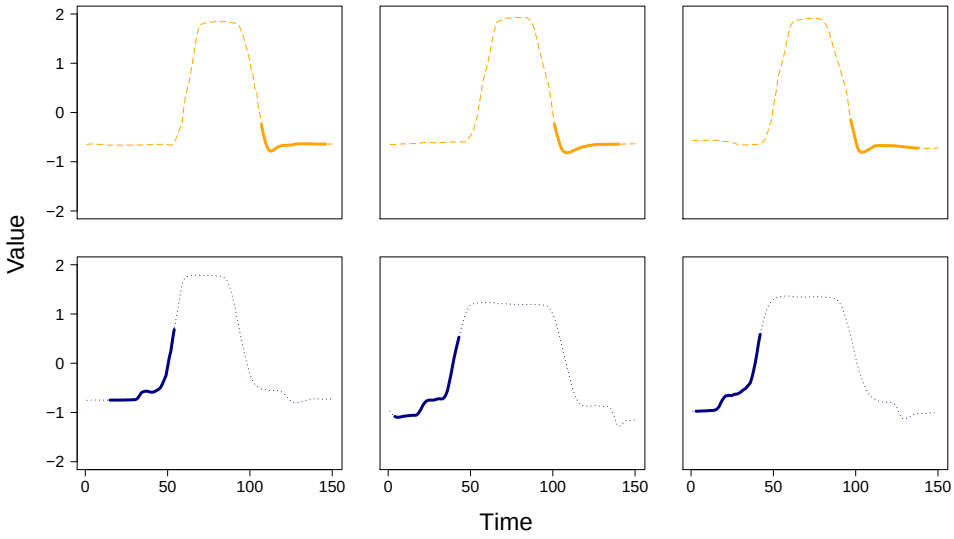



Figure 3.4: Illustrating three series of each class from the Gun Point dataset; one class in orange and the other in blue. For each class the discriminative segment is highlighted.

From a formal perspective, shapelets are a set of $K \in \mathbb{N}$ patterns, each having length $L$, and defined as $P \in \mathbb{R}^{K \times L}$. The minimum distances between shapelets and all the segments of each series constitute the shapelet-transformation, as defined in Equation 3.8. The challenge of this representation is to find the shapelets $P$, for which the resulting representation $T^{\text{SHAPELET}} \in \mathbb{R}^{N \times K}$ helps achieve the highest classification accuracy.

$$T_{i,k}^{\text{SHAPELET}} := \min_{j=1,\ldots,Q-L+1} \sum_{l=1}^{L} \left(T_{i,j:j+L-1} - P_{k,l}\right)^2 ; \; i = 1, \ldots, N, \; k = 1, \ldots, K \quad (3.8)$$

A description of the shapelet discovery approach is given in Algorithm 2. The method iterates over all the segments of the dataset (line 2-3) and considers each segment as a candidate (line 4). For each candidate we measure the distance against all the training series (line 5-8), as the distance to the closest segment of each series. The distances to the series can be perceived as a classification feature and the discrimination quality of such a feature can be measured using the Information Gain or F-stats measures [62]. The candidates and their qualities (proxy of accuracy) are stored in the lists $\mathcal{C}$ and $\mathcal{A}$. The operator $\oplus$ defines the addition of one element to the back of a list. In the end, we are interested to sort out the top $K$ shapelets which have the highest classification accuracy.

**Algorithm 2**: **ShapeletDiscovery**: Select the K most accurate shapelets

**Data**: Time-series data $T \in \mathbb{R}^{N \times Q}$, Labels $Y \in \mathbb{N}^N$, Number of shapelets $K \in \mathbb{N}$, Shapelets' length $L \in \mathbb{N}$

**Result**: Most accurate shapelets $P \in \mathbb{R}^{K \times L}$

$\mathcal{C} \leftarrow \emptyset$, $\mathcal{A} \leftarrow \emptyset$;
Compute the accuracy of each segments;
**for** $i = 1, \ldots, N$ **do**
  **for** $j = 1, \ldots, Q - L + 1$ **do**
    $c \leftarrow (T_{i,j}, T_{i,j+1}, \ldots, T_{i,j+L-1})$;
    $d^c \leftarrow 0^N$;
    **for** $i' = 1, \ldots, N$ **do**
      $d^c_{i'} \leftarrow \min_{j'=1,\ldots,Q-L+1} \sum_{l=1}^{L} (c_l - T_{i',j'+l-1})^2$;
    **end**
    $\mathcal{C} \leftarrow \mathcal{C} \oplus c$;
    $\mathcal{A} \leftarrow \mathcal{A} \oplus$ DiscriminationQuality($d^c$) (e.g. InfoGain, F-stat, ...) ;
  **end**
**end**
Record the K segments with the highest Discrimination Quality;
**for** $1, \ldots, K$ **do**
  $P \leftarrow P \oplus \left( \underset{\mathcal{C}_r \in C \wedge \mathcal{C}_r \notin P;\, \forall r \in \{1,\ldots,|\mathcal{C}|\}}{\operatorname{argmax}} \mathcal{A}_r \right)$;
**end**
**return** $P$

In terms of applicability, shapelets have been utilized in a battery of real-life domains. Unsupervised shapelets discovery, for instance, has been shown useful in clustering time series [129]. Shapelets have seen action in classifying/identifying humans through their gait patterns [108]. Gesture recognition is another application domain where the discovery of shapelets has played an instrumental role in improving the prediction accuracy [59, 60]. In the realm of medical and health informatics, interpretable shapelets have been shown to help the early classification of time series [124, 125]. In comparison to the state-of-the-art methods, in Chapter 6 we propose a novel method that **learns** near-to-optimal shapelets directly, without the need to search exhaustively among a pool of candidates extracted from time-series segments.

### 3.4.1 Fast Shapelet Discovery

The excessive amount of potential candidates makes the brute-force (exhaustive) shapelet discovery intractable for large datasets. Therefore, researchers have come up with various approaches for speeding up the search. Early abandoning of the Euclidean distance computation combined with an entropy pruning of the information gain metric is an early pioneer in that context [126]. Additional papers emphasize the reuse of computations and the pruning of the search space [86], while the projection of series to the SAX representation was also elaborated [95]. Furthermore, the discovery time of shapelets has been minimized by mining infrequent shapelet candidates [61]. Speed-ups have also been attempted by using hardware-based implementations, such as the usage of the processing power of GPUs for boosting the search time [24]. Another paper emphasized the need for randomly sampling candidates in order to significantly reduce the discovery time of shapelets [122]. In particular, the Fast Shapelet approach [95] has been shown to achieve very fast discovery times. The method can be verbally summarized in three steps.

1. Convert the time-series segments into the SAX representation,
2. Search for the shapelets in the SAX space,
3. Apply hash masking of SAX words to prune distance computations;

In contrast to the state-of-the-art methods, in Chapter 7 we propose a fast novel method that discovers shapelets by combining a direct similarity-based pruning strategy of candidates with an incremental classification technique.

## 3.5 A Brief Review of Relevant Classifiers

### 3.5.1 Nearest Neighbor

The nearest neighbor is a parameter-less classifier that outputs the label of the most similar training instance. For instance, given a training series $T_t^{\text{Test}} \in \mathbb{R}^Q$, we would like to estimate its target variable $\hat{Y}_t \in \mathbb{N}$ from the training series $T^{\text{Train}} \in \mathbb{R}^{N \times Q}$ and the training labels $Y^{\text{Train}} \in \mathbb{N}^N$ [2]. The standard nearest neighbor technique (denoted 1NN or NN) aims at predicting the target of the closest training instance, as formalized in Equation 3.9. The distance operator (denoted by $\mathcal{D}$) is one of the dis-similarity measures of Section 3.2.

$$\hat{Y}_t^{\text{Test}} := Y_{i^*}^{\text{Train}} \quad \text{where} \quad i^* = \underset{i=1,\ldots,N}{\operatorname{argmin}} \; \mathcal{D}(T_i^{\text{Train}}, T_t^{\text{Test}}) \tag{3.9}$$

The nearest neighbor can be extended toward aggregating the labels of multiple neighboring instances, also known as the K-nearest neighbors. The advantages of NN are its ease of implementation and the lack of parameters. For this reason, NN is often categorized as a *lazy learner*. On the other hand, the model of a nearest neighbor classifier is the whole training set, therefore the space complexity of the model is high. In addition, the runtime of classifying an instance is also high $\mathcal{O}(NQ)$, compared with the constant $\mathcal{O}(Q)$ and quasi-constant classification times for other methods such as logistic regression or Support Vector Machines.

### 3.5.2 Support Vector Machines

Support Vector Machines (SVMs) are considered to be one of the best off-the-shelf classifier for a wide application domains of machine learning. Since SVMs will be used in the forthcoming Chapters 4 and 5, they will be briefly described in this section.

The success of Support Vector Machines is based on the principle of finding the maximum margin decision boundary (hyperplane) that accurately splits class regions [32]. The training process of SVMs is done by optimizing the maximum margin objective, usually in the dual representation of the objective function. In order to overcome problems with non-linear separability of certain datasets, introduction of slack variables has been applied to allow regularized disobedience from the decision boundary. In addition kernel theory has been combined with the dual learning of SVMs in order to offer various types of non-linear expressiveness to the decision boundary [103].

As previously stated, Support Vector Machines, are a classifier that aims at finding the maximum margin separating decision boundary among class regions [31]. The decision boundary lies in the form of a hyperplane, denoted $W \in \mathbb{R}^{Q+1}$. For binary classification the target variable $Y \in \{-1, +1\}^N$. The classification of a test instance is computed as the sign of the dot product between the hyperplane and the instance vector, as shown in Equation 3.10. This formulation describes the prediction model of a linear SVM.

$$\hat{Y}_i = \text{sign}\left(W_0 + \sum_{q=1}^{Q} W_q T_{i,q}\right), \quad i = 1, \ldots, N \tag{3.10}$$

The maximum margin hyperplane is computed by solving the optimization function of Equation 3.11. Such a formulation is known as the soft-margin primal form [32]. The complexity hyper-parameter $C \in \mathbb{R}$ avoids the overfitting of the hyper-plane $W$ to the training instances.

$$\begin{aligned} \underset{W}{\text{argmin}} \quad & \frac{1}{2}||W||^2 + C\sum_{i=1}^{n}\xi_i \\ & \text{subject to:} \\ & Y_i(\langle W, T_i\rangle + W_0) \geq 1 - \xi_i, \text{and } \xi_i \geq 0, \;\; i = 1, ..., N \end{aligned} \tag{3.11}$$

The so called slack variables, defined in Equation 3.12 represent the violation of each series instance from the boundary with the objective of minimizing the violations.

$$\xi_i = \max(0, 1 - Y_i(\langle W, T_i\rangle + W_0)), \quad i = 1, \ldots, N \tag{3.12}$$

The primal form objective function is transformed by expressing the inequality conditions via Lagrange multipliers denoted $\alpha_i$, one per instance. Then the objective function is solved for $w$ and $w_0$ and the dual form is yield as shown in Equation 3.13. The dual formed is preferred because it gets rid of the inequality constraints of the primal form of Equation 3.11.

$$\begin{aligned} \max \quad & \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j Y_i Y_j K(T_i, T_j) \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq C, \; i = 1, ..., N \\ & \sum_{i=1}^{n}\alpha_i Y_i = 0 \end{aligned} \tag{3.13}$$

In order to classify datasets exhibiting non-linear separation, the dot product present in the dual objective function, $\langle T_i, T_j\rangle$, is substituted by the so called kernel trick which is denoted as $K(T_i, T_j)$ [103]. A typical kernel, the inhomogeneous polynomial one, is presented in Equation 3.14, where $d \in \mathbb{N}$ is the degree of the polynomial and $c \in \mathbb{R}$ a constant.

$$\langle x_i, x_j\rangle \rightarrow K(T_i, T_j) = (\langle T_i, T_j\rangle + c)^d \tag{3.14}$$

The purpose of the kernel trick approach is to express the feature space in terms of a higher dimensionality space, such that a linear decision boundary in the high dimensional space would represent a non-linear boundary in the original one. The solution of the dual problem is carried through dedicated optimization algorithms. The ultimate decision boundary can be formed as a function of the solved Lagrange multipliers as depicted by Equation 3.15. Please note that the non-zero $\alpha_j$ coefficients correspond to the so-called support vector instances $(T_j, T_j)$.

$$Y_i = \text{sign}\left(\sum_{j=1}^{N} \alpha_j Y_j K(T_j, T_i)\right) \tag{3.15}$$

Once the decision boundary is learned. a new instance can be classified as a summation iteration over the support vectors, which makes the operation of magnitude $O(ab)$, where $a$ is the number of support vectors and $b$ the complexity of the dot product of two instance's features. Please note that $a$ is a fraction of the number of instances, so still $O(N)$ at worst-case, while $b$ is linearly proportional to the number of features. So SVM's classification time for a single test instance deteriorates to $O(N)$ in the worst case scenario.

#### 3.5.2.1 Invariant Support Vector Machines

There is a stream of research that focuses on extending SVMs towards handling inter- and intra-class variations in image classification [36]. Even though the challenge of handling invariance in time series classification is rather new, it has been applied long ago to the domain of image classification. Significant effort to the problem of invariant classification was followed by the SVM community, where one of the initial and most successful approaches relied on creating virtual instances by replicating instances. Typically the support vectors are transformed, creating new instances called Virtual Support Vectors (VSV), with the aim of redefining the decision boundary [89, 102]. VSV has been reported to achieve competitive performance in image classification [36]. An alternative technique in handling variations relies in modifying the kernel of the SVM by adding a loss term in the dual objective function. Such a loss enforces the decision boundary to be tangent to the transformation vector [36, 82]. Other approaches have been focused on selecting an adequate set of instances for transformation [81]. Studies which target the adoption of SVM to the context of time series have primarily addressed the inclusion of DTW as a kernel [5, 106]. Unfortunately, all proposed

DTW-based kernels, we are aware of, are not efficiently obeying the positively semi-definite requirement [57]. The DTW based kernels have been reported to not perform optimally compared to state-of-art [131]. As a follow-up remedy, positive semi-definite kernels like the Gaussian elastic metric kernel arose [131]. Consecutively, another study has proposed a Gaussian elastic kernel for SVMs [131]. A method which produces a semi-definite kernel is called Global Alignment Kernels and builds an average statistics from all possible warping paths of time indexes [33]. Generating new instances based on the pairwise similarities has also been applied [57], with limited success compared to DTW-NN. In comparison, our method of Chapter 4 applies a novel transformation technique along with the VSV approach.

Intentional transformations and deformations of time series has shown little interest because of the limited studies of VSV classifiers in the domain. Among the initiatives, morphing transformation from a time series to another has been inspected [93]. However, deformations have been more principally investigated in the domain of images. Moving Least Squares is a state-of-art technique to produce realistic deformations [101]. In Chapter 4 we use the Moving Least Squares method for applying the transformations over series.

## 3.6 Brief Introduction to Matrix Factorization

Matrix Factorization (MF) is a technique that will be used for reducing the dimensionality of time series in Chapter 5. For this reason, this section provides a review of the Matrix Factorization method and its supervised variant. Matrix factorization is a variant of dimensionality reduction that projects data into a reduced/latent/hidden data space which usually consists of lower dimensions than the original space [71, 110]. Moreover, matrix factorization has been applied in domains involving time-series data as in music transcription [109], up to EEG processing [99].

Stated otherwise, factorization exploits hidden features of a matrix by learning latent matrices whose dot product represent the original [46, 110]. Unification of different MF techniques has been formalized as minimizing a generalized Bregman divergence [107]. Stochastic gradient descent learning has been successfully applied for unsupervised factorization of data sets with missing values [71]. For instance, in recommender systems various MF strategies have been recently adopted to collaborative filtering of sparse user-item ratings with the aim of recommending unrated pairs [98]. Supervision, in the context of matrix factorization, enables the linear separation of classes in the projected data space [84].

The objective function of the factorization has been shaped to enforce specific properties of the latent projected data, for example, geometric structure of affinity can be used to position neighbor instances closer in the projected space [19]. MF has been also applied to the analysis of time series, for instance in music transcription [109], or EEG preprocessing [99].

The unsupervised Matrix Factorization [71] is guided only by the reconstruction loss, i.e. it only reconstructs the predictors $T$ and not the labels $Y$. Such an approach does not take into consideration the classification accuracy impact of the extracted features, therefore the produced reduced dimensionality data is not optimized to improve accuracy. In order to overcome such a drawback, the so called supervised dimensionality reduction has been proposed [50, 56]. The key commonalities of those supervised dimensionality methods rely on defining a joint optimization function, consisting of the reconstruction loss terms and the classification accuracy terms.

The typical classification accuracy loss term focuses on defining a classifier in the latent space, i.e. $U \in \mathbb{R}^{N \times D}$, via a hyperplane defined by the weights vector $W \in \mathbb{R}^{D}$, such that the weights can correctly classify the training instances of $U$ in order to match observed label $Y \in \mathbb{N}^{N}$. Equation 3.16 defines a cumulative joint optimization function using a reconstruction term for the predictors, denoted $\mathcal{L}_R(T, U, V)$, and a classification accuracy term, denoted $\mathcal{L}_{CA}(Y, U, W)$. The trick of such a joint optimization constitutes on updating the low-rank data $U$ simultaneously, in order to minimize both $\mathcal{L}_R$ and $\mathcal{L}_{CA}$ via gradient descent on both loss terms. The hyper parameter $\beta$ is a switch which balances the impact of reconstruction vs classification accuracy. Throughout this section we evaluate the binary classification problem, even though the explained methods could be trivially transferred to multi-nominal target variables by employing the one-vs-all technique. Should that be needed, one would have to build as many classifiers as there are categories in the target variable, while each classifier would treat one category value as the positive class and all the remaining categories as the negative class. In addition to the reconstruction $\mathcal{L}_R$ and the classification accuracy $\mathcal{L}_{CA}$ loss terms, the model has additive regularization terms hyper-parametrized by coefficients $\lambda_U, \lambda_V, \lambda_W$. Such a regularization helps the model avoid overfitting and enables a better generalization over the test instances.

$$\mathcal{L}(T, Y, U, V, W) \;=\; \beta\, \mathcal{L}_R(T, U, V) + (1 - \beta)\, \mathcal{L}_{CA}(Y, U, W) \qquad (3.16)$$

### 3.6.1 Matrix Factorization

Matrix factorization is a dimensionality reduction technique which decomposes the dataset $T \in \mathbb{R}^{N \times Q}$ of $N$ training instances and $Q$ features, into two smaller matrices of dimensions $U \in \mathbb{R}^{N \times D}$ and $V \in \mathbb{R}^{D \times Q}$ [71]. The latent/reduced projection of the original data $X$ is the latent matrix $U$, where $D$ is the dimensionality of the projected space. Typically $D$ is much smaller than $Q$, meaning that the dimensionality is reduced. In case $D < Q$, then $U$ is nominated as the low-rank representation of $T$. Otherwise, if $D > Q$ a *non-grata* inflation phenomenon is achieved. Such a decomposition is expressed in forms of a regularized reconstruction loss, denoted $\mathcal{L}_R(T, U, V)$ and depicted in Equation 3.17. The optimization of such a function aims at computing latent matrices $U, V$ such that their dot product approximates the original matrix $T$ via an Euclidean distance ($L2$ norm) loss. In addition to the $L2$ reconstruction norm, $L2$ regularization terms are added by being weighted with factors $\lambda_U, \lambda_V$ in order to avoid over-fitting.

$$\underset{U,V}{\operatorname{argmin}} \qquad \mathcal{L}_R(T, U, V) = ||T - UV||^2 + \lambda_U ||U||^2 + \lambda_V ||V||^2 \tag{3.17}$$

Due to brevity this section describes only the factorization model and not the learning algorithm. For an in depth description on how are those latent matrices $U, V, W$ computed, the reader is invited to consult [50, 56].

### 3.6.2 Supervised Matrix Factorization

The linear supervision of the dimensionality reduction refers to the inclusion of a linear classification loss term to the objective function. The addition of the linear classification loss term enforces the instances of different classes to be linearly separable in the low-rank space. Various loss terms have been proposed depending on the utilized linear classifier. Before explaining the different losses, we introduce the predicted value of instance $i$ as $\hat{Y}_i$ and defined in Equation 3.18. The predicted value is the dot product of the instance values $U_{i,:} \in \mathbb{R}^D$ and linear weights $W \in \mathbb{R}^D$. In addition, the bias of the classification weight vector $W_0 \in \mathbb{R}$ is summed up.

$$\hat{Y}_i \;=\; W_0 + \sum_{k=1}^{D} U_{i,k} W_k, \quad i = 1, \ldots, N \tag{3.18}$$

#### 3.6.2.1 Loss terms

Loss terms quantify the degree of violation that a classifier exhibits from the desired (perfect) prediction accuracy. Concretely the least square loss measures the L2 distance between the true targets $Y$ and predicted vales $\hat{Y}$. In the context of linearly supervised reduction [84], the least-squares loss term can be defined as shown in Equation 3.19. Similar to the regression case, least squares is adopted for classification by treating the target values as $Y \in \{-1, 1\}^N$, while predicted positive values $\hat{Y}$ indicate a positive class and vice versa.

$$\mathcal{L}_{CA}(Y, U, W) = \sum_{i=1}^{N} \left(Y_i - \hat{Y}_i\right)^2 + Reg(U, W) \tag{3.19}$$

The logistic loss has been used to guide the decomposition by minimizing the target prediction error along a sigmoid curve [52]. Equation 3.20 presents the loss, while the target values are expected to be in the range $Y \in \{0, 1\}^N$. Please note that the sigmoid function is defined as: $\sigma(\hat{Y}) = \frac{1}{1+e^{-\hat{Y}}}$.

$$\begin{aligned} \mathcal{L}_{CA}(Y, U, W) \quad = \quad & \sum_{i=1}^{N} -Y_i \ln(\sigma(\hat{Y}_i)) - (1 - Y_i) \ln\left(1 - \sigma(\hat{Y}_i)\right) \\ & + Reg(U, W) \end{aligned} \tag{3.20}$$

Another strong classifier is based on hinge loss, which represents the underlying foundation of the Support Vector Machines is depicted in Equation 3.21. The hinge loss has been also applied to supervised dimensionality reduction [35]. The hinge loss is also called a maximum margin loss because it tries to find a margin of unit size between the hyperplane $W$ and the region of each class.

$$\mathcal{L}_{CA}(Y, U, W) = \sum_{i=1}^{N} \max(0, 1 - Y_i \hat{Y}_i) + Reg(U, W) \tag{3.21}$$

The regularization term is a L2 norm and defined in Equation 3.22. The regularization parameters $\lambda_U, \lambda_W$ control the complexity of the model and avoid over-fitting.

$$Reg(U, W) = \lambda_U \sum_{i=1}^{N} \sum_{k=1}^{D} {U_{i,k}}^2 + \lambda_W \sum_{k=1}^{D} {W_k}^2 \tag{3.22}$$

#### 3.6.2.2 Illustrating How Supervised Factorization Works

Supervised factorization relies on using the label information to guide the projection. In that way, any noise which is present in the observed data can be eliminated in the low-rank representation. In order to show the advantage of the supervised decomposition, we present the experiment of Figure 3.5.

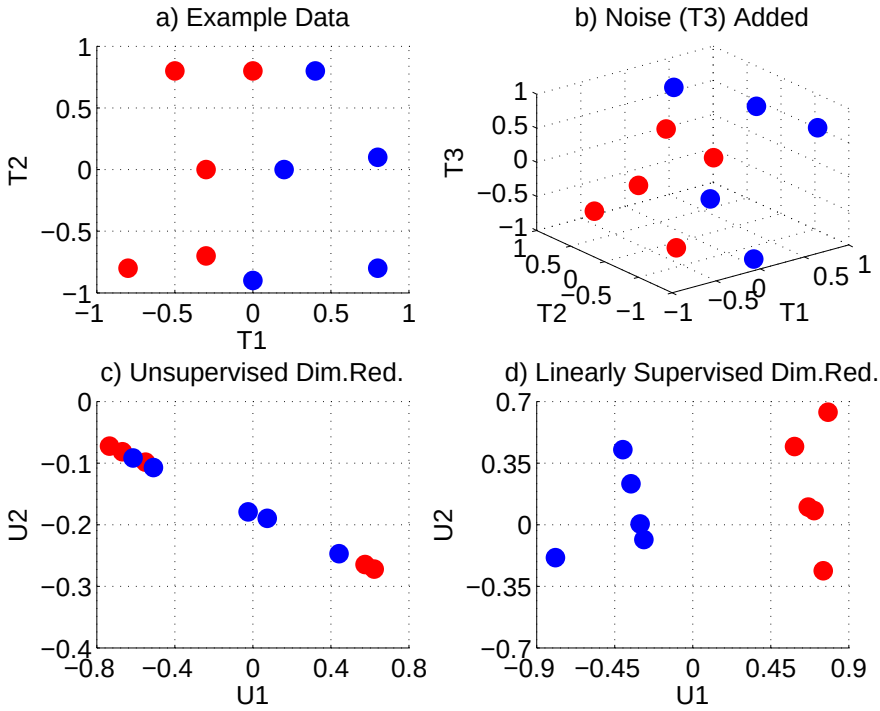



Figure 3.5: Supervised factorization: a) original data with two classes (blue and red); b) random noise variable (T3) added; c) unsupervised factorization (from b) to c) ); d) supervised factorization (from b) to d)

A 2-dimensional synthetic dataset of ten instances, belonging to two classes (red, blue) is depicted in sub-figure a). Please note that the original data are **linearly separable** by a hyperplane. Then, we added a random variable $T_3$ (shown in b) ) of uniform random values between $[-1, 1]$. The experiment aims at reducing the 3-dimensional noisy data back to 2-dimensions using both unsupervised and supervised dimensionality reductions. As can be observed, the unsupervised projection is affected by the added noise and the resulting 2-dimensional data in c) is not anymore linearly separable. In contrast, the supervised decomposition can benefit from a linear classification accuracy loss term to separate instances by

label. A smooth hinge loss supervised decomposition was applied to the decomposition of d). Please note that the resulting 2-dimensional projection depicted in d) is linearly separable as the original data. The experiment demonstrates that a supervised decomposition has a stronger immunity towards the presence of noise in the data.

### 3.6.3 Factorization of Time Series

In fact, dimensionality reduction has been previously used to project the time series into a low-rank data space [66], while a recent method incorporates class segregation into the projection [51]. In addition, there have been a few attempts in generating invariant time-series features through factorization. A shift-invariant sparse coding of signals has been proposed for reconstructing noisy or missing series segments [74]. In similar domains, sparse coding factorization has been applied for deriving shift and 2D rotation invariant features of hand writing data [8], and also invariant features of audio data [63]. Moreover, a temporal decomposition of multivariate streams has been used to discover patterns in patients' clinical events [114]. In contrast to the related work, we propose a segment-wise invariant factorization of time-series in Chapter 5.

## 3.7 Repetitive Time Series

The classification of repetitive time series focuses on long signals which are composed of one or more types of patterns appearing in unpredicted orders and frequencies. Principally, the classification of those series has been mainly conducted by detecting sub-sequence patterns and computing statistics over them. For instance, underlying series patterns have been expressed as motifs and the difference between the motif frequencies has been utilized [17]. Other approaches have explored the conversion of each sliding window segment into a literal word constructed by piecewise constant approximations and the SAX method [77, 78]. The words belonging to each time series are gathered in a 'bag' and a histogram of the words' frequencies is constructed. Such a representation has been shown to be rotation-invariant, because the occurrence of a pattern is not related to its position [77]. In addition to existing work, our work of Chapter 8 introduces an expressive histogram formulation based on literal words build from local pattern detection via polynomial approximations.

## 3.8 Image-Related Baseline

Image data can be perceived as two-dimensional time-series signals. A recent work has tried to use the strongest image-classification baseline, the Convolutional Neural Network (CNN), in order to classify time series [117]. First of all, the series are converted to image representations using the Gramian Angular and Markov Transition fields. Afterwards, a tiled CNN is applied over the image-transformed series. Unfortunately, the achieved empirical results are inferior with respect to the latest state-of-the-art, in particular compared to the results of Chapter 6.

## 3.9 Data Augmentation

Our forthcoming approach of Chapter 4 augments the training set by adding synthetic instances, which are generated as transformations of the existing training series. Nevertheless, data augmentation is a popular approach for data types that exhibit high degrees of intra-class variations. For instance, the training instances of images are often augmented using affine transformations before being fed into classifiers [91]. In the general context, a popular augmentation technique is SMOTE, which generates synthetic instances of the minority class through a nearest neighbor strategy [43]. We are going to empirically compare our data augmentation approach against SMOTE in Chapter 4.

# Chapter 4

# Data Augmentation

## Contents



**Highlights:** This chapter encapsulates the first attempt of this thesis in offering a classifier that is invariant to intra-class variations. The rationale of this chapter is to expand the training set, such that all possible variations within a class are added as instances. Therefore, the proposed method will

enable standard classifiers, here Support Vector Machines (SVMs), to achieve a significantly improved prediction accuracy.

SVMs are reported to perform non-optimally in the domain of time series, because they suffer detecting similarities in the lack of abundant training instances. In this chapter we present a novel time-series transformation method which significantly improves the performance of SVMs. Our novel transformation method is used to enlarge the training set through creating new transformed instances from the support vector instances. The new transformed instances encapsulate the necessary intra-class variations required to redefine the maximum margin decision boundary. The proposed transformation method utilizes the variance distributions from the intra-class warping maps to build transformation fields, which are applied to series instances using the Moving Least Squares algorithm. Extensive experimentations on 35 time series datasets demonstrate the superiority of the proposed method compared to both the Dynamic Time Warping version of the Nearest Neighbor and the SVMs classifiers, outperforming them in the majority of the experiments.

## 4.1 Introduction

Support Vector Machines (SVMs) are successful classifiers involved in solving a variety of learning and function estimation problems. Yet, experimental studies show that it performs non-optimally in the domain of time series [57]. We explore an approach to boost the classification of time series using SVMs, which is directly inspired by the nature of the problem and the reasons why SVMs fail to build optimal decision boundaries. In most time series datasets, the variations of instances belonging to the same class, denoted intra-class variation, are considerably numerous. Variations appear in different flavors. A pattern of a signal/time series can start at various time points (translational variance) and the duration of a pattern can vary in length (scaling variance). Even more challenging, such variances can partially occur in a signal, in multiple locations, by unexpected direction and magnitude of change. There exist more, theoretically infinitely many, possible variations of a particular class pattern, compared to the present number of instances in the dataset. Ergo, such lack of sufficient instances to cover all the possible variations can affect the maximum margin decision boundary in under-representing the ideal decision boundary of the problem.

In order to overcome the lack of instances, the insertion of virtual transformed instances to the training set has been proposed. In the case of SVMs, support

vectors are transformed/deformed, the new virtual support vectors added back to the training set and the model is finally retrained [36, 102]. An illustration of the effect of inserting virtual instances and its impact on redefining the decision boundary is shown in Figure 4.1. The most challenging aspect of this strategy is to define efficient transformation functions, which create new instances from existing ones, enabling the generated instances to represent the necessary variations in the feature space.

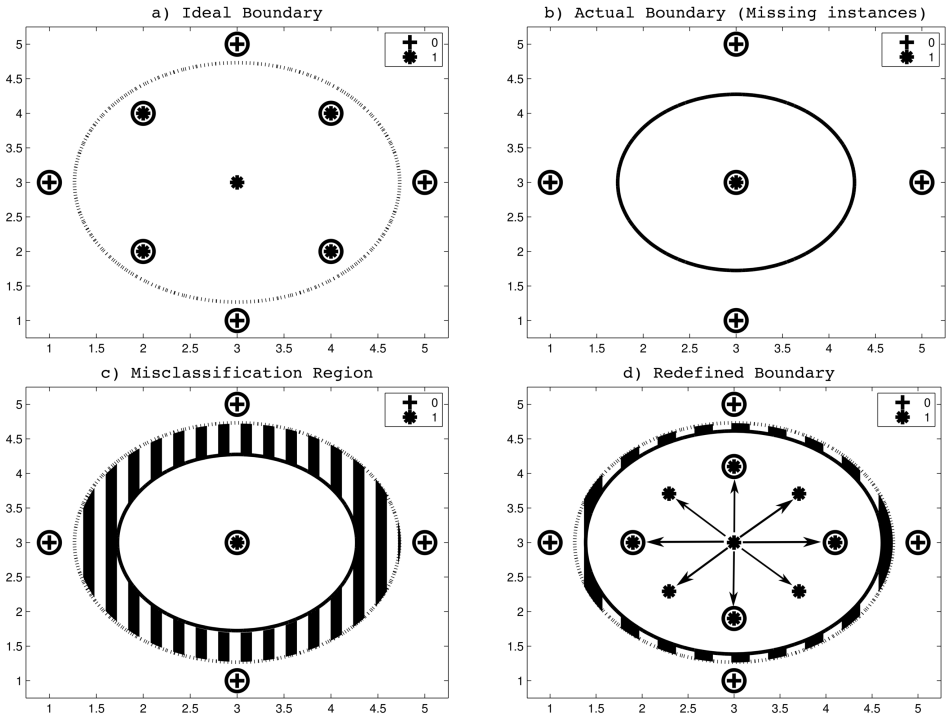



Figure 4.1: **a)** An ideal max-margin decision boundary for the depicted binary (0/1) classification problem. *Circled* points denote support vectors. **b)** An actual version of the problem where most class 1 instances are missing. The actual max-margin decision boundary differs from the ideal boundary in a). **c)** The under-fitting of the actual boundary (*solid*) to represent the ideal boundary (*dots*) produces the *shaded* misclassification region. **d)** Transforming the class 1 instance in coordinate (3,3) and inserting the transformed instances (*pointed by arrows*) back to the dataset, helps redefine the new max-margin boundary (*solid*). Consecutively, the area of the misclassification region is reduced.

The main contribution of our study is in defining a novel instance transformation method which improves the performance of SVMs in time series classification.

In our analysis the transformations should possess four characteristics. First the transformations should be data-driven, concretely an analysis of the intra-class variance distributions should be taken into account. Secondly, localized variations are required since the variations of series instances appears in forms of local deformations. Thirdly, in order to overcome the time complexity issues, only a representative subset of the instances should be selected for producing variations. Finally the transformations should accurately redefine the decision boundary without creating outliers or over-fit the training set.

The novel transformation method we introduce satisfies all the above raised requirements. The proposed method analyses the translational variance distributions by constructing warping alignment maps of intra-class instances. The time series is divided into a number of local regions and transformation fields/vectors are created to represent the direction and magnitude of the translational variance at every region center, based on the constructed variance distributions. Finally, the application of the transformation fields to time series is conducted using the Moving Least Squares algorithm [101].

The efficiency of the proposed method is verified through extensive experimentation on 35 datasets from the UCR collection [67]. Our method clearly outperforms DTW-NN on the vast majority of challenging datasets, while being on a par competitive with DTW-NN in the easy (low error) ones. Furthermore, the results indicate that our proposed method always improves the default SVM. The principal contributions of the study can be summarized as:

- A novel time series transformation method is presented
- For the first time, the approach of Invariant SVMs is proposed in time-series domain
- Extensive experimentations are conducted to demonstrate the superiority of the proposed method

## 4.2 Chapter Notations

This chapter will adhere to the notational conventions of Table 4.1.

| Symbol | Explanation |
|---|---|
| $R \in \mathbb{N}$ | The number of time-series regions |
| $\alpha, \beta \in \mathbb{N}^R$ | Control points |
| $\Phi \in \mathbb{N}^R$ | A transformation field |
| $W \in \mathbb{N}^{Q \times Q}$ | Cost matrix |
| $\tau \in \{\mathbb{N} \times \mathbb{N}\}^*$ | Warping path |
| $\mathcal{M} \in \mathbb{N}^{Q \times Q}$ | Warping map |

Table 4.1: Notational conventions of Chapter 4

## 4.3 Proposed Method

### 4.3.1 Principle

In order to boost the classification accuracy, our method needs to generate new instances via transformations. In order to capture the necessary patterns' intra-class variations, the transformation technique should aim for certain characteristics and requirements. In our analysis, the transformations should obey to the following list of properties:

- **Data-Driven:** Variance should be generated by analyzing the similarity distribution of instances inside a class.
- **Localized:** Intra-class variations are often expressed in local deformations, instead of global variations.
- **Selective:** Transforming all the instances becomes computationally expensive and many instances can be redundant w.r.t. the decision boundary. Therefore it is crucial to select only a few class-representative instances for generating variations.
- **Accurate:** The transformed instances should help redefine the decision boundary, however care should be payed to avoid excessive magnitudes of transformation, in order to avoid generating outliers.

### 4.3.2 Method Outline

The transformation method and the instance generation technique we are introducing, does answer all the requirements we raised in section 4.3.1. Initially we

define the local transformation fields in subsection 4.3.3, which are used to transform time series using the Moving Least Squares algorithm. The transformation fields are constructed by measuring the translational variance distributions subsection 4.3.5. The variance distributions are obtained by building the intra-class warping maps, defined in subsection 4.3.4. Finally, the transformation fields are applied only to the support vectors following the Virtual Support Vector classification approach, defined in subsection 4.3.6.

### 4.3.3 Transformation Fields and Moving Least Squares

In this subsection we present only the technicalities of how a time-series dataset can be transformed by a particular localized transformation, while the actual method that creates intelligent transformation magnitudes will be introduced in forthcoming subsections. Variations of time series are often occurring in localized forms, meaning that two series differ only in the deformation of a particular subsequence rather than a global difference. In order to include the mechanism of localized variances we first introduce the concept of a **transformation field**, denoted $\Phi \in \mathbb{R}^R$. We split the time series into $R$ many regions, and define the left/right translation that is required for each region. Each region will be transformed dedicatedly, while the transformation will be applied to the centroid of the region. Such centroids are denoted as **control points**. The amount of translational transformation applied to every control point (hence every region) is denoted as the transformation field vector $\Phi \in \mathbb{R}^R$. For instance Figure 4.2 shows the effect of applying a transformation field on two regions of a time series, where each region is denoted by its representative control point.

The mechanism of applying a transformation field to a series is conducted via the deformation algorithm called Moving Least Squares (MLS) [101], which is described in Algorithm 1. This algorithm is used to transform one signal that passes through a set of points $\alpha \in \mathbb{N}^R$, called control points. The transformation is defined by a new set of control points $\beta \in \mathbb{N}^R$, which are the transformed positions of the control points $\alpha$. The control points $\beta$ are obtained, in our implementation, by applying transformation fields $\Phi$ translations to the original control points $\alpha$. MLS applies the transformation by initially creating one local approximation function $l_v$ for each point $v$ of the time series. Thus, for every point we solve the best affine transformation that approximates the new control points $\beta$ [Line 3 of Alg 1]. There is a weight decay in the importance of the control points compared to the point for which we are defining a local transformation. In

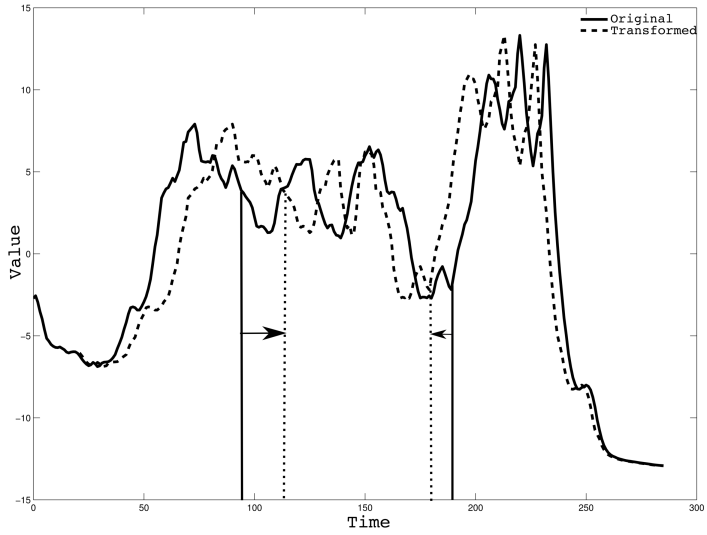



Figure 4.2: Demonstrating the effect of applying a transformation field vector of values [+20 -10] to the control points positioned at [95 190] highlighted with vertical lines, on an instance series from the *Coffee* dataset. The transformation algorithm (MLS) is described in Algorithm 1.

**Algorithm 1** MovingLeastSquares

**Require:** Series $T \in \mathbb{R}^Q$, Transformation field $\Phi \in \mathbb{R}^R$, Control points $\alpha \in \mathbb{N}^R$
**Ensure:** New transformed instance $T^* \in \mathbb{R}^Q$

1: $\beta \leftarrow [\alpha_1 + \Phi_1, \alpha_2 + \Phi_2, \ldots, \alpha_R + \Phi_R]$
2: **for** $v = 1$ to $Q$ **do**
3: Search $l_v$ that minimizes $\underset{l_v:\mathbb{N}\to\mathbb{N}}{\operatorname{argmin}} \sum_{r=1}^{R} w_r |l_v(\alpha_r) - \beta_r|^2$ , where $w_r = \frac{1}{|\alpha_r - v|^{2\gamma}}$
4: $T_v^* \leftarrow T_{l_v(v)}$
5: **end for**
6: **return** $T^*$

that way, the approximation of near control points get more impact. The speed of decay is controlled by a hyper parameter $\gamma \in \mathbb{R}^+$.

Once the local transformation function $l_v$ is computed, the value at point $v$ in the transformed series is computed by applying the learned transformation function over the value in the original series. In order for the transformed series to look similar to the original series, the transformation should be as rigid as possible, that is, the space of deformations should not even include uniform scal-

ing, therefore we follow the rigid transformations optimization [101] in solving line 3 of Alg. 1. This subsection only introduced the transformation fields and the underlying algorithm used to transform a series, while the successive subsections will show how to search for the best magnitudes of the transformation fields vector elements, in order for the transformed instances to encapsulate intra-class variations.

### 4.3.4 Warping Maps

Before introducing the main hub of our method concerning how the variance-generating transformation fields are created, we initially need to present some necessary concepts and means, which are used in the successive subsection to analyze the intra-class variance.

As previously explained in Section 3.2.1, DTW is an algorithm used to compute the similarity/distance between two time series. To help the clarity and make the description self-contained, we are briefly repeating its definition in this section. A cost matrix, denoted $W \in \mathbb{R}^{Q \times Q}$, is build progressively by computing the subtotal warping cost of aligning two series $T_x \in \mathbb{R}^Q$ and $T_z \in \mathbb{R}^Q$ for all pairs $\forall x, z \in [1, \ldots, N]$. The cost is computed recursively until reaching a stopping condition in aligning the first points of the series. The overall cost is accumulatively computed at the topmost index value of the matrix, whose indices correspond to the length of series.

$$\begin{aligned} \mathrm{DTW}(T_x, T_z) &= W_{Q,Q} \\ W_{1,1} &= (T_{x,1} - T_{z,1})^2 \\ W_{i,j} &= (T_{x,i} - T_{z,j})^2 + \min(W_{i-1,j}, W_{i,j-1}, W_{i-1,j-1}) \end{aligned} \tag{4.1}$$

An optimal **warping path** between series $T_x \in \mathbb{R}^Q$ and $T_z \in \mathbb{R}^Q$, defined as $\tau(T_x, T_z) \in \{\mathbb{N} \times \mathbb{N}\}^*$ or here shortly $\tau$, is a list of aligned indexes of points along the cost matrix. The sum of distances of the aligned points along the warping path should sum up to the exact distance cost of DTW. The list of the warping path indexes pairs $(i, j)$ corresponds to the chain of the recursive calls in the cost computation $W_{i,j}$. The sum of the distances among the values of the aligned indexes of two series, yields the minimum distance, which is equal to the DTW formulation.

$$\tau(T_x, T_z) = \{(i, j) \mid W_{i,j} \text{ called in the chain of recursion of } DTW(T_x, T_z)\} \tag{4.2}$$

A **warping map**, denoted $\mathcal{M} \in \mathbb{N}^{Q \times Q}$, is a square matrix whose elements are built by overlapping the warping paths of all-vs-all instances in an equi-length time series dataset. In this overlapping context, a cell of the warping map matrix denotes how often a warping alignment occur at that index. Equation 4.3 formalizes the procedure of building a warping map as a superposition (frequency) of warping paths of all time series pairs $T_x, T_z,\ \forall x, \forall z \in [1, \ldots, N]$ from dataset $T$.

$$\mathcal{M}(i,j) \leftarrow \mid \{(T_x, T_z) \in S^2 \mid (i,j) \in \tau(T_x, T_z)\} \mid \tag{4.3}$$

A filtered warping map is created similarly as shown in Algorithm 2, where we filter only those warping paths whose time indices that are either right (higher indices) or left (smaller indices) aligned at a specific point. For instance, if we need to filter for right alignment at a point $\alpha_r \in \alpha,\ r \in [1, \ldots, R]$, we need to build the DTW warping of any two series pairs, denoted $\tau$, and then check if the aligned index at the second series is higher than (right of) the index on the first series. For instance, the notation $\tau(T_x, T_z)_{\alpha_r}$ denotes the aligned index at series $T_z$ corresponding to time $\alpha_r$ of first series $T_x$.

**Algorithm 2** FilteredWarpingMap

**Require:** Time series $T \in \mathbb{R}^{N \times Q}$, Control Point $\alpha_r \in \mathbb{N}$, Direction $D \in \{right, left\}$
**Ensure:** Filtered warping map $\mathcal{M} \in \mathbb{R}^{\mathbf{Q} \times \mathbf{Q}}$
1: **if** $D = right$ **then**
2: $\quad \mathcal{M}(i,j) \leftarrow \mid \{(T_x, T_z) \in S^2 \mid (i,j) \in \tau(T_x, T_z) \wedge \alpha_r < \tau(T_x, T_z)_{\alpha_r}\} \mid$
3: **else**
4: $\quad \mathcal{M}(i,j) \leftarrow \mid \{(T_x, T_z) \in S^2 \mid (i,j) \in \tau(T_x, T_z) \wedge \alpha_r \geq \tau(T_x, T_z)_{\alpha_r}\} \mid$
5: **end if**
6: **return M**

In our forthcoming analysis we build warping paths by providing a filtered dataset of instances belonging to only one class. Therefore, we will construct one warping map per class.

### 4.3.5 Variance Distribution Analysis and Creation of Transformation Fields

In this section, we present the main method of creating the transformation which is based on the analysis of the variance distributions of warping paths. The trans-

formation fields represent local perturbation vectors of the predefined regions, $R$ many, by translating the representative control points. Each control point/region is translated both left and right, therefore creating $2 \times R$ total transformation fields. The amount of translational transformations to be applied to every region is computed by making use of the warping maps. For each region $R_i$, we filter in turn the right and left warping paths of instance warping alignments at that region, in order to analyze the general left/right translational variances of the warping alignments on other regions as an impact of a transformation at $R_i$. An illustration is found on Figure 4.3. The time series is divided into three regions defined by their centroid control points. In Figure 4.3.b-d) we show the filtered warping maps for the right warping alignments at every control points. Please note that three more filtered warping maps could be created for left alignments but are avoided due to lack of space.

Once we built the warping map, we can successively construct the distribution of the warped alignments at every control points. For every region where we apply left/right translational perturbations, the *transformation field is created to be equal to the means of the warped points*, as an impact of the perturbation. The means are selected as transformation field, because they represents the tendency of variations at every control point. An illustration of the distributions is depicted in Figure 4.4. Only two distribution plots are shown belonging to the warping maps in Figure 4.3.b-c).

**Algorithm 3** ComputeTransformationFields

**Require:** Time series: $T \in \mathbb{R}^{N \times Q}$, Time-series labels $Y \in \{1, \ldots, C\}^N$, Class of instances to be transformed: $y \in \{1, \ldots, C\}$, A list of control points $\alpha \in \mathbb{N}^R$
**Ensure:** List of transformation fields: $L$

$L \leftarrow \emptyset$
$T^L \leftarrow \{\ T_i \mid\ i \in \{1, \ldots, N\}\ \wedge\ Y_i = y\ \}$
**for** $r = 1, \ldots, R$ **do**
  **for** direction $\in \{\text{left}, \text{right}\}$ **do**
    $\mathcal{M} \leftarrow$FilteredWarpingMap$(T^L, \alpha_r, \text{direction})$ from Algorithm 2
    **for** $r' = 1, \ldots, R;\ j = 1, \ldots, R$ **do**
      $\Phi_j \leftarrow \frac{1}{\|\mathcal{M}_{j,*}\|} \sum_{k=1}^{|T^L|} (\mathcal{M}_{j,k} \cdot (k - \alpha_{r'}))$
    **end for**
    $L \leftarrow L \cup \{\Phi\}$
  **end for**
**end for**
**return** $L$

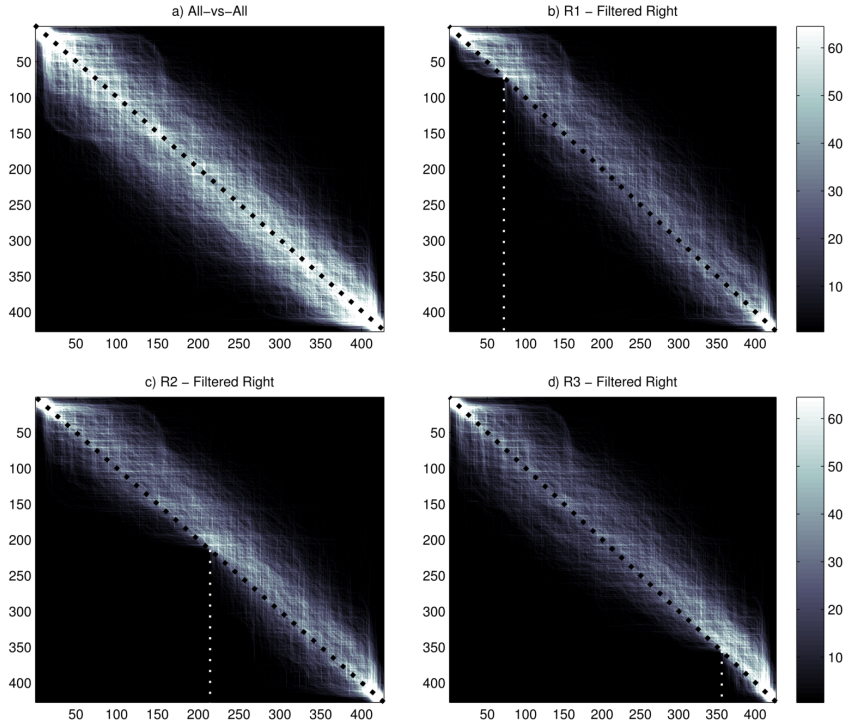



Figure 4.3: An illustration concerning the warping map belonging to label 0 of the *OSULeaf* dataset. The time series are divided into three regions $R_1, R_2, R_3$ for analysis. The center of each region is defined as a control point on time indices [71,213,355]. **a)** All-vs-all warping map. **b)** Warping map created by filtering only right warping paths at control point of $R_1$ at index 71. **c)** Warping map at control point of $R_2$ at index 213. **d)** A similar right warping filter of $R_3$ at 355.

For instance, the mean values of the distributions, which also represent the transformation fields at Figure 4.4.a), represent the warping distributions as an impact of perturbation of $R_1$, where the field is [34 24 0]. We can conclude that a right perturbation at $R_1$ causes a right translational impact on $R_2$ but fades away at $R_3$. Therefore the transformations of instances at $R_1$, will be in proportional to this distribution. Similarly in image Figure 4.4.b) there is a perturbation on $R_2$ which has a stronger impact on $R_1$ than on $R_3$.

Algorithm 3 describes the creation of transformation fields. For every control point [line 3], we analyze the right and left variations [line 4] and get respective filtered warping maps [line 5]. The impact of such variation on other control points [line 6] is taken into consideration by the weighted mean variance at each other control point [line 7].

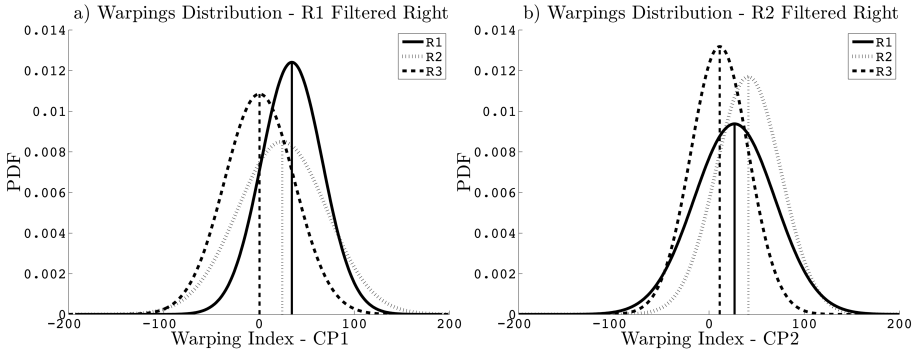



Figure 4.4: Approximation with Gaussian Probability Density Functions (PDF) regarding the warping distribution of filtered warping maps in Figure 4.3, images b and c. CP stands for Control Points (i.e. $\alpha, \beta$). **a)** The warping distribution as a result of right-warping occuring at R1/CP1. **b)** The warping distribution as a result of right-warping occuring at R2/CP2.

### 4.3.6 Learning and Virtual Support Vectors

The defined transformation fields are used during the classification of time series. Even though in principle various classifiers can benefit from larger training set, still transforming all instances deteriorates the learning time of methods. SVMs have a crucial advantage because they point out the important instances (support vectors) which are needed to be transformed. In our study only the support vectors of a SVM model are transformed, called Virtual Support Vectors (VSV) [102]. Such selective approach ensures that the decision boundary is redefined only by instances close to it, hence the support vectors. The training set is extended to include MLS transformations of the support vectors as shown in Equation 4.4. Given i) a list of control points, denoted $\alpha$; ii) a list of transformation fields for each class $c$, denoted $\bar{\Phi}_c, \forall c \in \{1, \ldots, C\}$; iii) a transformation scale factor $\mu$; then Equation 4.4 represents the addition of transformed support vectors obtained by building a model, denoted $svmModel$, from the training set $T_{train}$.

$$T^*_{train} := T_{train} \cup \{ \quad MLS \quad (sv, \mu \cdot \Phi, \alpha) \mid sv \in supportVectors(svmModel) \\ \wedge \quad \Phi \in \bar{\Phi}_{Y_{(sv)}} \} \tag{4.4}$$

Algorithm 4 describes the classification procedure in pseudo code style. The values of the transformation scales are computed by hyper-parameter search on a validation split search during the experiments.

**Algorithm 4** LearnModel

**Require:** Training set of time series $T_{train} \in \mathbb{R}^{N \times Q}$, Labels $Y_{train} \in \{1, \ldots, C\}^N$, SVM hyper parameters $\theta$, Transformation Fields $\bar{\Phi} \in \mathbb{N}^{C \times * \times R}$, Control Points $\alpha \in \mathbb{N}^R$, Transformation Scale: $\mu \in \mathbb{R}^+$
**Ensure:** SVM model: $svmModel$

```
svmModel ← svm.train(T_train, θ)
for sv ∈ supportVectors(svmModel) do
    for Φ ∈ Φ̄_{Y_sv} do
        VSV ← MLS(sv, μ · Φ, α) from Algorithm 1
        T_train ← T_train ∪ {VSV}
    end for
end for
svmModel ← svm.train(T_train, θ)
return svmModel
```

## 4.4 Experimental Setup

In order to evaluate the performance of our proposed method, denoted as Invariant SVM, or shortly ISVM, we implemented and evaluated the following set of baselines:

- **SVM:** The default SVM is a natural choice for a baseline. The performance of our method compared to the standard SVM will give us indications on the success of redefining the decision boundaries by injecting transformed support vectors.
- **DTW-NN:** Characterized as a hard-to-beat baseline in time series classification which has been reported to achieve hard-to-beat classification accuracy [123]. The relative performance of our method compared to DTW-NN will give hints whether a refined maximum-margin is competitive or not.

The UCR collection of time series dataset was selected for experimentation. The largest datasets, whose transformation fields creation exceeded one week of running time, were omitted. All the datasets were randomly divided into five subsets/folds of same size (5-folds cross-validation). Each random subset was stratified, meaning that the number of instances per label was kept equal on all subsets. In turn, each fold was used once for testing the method, while three out of the remaining four for training and one for validation. The inhomogeneous polynomial kernel, $k(x, y) = (\gamma\, x \cdot y + 1)^d$, was applied for both the standard

SVM as well as our method. The degree $d$ was found to perform overall optimal at value of 3. A hyper-parameter search was conducted in order to select the optimal values of the kernel's parameter $\gamma$ and the methods transformation scale $\mu$ of Algorithm 4, by searching for maximum performance on the validation set after building the model on the train set. SVM's parameter $C$ was searched among $\{0.25, 0.5, 1, 2, 4\}$. The number of regions/control points was found to be optimal around 10. The performance is finally tested with a cross-validation run over all the splits.

## 4.5 Results

A cumulative results table involving the experimentation results is found in Table 4.2. For every dataset, the mean and standard deviations of the cross validation error rates is reported in columns. The non-overlapping 1-$\sigma$ confidence interval results, representing significance of ISVM/SVM results versus DTW-NN, are annotated with a circle (°). We grouped the datasets into two categories, easy ones and challenging ones. The criteria of the split is based on an error rate threshold, where values greater than 5% of the default SVM are grouped as easy dataset. The values in bold indicate the best mean error rate for the respective dataset/row. The last row indicates a sum of the wins for each method, where draw points are split to ties. In brackets we denote the wins with significant and non-significant intervals.

The first message of the experiments is that the performance of our method is improving the accuracy of a standard SVM. In various cases like 50words, Cricket_X, Cricket_Y, Cricket_Z, Lighting7, OSULeaf and WordsSynonyms the improvement is very significant ranging from +5% up to +11% accuracy. Thus it is appropriate to use our method for boosting the SVM accuracy, without adding noise.

The second and more important message is that our method produces better mean error rates than DTW-NN, winning on the majority of the datasets. The performance on the easy datasets is even (7 to 7). However, our method outperforms DTW-NN on the majority (11 to 5) of the challenging datasets. The invariant SVM looses significantly only on Cricket_* and Lighting2. In contrast, it significantly outperforms DTW-NN on 50words, Fish, OliveOil, SwedishLeaf, uWaveGestureLibrary_* and WordsSynonyms. Thus, our experiments demonstrate the superiority of our approach to DTW-NN.

The transformation scale parameter introduced in Algorithm 3, controls the scale of the transformation fields perturbations to be applied to the instances.

Table 4.2: **5-fold Cross-Validation Experiments Results** - Error Rate Fractions; *(i)* **ISVM** *: Our proposed method Invariant Support Vector Machines, (ii)* **DTW-NN** *: Nearest Neighbor with Dynamic Time Warping, (iii)* **SVM***: The default Support Vector Machines*

| **Dataset** | **ISVM** | | **DTW-NN** | | **SVM** | |
|---|---|---|---|---|---|---|
| | *mean* | *st.dev.* | *mean* | *st.dev.* | *mean* | *st.dev.* |
| **Easy Datasets** | | | | | | |
| CBF | 0.002 | 0.003 | **0.000** | 0.000 | 0.002 | 0.003 |
| Coffee | °**0.000** | 0.000 | 0.073 | 0.010 | °**0.000** | 0.000 |
| DiatomSizeReduction | °**0.000** | 0.000 | 0.006 | 0.000 | °**0.000** | 0.000 |
| ECGFiveDays | °**0.001** | 0.003 | 0.007 | 0.000 | °**0.001** | 0.003 |
| FaceAll | **0.020** | 0.007 | 0.024 | 0.000 | 0.027 | 0.005 |
| FaceFour | **0.036** | 0.038 | 0.054 | 0.006 | 0.045 | 0.056 |
| FacesUCR | °**0.021** | 0.007 | 0.031 | 0.000 | 0.026 | 0.010 |
| Gun_Point | °**0.030** | 0.027 | 0.075 | 0.001 | 0.050 | 0.040 |
| ItalyPowerDemand | °**0.026** | 0.019 | 0.051 | 0.000 | °**0.026** | 0.019 |
| MoteStrain | 0.050 | 0.016 | **0.045** | 0.000 | 0.060 | 0.020 |
| SonyAIBORobotSurface | °**0.008** | 0.011 | 0.027 | 0.000 | °**0.008** | 0.011 |
| SonyAIBORobotSurfaceII | 0.004 | 0.004 | 0.029 | 0.000 | °**0.003** | 0.005 |
| Symbols | 0.027 | 0.016 | **0.019** | 0.000 | 0.029 | 0.012 |
| synthetic_control | 0.020 | 0.013 | °**0.007** | 0.000 | 0.022 | 0.015 |
| Trace | 0.030 | 0.027 | °**0.000** | 0.000 | 0.045 | 0.048 |
| TwoLeadECG | 0.002 | 0.002 | **0.001** | 0.000 | 0.002 | 0.002 |
| Two_Patterns | 0.001 | 0.001 | °**0.000** | 0.000 | 0.006 | 0.001 |
| wafer | °**0.002** | 0.002 | 0.006 | 0.000 | °**0.002** | 0.001 |
| **Wins (Sig./Non-Sig.)** | **7 (5/2)** | | **7 (3/4)** | | **4 (4/0)** | |
| **Challenging Datasets** | | | | | | |
| 50words | °**0.199** | 0.008 | 0.287 | 0.001 | 0.272 | 0.035 |
| Adiac | °**0.202** | 0.038 | 0.341 | 0.001 | 0.206 | 0.037 |
| Beef | °**0.267** | 0.109 | 0.467 | 0.009 | °**0.267** | 0.109 |
| Cricket_X | 0.300 | 0.025 | °**0.197** | 0.001 | 0.391 | 0.033 |
| Cricket_Y | 0.271 | 0.021 | °**0.213** | 0.001 | 0.388 | 0.045 |
| Cricket_Z | 0.306 | 0.043 | °**0.188** | 0.000 | 0.399 | 0.031 |
| ECG200 | **0.110** | 0.052 | 0.160 | 0.003 | 0.125 | 0.040 |
| Fish | °**0.094** | 0.031 | 0.211 | 0.000 | 0.103 | 0.031 |
| Lighting2 | 0.289 | 0.025 | °**0.099** | 0.002 | 0.297 | 0.013 |
| Lighting7 | **0.252** | 0.054 | 0.285 | 0.010 | 0.357 | 0.068 |
| OliveOil | **0.083** | 0.083 | 0.133 | 0.006 | **0.083** | 0.083 |
| OSULeaf | 0.296 | 0.039 | **0.285** | 0.003 | 0.346 | 0.059 |
| SwedishLeaf | °**0.079** | 0.017 | 0.185 | 0.001 | 0.082 | 0.014 |
| uWaveGestureLibrary_X | °**0.193** | 0.012 | 0.251 | 0.000 | 0.197 | 0.013 |
| uWaveGestureLibrary_Y | °**0.253** | 0.009 | 0.342 | 0.000 | 0.259 | 0.009 |
| uWaveGestureLibrary_Z | °**0.249** | 0.008 | 0.301 | 0.000 | 0.256 | 0.011 |
| WordsSynonyms | °**0.200** | 0.038 | 0.270 | 0.000 | 0.261 | 0.027 |
| **Wins (Sig./Non-Sig.)** | **11 (9/2)** | | **5 (4/1)** | | **1 (0.5/0.5)** | |

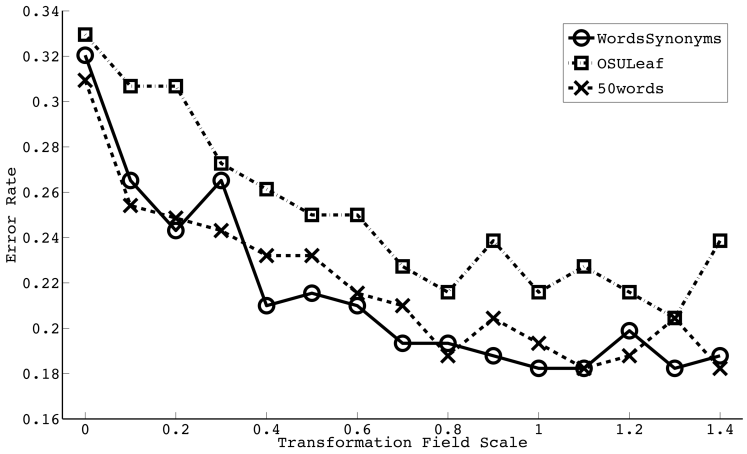



Figure 4.5: The effect of increasing the transformation field scale on three typical datasets. The accuracy improves proportionally with the scale, until a minimum point is reached. After the optimal error rate, the large transformations produce noise and deteriorate accuracy, as for instance in OSULeaf, after minimum at scale value 1.3.

Intuitively, optimal transformation fields redefine the decision boundary while excessive magnitudes of transformations deteriorate into noisy instances. A demonstration of the transformation fields' scale parameter behavior is presented in Figure 4.5.

Finally, it is worth mentioning that the time complexity of our method is obviously worse than that of a normal SVM, because of the enlarged training set. Yet, the computational time is not prohibitive in terms of run-time feasibility. In Table 4.3 you can find some test run times in minutes, of typical prototypes of easy and challenging datasets, with the aim of demonstrating the run-time feasibility of our method compared to DTW-NN. The run-time minutes shown on the last column are measured over the same random dataset fold.

## 4.6 Comparison to General Data Augmentation

Are existing off-the-shelf data augmentation techniques sufficient for classifying time-series data? In other words, are standard methods that augment generic feature-vector data able to compete with our tailored time-series transformation approach. In order to answer this question, we compared the prediction quality of our method against a popular data augmentation baseline called SMOTE [43].

Table 4.3: Classification Run Times of Typical Dataset

| **Dataset** | **# labels** | **# instances** | **length** | **ISVM** (min) | **SVM** (sec) |
|---|---|---|---|---|---|
| Coffee | 2 | 56 | 286 | 0.01 | 0.59 |
| ECG200 | 2 | 200 | 96 | 0.04 | 0.94 |
| wafer | 2 | 7174 | 152 | 5.48 | 2.71 |
| FacesUCR | 14 | 2250 | 131 | 8.73 | 2.60 |
| 50words | 50 | 905 | 270 | 14.17 | 14.8 |
| Cricket_X | 12 | 780 | 300 | 24.00 | 3.1 |

Table 4.4: MCRs of ISVM versus SMOTE

| **Dataset** | **ISVM** | **SMOTE** | **Dataset** | **ISVM** | **SMOTE** |
|---|---|---|---|---|---|
| 50words | **0.199** | 0.273 | Lighting7 | **0.252** | 0.287 |
| Adiac | **0.202** | 0.206 | MoteStrain | **0.050** | 0.073 |
| Beef | 0.267 | **0.133** | OliveOil | 0.083 | **0.067** |
| CBF | **0.002** | 0.008 | OSULeaf | **0.296** | 0.355 |
| Coffee | **0.000** | **0.000** | Sony | **0.008** | 0.010 |
| CricketX | **0.300** | 0.442 | SonyII | **0.004** | 0.018 |
| CricketY | **0.271** | 0.524 | Swedish. | **0.079** | 0.105 |
| CricketZ | **0.306** | 0.451 | Symbols | **0.027** | 0.030 |
| Diatom | **0.000** | **0.000** | synthetic. | **0.020** | 0.045 |
| ECG200 | **0.110** | 0.115 | Trace | **0.030** | 0.080 |
| ECGF. | 0.001 | **0.000** | TwoPatt. | **0.001** | 0.058 |
| FaceAll | **0.020** | 0.034 | TwoL. | 0.002 | **0.001** |
| FaceFour | **0.036** | 0.054 | uWaveX | **0.193** | 0.246 |
| FacesUCR | **0.021** | 0.039 | uWaveY | **0.253** | 0.318 |
| Fish | **0.094** | 0.106 | uWaveZ | **0.249** | 0.296 |
| GunPoint | 0.030 | **0.025** | Wafer | **0.002** | **0.002** |
| ItalyPower | **0.026** | 0.033 | WordsS. | **0.200** | 0.288 |
| Lighting2 | 0.289 | **0.248** | | | |
| **Wins** | **13** | **5** | **Wins** | **15** | **2.5** |

A SVM was used to classify the data augmented by SMOTE, while the hyper-parameters of the SVM were selected in the same manner as ISVM, specified in Section 4.4. The results of the experiment are provided in Table 4.4. As can be clearly seen, the method proposed in this chapter, denoted as ISVM, outperforms SMOTE in 28 out of 35 datasets. Given the large superiority, it is evident that one needs specific augmentation approaches (as our transformations). Consequently, off-the-shelf classifiers that treat time-series as feature vectors under-perform in terms of prediction accuracy.

## 4.7 Conclusion

In this chapter we introduced a novel instance transformation method, which is used to boost the performance of SVM via transforming the support vectors. The proposed method utilizes the distribution of warping variances, yield from warping alignment maps, in order to define transformation fields, which represent variances at a predefined set of local regions of a particular class. Therefore, the virtual support vectors which are generated by applying the transformation fields, represent the necessary intraclass variation and redefines the maximum margin decision boundary. The superiority of our method is demonstrated by extensive experimentations on 35 datasets of the UCR collection. In a group of easy datasets, the presented method is *on par* competitive to the baselines, while being clearly superior on a set of challenging datasets.

# Chapter 5

# Time-Series Factorization

## Contents

**Highlights:** Intra-class variations occur in local segments of time-series, which are either shifted in time, or distorted. This chapter presents a novel method that extracts local segments from time series independent of their location. The presented approach is a dimensionality reduction tailored for time series, which converts the series into a new representation, which reflects sums of local patterns. In that way, the new representation is invariant to the location of the patterns. Moreover, the dimensionality reduction avoids including noisy patterns that arise from local distortions.

In contrast to existing approaches, the new representation proposed in this paper decomposes a time-series dataset into latent patterns and membership weights of local segments to those patterns. The process is formalized as a constrained objective function and a tailored stochastic coordinate descent optimization is applied. The time-series are projected to a new feature representation consisting of the sums of the membership weights, which captures frequencies of local patterns. Features from various sliding window sizes are concatenated in order to encapsulate the interaction of patterns from different sizes. The derived representation offers a set of features that boosts classification accuracy. Finally, a large-scale experimental comparison, against 11 baselines over 43 real life datasets, indicates that the proposed method achieves state-of-the-art prediction accuracy results.

## 5.1 Introduction

This chapter introduces a new representation of time series, which can capture patterns that are invariant to shifts and scales. We assume that time series are generated by a set of latent (hidden) patterns which occur at different time stamps and different frequencies across instances. In addition those patterns might be

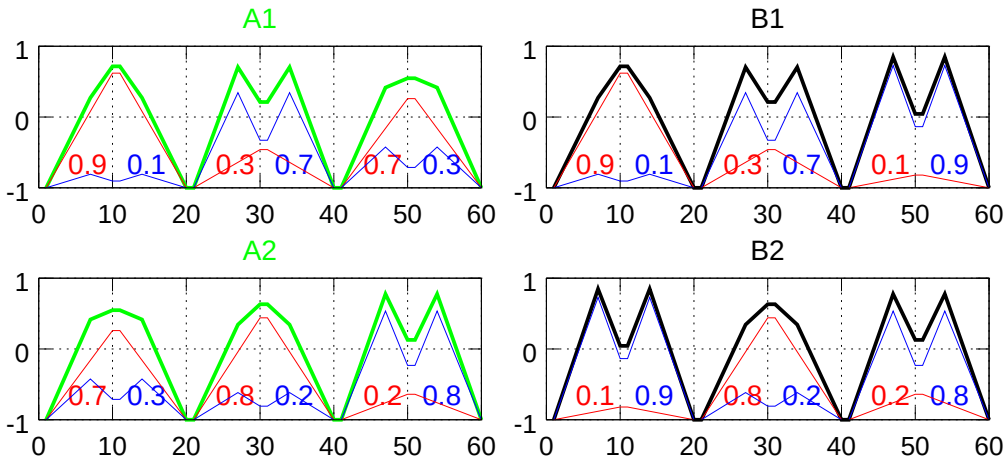



Figure 5.1: Four series of two classes A={A1,A2} and B={B1,B2}, each generated as a convolution of latent patterns

convoluted and/or distorted to produce derived local patterns. We would like to introduce the concept through the illustration of Figure 5.1. A synthetic dataset consists of two classes A (green) and B (black), each having two instances. All the time series are composed of three segments of 20 points, while each segment is a convolutional derivative of two latent patterns depicted in red and blue. In other words, each segment is a weighted sum of a single-peaked and double-peaked pattern. The shown coefficients of the convolution are degrees of membership that each local segment has to one of those two latent patterns.

Both Euclidean and DTW based nearest neighbor classifiers have 100% error on a leave-one-out experiment on the dataset of Figure 5.1. As can be observed, instance A1 is closer to B1 than A2, and the same applies for all other series. In fact the rationale behind this dataset is that A has a higher frequency of the red single-peaked pattern, while B has a higher domination of the blue double-peaked pattern. The method presented in this chapter detects the latent patterns, measures the degrees of membership and sums them up into a bag-of-pattern approach. Our approach converts the series of Figure 5.1 into a new representation F, concretely: $F_{A1} = [1.9, 1.1], F_{A2} = [1.7, 1.3], F_{B1} = [1.3, 1.7], F_{B2} = [1.1, 1.9]$. A nearest neighbor classifier over the new representation F yields 0% error.

In this chapter, we will propose a method which detects a set of latent patterns for a time series dataset together with a convolutional degree of membership weights. Such a decomposition is a tailored dimensionality reduction for time-series. The product of the membership weights with the patterns approximates

the original segments. In contrast to the aforementioned synthetic example, real datasets have segments occurring at arbitrary locations and being of different sizes. Our method employs a sliding window approach to split the series into overlapping local segments and utilizes a factorization model to decompose the segments into latent patterns and weights. We formalize the objective function of the factorization and propose a stochastic coordinate descent technique in order to optimize the objective. The sum of the learned membership degrees is used to project the time series into a new representation. Ultimately, in order to resolve the scale invariance of the patterns, sums of memberships from different sliding window sizes are concatenated.

Patterns in time-series often occur at different time indices, which is a behavior known as "shifts". The factorization method we propose in this chapter, captures weights of patterns in a sliding window segmentation and is invariant to the position of a pattern. In addition, time-series patterns often appear in different scales/sizes. Our method is invariant to the scale of the pattern because we factorize patterns corresponding to various sliding window sizes. Therefore, we name the proposed factorization as "invariant" to shifts and scale variations of patterns.

A thorough experimental comparison is conducted on 43 datasets of the UCR time-series collection against six state of the art baselines. Our method achieves state-of-the-art results in terms of prediction accuracy.

## 5.2 Chapter Notations

This chapter will adhere to the notational conventions of Table 5.1 and the forthcoming brief explanation.

| Symbol | Explanation |
|---|---|
| $D \in \mathbb{R}^{N \times M \times K}$ | Degrees of Membership |
| $\mathcal{C} : \mathbb{R}^L \to \mathbb{R}$ | Distance of a segment to the closest centroid |
| $\Phi \in \mathbb{N}$ | Scales of patterns |
| $\alpha \in \mathbb{R}$ | Sum of two segments' membership weights |

Table 5.1: Notational Conventions of Chapter 5

- **Degrees of Membership:** Each instance of a dataset will be approximated via the product of latent patterns and the set of membership degrees to those patterns. Each segment of a series will have one membership weight

to each of the $K$ latent patterns. Consequently, the degrees of membership of all time-series are defined as $D \in \mathbb{R}^{N \times M \times K}$.

## 5.3 Factorization of Time Series

The method presented in this chapter is a new feature representation for time series data. The representation reduces the dimensionality of the original series by factorizing the series data into a set of patterns and weights of segments to those patterns. The sum of weights over all the sliding window segments of a time-series is the new feature vector that our method constructs. The factorization process is fully unsupervised and does not take into account the label information. However, the derived representation provides a set of features that boost the classification accuracy of standard classifiers because the new representation is invariant to shifts and scales of patterns. This section will walk the reader through the details of our method, including the learning algorithms.

### 5.3.1 Segmentation of Time Series

As a first step, the series of the dataset are segmented in a sliding window approach having size $L$ and increment $\delta$. The segmentation of each series is described in Algorithm 5. Once derived, the segments are normalized to mean 0 and deviation 1.

**Algorithm 5** SegmentSeries

**Require:** Time series $T \in \mathbb{R}^{N \times Q}$, Segment length $L \in \mathbb{N}$, Sliding window increment $\delta \in \mathbb{N}$
**Ensure:** $S \in \mathbb{R}^{N \times M \times L}$
1: $M \leftarrow \frac{Q-L}{\delta}$
2: **for** $i = 1, \ldots, N,\ j = 1, \ldots, M$ **do**
3:  **for** $l = 1, \ldots, L$ **do**
4:  $S_{i,j,l} \leftarrow T_{i,\,\delta(j-1)+l}$
5:  **end for**
6:  $S_{i,j} \leftarrow \text{normalize}(S_{i,j})$
7: **end for**
8: **return** $S$

### 5.3.2 From the Problem Definition to an Objective Function

The definition of our problem is to learn degrees of memberships $D \in \mathbb{R}^{N \times M \times K}$ and the patterns $P \in \mathbb{R}^{K \times L}$ that reconstruct/approximate the segments $S \in \mathbb{R}^{N \times M \times L}$. A linear convolution $DP$ is used as the model that reconstructs the segments. The loss of reconstruction is measured using the squared-error (L2) error, while two different types of regularization are applied on $D$ and $P$.

Therefore, the objective function is defined as a regularized loss and is described in Equation 5.1. The solution of the objective function returns the optimal $D, P$ matrices that cause the loss achieve its minimal value.

$$\underset{D,P}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{l=1}^{L} \left( S_{i,j,l} - \sum_{k=1}^{K} D_{i,j,k} P_{k,l} \right)^2 + \lambda_P \sum_{k=1}^{K} \sum_{l=1}^{L} P_{k,l}^2 \tag{5.1}$$

Subject To:

$$\sum_{k=1}^{K} D_{i,j,k} = 1, \quad D_{i,j,k} \geq 0, \quad \forall i, j, k$$

The objective function is composed of two loss terms and one constraint. Firstly, the latent patterns $P$ and the memberships $D$ should approximate the normalized segments of the series dataset. Therefore, minimizing the L2 norm of the reconstruction error achieves the goal. In addition, a second regularization loss term is added in order to prohibit the patterns $P$ from over-fitting. A hyper-parameter $\lambda_P$ controls the degree of regularization. Finally, we impose equality and positivity constraints on the membership degrees. The membership degrees of every segment $D_{i,j}$ sum-up to one, because each segment needs to have the same impact factor. Otherwise, in a bag-of-patterns representation of series, different segments would have different scales of memberships. The positivity constraint, on the other hand, prohibits non-interpretable negative memberships. It is worth noting that the model is similar in spirit to factorization models for soft-clustering [38]. The novelty here relies on applying a per-segment factorization tailored for time-series data.

We would like to illustrate the invariant factorization objective with a concrete illustration, shown in Figure 5.2. A learned decomposition, as in Equation 5.1, is depicted for the Gun Point dataset. On the left top, a series instance is presented, while the dataset's latent patterns and the membership degrees of the instance are found below. The product of the patterns and memberships yield the series

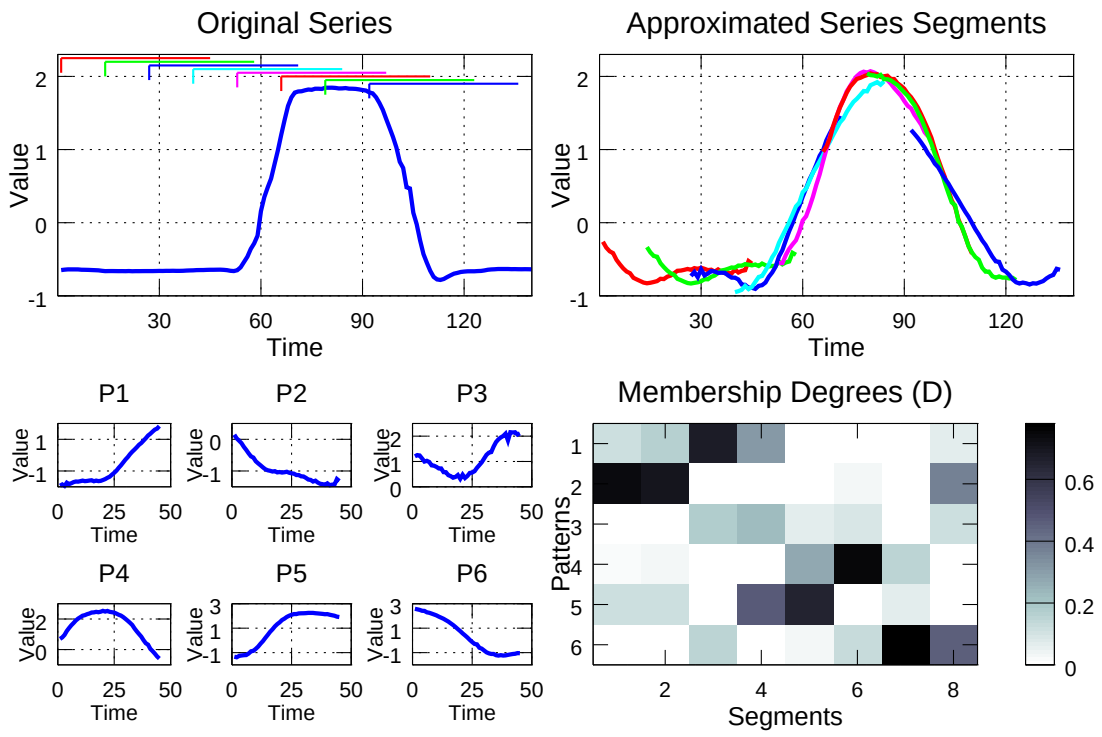



Figure 5.2: A factorized instance of the Gun Point dataset with parameters $K = 6, L = 45, \delta = 13, \lambda_P = 1$

approximation shown in the right top chart. The series is split into 8 overlapping segments of size 45, each starting at an offset of 13 points. For instance, the 7-th segment starts at 79 and has a high membership value to the 6-th pattern, which matches the descending structure. However, please note that other patterns also contribute with smaller membership degrees (patterns 4 and 5) in order to fit exactly the original segment content.

### 5.3.3 Learning the Patterns and Memberships

In order to learn the latent patterns and the memberships we are going to optimize the objective function of Equation 5.1 via **stochastic coordinate descent**, which operates by updating each cell of $D, P$ in the direction of the first derivative of the objective.

#### 5.3.3.1 Update Rules for Latent Patterns

In order to compute the update rules for the patterns, we first define in Equation 5.2 the error in approximating a point $l$ of the segment $j$, in time-series $i$, as $\xi_{i,j,l}$. A stochastic coordinate descent optimization fixes the error of approximating all points $S_{ijl}$ and is different to gradient approaches that consider the full error.

$$\text{Let } \xi_{i,j,l} := S_{i,j,l} - \sum_{k=1}^{K} D_{i,j,k} P_{k,l} \tag{5.2}$$

The optimization technique learns the optimal values of the patterns and membership weights, which eliminate the residual of each point $\xi_{i,j,l}$. In the case of a pattern point $P_{k,l}$, the optimal value can be found by isolating the residual that the point contributes from the error. Such an optimization technique is named as "Coordinate Descent" and is popular for the factorization community [128]. In our context, we try to isolate the residual error of $P_{k,l}$ by introducing a placeholder variable $z$ as is shown in Equation 5.3. The optimal value of $z$ that minimizes the error $\xi_{i,j,l}$, subject to the regularization, is denoted as $P^*_{k,l}$

$$P^*_{k,l} := \underset{z}{\operatorname{argmin}} \left( \lambda_P z^2 + \sum_{i,j} \left( \xi_{i,j,l} + P_{k,l} D_{i,j,k} - z D_{i,j,k} \right)^2 \right) \tag{5.3}$$

Subsequently the optimal value of every point $l$ of a latent pattern $k$ (denoted as $P^*_{k,l}$) is found by solving the first derivative as presented in Equation 5.4.

$$2\lambda_P P^*_{k,l} - 2 \sum_{i,j} \left( \xi_{i,j,l} + D_{i,j,k} (P_{k,l} - P^*_{k,l}) \right) D_{i,j,k} = 0 \tag{5.4}$$

Therefore, the optimal value $P^*_{k,l}$ is defined in Equation 5.5 as a derivation of Equation 5.4. Please note that our learning algorithm will iterate through the error of all the segment points $\xi_{i,j,l}$ and then update all $P_{k,l}$ cells to the optimal value with respect to the $\xi_{i,j,l}$.

$$P^*_{k,l} := \frac{\sum_{i,j} \left( \xi_{i,j,l} + D_{i,j,k} P_{k,l} \right) D_{i,j,k}}{\lambda_P + \sum_{i,j} D^2_{i,j,k}} \tag{5.5}$$

Please note that the error values don't have to be recomputed for each point over all latent patterns, instead we can incrementally update the error terms [128]. Equation 5.6 refreshes the error terms after the change of the pattern value.

$$\xi_{i,j,l} \leftarrow \xi_{i,j,l} - (P^*_{k,l} - P_{k,l})D_{i,j,k} \tag{5.6}$$

#### 5.3.3.2 Update Rules for Membership Degrees

The update rules for the membership degrees needs to preserve an equality constraint, which enforce the memberships of a segment to sum to one. Therefore, any direct update of a membership $D_{i,j,k}$ will violate the constraint. In order to avoid this bottleneck, we propose to update the memberships in pairs, inspired by a similar strategy known as the Sequential Minimal Optimization algorithm [92]. The idea is to draw two random membership weights $D_{i,j,k}, D_{i,j,w}$ and update them such that their sum, denoted $\alpha = D_{i,j,k} + D_{i,j,w}$, remains equal before and after the updates. In that way, if we increase one membership, the other would have to decrease and vice versa, while the aim is to find the combination which yields the smallest approximation error.

Since the value of $D_{i,j,k}$ depends on the other value of the pair $D_{i,j,w}$, we can no longer find the minimum value of the points separately. The optimization should find the combination of pair values that both minimize the residual error $\xi_{i,j,l}$ of approximating a segment point $S_{i,j,l}$. Therefore, the optimal value of $D_{i,j,k}$ will be denoted by $D^*_{i,j,k}$ and can be solved by isolating the residual error of both values by introducing a variable $z$ and creating an optimization sub-problem [128]. Since the sum of the pair of membership values is bound to $\alpha$, we can replace the optimal value of $D_{i,j,w}$ as $\alpha - z$. The resulting optimal value is expressed in Equation 5.7.

$$D^*_{i,j,k} = \underset{z}{\operatorname{argmin}} \sum_l \left(\xi_{i,j,l} + D_{i,j,k}P_{k,l} + D_{i,j,w}P_{w,l} - \alpha P_{w,l} + z(P_{w,l} - P_{k,l})\right)^2 \tag{5.7}$$

The solution of Equation 5.7 can be algebraically derived as the solution of the first derivative and is presented in Equation 5.8. Our forthcoming learning algorithm will update pairs of membership weights for all the points of series segments in a series of iterations.

$$D^*_{i,j,k} = \frac{-\sum_l \left(\xi_{i,j,l} - D_{i,j,k}\left(P_{w,l} - P_{k,l}\right)\right)\left(P_{w,l} - P_{k,l}\right)}{\sum_l \left(P_{w,l} - P_{k,l}\right)^2} \tag{5.8}$$

Once the optimal value $D^*_{i,j,k}$ is defined, we have to ensure the constraints. Equation 5.9 crops the optimal value to be nonnegative and not exceed the sum $\alpha$ of the membership pairs. As mentioned during the description of the objective function, we constraint the membership values to be non-negative and sum to one.

After updating the pair of membership weights, we can also update the reconstruction error term $\xi_{i,j,l}$. The error terms are refreshed in order to avoid recomputing the error of Equation 5.2 before every update. Therefore the computation of the error terms can be reduced from $\mathcal{O}(K)$ to $\mathcal{O}(1)$. Equations 5.10 and 5.11 define the steps of updating the errors as a result of changing $D_{i,j,k}$ and $D_{i,j,w}$.

As a last step, we can commit the optimal values by preserving their sum before the updates. As Equation 5.12 represents, the best value of $D_{i,j,w}$ can be deduced from the optimal value of $D_{i,j,k}$.

- Crop value of $D_{i,j,k}$, keep it between $[0, \alpha]$:
$$D^*_{i,j,k} \leftarrow \max(0, \min(\alpha, D^*_{i,j,k})) \tag{5.9}$$
- Update error $\xi_{i,j,l}$ pursuant to changing $D_{i,j,k}$:
$$\xi_{i,j,l} \leftarrow \xi_{i,j,l} - (D^*_{i,j,k} - D_{i,j,k})P_{k,l} \tag{5.10}$$
- Update error $\xi_{i,j,l}$ pursuant to changing $D_{i,j,w}$:
$$\xi_{i,j,l} \leftarrow \xi_{i,j,l} - (\alpha - D^*_{i,j,k} - D_{i,j,w})P_{w,l} \tag{5.11}$$
- Set $D_{i,j,k}, D_{i,j,w}$ to their optimal values:
$$D_{i,j,k} \leftarrow D^*_{i,j,k}, \quad D_{i,j,w} \leftarrow \alpha - D^*_{i,j,k} \tag{5.12}$$

### 5.3.4 Analogies to Fuzzy Clustering

The factorization approach of this chapter is similar to a fuzzy K-Means clustering [14]. The latent patterns $P$ of our approach can be perceived as cluster centroids. In a similar fashion, every time-series segment can be assigned to each of the centroids $P$, through cluster membership weights $D$. The per-segment weights $D$ are similar to cluster membership degrees. Therefore, one can view our approach as an application of fuzzy clustering to series segments. However, it is worth pointing that in contrast to the EM-style heuristic optimization of the fuzzy K-Means [14], we proposed a tailored stochastic coordinate descent. In addition, our method has a L2 regularization of the patterns, while the fuzzy clustering applies no regularization to the centroids.

### 5.3.5 Efficient Initialization

Since the objective function of Equation 5.1 is non-convex in terms of $P$ and $D$, a coordinate descent optimization is not guaranteed to avoid local optima. Therefore, good initial values of the patterns and the memberships are crucial for the learning process. The intuition leads into assigning some of the segments as initial patterns. Unfortunately, it is not obvious which of them provide the best initialization.

The answer is addressed via a technique (known as KMeans++) utilized to find the initial centroids in a clustering setup [4]. The patterns (analogy to centroids) are initialized to segments with a probability proportional to the distance to all the other segments [4]. Therefore, we are assured to pick centroid segments which are evenly distributed across the space of all series segments. The initialization steps are detailed in Algorithm 6. Please note that the first pattern has to be drawn randomly in a uniform distribution, while the other patterns are chosen randomly from the dataset segments based on the probability of their distance to the existing patterns. The function $\mathcal{C}$ measures the distance of a segment to the closest existing pattern.

**Algorithm 6** Initialize

**Require:** $S \in \mathbb{R}^{N \times M \times L}, L \in \mathbb{N}, K \in \mathbb{N}$
**Ensure:** $D \in \mathbb{R}^{N \times M \times K}, P \in \mathbb{R}^{K \times L}$
1: $P_1 \leftarrow S_{i',j'}$, drawn $i', j' \sim \mathcal{U}(N, M)$
2: **for** $k = 2, \ldots, K$ **do**
3: $\quad P_k \leftarrow S_{i',j'}$, with probability weights $\frac{\mathcal{C}(S_{i',j'})^2}{\sum_{i,j} \mathcal{C}(S_{i,j})^2}$
4: **end for**
5: **for** $i = 1, \ldots, N;\ j = 1, \ldots, M$ **do**
6: $\quad k' = \text{argmin}_{k \in \{1,\ldots,K\}} ||S_{i,j} - P_k||^2$
7: $\quad D_{i,j,k} \leftarrow \begin{cases} 1 & k = k' \\ 0 & k \neq k' \end{cases}, \quad k = 1, \ldots, K$
8: **end for**
9: **return** $D, P$

The initialization of the membership degrees is more trivial than patterns. The degree index $k'$ denotes that pattern $P_{k'}$ is the closest to segment $S_{i,j}$ and its membership $D_{i,j,k'}$ is set to 1, while all the other membership degrees are initialized to zero.

### 5.3.6 Learning Algorithm

Algorithm 7 finally combines all the steps of the factorization process. In the beginning, the memberships and the patterns are initialized using Algorithm 6. Next the errors are initialized, then the coordinate descent technique updates all the parameters in a number of iterations, denoted as a hyper-parameter $\mathcal{I}$. Subsequently, the degrees of membership and the patterns are learned by setting the aforementioned optimal values. The membership and pattern indexes are visited in random order to speed up the convergence.

For the sake of clarity, we are going to describe the steps of the algorithm in a line by line fashion. In line 1 the segmentation of series using Algorithm 5 is run, while in line 2 we initialize the membership weights $D$ and the patterns $P$ via Algorithm 6. The lines 4 to 6 initialize the errors $\xi$ with the initial reconstruction error between segments $S$ and their convolutional reconstruction $DP$. Once the initializations are completed, the invariant factorization learns the matrices $D, P$ in a series of iterations that are located in lines 8-37. For all the series $i$ of the dataset and the sliding window segments $j$, our method learns all the $K$-many pairs of degrees of memberships and patterns. The learning of membership degrees and the refreshing of errors as a result of the update of degrees, located in lines 8-26, is a direct mirroring of Equations 7-12. The update of the latent patterns is conducted through lines 28-37 of the algorithm and uses the defined update rules of Equations 5-6.

### 5.3.7 A New Invariant Representation

The final representation will sum the membership degrees in a bag-of-patterns strategy. It enables a quantification of which local patterns appear in a series and how often. The shift invariance is achieved by segmenting the series in a sliding window approach and the scale invariance is addressed using different sliding window sizes. Algorithm 8 describes the algorithmic steps. The algorithm iterates over $\Phi$ many different scales of an initial sliding windows size $L$ and solves an invariant factorization from Algorithm 7 per each size. The frequencies of the learned memberships are summed up for all $K$ patterns and the procedure is repeated for every sliding window size. Finally each time series contains $K\Phi$ many features, which denote the frequencies of patterns at different sizes and positions.

More concretely, in line 3 we apply a factorization with an initial sliding window size $L'$ and receive in return the factorized membership weights $D$ from Algorithm 7. Then a set of $K$ features, denoted $F$, can be constructed as the

**Algorithm 7** InvariantFactorization

**Require:** Time series $T \in \mathbb{R}^{N \times Q}$, Segment and pattern length $L \in \mathbb{N}$, Sliding window segments $\delta \in \mathbb{N}$, Number of patterns $K \in \mathbb{N}$, Regularization hyper-parameter $\lambda_P \in \mathbb{R}$, Number of iterations $\mathcal{I} \in \mathbb{N}$

**Ensure:** Degrees of membership $D \in \mathbb{R}^{N \times M \times K}$, Patterns $P \in \mathbb{R}^{K \times L}$

$S \leftarrow \text{SegmentSeries}(T, L, \delta)$
$(D, P) \leftarrow \text{Initialize}(S, L, K)$
{Initialize the errors}
**for** $i = 1, \ldots, N;\ j = 1, \ldots, M;\ l = 1, \ldots, L$ **do**
    $\xi_{i,j,l} := S_{i,j,l} - \sum_{k=1}^{K} D_{i,j,k} P_{k,l}$
**end for**
{Update the patterns&memberships iteratively}
**for** $1, \ldots, \mathcal{I}$ **do**
    {Update all degrees of membership}
    **for** $i = 1, \ldots, N;\ j = 1, \ldots, M$ randomly **do**
        **for** $1, \ldots, K$, {Draw K-many pairs} **do**
            $k, w \sim \mathcal{U}(K, K)$, s.t. $D_{i,j,k} + D_{i,j,w} \neq 0$
            $\alpha \leftarrow D_{i,j,k} + D_{i,j,w}$
            {Solve and crop the optimal memberships}
            $D^*_{i,j,k} = \frac{-\sum_l (\xi_{i,j,l} - D_{i,j,k}(P_{w,l} - P_{k,l}))(P_{w,l} - P_{k,l})}{\sum_l (P_{w,l} - P_{k,l})^2}$
            $D^*_{i,j,k} \leftarrow \max\left(0, \min(\alpha, D^*_{i,j,k})\right)$
            {Update the error terms}
            **for** $l = 1, \ldots, L$ **do**
                $\xi_{i,j,l} \leftarrow \xi_{i,j,l} - (D^*_{i,j,k} - D_{i,j,k}) P_{k,l}$
                $\xi_{i,j,l} \leftarrow \xi_{i,j,l} - (\alpha - D^*_{i,j,k} - D_{i,j,w}) P_{w,l}$
            **end for**
            {Commit the values of the pair}
            $D_{i,j,k} \leftarrow D^*_{i,j,k}$
            $D_{i,j,w} \leftarrow \alpha - D^*_{i,j,k}$
        **end for**
    **end for**
    {Update all patterns}
    **for** $k = 1, \ldots, K;\ l = 1, \ldots, L,$ randomly **do**
        $P^*_{k,l} = \frac{\sum_{i,j} (\xi_{i,j,l} + D_{i,j,k} P_{k,l}) D_{i,j,k}}{\lambda_P + \sum_{i,j} D^2_{i,j,k}}$
        {Update the error terms}
        **for** $i = 1, \ldots, N;\ j = 1, \ldots, M$ **do**
            $\xi_{i,j,l} \leftarrow \xi_{i,j,l} - (P^*_{k,l} - P_{k,l}) D_{i,j,k}$
        **end for**
        {Commit the pattern's point value}
        $P_{k,l} \leftarrow P^*_{k,l}$
    **end for**
**end for**
**return** $D, P$

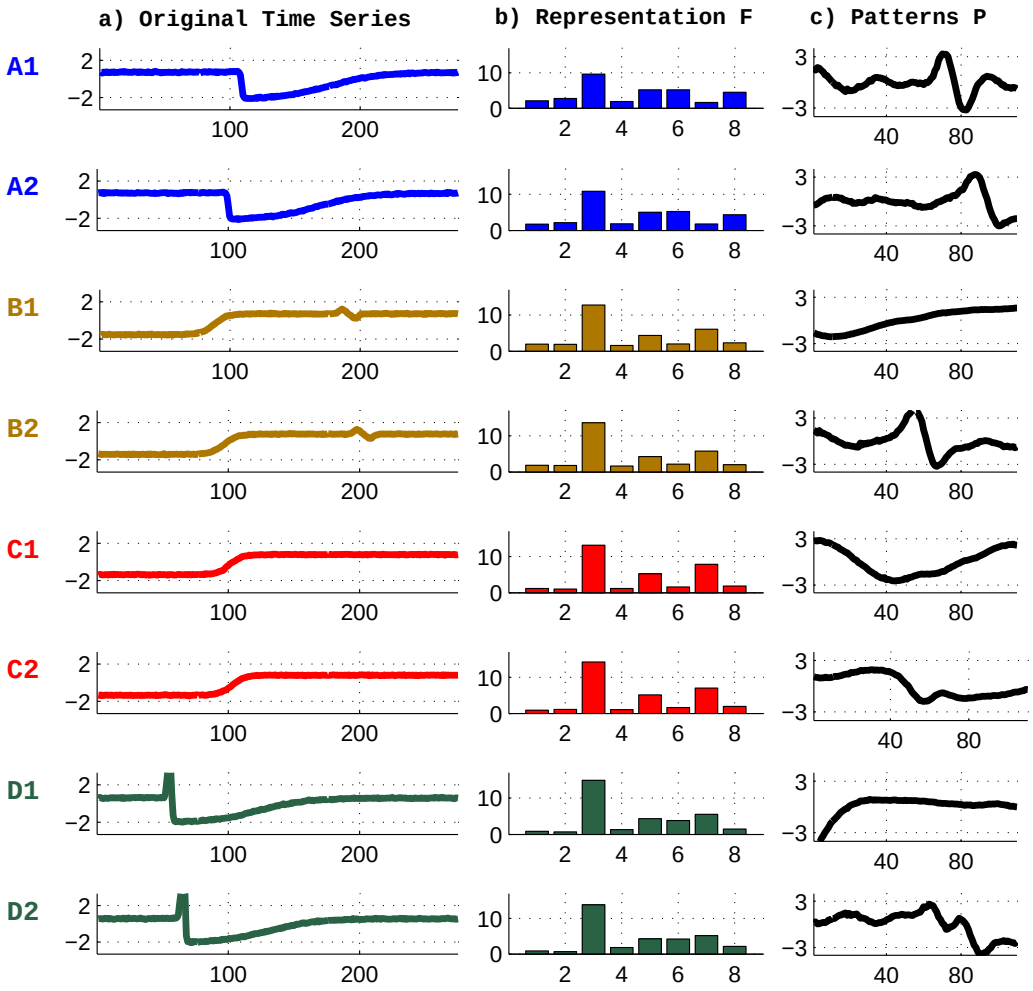



Figure 5.3: Illustration of the original time series (left) and the factorized representation $F$ (center) of eight series from the Trace dataset. The series belong to four different classes (A,B,C,D) shown in colors. On the right, the patterns of the factorization are displayed. Parameters: $K = 8$, $L = 110$, $\Phi = 1$, $\delta = 1$, $\lambda_P = 0.01$, $I = 15$.

sum of the $K$-many dimensions of $D$ by summing up the weights of every sliding window of a series. The summation step is located in line 6. Then the sliding window size $L'$ is incremented in line 8 and the algorithm continues with a bigger sliding window size. In each iteration we extract $K$ factorization features that are appended to $F$.

The new representation will be used for classification, instead of the original time series. We deployed a polynomial kernel Support Vector Machines, because we need to capture the interaction among features, i.e. the interaction among patterns of various sizes.

**Algorithm 8** InvariantRepresentation

**Require:** Time-series data $T \in \mathbb{R}^{N \times Q}$, Segment and pattern length $L \in \mathbb{N}$, Sliding window threshold $\delta \in \mathbb{N}$, Number of patterns $K \in \mathbb{N}$, Hyper-parameter settings $\lambda_P \in \mathbb{R}$, Number of iterations $\mathcal{I} \in \mathbb{N}$, Scales of pattern lengths $\Phi \in N$

**Ensure:** New representation $F \in \mathbb{R}^{N \times (K\Phi)}$

1: $L' \leftarrow L$
2: **for** $s = 1, \ldots, \Phi$ **do**
3: $\quad D \leftarrow \text{InvariantFactorization}(T, \underline{L'}, \delta, K, \lambda_P, I)$
4: $\quad$ **for** $i = 1, \ldots, N;\ k = 1, \ldots, K$ **do**
5: $\quad\quad M \leftarrow \frac{Q - L'}{\delta}$
6: $\quad\quad F_{i,k+(s-1)K} \leftarrow \sum_{j=1}^{M} D_{i,j,k}$
7: $\quad$ **end for**
8: $\quad L' \leftarrow L' + L$
9: **end for**
10: **return** $F$

We would like to explain the resulting representation ($F$) that is produced by the proposed method, with the aid of Figure 5.3. Eight time series from the Trace dataset (shown in the left side) are factorized into a set of weights ($D$) and a set of eight patterns ($P$) (shown on the right side). We assign an ordinal number to the patterns from 1 to 8 in a top-down order. The new representation (middle plots) is the sum over all the sliding window weights of each pattern (line 6 in Algorithm 8). The reduced dimensionality not only is more compact, but also can help understand the differences among classes. For instance, classes B and C have a very subtle difference that is reflected as a perturbation around point 200. In our latent representation, class B instances have higher weights of patterns 1 and 2 that reflect the subtle perturbation while class C instances have lower weights for the patterns 1 and 2. It is possible to realize with mere human inspection that the factorized representation could detect the difference

between classes by inspecting the derived representation $F$, which is the sum of $D$ membership degrees. In fact, a SVM applied over the derived representation $F$ achieves 0% error on the Trace dataset.

### 5.3.8 Algorithmic Complexity

The run-time complexity of the method is dominated by the updates of memberships and has an order $O(NMKL\mathcal{I})$. Concretely, our method needs 48.4 hours to compute on the StarLightCurves (the largest) dataset, while for instance DTW needs 87 hours. We used the standard DTW version, without lower bounding pruning and warping window restrictions. The space complexity of our method depends on the storage of the segments $S$ and the memberships $D$, which is $O(NM \max(K, L))$.

### 5.3.9 Inductive and Transductive Factorizations

Our factorization method is a fully unsupervised dimensionality reduction that projects all the time series of a dataset to a new, reduced representation. As Algorithm 7 explained, the learning process needs to have access to a batch of time-series instances, in order to jointly minimize the reconstruction error of Equation 5.1. However, the method proposed in this chapter can operate in two modes: *Inductive* and *Transductive*. The transductive case factorizes all the instances jointly, using both the training and the testing predictors (not labels). On the other hand, the inductive mode factorizes the training instances first and then *folds in* the testing predictors *one at a time* to the latent representation, without modifying the patterns.

It is worth clarifying that in contrast to modern semi-supervised methods that use the test predictors to **directly** alter the decision boundary of their classifiers during the learning of the discriminative model, our transductive dimensionality reduction is unsupervised. Once the data is factorized to a reduced representation, we classify the reduced instances using a standard SVM that does not utilize the predictors of the test instances.

The strategy of enabling an inductive operation of our method relies on a so-called 'Fold In' factorization. The 'Fold In' learns a factorization from a batch of available training time-series data and stores the $D, P$ representation. Then, the 'folding in' refers to the process of converting a test time series $T_t \in \mathbb{R}^Q$ to $D_t \in \mathbb{R}^{M \times L}$ by learning only one instance $D_t$ from Equation 5.1, keeping all the other $D, P$ constant. For triviality, the update rule of learning $D_t$ is avoided because it is the same as lines 10-25 of Algorithm 7. In Figure 5.4 the inductive

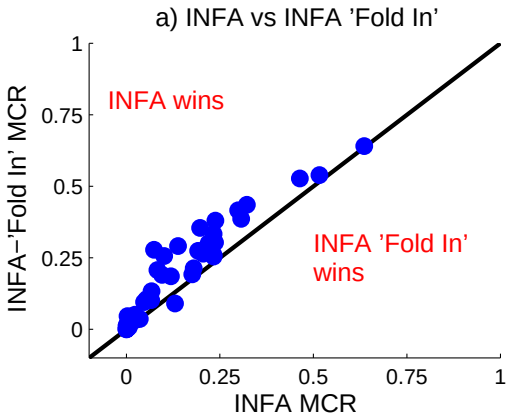



Figure 5.4: Error rate comparison of the transductive INFA against the inductive 'Fold In' variant of INFA.

'Fold In' (denoted as INFA 'Fold In') mode is compared to the transductive factorization (denoted as INFA) in terms of error rates. Results show that the accuracy of the inductive 'Fold In' mode is inferior to the transductive INFA, which is due to the fact that folding in does not fully minimize the reconstruction objective of Equation 5.1, but simply the weights $D$ of one test instance at a time. As a result, the total reconstruction error will be higher, meaning that the factorized data will not be anymore a close representation of the original data, leading to a deteriorated prediction accuracy in case the inductive representation is fed to a SVM.

## 5.4 Experimental Results

We will keep refering to our Invariant Factorization via acronyms, INFA for the transductive case and shortly INFAI for the inductive case, throughout the remaining parts of this document. This chapter proposes a new representation of time series that is computed using an invariant factorization, not at all a classification method. Nevertheless, we apply a standard Support Vector Machines (SVM) upon the reduced dimensionality representation, in order to classify time series. For making the narration smoother, we are often going to refer to INFA as a joint method that includes a factorized representation plus an SVM over the factorized instances.

### 5.4.1 Baselines

We compared the prediction accuracy of our method, denoted **INFA** for the transductive mode and **INFAI** for the inductive mode, against the following seven state of the art baselines:

- **TSBF:** The bag-of-features framework for time series (TSBF) uses a supervised codebook to extract features for a random forest classifier [12].
- **SSSK:** Sparse Spatial Similarity Kernel (SSSK) measures sequence similarity through sampling sequence features at different resolutions [73].
- **BOW:** The Bag of Words (BOW) method decomposes the series into local SAX words and uses a histogram representation of words as the new feature representation [77, 78].
- **NN:** The nearest neighbor classifier with Euclidean distance based similarity of series is a classical, yet competitive, approach in the time-series domain.
- **DTW:** Dynamic Time Warping (DTW) computes the best alignment of time indexes resulting in the mininal distance [69, 94].
- **CID:** The complexity invariant distance (CID) adds a L2-based total variation regularization term into the Euclidean distance [9].
- **FSH:** Fast shapelet (FSH) extracts the most discriminative segment of the series dataset, such that the distance from the dataset instances to the optimal shapelet can be used as a feature for classification [95].

### 5.4.2 Setup and Reproducibility

We conducted a large-scale experimentation in 43 time-series dataset from the UCR collection[1]. Our protocol complied to the default train/test split of the data, which is an established benchmark split and is used by the baselines. The metric of comparison is the error rate, i.e. the misclassification rate. Table 5.2 shows the datasets used for experimentation together with the number of classes, the number of training instances, the number of testing instances and the length of the series.

[1] `www.cs.ucr.edu/~eamonn/time_series_data`

Table 5.2: Error Rates - Comparison of Prediction Accuracies on the UCR Collection of Datasets

| Dataset | Cls. | Train | Test | Len. | INFA | INFAI | TSBF | SSSK | BOW | ENN | DTW | CID | FSH |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 50words | 50 | 450 | 455 | 270 | 0.220 | 0.301 | **0.209** | 0.488 | 0.316 | 0.369 | 0.310 | 0.336 | 0.557 |
| Adiac | 37 | 390 | 391 | 176 | 0.322 | 0.435 | **0.245** | 0.575 | 0.325 | 0.389 | 0.396 | 0.373 | 0.514 |
| Beef | 5 | 30 | 30 | 470 | **0.233** | 0.333 | 0.287 | 0.633 | 0.267 | 0.333 | 0.500 | 0.367 | 0.447 |
| CBF | 3 | 30 | 900 | 128 | **0.001** | 0.001 | 0.009 | 0.090 | 0.048 | 0.148 | 0.003 | 0.016 | 0.053 |
| Chlorine | 3 | 467 | 3840 | 166 | 0.464 | 0.526 | **0.336** | 0.428 | 0.405 | 0.350 | 0.352 | 0.351 | 0.417 |
| CinCECG | 4 | 40 | 1380 | 1639 | 0.138 | 0.291 | 0.262 | 0.438 | 0.164 | **0.103** | 0.349 | 0.084 | 0.174 |
| Coffee | 2 | 28 | 28 | 286 | **0.000** | 0.000 | 0.004 | 0.071 | 0.036 | **0.000** | 0.179 | **0.000** | 0.068 |
| CricketX | 12 | 390 | 390 | 300 | **0.205** | 0.264 | 0.278 | 0.585 | 0.305 | 0.423 | 0.223 | 0.372 | 0.527 |
| CricketY | 12 | 390 | 390 | 300 | **0.197** | 0.354 | 0.259 | 0.654 | 0.313 | 0.433 | 0.208 | 0.421 | 0.505 |
| CricketZ | 12 | 390 | 390 | 300 | **0.192** | 0.274 | 0.263 | 0.574 | 0.295 | 0.413 | 0.208 | 0.405 | 0.547 |
| Diatom | 4 | 16 | 306 | 345 | **0.003** | 0.046 | 0.126 | 0.173 | 0.111 | 0.065 | 0.033 | 0.065 | 0.117 |
| ECG200 | 2 | 100 | 100 | 96 | 0.130 | 0.090 | 0.145 | 0.220 | **0.110** | 0.120 | 0.230 | **0.110** | 0.227 |
| ECGF. | 2 | 23 | 861 | 136 | **0.001** | 0.001 | 0.183 | 0.360 | 0.164 | 0.203 | 0.232 | 0.218 | 0.004 |
| FaceAll | 14 | 560 | 1690 | 131 | 0.238 | 0.380 | 0.234 | 0.369 | 0.238 | 0.286 | **0.192** | 0.269 | 0.411 |
| FaceFour | 4 | 24 | 88 | 350 | **0.000** | 0.011 | 0.051 | 0.102 | 0.102 | 0.216 | 0.170 | 0.193 | 0.090 |
| FacesUCR | 14 | 200 | 2050 | 131 | **0.083** | 0.207 | 0.090 | 0.356 | 0.137 | 0.231 | 0.095 | 0.235 | 0.328 |
| Fish | 7 | 175 | 175 | 463 | **0.023** | 0.051 | 0.080 | 0.177 | 0.029 | 0.217 | 0.167 | 0.217 | 0.197 |
| GunPoint | 2 | 50 | 150 | 150 | **0.007** | 0.007 | 0.011 | 0.133 | 0.407 | 0.087 | 0.093 | 0.073 | 0.061 |
| Haptics | 5 | 155 | 308 | 1092 | 0.516 | 0.539 | **0.488** | 0.591 | 0.630 | 0.630 | 0.623 | 0.584 | 0.616 |
| InlineSkate | 7 | 100 | 550 | 1882 | 0.636 | 0.640 | **0.603** | 0.729 | 0.629 | 0.658 | 0.616 | 0.629 | 0.741 |
| ItalyPower | 2 | 67 | 1029 | 24 | **0.036** | 0.036 | 0.096 | 0.101 | 0.044 | 0.045 | 0.050 | 0.044 | 0.095 |
| Lighting2 | 2 | 60 | 61 | 637 | 0.180 | 0.213 | 0.257 | 0.393 | 0.328 | 0.246 | **0.131** | 0.246 | 0.295 |
| Lighting7 | 7 | 70 | 73 | 319 | **0.233** | 0.260 | 0.262 | 0.438 | 0.370 | 0.425 | 0.274 | 0.397 | 0.403 |
| MALLAT | 8 | 55 | 2345 | 1024 | 0.047 | 0.095 | 0.037 | 0.153 | 0.098 | 0.086 | 0.066 | 0.075 | **0.033** |
| Medical. | 10 | 381 | 760 | 99 | 0.299 | 0.102 | 0.269 | 0.463 | 0.401 | 0.316 | **0.263** | 0.309 | 0.433 |
| MoteStrain | 2 | 20 | 1252 | 84 | **0.066** | 0.102 | 0.135 | 0.166 | 0.177 | 0.121 | 0.165 | 0.212 | 0.217 |
| OliveOil | 4 | 30 | 30 | 570 | **0.067** | 0.133 | 0.090 | 0.300 | 0.233 | 0.133 | 0.133 | 0.133 | 0.213 |
| OSULeaf | 6 | 200 | 242 | 427 | **0.095** | 0.190 | 0.329 | 0.326 | 0.153 | 0.479 | 0.409 | 0.438 | 0.359 |
| Sony | 2 | 20 | 601 | 70 | **0.101** | 0.256 | 0.175 | 0.376 | 0.409 | 0.304 | 0.275 | 0.185 | 0.315 |
| SonyII | 2 | 27 | 953 | 65 | **0.054** | 0.105 | 0.196 | 0.339 | 0.154 | 0.141 | 0.169 | 0.123 | 0.215 |
| StarLight. | 3 | 1000 | 8236 | 1024 | **0.021** | 0.031 | 0.022 | 0.135 | **0.021** | 0.151 | 0.093 | 0.057 | 0.063 |
| Swedish. | 15 | 500 | 625 | 128 | **0.074** | 0.278 | 0.075 | 0.339 | 0.125 | 0.211 | 0.210 | 0.123 | 0.269 |
| Symbols | 6 | 25 | 995 | 398 | **0.026** | 0.034 | 0.034 | 0.184 | 0.088 | 0.101 | 0.050 | 0.084 | 0.068 |
| synthetic. | 6 | 300 | 300 | 60 | 0.013 | 0.033 | 0.008 | 0.067 | 0.017 | 0.120 | **0.007** | 0.050 | 0.081 |
| Trace | 4 | 100 | 100 | 275 | **0.000** | 0.000 | 0.020 | 0.300 | **0.000** | 0.240 | **0.000** | 0.140 | 0.002 |
| TwoPatt. | 4 | 1000 | 4000 | 128 | 0.003 | 0.016 | 0.001 | 0.087 | 0.010 | 0.093 | **0.000** | 0.121 | 0.114 |
| TwoL. | 2 | 23 | 1139 | 82 | **0.002** | 0.018 | 0.046 | 0.257 | 0.248 | 0.253 | 0.096 | 0.232 | 0.090 |
| uWaveX | 8 | 896 | 3582 | 315 | 0.176 | 0.192 | **0.164** | 0.358 | 0.242 | 0.261 | 0.273 | 0.238 | 0.293 |
| uWaveY | 8 | 896 | 3582 | 315 | **0.237** | 0.303 | 0.249 | 0.493 | 0.352 | 0.338 | 0.366 | 0.290 | 0.392 |
| uWaveZ | 8 | 896 | 3582 | 315 | 0.233 | 0.254 | **0.217** | 0.439 | 0.325 | 0.350 | 0.342 | 0.291 | 0.364 |
| Wafer | 2 | 1000 | 6174 | 152 | **0.002** | 0.003 | 0.004 | 0.029 | 0.010 | 0.005 | 0.020 | 0.006 | 0.004 |
| WordsS. | 25 | 267 | 638 | 270 | 0.307 | 0.386 | **0.302** | 0.553 | 0.371 | 0.382 | 0.351 | 0.357 | 0.594 |
| yoga | 2 | 300 | 3000 | 426 | **0.119** | 0.185 | 0.149 | 0.172 | 0.145 | 0.170 | 0.164 | 0.164 | 0.269 |
| **Absolute Total Wins** | | | | | **25.16** | | **8.00** | **0.00** | **1.33** | **1.33** | **5.33** | **0.83** | **1.00** |
| **INFAI One-To-One Wins** | | | | | | | **18** | **40** | **25** | **31** | **24** | **29** | **39** |
| **INFAI One-To-One Draws** | | | | | | | **0** | **0** | **1** | **3** | **1** | **2** | **0** |
| **INFAI One-To-One Losses** | | | | | | | **25** | **3** | **17** | **9** | **18** | **12** | **4** |
| **INFA One-To-One Wins** | | | | | | | **30** | **42** | **38** | **39** | **35** | **38** | **41** |
| **INFA One-To-One Draws** | | | | | | | **0** | **0** | **1** | **1** | **1** | **1** | **0** |
| **INFA One-To-One Losses** | | | | | | | **13** | **1** | **4** | **3** | **7** | **4** | **2** |
| **Wilcoxon Signed-Rank Test ($p$-values)** | | | | | | | **0.004** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** |

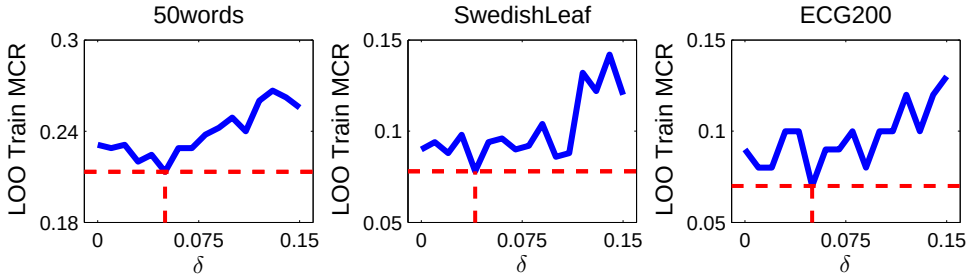



Figure 5.5: Effect of alternating the sliding window threshold $\delta$ on the Leave-One-Out (LOO) train miss-classification error (MCR). Other parameters: $K = 0.5 \times Q, L = 0.2 \times Q, \Phi = 4, C = 1.0, I = 15$.

Our method has a relatively high number of hyper-parameters, however they can be elegantly tuned via a grid hyper-parameter search. First of all, the set of eligible regularization parameters is $\lambda_P \in \{0.001, 1\}$. The sliding window size was searched from $L = \{0.15, 0.2\} \times Q$, and the number of latent dimensions from $K \in \{0.25, 0.5\} \times Q$. The sliding window scale was picked from $\Phi = \{3, 4\}$, while the sliding window offset was searched from $\delta = \{0.00, 0.05\} \times L$. The maximum number of iterations was set to $\mathcal{I} = 15$ in all cases. The applied classifier was a polynomial kernel SVM with a polynomial degree being 3 and the complexity parameter searched among $C \in \{0.1, 1, 10\}$. Out parameter tuning runs the invariant factorization and the subsequent SVM using all the combinations of parameters and selects that parameter combination which achieve the smallest leave-one-out (LOO) error on the training set. We define the parameters that result in the smallest LOO error as the optimal values. The error rate values to be detailed in this chapter refer to the test set results when run with the optimal parameter combination. Since the algorithm is based on a probabilistic initialization, it might be possible that it converges to different closeby optima in each execution. However, in our experiments, those optima were very close and the final prediction accuracy results have insignificant differences.

Most methods increment the sliding window by an offset of a single point at a time. While such an approach is practical and avoids the need to fit the offset parameter, it doesn't provide the optimal accuracy. Figure 5.5 illustrate the sensitivity of the Leave-One-Out (LOO) training error as a result of changing the $\delta$ parameter in a *ceteris-paribus* principle (all other parameters kept constant). The scale of $\delta$ is the percentage of the sliding window length $L$. As can be observed the optimal offset is a value that is small enough, but not the smallest, i.e. not one. Our method selects the optimal sliding window increment by finding the parameter $\delta$ that minimizes the leave-one-out cross validation search, in a grid search.

### 5.4.3 Results

The error rate results of the six state-of-the-art baselines and our method INFA are presented in Table 5.2. The best performing method for each dataset (row) is emphasized in bold. In order to compare multiple classifiers across a large number of datasets we follow the established benchmarks of counting wins and *Wilcoxon's Signed-Rank* test for statistical significance [37]. To be fair with the baselines, we retrieved the results from the baselines' publications [9, 12, 95] over the **same** data splits as INFA. In addition, we verified the published results of the baselines with our own experimental verifications.

Three comparative figures are conducted, the first of which counts the absolute number of wins. Each dataset awards a total value of 1, which is split into equal fractions in case methods have equal error rate scores. The "Absolute wins" row, in the bottom of the table, counts the datasets where a method has the best prediction accuracy. As can be trivially deduced, our method has a clear superiority in terms of absolute wins, scoring 25.16 wins against 8.00 wins of the second best method. In addition, the transductive INFA outperforms by large margins all the baselines in an one-to-one comparisons of wins. INFA has more wins, yet the predominant analysis is whether or not those wins represent statistically significant differences. Each cell on the bottom row represents the p value of the Wilcoxon Signed-Rank test on the error rate values of INFA against each baseline. Our method has a statistically significant difference over the error results of all baselines with a two-tailed hypothesis and the standard significance level of 95% confidence ($p \leq 0.05$).

The transductive mode INFA is obviously always better than the inductive version INFAI, because INFAI learns a partial objective function, as previously explained and demonstrated in Section 5.3.9. Yet, the results for the 'Fold-In' mode (INFAI) indicate that our inductive variant is also very competitive. In a one-to-one comparison the inductive mode outperforms all the baselines, except TSBF. Moreover, the difference of TSBF to INFAI is not significant according to the Wilcoxon Signed-Rank test ($p = 0.075$). On the other hand, INFAI outperforms strong popular methods such as DTW by 24 wins, 1 draws and 18 losses; or similarly CID by 29 wins, 2 draws and 12 losses.

### 5.4.4 On the Need of INFA as a Dimensionality Reduction Method

In its essence, our method can be perceived as a special variant of a PCA or Matrix Factorization that is tailored for time-series data. Instead of reducing

the dimensionality (a.k.a. factorization) of the full time series, our method INFA factorizes sliding window segments and sums up the factorization coefficients (analogous to PCA weights).

Factorizing local segments has several advantages compared to the factorization of the full segment. First the weights of patterns are captured independent to the locations of those patterns (shift variations of series). Secondly, INFA takes into account the size of patterns by applying a factorization of various sliding window sizes. Furthermore, since the sliding window can be arbitrarily large (up to the full series length), then INFA includes PCA in terms of functionality.

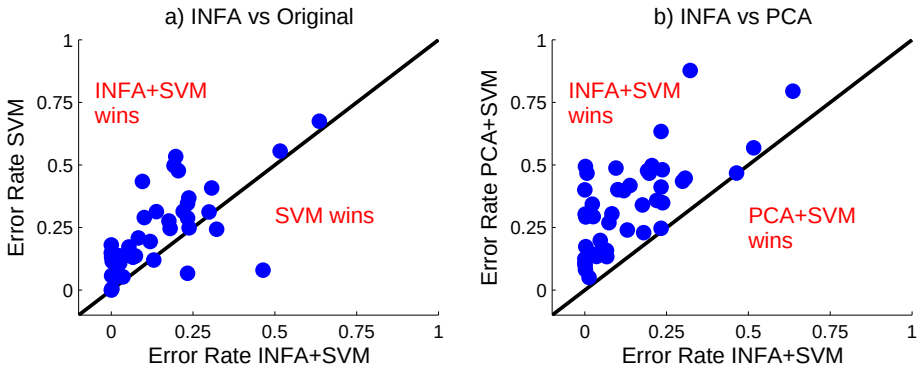



Figure 5.6: a) Comparison of Error Rates of INFA+SVM against SVM without dimensionality reduction; b) Comparison of error rates over time-series factorized using two different dimensionality reduction techniques INFA vs. PCA.

Nevertheless, two experimental tests are required to support our claims. First of all, we should be able to empirically demonstrate whether or not our proposed factorization is required at all, meaning to check how the same classifier (SVM) would perform on the original time-series data. Figure 5.6, sub-plot a), shows the error rates of INFA+SVM against SVM over the original data using all the 43 time series of our experimental setup. As can be easily deduced from the illustration INFA outperforms largely a SVM without factorization. Yet, we would like to also show that the success belongs to our specific factorization method and not due to any dimensionality reduction method. In sub-plot b) we compare the error rates of SVM over factorized time-series using either our method or a standard PCA. The results indicate that our method, as expected, largely outperform the PCA representation.

### 5.4.5 Comparison To Semi-supervised Methods

Whilst our factorization method is unsupervised, yet the transductive operation mode factorizes all predictors of a dataset, including the testing predictors (detailed in Section 5.3.9). The utilization of all predictors (including train and test) for dimensionality reduction, even-though fully unsupervised, raises questions on comparisons against semi-supervised methods. Therefore, this section is dedicated to comparing the prediction quality of our method INFA against state-of-the-art methods that focus on the semi-supervised classification of time series.

We selected two state-of-the-art methods for comparison. The first method by Wei et al. is a well-established alternative in semi-supervised classification of time series [118]. The classifier is trained on an initial training set with positive labeled instances and in an iterative manner the unlabeled instances are mined for enlarging the training set. In addition, a recent method in semi-supervised time-series classification, named SUCCESS, utilizes a combination of constrained hierarchical clustering and DTW [83].

Table 5.3 contains the results of our method INFA against the two state-of-the-art methods denoted as Wei et al. [118] and SUCCESS [83]. The time series belong to the UCR collection of datasets, same as the ones described in Section 5.3. Both leave-one-out training and testing error rate results are shown for all the methods, with the emphasis naturally being on the test scores. The method that achieves the minimum error is highlighted for every dataset. Two different comparative figures are derived. First of all, the total wins indicate that our method INFA has 33.5 wins against the 6.5 wins of SUCCESS, the closest baseline. Secondly, the same superiority is demonstrated by the very high number of one-to-one wins of INFA against both Wei et al. and SUCCESS. All the one-to-one wins are statistically significant as shown in the last row by the Wilcoxon Signed Rank test (significant for $p < 0.05$).

### 5.4.6 Prediction Ahead of Time

Sometimes it is important to be able to tell ahead of time the prediction accuracy of the method, by looking only at the training set [9]. In order to conduct this comparison, first we define the concept of accuracy gain. The gain is defined as the ration of the accuracies among our proposed method INFA and selected baselines, i.e.: $\mathcal{G}\text{ain} = \frac{1-\text{Error INFA}}{1-\text{Error Baseline}}$. The smaller the error of INFA compared to the baseline, the bigger the gain in accuracy. The 'Expected Gain' is defined as

Table 5.3: Error rate results of INFA against state-of-the-art semi-supervised methods

| Dataset | Classes | Train Error | | | Test Error | | |
|---|---|---|---|---|---|---|---|
| | | Wei et al. | SUCCESS | INFA | Wei et al. | SUCCESS | INFA |
| 50words | 50 | 0.432 | 0.398 | **0.213** | 0.436 | 0.414 | **0.220** |
| Adiac | 37 | 0.607 | 0.582 | **0.326** | 0.601 | 0.595 | **0.322** |
| Beef | 5 | 0.683 | 0.656 | **0.400** | 0.617 | 0.600 | **0.233** |
| CBF | 3 | 0.007 | 0.002 | **0.000** | 0.005 | 0.003 | **0.001** |
| ChlorineConcentration | 3 | 0.373 | **0.062** | 0.381 | 0.350 | **0.101** | 0.464 |
| CinCECGTorso | 4 | 0.021 | **0.001** | 0.125 | 0.019 | **0.001** | 0.138 |
| Coffee | 2 | 0.429 | 0.368 | **0.000** | 0.460 | 0.440 | **0.000** |
| CricketX | 12 | 0.477 | 0.425 | **0.167** | 0.465 | 0.444 | **0.205** |
| CricketY | 12 | 0.463 | 0.405 | **0.154** | 0.433 | 0.396 | **0.197** |
| CricketZ | 12 | 0.443 | 0.395 | **0.167** | 0.459 | 0.423 | **0.192** |
| DiatomSizeReduction | 4 | 0.018 | **0.017** | 0.063 | 0.031 | 0.025 | **0.003** |
| ECG200 | 2 | 0.237 | 0.225 | **0.080** | 0.239 | 0.195 | **0.130** |
| ECGFiveDays | 2 | 0.051 | 0.021 | **0.000** | 0.053 | 0.030 | **0.001** |
| FaceFour | 4 | 0.201 | 0.191 | **0.000** | 0.182 | 0.200 | **0.000** |
| FacesUCR | 14 | 0.080 | 0.062 | **0.060** | 0.083 | **0.070** | 0.083 |
| Fish | 7 | 0.424 | 0.449 | **0.040** | 0.403 | 0.434 | **0.023** |
| GunPoint | 2 | 0.089 | 0.039 | **0.000** | 0.075 | 0.045 | **0.007** |
| Haptics | 5 | 0.671 | 0.706 | **0.426** | 0.704 | 0.730 | **0.516** |
| InlineSkate | 7 | 0.693 | 0.679 | **0.470** | 0.683 | 0.663 | **0.636** |
| ItalyPowerDemand | 2 | 0.063 | 0.073 | **0.030** | 0.066 | 0.076 | **0.036** |
| Lighting2 | 2 | 0.355 | 0.322 | **0.150** | 0.342 | 0.317 | **0.180** |
| Lighting7 | 7 | 0.463 | 0.477 | **0.243** | 0.536 | 0.529 | **0.233** |
| Mallat | 8 | 0.042 | 0.041 | **0.018** | **0.042** | 0.037 | 0.047 |
| MedicalImages | 10 | 0.379 | 0.386 | **0.252** | 0.394 | 0.393 | **0.299** |
| MoteStrain | 2 | 0.124 | 0.129 | **0.050** | 0.115 | 0.107 | **0.066** |
| OliveOil | 4 | 0.300 | 0.315 | **0.100** | 0.367 | 0.383 | **0.067** |
| OSULeaf | 6 | 0.550 | 0.512 | **0.105** | 0.532 | 0.466 | **0.095** |
| SonyAIBORobotS. | 2 | 0.052 | 0.090 | **0.050** | **0.060** | 0.110 | 0.101 |
| SonyAIBORobotS.II | 2 | 0.088 | 0.094 | **0.074** | 0.079 | 0.087 | **0.054** |
| StarLightCurves | 3 | 0.119 | 0.200 | **0.022** | 0.140 | 0.200 | **0.021** |
| SwedishLeaf | 15 | 0.330 | 0.369 | **0.068** | 0.364 | 0.379 | **0.074** |
| Symbols | 6 | 0.033 | **0.022** | 0.040 | 0.025 | **0.019** | 0.026 |
| SyntheticControl | 6 | 0.051 | 0.029 | **0.010** | 0.065 | 0.045 | **0.013** |
| Trace | 4 | 0.054 | 0.001 | **0.000** | 0.050 | **0.000** | **0.000** |
| TwoPatterns | 4 | **0.000** | **0.000** | 0.002 | 0.000 | **0.000** | 0.003 |
| TwoLeadECG | 2 | 0.004 | 0.001 | **0.000** | 0.003 | **0.001** | 0.002 |
| uWaveGestureX | 8 | 0.276 | 0.284 | **0.173** | 0.284 | 0.286 | **0.176** |
| uWaveGestureY | 8 | 0.356 | 0.368 | **0.211** | 0.377 | 0.377 | **0.237** |
| uWaveGestureZ | 8 | 0.359 | 0.378 | **0.219** | 0.368 | 0.385 | **0.233** |
| Wafer | 2 | 0.009 | 0.009 | **0.000** | 0.009 | 0.009 | **0.002** |
| WordsSynonyms | 25 | 0.414 | 0.378 | **0.247** | 0.410 | 0.382 | **0.307** |
| Yoga | 2 | 0.148 | 0.149 | **0.130** | 0.152 | 0.151 | **0.119** |
| **Absolute Total Wins** | | **0.50** | **4.50** | **37.00** | **2.00** | **6.50** | **33.50** |
| **INFA One-to-one Wins** | | **37** | **40** | **-** | **36** | **37** | **-** |
| **INFA One-to-one Draws** | | **0** | **0** | **-** | **0** | **1** | **-** |
| **INFA One-to-one Losses** | | **5** | **2** | **-** | **6** | **4** | **-** |
| **Wilcoxon Signed Rank (p value)** | | **0.000** | **0.000** | **-** | **0.000** | **0.000** | **-** |

the leave-one-out error on the training set, while the the 'Real Gain' is defined as the test error.

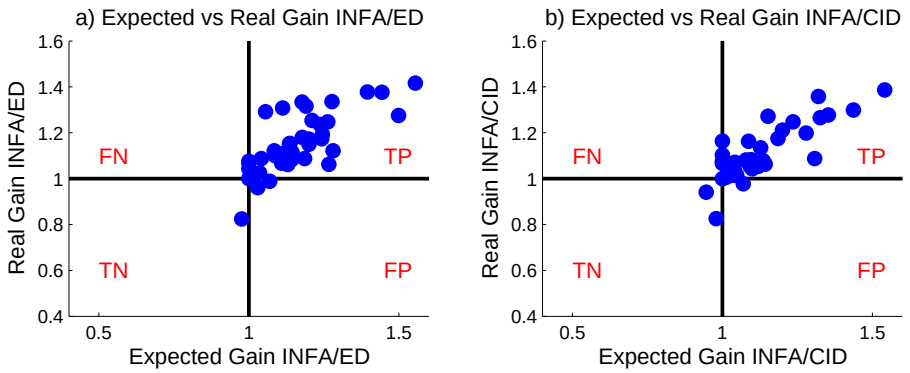



Figure 5.7: Illustration of the Expected (Train) versus Real (Test) gain against a) Euclidean Distance (ED) and b) Complexity Invariant Distance (CID).

The scatter plot of Expected vs Real Gain against two methods is shown in Figure 5.7. We selected two baselines, namely the Euclidean distance and Complexity-Invariant distance, as used in the original paper [9]. The scatter plot can be divided into four quadrants that represent regions of True/False Positives/Negatives. As can be seen, our method has a very large number of True Positive results over the baselines using the same 43 datasets of the UCR collections. The semantics of the test is to show that there is a consistent pattern where our method wins, which can be estimated by using only the training series.

## 5.5 Semi-supervised Time-Series Factorization

The factorization so-far presented in this chapter decomposes the series in a totally unsupervised way, meaning that the labels of the training instances are not taken into consideration. In that context, it is interesting to question whether a semi-supervised factorization, as introduced in Section 3.6.2, would improve the prediction accuracy. This section presents a semi-supervised factorization approach for time series, which learns a decomposed representation of each segment, such that i) the per-segment weights approximate the segments' values, and ii) the frequency of a series' segments' weights serve as predictors for a classification loss. In that light, a semi-supervised factorization learns *jointly* a reconstruction and a classification loss term.

Before further detailing the factorization mechanism, we would like to define the data segments as $S \in \mathbb{R}^{N \times \Phi \times M_* \times L_*}$, where we have extracted sliding window segments of various lengths $\Phi \in \mathbb{N}$. The number of series segments, given a segment length $L_\phi \in \mathbb{N}$, $\forall \phi \in \{1, \ldots, \Phi\}$ is denoted as $M_\phi \in \mathbb{N}$. The objective of the semi-supervised factorization is composed of two losses, as formalized in Equation 5.13, concretely a reconstruction loss denoted by $\mathcal{L}_R(S, D, P)$ and a classification accuracy loss denoted by $\mathcal{L}_A(Y, D, W)$. The reconstruction loss $\mathcal{L}_R$ measures the accuracy of reconstructing the segments $S$ as a dot product of patterns $P \in \mathbb{R}^{\Phi \times K \times L_*}$ and per-segment weights to each pattern, denoted by $D \in \mathbb{R}^{N \times \Phi \times M_* \times K}$. In other words, $K \in \mathbb{N}$ many patterns will be learned for each scale of segment lengths. Moreover, the segment lengths are multiples of an initial hyper-parameter length $L^{\text{Init}} \in \mathbb{N}$, literally $L^{\text{Init}}, 2L^{\text{Init}}, \ldots, \Phi L^{\text{Init}}$. It is worth noting that the reconstruction loss is similar to the unsupervised factorization detailed in the previous sections of this chapter. The only difference is the L2 regularization of $D$, instead of the sum-to-one constraint, which makes it easier to define the gradients of both loss terms.

**Optimize:**

$$\underset{D,P,W}{\operatorname{argmin}} \quad \mathcal{L}_R(S, D, P) + \mathcal{L}_A(Y, D, W) \tag{5.13}$$

$$\mathcal{L}_R(S, D, P) = \beta \sum_{i=1}^{N} \sum_{\phi=1}^{\Phi} \sum_{j=1}^{M_\phi} \sum_{l=1}^{L_\phi} \left( S_{i,\phi,j,l} - \hat{S}_{i,\phi,j,l} \right)^2 + \frac{\lambda_D}{2} ||D||^2 + \lambda_P ||P||^2$$

$$\mathcal{L}_A(Y, D, W) = (1 - \beta) \sum_{i=1}^{N^{\text{Train}}} \sum_{i=1}^{C} -Y_{i,c} \ln \left( \sigma \left( \hat{Y}_{i,c} \right) \right) - (1 - Y_{i,c}) \ln \left( 1 - \sigma \left( \hat{Y}_{i,c} \right) \right)$$
$$+ \frac{\lambda_D}{2} ||D||^2 + \lambda_W ||W||^2$$

**Such that:**

$$\hat{S}_{i,\phi,j,l} = \sum_{k=1}^{K} D_{i,\phi,j,k} P_{\phi,k,l}$$

$$\hat{Y}_{i,c} = W_{c,0} + \sum_{\phi=1}^{\Phi} \sum_{k=1}^{K} F_{i,\phi,k} W_{c,\phi,k}$$

$$F_{i,\phi,k} = \sum_{j=1}^{M_\phi} D_{i,\phi,j,k}$$

In addition, the classification accuracy loss $\mathcal{L}_A(Y, D, W)$ learns the weights $D$ in a way that the sum of weight's frequencies serve as classification predictors. The labels are denoted as the one-vs-all binary targets $Y \in \{0,1\}^{N^{\text{Train}} \times C}$, while the number of labeled instances is $N^{\text{Train}} \in \mathbb{N}$, $N^{\text{Train}} << N$. The accuracy is measured by a logistic loss that quantifies the success of the estimated targets $\hat{Y} \in \mathbb{R}^{\text{Train} \times C}$ in predicting the ground truth $Y$. The predictors are the frequencies of local weights, denoted by $F \in \mathbb{R}^{N^{\text{Train}} \times \Phi \times K}$, while the classification weights are denoted by $W \in \mathbb{R}^{C \times \Phi \times K}$. The frequencies $F$ can be interpreted as a learnable bag-of-patterns (a.k.a. bag-of-words) style of features. In addition, the prediction model of $\hat{Y}$ incorporates bias terms $W_{*,0} \in \mathbb{R}^C$. A L2 regularization for the weights $W$ controls the over-fitting aspects of the model. The switching hyper-parameter, denoted $\beta \in \mathbb{R} \wedge 0 \leq \beta \leq 1$, controls the importance of the two loss terms. Overall this factorization approach has a series of hyper-parameters, namely $\beta, \lambda_P, \lambda_W, \lambda_D, K, \Phi, L^{\text{Init}}$.

We would like to note down that, while being intuitive enough, to the best of our knowledge this semi-supervised factorization approach is novel in the realm of time-series classification. In fact we believe it to be the first semi-supervised bag-of-patterns approach.

### 5.5.1 Prediction of A Test Series

The prediction of the label of a test instance, indexed by $t = N^{\text{Train}} + 1, \ldots, N$, is done by taking the highest probability among the one-vs-all prediction models $\hat{Y}_t \in \mathbb{R}^C$. Equation 5.14 formalizes the prediction of the label $c_t$ of the $t$-th test instance.

$$c_t \leftarrow \underset{c^*=1,\ldots,C}{\text{argmax}} \quad \sigma\left(\hat{Y}_{t,c^*}\right) \tag{5.14}$$

### 5.5.2 Reconstruction Loss and Gradients

The factorization of Equation 5.13 will be learned using Stochastic Gradient Descent. However, in order to move forwards with the learning algorithm, we need to decompose the loss term and define the gradients. Equation 5.15 describes how the reconstruction loss is divided into smaller atomic loss terms, denoted as $\mathcal{L}_{Ri,\phi,j,l}$.

$$\mathcal{L}_R(S,D,P) = \sum_{i=1}^{N}\sum_{\phi=1}^{\Phi}\sum_{j=1}^{M_\phi}\sum_{l=1}^{L_\phi}\mathcal{L}_{Ri,\phi,j,l} \tag{5.15}$$

$$\mathcal{L}_{Ri,\phi,j,l} = \beta\left(S_{i,j,l}-\hat{S}_{i,j,l}\right)^2 + \frac{\lambda_D}{2L_\phi}\sum_{k=1}^{K}{D_{i,\phi,j,k}}^2 + \frac{\lambda_P}{NM_\phi}\sum_{k=1}^{K}{P_{\phi,k,l}}^2$$

The gradients of the decomposed reconstruction loss $\mathcal{L}_{Ri,\phi,j,l}$ with respect to the per-segment weights $D$ and patterns $P$ are detailed in Equations 5.16-5.17 .

$$\frac{\partial\mathcal{L}_{Ri,\phi,j,l}}{\partial D_{i,\phi,j,k}} = -2\beta\left(S_{i,\phi,j,l}-\hat{S}_{i,\phi,j,l}\right)P_{\phi,k,l} + \frac{\lambda_D}{L_\phi}D_{i,\phi,j,k} \tag{5.16}$$

$$\frac{\partial\mathcal{L}_{Ri,\phi,j,l}}{\partial P_{\phi,k,l}} = -2\beta\left(S_{i,m,j,l}-\hat{S}_{i,\phi,j,l}\right)D_{i,\phi,j,k} + \frac{2\lambda_P}{NM_\phi}P_{\phi,k,l} \tag{5.17}$$

### 5.5.3 Accuracy Loss and Gradients

Similarly the classification accuracy loss can be decomposed into smaller atomic losses, as shown in Equation 5.18.

$$\mathcal{L}_A(Y,D,W) = \sum_{i=1}^{N^{\text{Train}}}\sum_{c=1}^{C}\mathcal{L}_{Ai,c} \tag{5.18}$$

$$\mathcal{L}_{Ai,c} = (1-\beta)\left(-Y_{i,c}\ln\left(\sigma\left(\hat{Y}_{i,c}\right)\right)-(1-Y_{i,c})\ln\left(1-\sigma\left(\hat{Y}_{i,c}\right)\right)\right) + \frac{\lambda_D}{2C}\sum_{\phi=1}^{\Phi}\sum_{j=1}^{M_\phi}\sum_{k=1}^{K}{D_{i,\phi,j,k}}^2 + \frac{\lambda_W}{N^{\text{Train}}}\sum_{\phi=1}^{\Phi}\sum_{k=1}^{K}{W_{c,\phi,k}}^2$$

Furthermore, Equations 5.19-5.21 present the gradients of the decomposed classification accuracy loss $\mathcal{L}_{Ai,c}$ with respect to the classification weights $W$, the bias weights $W_{*,0}$ and the per-segment weights $D$.

$$\frac{\partial\mathcal{L}_{Ai,c}}{\partial W_{c,\phi,k}} = -(1-\beta)\left(Y_{i,c}-\sigma\left(\hat{Y}_{i,c}\right)\right)F_{i,\phi,k} + \frac{2\lambda_W}{N^{\text{Train}}}W_{c,\phi,k} \tag{5.19}$$

$$\frac{\partial\mathcal{L}_{Ai,c}}{\partial W_{c,0}} = -(1-\beta)\left(Y_{i,c}-\sigma\left(\hat{Y}_{i,c}\right)\right) \tag{5.20}$$

$$\frac{\partial\mathcal{L}_{Ai,c}}{\partial D_{i,\phi,j,k}} = -(1-\beta)\left(Y_{i,c}-\sigma\left(\hat{Y}_{i,c}\right)\right)W_{c,\phi,k} + \frac{\lambda_D}{C}D_{i,\phi,j,k} \tag{5.21}$$

### 5.5.4 Learning Algorithm

**Algorithm 9** Learning Semi-supervised Time-series Factorization

**Require:** $S \in \mathbb{R}^{N \times \Phi \times M_* \times L_*}$, Labels $Y \in \{0,1\}^{N^{\text{Train}} \times C}$, Number of patterns $K \in \mathbb{N}$, Learn Rate $\eta \in \mathbb{R}$, Regularization $\lambda_P \in \mathbb{R}, \lambda_D \in \mathbb{R}, \lambda_W \in \mathbb{R}$, Initial Segment Length $L^{\text{Init}} \in \mathbb{N}$, Magnitudes/Scales $\Phi \in \mathbb{N}$, Number of Iterations: $I \in \mathbb{N}$

**Ensure:** $D \in \mathbb{R}^{N \times \Phi \times M_* \times K}, P \in \mathbb{R}^{\Phi \times K \times L_*}, W \in \mathbb{R}^{C \times \Phi \times K}$

**for** $1, \ldots, I$ **do**
  **for** $i = 1, \ldots, N, \phi = 1, \ldots, \Phi, j = 1, \ldots, M_\phi, k = 1, \ldots, K, l = 1, \ldots, L_\phi$ **do**
    $D_{i,\phi,j,k} \leftarrow D_{i,\phi,j,k} - \eta \frac{\partial \mathcal{L}_{R\,i,\phi,j,l}}{\partial D_{i,\phi,j,k}}$
    $P_{m,k,l} \leftarrow P_{\phi,k,l} - \eta \frac{\partial \mathcal{L}_{R\,i,\phi,j,l}}{\partial P_{\phi,k,l}}$
  **end for**
  **for** $i = 1, \ldots, N^{\text{Train}}$, $c = 1, \ldots, C$ **do**
    **for** $\phi = 1, \ldots, \Phi$, $k = 1, \ldots, K$ **do**
      **for** $j = 1, \ldots, M_\phi$ **do**
        $D_{i,\phi,j,k} \leftarrow D_{i,\phi,j,k} - \eta \frac{\partial \mathcal{L}_{A\,i,c}}{\partial D_{i,\phi,j,k}}$
      **end for**
      $W_{c,\phi,k} \leftarrow W_{c,\phi,k} - \eta \frac{\partial \mathcal{L}_{A\,i,c}}{\partial W_{c,\phi,k}}$
    **end for**
    $W_{c,0} \leftarrow W_{c,0} - \eta \frac{\partial \mathcal{L}_{A\,i,c}}{\partial W_{c,0}}$
  **end for**
**end for**
**return** $D, P, W$

Having defined the gradients, we present the overall learning procedure in Algorithm 9. The optimization is iterated through $I \in \mathbb{N}$ epochs, while in each iteration the method first updates the patterns and the per-segment weights for minimizing the reconstruction loss. Jointly, in each iteration, the algorithm updates the weights $W$ and the per-segment weights with respect to the classification loss term. The updates of the parameters $D, P, W$ are conducted using a learning rate hyper-parameter $\eta \in \mathbb{R}$.

### 5.5.5 Does Semi-supervised Factorization Improve Accuracy?

Does the addition of a classification accuracy loss term, in Equation 5.13, improve the prediction quality, compared to an entirely unsupervised factorization? In order to answer this question, we would run an experimental analysis by creating three models, as a result of altering the $\beta$ hyper-parameter. The first model sets $\beta = 1$, so the classification loss term becomes 0 and the factorization is unsupervised. Afterwards we additionally learn only the weights $W$ using the per-weight segments $D$ of the unsupervised factorization. The two other models are variations of the parameter beta for $\beta = 0.1$ and $\beta = 0.5$. While $\beta = 0.5$ gives an equal importance to both terms, $\beta = 0.1$ gives higher priority to the classification loss.

All datasets used in this experimental analysis are again taken from the UCR collection [67]. The hyper-parameters used for the analysis are $L^{\text{Init}} = 0.3Q$, $R = 2$, $K = 0.5Q$, $\lambda_W = \lambda_D = \lambda_P = 0.1$, $\eta = 0.03$ and $I = 100$. The results do not include the 7 largest datasets, where the semi-supervised decomposition needed more than 10 GB main memory to finish, as a result of a practical computing constraint for this experiment.

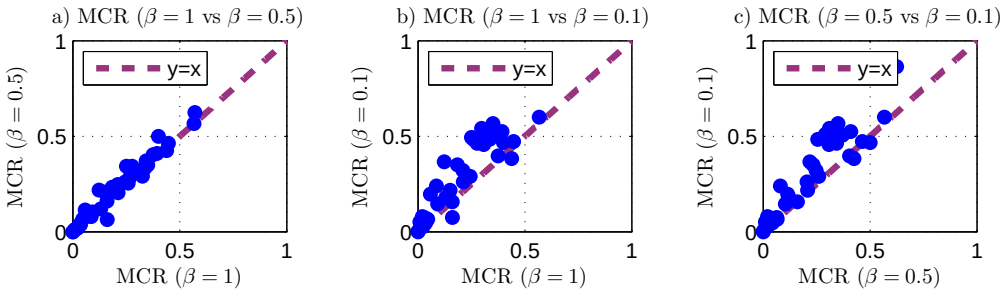



Figure 5.8: Efficiency of Semi-supervised Factorization

Figure 5.8 presents the comparative results of i) the unsupervised factorization ($\beta = 1$), ii) the moderately supervised factorization ($\beta = 0.5$) and iii) the strongly supervised version ($\beta = 0.1$). The unsupervised factorization achieves the best prediction accuracy, outperforming the moderate factorization ($\beta = 0.5$) in 28 out of 38 datasets. The difference is statistically significant according to a Wilcoxon Signed Rank test with a two tailed hypothesis and 0.05 significance level. Concretely the p-value of the test is 0.00544, indicating a clear significance of the difference. One might question though, whether adding more supervision

(i.e. smaller $\beta$) would help the supervised factorization increase its accuracy. Unfortunately, the results against the strongly supervised factorization(i.e. $\beta = 1$ vs $\beta = 0.1$) demonstrate that the unsupervised version is better in 35 of the 38 datasets, with an absolute statistical significance (p-value is 0). Evidently more supervision deteriorates the accuracy, as is also illustrated by Figure 5.8.c.

As can be deduced from the provided results, a semi-supervised factorization fails to deliver the expected qualitative boost, compared to an unsupervised transductive factorization. Such a finding indicate that in the realm of time-series data, updating the per-segment weights for the classification loss has non-positive impact on accuracy. Regarding other aspects, the semi-supervised factorization is limited to a linear classifier, while the unsupervised approach profits from the usage of a nonlinear polynomial-kernel SVM. In addition, the semi-supervised factorization has 9 hyper-parameters $(\beta, \lambda_P, \lambda_W, \lambda_D, K, \Phi, L^{\text{Init}}, \eta, I)$, while the unsupervised factorization has only 6 of them. Given the empirical evidences, we can assert that semi-supervised factorization is not a promising direction for time-series classification.

## 5.6 Conclusions

In this chapter we presented Invariant Factorization, a method that initially decomposes the time series into a set of overlapping segments via a sliding window approach. The segments are approximated by learning a set of latent patterns and degrees of memberships of each segment to each pattern. We formalized the factorization as a constraint objective function and proposed a stochastic coordinate descent method to solve it. The new representation of time series are the sums of the membership weights, which represent frequencies of local patterns. Features from various sliding window sizes were concatenated to encapsulate interaction among patterns of various scales. Finally, we conducted a thorough experimental comparison against totally 11 state of the art baselines in 43 real-life time series datasets. Our method introduces state-of-the-art results in the realm of time-series classification, regarding the UCR collection of datasets.

# Chapter 6

# Target-Driven Time-Series Feature Extraction

## Contents

**Highlights:** This chapter presents the best contribution of this thesis in terms of building an accurate classifier that tackles the intra-class series variations. In comparison to the previous thesis methods which augmented the training set, or decomposed the series segments, here we learn discriminative series patterns. Those patterns are called shapelets and are learned directly on the classification loss, therefore assuring state-of-the-art prediction quality.

Shapelets are discriminative sub-sequences of time series that best predict the target variable. For this reason, shapelet discovery has recently attracted considerable interest within the time-series research community. Currently shapelets are found by evaluating the prediction qualities of numerous candidates extracted from the series segments. In contrast to the state-of-the-art, this chapter proposes a novel perspective in terms of learning shapelets. A new mathematical formalization of the task via a classification objective function is proposed and a tailored stochastic gradient learning algorithm is applied. The proposed method of this chapter enables learning near-to-optimal shapelets directly without the need to try out lots of candidates. Furthermore, our method can learn true top-K shapelets by capturing their interaction. Extensive experimentation demonstrates statistically significant improvement in terms of wins and ranks against 13 baselines over 28 time-series datasets.

## 6.1 Introduction

As clarified early in Chapters 1-3, time-series data often exhibit inter-class differences in terms of small sub-sequencies rather than the full series structure [126]. A recently introduced concept, named *shapelet*, represents a maximally discriminative sub-sequence of time series data. Stated more directly, shapelets identify short discriminative series segments [86, 126]. Apart from their high prediction accuracy, shapelets also offer interpretable features to domain experts. Moreover, discovering shapelets has been a hot topic in the time-series domain during the last five years [62, 79, 80, 86, 95, 126, 129].

State-of-the-art methods discover shapelets by trying a pool of candidate subsequences from all possible series segments [80, 126] and then sorting the top performing segments according to their target prediction qualities. Distances between series and shapelets represent shapelet-transformed [80] classification features for a series of segregation metrics, such as information gain [86, 126], F-Stat [62] or Kruskall-Wallis [79]. The brute-force candidates search approach, based on an exhaustive search of candidates, suffers from a high runtime complexity, therefore several speed-up techniques have aimed at reducing the discovery time of shapelets [25, 86, 95]. In terms of classification performance, the shapelet-transformation method constructs qualitative predictors for standard classifiers and has recently shown improvements with respect to prediction accuracy [62, 80].

This chapter proposes an entirely new perspective on time-series shapelets. For the first time, we propose a mathematical formulation of the shapelet learning task as an optimization of a classification objective function. Furthermore, we propose a learning method that *learns* (not searches for) the shapelets which optimize the objective function. Concretely, we learn shapelets whose distances to series can linearly separate the time series instances by their targets, as shown in Figure 6.1. In comparison to existing approaches, our method can learn near-to-optimal shapelets and **true** top-K shapelet interactions. In a large pool of 28 datasets we demonstrate that the proposed method yields a large and statistically significant improvement over 13 baselines.

## 6.2 Chapter Notations

This chapter will adhere to the notational conventions of Table 6.1 and the forthcoming brief explanation. In order to avoid notational conflicts with previous chapters, we redefine the sliding window segments as:

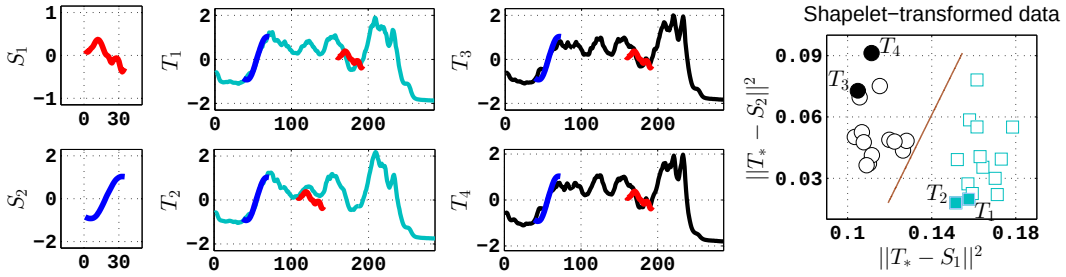



Figure 6.1: An illustration of two shapelets $S_1, S_2$ (leftmost plots) learned on the Coffee dataset. Series' distances to shapelets can optimally project the series into a 2-dimensional space, called the shapelet-transformed representation [80] (rightmost plot). The middle plots show the closest matches of the shapelets on series of two classes having light-blue and black colors.

| Symbol | Explanation |
|---|---|
| $J \in N$ | Number of series segments, (defined as $M$ in previous chapters) |
| $D \in \mathbb{R}^{N \times K \times M}$ | Distance between a shapelet and segments of time series |
| $M \in \mathbb{R}^{N \times K}$ | Minimum distances between a series and a shapelet |
| $R \in N$ | Scales (sizes) of patterns |

Table 6.1: Notational Conventions of Chapter 6

- A **sliding window segment** of length $L$ is an ordered sub-sequence of a series. Concretely, the segment starting at time $j$ inside the $i$-th series is defined as $(T_{i,j}, \ldots, T_{i,j+L-1})$. There are totally $J := Q - L + 1$ segments in a time series provided the starting index of the sliding window is incremented by one.

## 6.3 Proposed Method

### Distances Between Shapelets and Series

The distance between the $i$-th series $T_i$ and the $k$-th shapelet $P_k$ is defined as the minimum distance $M_{i,k}$ (shown in Equation 6.1) among the distances between the shapelet $P_k$ and each segment $j$ of $T_i$ [126, 127]. Informally speaking, it is the distance of a shapelet to the most similar series segment, as shown in Figure 6.1.

$$M_{i,k} = \min_{j=1,\dots,J} \frac{1}{L} \sum_{l=1}^{L} (T_{i,j+l-1} - P_{k,l})^2 \tag{6.1}$$

**Shapelet Transformation**

Minimum distances to shapelets can be characterized as a transformation of the time-series data $T \in \mathbb{R}^{N \times Q}$ into a new representation $M \in \mathbb{R}^{N \times K}$ [80]. Such a transformation reduces the dimensionality of the original time-series, because typically $K < Q$. General purpose classifiers (e.g.: SVMs, Bayesian Network, ...) have been recently shown to achieve high prediction accuracy over the new representation $M$ [62].

**Logistic Sigmoid Function**

The logistic sigmoid function is an S shaped instance of the logistic function and is defined as $\sigma(Y) = \left(1 + e^{-Y}\right)^{-1}$. We are going to use the sigmoid function for the prediction of target variables via a logistic regression loss.

### 6.3.1 A Novel Principle

In this chapter we propose a novel principle in learning time-series shapelets. Instead of searching among possible shapelet candidates from the series segments [80, 126], we propose a formal method that can directly learn optimal shapelets without needing to explore all possible candidates. Our principle can be summarized in two steps: (i) Start with rough initial guesses for the shapelets, (ii) Iteratively learn/optimize the shapelets by minimizing a classification loss function. In order to conduct the shapelet optimization, we define a novel classification model that is differentiable with respect to shapelets. Therefore, shapelets can be updated in a stochastic gradient descent optimization fashion, by taking steps towards the minimum of the classification loss function (i.e. towards maximal prediction accuracy).

### 6.3.2 Objective Function

For the sake of simplicity, the model introduced in this section will be focused only on binary targets $Y \in \{0, 1\}^N$ and a fixed shapelet length $L$. A general version of the model, with extended properties, is described Section 6.5.

#### 6.3.2.1 Prediction Model

Since the minimum distances $M$ are the new predictors in the transformed shapelets space, a linear learning model can predict approximate target values $\hat{Y} \in \mathbb{R}^{N \times K}$ via the predictors $M$ and linear weights $W \in \mathbb{R}^K$ (plus bias $W_0 \in \mathbb{R}$), as shown in Equation 6.2.

$$\hat{Y}_i = W_0 + \sum_{k=1}^{K} M_{i,k} W_k, \quad \forall i \in \{1, \ldots, N\} \tag{6.2}$$

#### 6.3.2.2 Loss Function

In this chapter we are going to exploit the logistic regression classification model, because it provides an option to interpret predicted binary targets as probabilistic confidences. Such a probabilistic interpretation will ensure extending our approach to the multi-class case in Section 6.5. The logistic regression operates by minimizing the logistic loss, defined in Equation 6.3, between true targets $Y$ and estimated ones $\hat{Y}$.

$$\mathcal{L}(Y, \hat{Y}) = -Y \ln \sigma(\hat{Y}) - (1 - Y) \ln \left(1 - \sigma(\hat{Y})\right) \tag{6.3}$$

#### 6.3.2.3 Regularized Objective Function

The logistic loss function together with regularization terms represent the regularized objective function, denoted as $\mathcal{O}$ in Equation 6.4. The idea of this chapter is to jointly learn the optimal shapelets $P$ and the optimal linear hyper-plane $W$ that minimize the classification objective $\mathcal{O}$.

$$\underset{S,W}{\operatorname{argmin}} \ \mathcal{O}(P, W) = \underset{P,W}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}(Y_i, \hat{Y}_i) + \lambda_W ||W||^2 \tag{6.4}$$

### 6.3.3 Differentiable Soft-Minimum Function

In order to compute the derivative of the objective function, all the involved functions of the model need to be differentiable. Unfortunately, the minimum function of Equation 6.1 is not differentiable and the partial derivative $\frac{\partial M}{\partial P}$ is not

defined. A differentiable approximation to the minimum function is introduced in this section.

For the sake of organizational clarity, we will introduce the distance between the $j$-th segment of series $i$ and the $k$-th shapelet as $D_{i,k,j}$ and define it in Equation 6.5.

$$D_{i,k,j} \quad := \quad \frac{1}{L} \sum_{l=1}^{L} \left( T_{i,j+l-1} - P_{k,l} \right)^2 \tag{6.5}$$

A differentiable approximation of the minimum function is the popular *Soft Minimum* function that is depicted in Equation 6.6. A parameter $\alpha$ controls the *precision* of the function and the soft minimum approaches the true minimum for $\alpha \to -\infty$.

$$M_{i,k} \quad \approx \quad \hat{M}_{i,k} = \frac{\sum_{j=1}^{J} D_{i,k,j} \; e^{\alpha \, D_{i,k,j}}}{\sum_{j'=1}^{J} e^{\alpha \, D_{i,k,j'}}} \tag{6.6}$$

Please note that the soft minimum has the shapelets as the only varying input, which appear embedded inside the distance definition $D$. We would like to explain the operating principle of the soft minimum with the aid of Figure 6.2.

A series from the FaceFour dataset and a shapelet are depicted in the upper plot of Figure 6.2. The shapelet is a slightly distorted variant of the series segment starting at time index 51. If we slide the shapelet over all the series segments and record the distance of shapelets to segments (i.e. Equation 6.5), then the Euclidean distances' plot in blue is achieved. Two plots in red (bottommost) illustrate the operation of the soft minimum function. Each point $j$ of the soft minimum plots correspond to $\frac{D_{i,k,j} \; e^{\alpha \, D_{i,k,j}}}{\sum_{j'=1}^{J} e^{\alpha \, D_{i,k,j'}}}$, while the **area** under the soft-minimum plots sums up to the true minimum distance between the shapelet and the series (i.e. Equation 6.1). It is important to realize in the third plot ($\alpha = -20$) that the amount by which a segment distance impacts the overall minimum is directly related to how small is that segment's distance compared to other segment distances. As can be seen in the bottom plot, if $\alpha = -100$, then only the true minimum segment distance is allowed to contribute to the grand total minimum. We found out that $\alpha = -100$ is small enough to make the soft minumum yield exactly the same results as the true minimum. The $\alpha$ value works fine because all segments are Z-normalized, i.e. all have a mean 0 and standard deviation of 1. Therefore, we kept this value fixed throughout all our experiments.

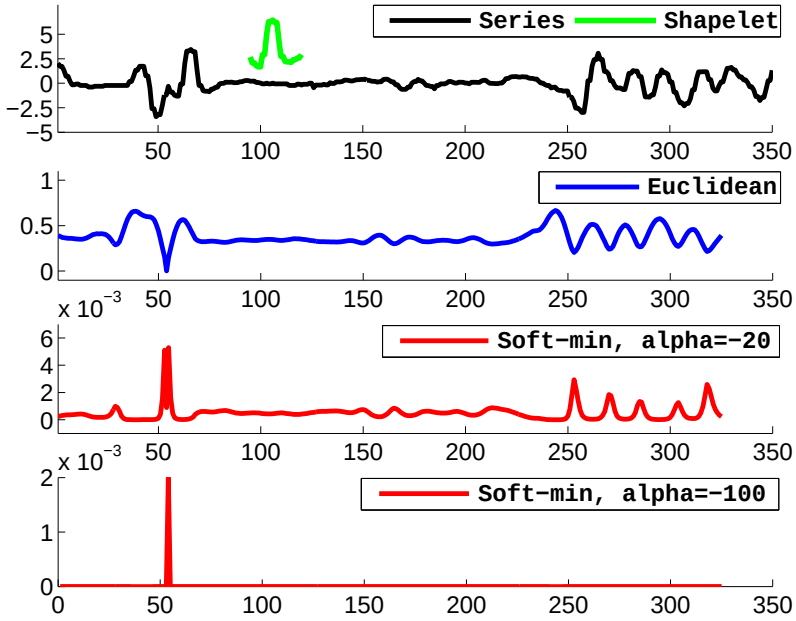



Figure 6.2: Illustration of the soft minimum between a shapelet (green) and all the segments of a series (black) from the FaceFour dataset

### 6.3.4 Per-Instance Objective

The optimization we will adopt in this chapter is a stochastic gradient descent approach that remedies the classification error caused by one instance at a time. The Equation 6.7 demonstrates the decomposed objective function $\mathcal{O}_i$, which corresponds to a division of the objective of Equation 6.4 into per-instance losses for each time series.

$$\mathcal{O}_i = \mathcal{L}(Y_i, \hat{Y}_i) + \frac{\lambda_W}{N} \sum_{k=1}^{K} {W_k}^2 \tag{6.7}$$

### 6.3.5 Gradients for Shapelets

The learning algorithm requires the definition of the gradients of the objective function with respect to the shapelets. The gradient of point $l$ in shapelet $k$ with respect to the objective of the $i$-th time series is defined in Equation 6.8 and is derived through the chain rule of derivation.

$$\frac{\partial \mathcal{O}_i}{\partial P_{k,l}} = \frac{\partial \mathcal{L}(Y_i, \hat{Y}_i)}{\partial \hat{Y}_i} \frac{\partial \hat{Y}_i}{\partial \hat{M}_{i,k}} \sum_{j=1}^{J} \frac{\partial \hat{M}_{i,k}}{\partial D_{i,k,j}} \frac{\partial D_{i,k,j}}{\partial P_{k,l}} \tag{6.8}$$

Furthermore, the gradient of the loss with respect to the predicted target is defined in Equation 6.9, while the gradient of the minimum distances with respect to the estimated target is shown in Equation 6.10.

$$\frac{\partial \mathcal{L}(Y_i, \hat{Y}_i)}{\partial \hat{Y}_i} = -\left(Y_i - \sigma\left(\hat{Y}_i\right)\right) \tag{6.9}$$

$$\frac{\partial \hat{Y}_i}{\partial \hat{M}_{i,k}} = W_k \tag{6.10}$$

In addition, the gradient of the overall minimum distance with respect to a segment distance is presented in Equation 6.11 and the gradient of a segment distance with respect to a shapelet point is derived in Equation 6.12.

$$\frac{\partial \hat{M}_{i,k}}{\partial D_{i,k,j}} = \frac{e^{\alpha D_{i,k,j}} \left(1 + \alpha \left(D_{i,k,j} - \hat{F}_{i,k}\right)\right)}{\sum_{j'=1}^{M} e^{\alpha D_{i,k,j'}}} \tag{6.11}$$

$$\frac{\partial D_{i,k,j}}{\partial P_{k,l}} = \frac{2}{L}\left(P_{k,l} - T_{i,j+l-1}\right) \tag{6.12}$$

### 6.3.6 Gradients for Classification Weights

The hyper-plane weights $W$ can also be learned to minimize the classification objective via stochastic gradient descent. Equation 6.13 shows the partial gradient of updating each weight $W_k$ and Equation 6.14 presents the bias term $W_0$.

$$\frac{\partial \mathcal{O}_i}{\partial W_k} = -\left(Y_i - \sigma\left(\hat{Y}_i\right)\right) \hat{M}_{i,k} + \frac{2\lambda_W}{N} W_k \tag{6.13}$$

$$\frac{\partial \mathcal{O}_i}{\partial W_0} = -\left(Y_i - \sigma\left(\hat{Y}_i\right)\right) \tag{6.14}$$

**Algorithm 10** Learning Time-Series Shapelets

**Require:** $T \in \mathbb{R}^{N \times Q}$, Number of Shapelets $K \in \mathbb{N}$, Length of a shapelet $L \in \mathbb{N}$, Regularization $\lambda_W \in \mathbb{R}$, Learning Rate $\eta \in \mathbb{R}$, Number of iterations: $I \in \mathbb{N}$
**Ensure:** Shapelets $P \in \mathbb{R}^{K \times L}$, Classification weights $W \in \mathbb{R}^K$, Bias $W_0 \in \mathbb{R}$

**for** $1, \ldots, I$ **do**
  **for** $i = 1, \ldots, N$ **do**
    Pre-compute errors needed for $\mathcal{O}_i$
    **for** $k = 1, \ldots, K$ **do**
      $W_k \leftarrow W_k - \eta \frac{\partial \mathcal{O}_i}{\partial W_k}$
      **for** $l = 1, \ldots, L$ **do**
        $P_{k,l} \leftarrow P_{k,l} - \eta \frac{\partial \mathcal{O}_i}{\partial P_{k,l}}$
      **end for**
    **end for**
    $W_0 \leftarrow W_0 - \eta \frac{\partial \mathcal{O}_i}{\partial W_0}$
  **end for**
**end for**
**return** $O, W, W_0$

### 6.3.7 Learning Algorithm

After having derived the gradients of the shapelets and the weights, we can introduce the overall learning algorithm. Our approach iterates in a series of epochs and updates the values of the shapelets and weights in the negative direction of the derivative with respect to the classification objective of each training instance.

The steps of the learning process are shown in Algorithm 10. The pseudo-code iterates over all training instances and updates all $K$ shapelets $P$ and the weights $W, W_0$ by a learning rate $\eta$.

### 6.3.8 Convergence

The convergence of Algorithm 10 depends on two parameters, the learning rate $\eta$ and the maximum number of iterations. High values for the learning rate can minimize the objective in less iterations, but pose the risk of divergence, while small learning rates require more iterations. Subsequently, the learning rate and the number of iterations should be learned via cross-validation from the training data.

For instance, Figure 6.3 illustrates the convergence of the learning algorithm on the Coffee dataset for 57 shapelets. Both the training and the testing loss

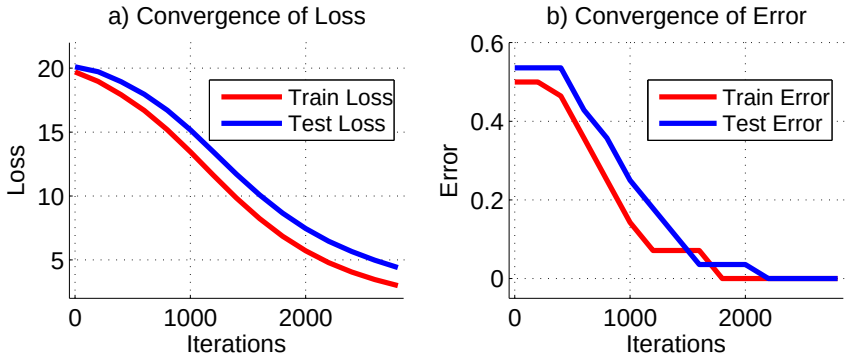



Figure 6.3: Convergence of the Coffee dataset, Parameters: $K = 57$, $L = 143$, $\eta = 0.01$, $\lambda_W = 0.001$, $\alpha = -100$

converge very smoothly for $\eta = 0.01$. As a consequence of optimizing the training loss, the train and test errors also decrease simultaneously. In addition, the regularization weight $\lambda_W = 0.001$ ensures that the train and test loss have small differences, which can be interpreted as a generalization quality without any overfitting effect.

### 6.3.9 Model Initialization

Equation 6.4 is a non-convex function in terms of $P$ and $W$ because of the product term between both variables. Gradient descent techniques do not theoretically guarantee the discovery of global optima in non-linear functions. Unfortunately, non-convex optimization techniques are very slow for data mining problems, therefore gradient based approaches are often selected as a compromise between feasibility and optimality [120].

Gradient descent optimization requires a good initialization of the parameters when applied to non-convex functions. In other words, if the initialization starts the learning around a region where the global optimum is located, then the gradient can update the parameters to the exact location of the optimum.

Initialization can influence a gradient based technique significantly. We are going to illustrate the sensitivity of shapelets initialization through an experiment shown in Figure 6.4. For the sake of two-dimensional illustration, we initialized one shapelet, denoted $S$ in Figure 6.4, in the Gun-Point dataset using two values. The first 15 points of a 30 points long shapelet ($S_{1:15}$) were given a fixed initial value, while the other half points of the shapelet $S_{16:30}$ were initialized with another fixed value. Figure 6.4 demonstrate that different initial values of

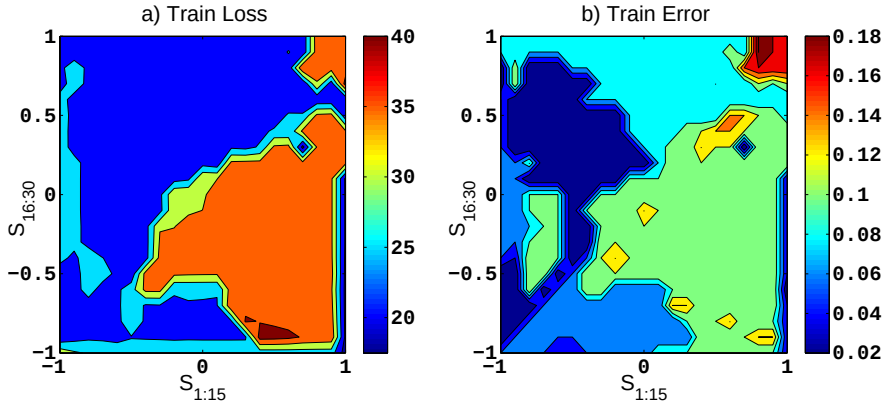



Figure 6.4: Sensitivity of Shapelet (S) Initialization, Gun-Point dataset, Parameters: $L = 30$, $\eta = 0.01$, $\lambda_W = 0.01$, Iterations= 3000, $\alpha = -100$

the shapelet can result in different loss values and error rates over the training instances.

In order to robustify the initialization guesses, we use the K-Means centroids of all segments as initial values for the shapelets. Since centroids represent typical patterns of the data, they offer a good variety of shapes for initializing shapelets and help our method achieve high prediction accuracy. The initialization is conducted before Algorithm 10 starts, while $W$ is also initialized randomly around 0.

### 6.3.10 Illustrating The Mechanism

An illustration of the learning algorithm is depicted in Figure 6.5. Two shapelets of length 40 are learned from the Gun-Point dataset. Sub-figure *a*) demonstrates the initialization values of the shapelets and the arrangement of the minimum values of the time series to shapelets. As can be seen, a linear hyper-plane $W$ cannot easily separate the two classes. After 400 iterations of our method, the shapelets are updated as shown in sub-figure *b*). In addition, the shapelet transformed data representation $M$ becomes almost linearly separable with few exceptions. Finally, the algorithm approaches convergence in sub-figure *c*) after 800 iterations. The linear hyper-plane $W$ separates the shapelet-transformed instances of the binary dataset with just a single error (in red).

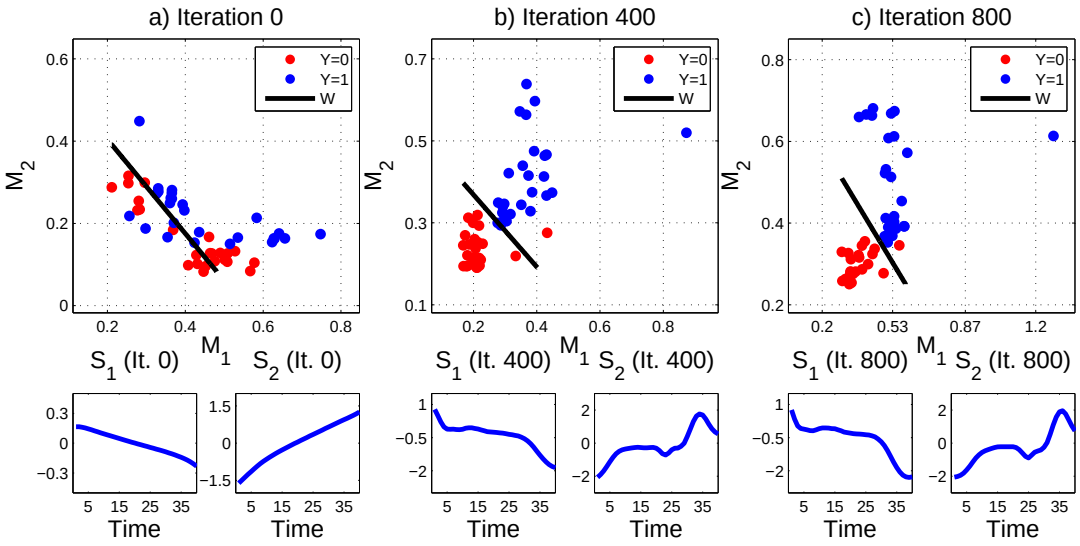



Figure 6.5: Learning Two Shapelets on the Gun-Point Dataset: Parameters $L = 40, \eta = 0.01, \lambda_W = 0.01, \alpha = -100$

## 6.4 Analysis of Our Method

### 6.4.1 Algorithmic Complexity

The baseline method which exhaustively tries candidates from series segments [80, 126] requires $\mathcal{O}(N^2Q^3)$ running time for discovering the best shapelet of a particular length $Q$. On the other hand, our method requires $\mathcal{O}(NQ^2 \times \mathrm{I})$, therefore our algorithm finds the best shapelet in a faster time, given that usually $\mathrm{I} << NQ$.

### 6.4.2 Comparison to State of the Art

#### 6.4.2.1 Learning Near-To-Optimal Shapelets

The optimal solution of Equation 6.4 gives the optimal shapelets, while a gradient descent approach can find a near-to-optimal minimum given an appropriate initialization.

The baseline approaches, on the other hand, provide no guarantee of optimal solutions for two primary reasons. First of all, the baselines are bound to shapelet candidates from the pool of series segments and cannot explore candidates which do not appear literally as segments. Secondly, minimizing the

classification objective through candidate guesses has no guarantee of optimality, while a gradient-based optimization guarantees at least a local minima.

#### 6.4.2.2 Capturing Interactions Among Shapelets

The baselines find the score of each shapelet independently and then sort the individual quality of each shapelet, in order to select the top performers. However, such an approach does not take into account interactions among patterns. In other words, two shapelets can be individually sub-optimal, but when combined together they can improve the results. In fact, this problem is well known in data mining and referred to as variable subset selection [34].

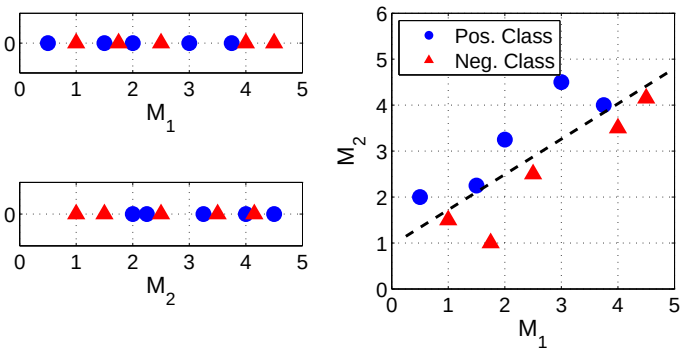



Figure 6.6: Interactions among Shapelets Enable Individually Unsuccessful Shapelets (left plots) to Excel in Cooperation (right plot)

For instance, Figure 6.6 demonstrates an example on how interactions among shapelets can become a game changing factor. On the left plots, we show the minimum distances of series to two shapelets. As can be observed, the individual discriminative quality of the shapelets is poor. On the other hand, a simple 2-dimensional interaction of exactly the **same** distances $M_1, M_2$ can yield drastically improved results, as shown on the right plot. When combined together, the distances to those shapelets can create a linearly separable discrimination, i.e. a perfect classification accuracy.

If the baseline's exhaustive discovery approach would attempt to select the **true** top-K interaction of shapelets out of $N(Q-L+1)$ candidates, then it will need to check the interaction of: $\binom{N(Q-L+1)}{K} = \frac{(N(Q-L+1))!}{K!(N(Q-L+1)-K)!}$ many combinations of candidates. For instance finding the **true** top 100 shapelets of length 30 from the Adiac dataset with $N = 390$ and $Q = 176$ requires checking $3.42 \times 10^{317}$ combination trials using the baseline's approach. Clearly, the exhaustive search baseline cannot find **true**

top-K shapelet interactions within a feasible time-frame. On the contrary, our method can find the interactions at a simple linear scale $K$, due to the property of jointly learning the shapelets and their interactions.

#### 6.4.2.3 Weaker Aspects of Our Method

Our method relies on more hyper-parameters than the baselines, such as the learning rate $\eta$, the number of iterations, the regularization parameter $\lambda_W$ and the soft-min precision $\alpha$. However, given the high prediction accuracy that will be demonstrated in Section 6.6, we argue that the very high accuracy by far out-weights the model's learning efforts.

The time needed for the baselines to compute the top-K shapelets is not significantly large with respect to the time needed to find a single shapelet. Such a behavior comes from the fact that the quality of each candidate is known and ready in the end of the discovery. On the other hand, our method needs $K$-many time units for $K$ shapelets, w.r.t. learning one shapelet. However, such a disadvantage in time for large $K$ is well spent in terms of accuracy, because our method can learn **true** top-K shapelet interactions and significantly improve the classification accuracy. Moreover, we believe that our method may yield to further improvements in efficiency by exploiting "early abandoning" and caching of partial results (as in [7]). For brevity we ignore such issues here and focus on forcefully demonstrating the improvements in accuracy.

## 6.5 Learning General Shapelets

The model presented in Section 6.3.2 can be generalized to multi-class labels and multi-size shapelets. Basically the model is extended to classify multi-class targets and capture interactions among shapelets of various sizes.

### 6.5.1 Decomposition of the Multi-Class Problem Into One-vs-all Subproblems

In order to learn from multi-class targets $Y \in \{1, \ldots, C\}^N$ with $C$ categories, we will convert the problem into $C$-many one-vs-all sub-problems. Each sub-problem will discriminate one class against all the others. The one-vs-all binary targets $Y^b \in \{0, 1\}^{N \times C}$ are defined in Equation 6.15. We would like to specify that the choice of one-vs-all model is carried on due to the practicality in terms

of implementation. In contrast, the multi-class learning in Chapters 4-5 was conducted by off-the-shelf optimization libraries for SVM.

$$Y_{i,c}^{b} = \begin{cases} 1 & Y_i = c \\ 0 & Y_i \neq c \end{cases}, \quad \forall i \in \{1, \ldots, N\}, \ \ \forall c \in \{1, \ldots, C\} \tag{6.15}$$

In fact, the conversion to one-vs-all sub-problems will be useful for the operation of the logistic regression classifier. The output of the logistic regression for a binary problem can be perceived as a confidence probability. Therefore, the index of the most confident among the $C$-many classifiers is selected as the predicted categorical value of a test instance.

### 6.5.2 Interactions among Shapelets having Various Lengths

Capturing interactions among shapelets having various lengths is another aspect of the extended method. Our generalized model learns $R$ different scales of shapelet lengths starting at a minimum $L^{\min}$ as $\{L^{\min}, 2L^{\min}, \ldots, RL^{\min}\}$. The shapelets therefore will be defined as $P \in \mathbb{R}^{R \times K \times *}$, where $P_r \in \mathbb{R}^{K \times rL^{\min}}$, $r \in \{1, \ldots, R\}$, and represent $K$-many shapelets for each scale $R$, i.e. totally $KR$ shapelets. The length of a shapelet at scale $r \in \{1, \ldots, R\}$ is $r \cdot L^{min}$. Consequently, the number of segments in a time series depends on the scale of the shapelet's length to be matched against and is $J(r) = Q - r \cdot L^{\min} + 1$.

### 6.5.3 Generalized Objective Function

The objective function of the generalized model is presented in Equation 6.16, which is a regularized logistic regression loss between the true targets and the predicted ones shown in Equation 6.17. The notation $M_{r,i,k}$ identifies the minimum distance of the $i$-th series to the $k$-th shapelet of scale $r$, i.e. to $P_{r,k} \in \mathbb{R}^{r \cdot L^{min}}$. In addition, the weight $W_{c,r,k}$ identifies the class $c$ classifier and the weight of the $k$-th shapelet at scale $r$.

$$\underset{S,W}{\operatorname{argmin}} \quad \mathcal{O} \quad = \quad \underset{S,W}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{c=1}^{C} \mathcal{L}(Y_{i,c}^{b}, \hat{Y}_{i,c}^{b}) + \lambda_W ||W||^2 \tag{6.16}$$

$$\hat{Y}_{i,c}^{b} \quad = \quad W_{c,0} + \sum_{r=1}^{R} \sum_{k=1}^{K} M_{r,i,k} W_{c,r,k} \tag{6.17}$$

### 6.5.4 Classification of Test Instances

Once the model is learned, a test instance indexed $t$ is classified as the one-vs-all classifier which yields maximum confidence, as presented in Equation 6.18. The algorithmic complexity of classifying a test instance is $O(CRKJ)$, but since $C, R, K$ are asymptotically smaller values than $J$, the asymptotic complexity is $O(J)$.

$$\hat{Y}_{\mathbf{t}} \leftarrow \underset{c\in\{1,\ldots,C\}}{\operatorname{argmax}} \; \sigma\left(\hat{Y}^b_{\mathbf{t},c}\right), \qquad \forall t \in \{1,\ldots,N^{Test}\} \tag{6.18}$$

### 6.5.5 Generalized Soft-Minimum

The soft minimum function can be trivially generalized to include the notation for the scales $r$ as shown in Equation 6.19. The distance between the $k$-th shapelet at scale $r$ and the $j$-th segment of time series $i$ is denoted as $D_{r,i,k,j}$ in Equation 6.20.

$$M_{r,i,k} \approx \hat{M}_{r,i,k} = \frac{\sum_{j=1}^{J(r)} D_{r,i,k,j}\; e^{\alpha\, D_{r,i,k,j}}}{\sum_{j'=1}^{J(r)} e^{\alpha\, D_{r,i,k,j'}}} \tag{6.19}$$

$$D_{r,i,k,j} = \frac{1}{r\cdot L^{\min}} \sum_{l=1}^{r\cdot L^{\min}} \left(T_{i,j+l-1} - P_{r,k,l}\right)^2 \tag{6.20}$$

### 6.5.6 Gradients of Generalized Objective Function

The objective function can be split per each instance $i$ and the loss of each one-vs-all classifier $c$ and denoted in Equation 6.21 as $\mathcal{O}_{i,c}$.

$$\mathcal{O}_{i,c} = \mathcal{L}(Y^b_{i,c}, \hat{Y}^b_{i,c}) + \frac{\lambda_W}{NC} \sum_{r=1}^{R}\sum_{k=1}^{K} {W_{c,r,k}}^2 \tag{6.21}$$

#### 6.5.6.1 Shapelet Gradients

The derivative of the per-cell objective $O_{i,c}$ with respect to each shapelet $P_{r,k,l}$ is shown in Equation 6.22.

$$\frac{\partial \mathcal{O}_{i,c}}{\partial P_{r,k,l}} = -\left(Y^b_{i,c} - \sigma\left(\hat{Y}^b_{i,c}\right)\right) \frac{\partial \hat{M}_{r,i,k}}{\partial P_{r,k,l}} W_{c,r,k} \tag{6.22}$$

Moreover, the derivative of the minimum distances with respect to the generalized shapelets is defined in Equations 6.23-6.24.

$$\frac{\partial \hat{M}_{r,i,k}}{\partial D_{r,i,k,j}} = \frac{e^{\alpha D_{r,i,k,j}} \left(1 + \alpha \left(D_{r,i,k,j} - \hat{M}_{r,i,k}\right)\right)}{\sum_{j'=1}^{J(r)} e^{\alpha D_{r,i,k,j'}}} \tag{6.23}$$

$$\frac{\partial D_{r,i,k,j}}{\partial P_{r,k,l}} = \frac{2\left(P_{r,k,l} - T_{i,j+l-1}\right)}{r \cdot L^{\min}} \tag{6.24}$$

#### 6.5.6.2 Classification Weights' Gradients

The gradients of the per-cell objective with respect to the generalized weights and the bias terms are presented in Equations 6.25-6.26.

$$\frac{\partial \mathcal{O}_{i,c}}{\partial W_{c,r,k}} = -\left(Y^b_{i,c} - \sigma\left(\hat{Y}^b_{i,c}\right)\right)\hat{M}_{r,i,k} + \frac{\lambda_W W_{c,r,k}}{NC} \tag{6.25}$$

$$\frac{\partial \mathcal{O}_{i,c}}{\partial W_{c,0}} = -\left(Y^b_{i,c} - \sigma\left(\hat{Y}^b_{i,c}\right)\right) \tag{6.26}$$

#### 6.5.6.3 Optimized Learning Algorithm

Algorithm 11 summarizes all the steps of the learning process. The first section of the procedure pre-computes terms which are used frequently in the gradients of the shapelets, such as $\xi, D, \psi, \vartheta$. The pre-computations boost the learning time and avoid computing the same terms repeatedly. The second part of the algorithm updates the weights and the shapelets using the defined gradients and the precomputed terms.

## 6.6 Experimental Results

### 6.6.1 Setup And Reproducibility

#### 6.6.1.1 Datasets

For the sake of equivalent comparison, we selected exactly the same set of datasets as the closest baselines [62, 80]. A large pool of 28 datasets consisting of time-series datasets having various numbers of instances, lengths and number of classes is selected and details are shown in Table 6.2. In order to ensure a fair comparison with the baselines, we used the default train and test data splits, **same** as

**Algorithm 11** Generalized Shapelets Learning

**Require:** Time series $T \in \mathbb{R}^{N \times Q}$, Binary labels $Y^b \in \mathbb{R}^{N \times C}$, Number of shapelets $K$, Learn Rate $\eta$, Regularization $\lambda_W$, Scales of shapelet lengths $R \in \mathbb{N}$, Minimum Shapelet Length $L^{\min}$, Number of Iterations: $I \in \mathbb{N}$

**Ensure:** Shapelets $P \in \mathbb{R}^{R \times K \times *}$, Classification weights $W \in \mathbb{R}^{R \times C \times K}, W_0 \in \mathbb{R}^C$

Initialize $P, W, W_0$
**for** iteration=$\{1, \ldots, \mathrm{I}\}$ **do**
  **for** $i = \{1, \ldots, N\}$ **do**
    {Pre-compute Terms}
    **for** $r = \{1, \ldots, R\},\ k = \{1, \ldots, K\}$ **do**
      **for** $j = \{1, \ldots, J(r)\}$ **do**
        $D_{r,i,k,j} := \frac{1}{r \cdot L^{\min}} \sum_{l=1}^{r \cdot L^{\min}} \left(T_{i,j+l-1} - P_{r,k,l}\right)^2$
        $\xi_{r,i,k,j} := e^{\alpha\, D_{r,i,k,j}}$
      **end for**
      $\psi_{r,i,k} := \sum_{j=1}^{J(r)} \xi_{r,i,k,j}$
      $\hat{M}_{r,i,k} := \frac{1}{\psi_{r,i,k}} \sum_{j=1}^{J(r)} D_{r,i,k,j}\ \xi_{r,i,k,j}$
    **end for**
    **for** $c = \{1, \ldots, C\}$ **do**
      $\sigma(\hat{Y}^b_{i,c}) := \left(1 + e^{-\sum_{r=1}^{R} \sum_{k=1}^{K} \hat{M}_{r,i,k} W_{c,r,k}}\right)^{-1}$
      $\vartheta_{i,c} := Y^b_{i,c} - \sigma(\hat{Y}^b_{i,c})$
    **end for**
    {Learn Shapelets and Classification Weights}
    **for** $c = \{1, \ldots, C\}$ **do**
      **for** $r = \{1, \ldots, R\},\ k = \{1, \ldots, K\}$ **do**
        $\mathbf{W}_{c,r,k}\ +=\ \eta \left(\vartheta_{i,c} \hat{M}_{r,i,k} - \frac{2\lambda_W}{NC} \mathbf{W}_{c,r,k}\right)$
        **for** $j = \{1, \ldots, J(r)\}$ **do**
          $\phi_{r,i,k,j} := \frac{2\xi_{r,i,k,j}\left(1 + \alpha\left(D_{r,i,k,j} - \hat{M}_{r,i,k}\right)\right)}{r \cdot L^{\min} \psi_{r,i,k}}$
          **for** $l = \{1, \ldots, r \cdot L^{\min}\}$ **do**
            $\mathbf{P}_{r,k,l} +=\ \eta\ \vartheta_{i,c}\ \phi_{r,i,k,j} \times$
            $\left(\mathbf{P}_{r,k,l} - T_{i,j+l-1}\right) W_{c,r,k}$
          **end for**
        **end for**
      **end for**
      $\mathbf{W}_{c,0} +=\ \eta\ \vartheta_{i,c}$
    **end for**
  **end for**
**end for**
**return** $P, W, W_0$

Table 6.2: Dataset Information, Parameter Search Results and Running Time for The Best Shapelet [62]

| Dataset Information | | | | Parameter Values (LTS) | | | | | Run-Time (Best Shap.) | |
|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | Train/Test | Length | Cls. | $K$ | $L^{\min}$ | $R$ | $\lambda_W$ | maxIter | F-Stat (Sec) | LTS (Sec) |
| Adiac | 390/391 | 176 | 37 | 0.3 | 0.2 | 3 | 0.01 | 10000 | 4509.91 | 3017.23 |
| Beef | 30/30 | 470 | 5 | 0.15 | 0.125 | 3 | 0.01 | 10000 | 1251.21 | 293.68 |
| Beetle/Fly | 20/20 | 512 | 2 | 0.15 | 0.125 | 1 | 0.01 | 5000 | 21496.51 | 131.015 |
| Bird/Chicken | 20/20 | 512 | 2 | 0.3 | 0.075 | 1 | 0.1 | 10000 | 20465.63 | 81.405 |
| Chlorine. | 467/3840 | 166 | 3 | 0.3 | 0.2 | 3 | 0.01 | 10000 | 15681.39 | 558.51 |
| Coffee | 28/28 | 286 | 2 | 0.05 | 0.075 | 2 | 0.01 | 5000 | 258.15 | 90.96 |
| Diatom. | 16/306 | 345 | 4 | 0.3 | 0.175 | 2 | 0.01 | 10000 | 53.91 | 173.1 |
| DP_Little | 400/645 | 250 | 3 | 0.15 | 0.175 | 1 | 1 | 5000 | 78005.7 | 1525.595 |
| DP_Middle | 400/645 | 250 | 3 | 0.3 | 0.025 | 3 | 1 | 10000 | 91208.52 | 910.33 |
| DP_Thumb | 400/645 | 250 | 3 | 0.05 | 0.175 | 3 | 0.1 | 5000 | 123766.49 | 963.765 |
| ECGFiveDays | 23/861 | 136 | 2 | 0.05 | 0.125 | 2 | 0.01 | 5000 | 149.1 | 29.365 |
| FaceFour | 24/88 | 350 | 4 | 0.3 | 0.175 | 3 | 1 | 5000 | 4556.41 | 386.45 |
| GunPoint | 50/150 | 150 | 2 | 0.15 | 0.2 | 3 | 0.1 | 10000 | 569.42 | 46.69 |
| ItalyPower. | 67/1029 | 24 | 2 | 0.3 | 0.2 | 3 | 0.01 | 5000 | 1.75 | 10.285 |
| Lighting7 | 70/73 | 319 | 7 | 0.05 | 0.075 | 3 | 1 | 5000 | 14912.74 | 394.44 |
| MedicalImages | 381/760 | 99 | 10 | 0.3 | 0.2 | 2 | 1 | 10000 | 7742.97 | 406.725 |
| MoteStrain | 20/1252 | 84 | 2 | 0.3 | 0.2 | 3 | 1 | 10000 | 10.76 | 16.875 |
| MP_Little | 400/645 | 250 | 3 | 0.3 | 0.2 | 3 | 0.01 | 5000 | 88071.5 | 965.27 |
| MP_Middle | 400/645 | 250 | 3 | 0.05 | 0.2 | 2 | 0.01 | 5000 | 134731.54 | 940.555 |
| Otoliths | 64/64 | 512 | 2 | 0.15 | 0.125 | 3 | 0.01 | 2000 | 55874.19 | 407.835 |
| PP_Little | 400/645 | 250 | 3 | 0.15 | 0.125 | 3 | 1 | 10000 | 79993.31 | 890.925 |
| PP_Middle | 400/645 | 250 | 3 | 0.15 | 0.175 | 2 | 0.01 | 10000 | 57815.02 | 1574.805 |
| PP_Thumb | 400/645 | 250 | 3 | 0.3 | 0.175 | 2 | 0.1 | 10000 | 91401.49 | 1449.36 |
| SonyAIBO. | 20/601 | 70 | 2 | 0.3 | 0.125 | 2 | 0.01 | 10000 | 6.73 | 11.415 |
| Symbols | 25/995 | 398 | 6 | 0.05 | 0.175 | 1 | 0.1 | 5000 | 8901.28 | 308.99 |
| SyntheticControl | 300/300 | 60 | 6 | 0.15 | 0.125 | 3 | 0.01 | 5000 | 984.36 | 219.97 |
| Trace | 100/100 | 275 | 4 | 0.15 | 0.125 | 2 | 0.1 | 10000 | 54128.53 | 275.375 |
| TwoLeadECG | 23/1139 | 82 | 2 | 0.3 | 0.075 | 1 | 0.1 | 10000 | 3.12 | 15.415 |

the baselines [62, 80]. The datasets are available through the UCR[1] and UEA[2] websites.

#### 6.6.1.2 Reproducibility and Hyper-parameter Search

Our method (hereafter denoted as LTS, for **L**earning **T**ime-Series **S**hapelets) requires the tuning of a series of hyper-parameters, which were found through a grid search approach using cross-validation over the training data. The number of shapelets was searched in a range of $K \in \{0.05, 0.15, 0.3\}$, which is a fraction of the series length, e.g. $K = 0.3$ means 30% of $Q$. Similarly, $L^{\min} \in \{0.025, 0.075, 0.125, 0.175, 0.2\} \times 100\%$ of $Q$, while three scales of shapelet lengths were searched from $R \in \{1, 2, 3\}$. The regularization parameter was one of $\lambda_W \in \{0.01, 0.1, 1\}$ .

[1] `http://www.cs.ucr.edu/~eamonn/time_series_data/`

[2] `http://www.uea.ac.uk/computing/machine-learning/shapelets/shapelet-data`

The learning rate was kept fixed at a small value of $\eta = 0.01$, while the number of iterations is selected from $I \in \{2000, 5000, 10000\}$. All our method' parameters for all datasets are shown in Table 6.2.

#### 6.6.1.3 Baselines

Thirteen different baselines were compared against, which are grouped into the following four clusters. The vast majority of methods are shapelet-related and fit directly the scope of this chapter.

- **Shapelet Tree Methods**, constructed from shapelets whose qualities are measured using: *i)* Information gain quality criterion (IG) [86, 126], *ii)* Kruskall-Wallis quality criterion (KW) [62], *iii)* F-Stats quality criterion (FST) [80] and *iv)* the Mood's Median Criterion (MM) [62].
- **Basic Classifiers** [80], learned over shapelet-transformed data, such as: Nearest Neighbors (1NN), Naive Bayes (NB) and C4.5 tree (C4.5).
- **More Complex Classifiers** [62], learned over shapelet transformed data, such as: Bayesian Networks (BN), Random Forest (RAF), Rotation Forest (ROF) and Support Vector Machines (SVM).
- **Other Related Methods:** The Fast Shapelets (FSH) [95] exploits a fast random projection technique on the SAX representation, while the Dynamic Time Warping (DTW) classifier on the raw time-series data is also selected due to its reputation as a strong similarity metric [39].

### 6.6.2 Very High Prediction Accuracy

We compared our method of learning shapelets (denoted as LTS) against the selected baselines in terms of classification accuracy ratio (fraction of correct classifications) as shown in Table 6.3. The best method per dataset is highlighted in **bold**.

Our method LTS has a very large superiority in terms of Absolute Wins (17.28 absolute wins in 28 datasets against 13 baselines) and 1-to-1 wins, as indicated by the respected rows in the end of the table. Each dataset awards one point, which is split into fractions in case of draws. In addition, we compared the ranks of the classifiers and found out that LTS has a significantly better rank of $1.946 \pm 0.536$ against the closest baseline's (SVM) rank $4.554 \pm 1.180$. In order to bullet-proof

Table 6.3: Accuracy Ratios involving 28 Time Series Datasets and 13 Baselines

| # | Dataset | IG | KW | FST | MM | DTW | C4.5 | 1NN | NB | BN | RAF | ROF | SVM | FSH | LTS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Adiac | 0.299 | 0.266 | 0.156 | 0.271 | 0.491 | 0.243 | 0.253 | 0.281 | 0.251 | 0.304 | 0.307 | 0.238 | **0.558** | 0.542 |
| 2 | Beef | 0.500 | 0.333 | 0.567 | 0.300 | 0.433 | 0.600 | 0.833 | 0.733 | **0.900** | 0.600 | 0.700 | 0.867 | 0.517 | 0.800 |
| 3 | Beetle/Fly | 0.775 | 0.700 | 0.900 | 0.800 | 0.650 | 0.750 | **1.000** | 0.925 | 0.975 | 0.900 | 0.950 | 0.975 | 0.742 | 0.950 |
| 4 | Bird/Chicken | 0.850 | 0.875 | 0.900 | 0.875 | 0.725 | 0.900 | 0.975 | 0.875 | 0.950 | 0.950 | 0.925 | 0.950 | 0.843 | **1.000** |
| 5 | ChlorineConcentration | 0.588 | 0.520 | 0.535 | 0.521 | 0.634 | 0.565 | 0.569 | 0.460 | 0.571 | 0.576 | 0.635 | 0.562 | 0.578 | **0.743** |
| 6 | Coffee | 0.964 | 0.857 | **1.000** | 0.857 | 0.464 | 0.857 | **1.000** | 0.929 | 0.964 | **1.000** | 0.893 | **1.000** | 0.925 | **1.000** |
| 7 | DiatomSizeReduction | 0.722 | 0.621 | 0.765 | 0.448 | 0.925 | 0.752 | 0.935 | 0.788 | 0.902 | 0.804 | 0.830 | 0.922 | 0.869 | **0.951** |
| 8 | DP_Little | 0.654 | 0.680 | 0.603 | 0.710 | 0.493 | 0.659 | 0.728 | 0.735 | 0.729 | 0.730 | 0.747 | **0.752** | 0.566 | 0.727 |
| 9 | DP_Middle | 0.705 | 0.693 | 0.619 | 0.737 | 0.546 | 0.712 | 0.737 | 0.740 | 0.747 | 0.755 | 0.768 | **0.796** | 0.581 | 0.758 |
| 10 | DP_Thumb | 0.581 | 0.720 | 0.560 | 0.703 | 0.530 | 0.580 | 0.607 | 0.630 | 0.639 | 0.641 | 0.671 | 0.698 | 0.580 | **0.740** |
| 11 | ECGFiveDays | 0.775 | 0.872 | 0.990 | 0.928 | 0.828 | 0.962 | 0.984 | 0.964 | 0.995 | 0.933 | 0.986 | 0.990 | 0.996 | **1.000** |
| 12 | FaceFour | 0.841 | 0.443 | 0.750 | 0.409 | 0.830 | 0.761 | **1.000** | 0.977 | **1.000** | 0.875 | 0.989 | 0.977 | 0.909 | **1.000** |
| 13 | GunPoint | 0.893 | 0.940 | 0.953 | 0.920 | 0.913 | 0.907 | 0.980 | 0.920 | 0.993 | 0.960 | 0.987 | **1.000** | 0.932 | **1.000** |
| 14 | ItalyPowerDemand | 0.892 | 0.910 | 0.931 | 0.911 | 0.961 | 0.910 | 0.921 | 0.925 | 0.924 | 0.930 | 0.920 | 0.921 | 0.921 | **0.962** |
| 15 | Lighting7 | 0.493 | 0.480 | 0.411 | 0.274 | 0.726 | 0.534 | 0.493 | 0.575 | 0.658 | 0.644 | 0.658 | 0.699 | 0.601 | **0.877** |
| 16 | MedicalImages | 0.488 | 0.471 | 0.508 | 0.490 | 0.690 | 0.449 | 0.457 | 0.174 | 0.282 | 0.508 | 0.515 | 0.525 | 0.608 | **0.734** |
| 17 | MoteStrain | 0.825 | 0.840 | 0.840 | 0.840 | 0.816 | 0.844 | 0.903 | 0.888 | 0.891 | 0.846 | 0.870 | 0.887 | 0.785 | **0.913** |
| 18 | MP_Little | 0.664 | 0.697 | 0.578 | 0.703 | 0.558 | 0.634 | 0.685 | 0.688 | 0.695 | 0.714 | 0.752 | 0.750 | 0.565 | **0.758** |
| 19 | MP_Middle | 0.710 | 0.750 | 0.609 | 0.720 | 0.470 | 0.733 | 0.709 | 0.720 | 0.711 | 0.752 | 0.747 | 0.769 | 0.605 | **0.780** |
| 20 | Otoliths | 0.672 | 0.609 | 0.578 | 0.547 | 0.594 | 0.656 | 0.719 | 0.688 | 0.641 | 0.656 | 0.594 | 0.641 | 0.560 | **0.766** |
| 21 | PP_Little | 0.596 | 0.720 | 0.586 | 0.673 | 0.495 | 0.574 | 0.672 | 0.692 | 0.701 | 0.666 | 0.698 | **0.721** | 0.549 | 0.710 |
| 22 | PP_Middle | 0.614 | 0.683 | 0.581 | 0.697 | 0.499 | 0.625 | 0.685 | 0.698 | 0.714 | 0.705 | 0.754 | 0.759 | 0.580 | **0.767** |
| 23 | PP_Thumb | 0.608 | 0.713 | 0.591 | 0.730 | 0.526 | 0.595 | 0.677 | 0.694 | 0.695 | 0.678 | 0.728 | **0.755** | 0.536 | 0.715 |
| 24 | SonyAIBORobotSurface | 0.845 | 0.727 | **0.953** | 0.749 | 0.699 | 0.845 | 0.840 | 0.790 | 0.897 | 0.852 | 0.890 | 0.867 | 0.698 | 0.952 |
| 25 | Symbols | 0.780 | 0.557 | 0.801 | 0.574 | 0.934 | 0.471 | 0.856 | 0.780 | 0.923 | 0.846 | 0.844 | 0.846 | 0.930 | **0.959** |
| 26 | SyntheticControl | 0.943 | 0.900 | 0.957 | 0.857 | 0.973 | 0.903 | 0.930 | 0.780 | 0.767 | 0.890 | 0.920 | 0.873 | 0.917 | **1.000** |
| 27 | Trace | 0.980 | 0.940 | 0.980 | **1.000** | 0.990 | 0.980 | 0.980 | 0.980 | **1.000** | 0.980 | 0.980 | 0.980 | **1.000** | **1.000** |
| 28 | TwoLeadECG | 0.851 | 0.764 | 0.970 | 0.853 | 0.795 | 0.853 | 0.995 | 0.991 | 0.988 | 0.961 | 0.980 | 0.993 | 0.922 | **1.000** |
| **Absolute Wins** | | **0.00** | **0.00** | **1.20** | **0.25** | **0.00** | **0.00** | **1.53** | **0.00** | **1.58** | **0.20** | **0.00** | **4.70** | **1.25** | **17.28** |
| **LTS 1-to-1 Wins** | | **28** | **27** | **26** | **26** | **28** | **28** | **23** | **27** | **23** | **26** | **24** | **20** | **26** | **-** |
| **LTS 1-to-1 Draws** | | **0** | **0** | **1** | **1** | **0** | **0** | **2** | **0** | **2** | **1** | **1** | **2** | **1** | **-** |
| **LTS 1-to-1 Losses** | | **0** | **1** | **1** | **1** | **0** | **0** | **3** | **1** | **3** | **1** | **3** | **6** | **1** | **-** |
| **Rank Mean** | | **9.768** | **9.982** | **9.107** | **9.500** | **9.804** | **10.036** | **6.196** | **7.714** | **5.518** | **6.321** | **5.357** | **4.554** | **9.196** | **1.946** |
| **Rank Confidence Interval** | | **1.016** | **1.259** | **1.318** | **1.273** | **1.867** | **0.781** | **1.195** | **1.091** | **1.150** | **0.743** | **0.898** | **1.180** | **1.519** | **0.536** |
| **Rank Standard Deviation** | | **2.743** | **3.398** | **3.559** | **3.436** | **5.040** | **2.108** | **3.228** | **2.944** | **3.104** | **2.005** | **2.423** | **3.186** | **4.102** | **1.448** |
| **Wilcoxon Test p-values** | | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.000** | **0.003** | **0.000** | **-** |

our claim, we ran the well-known Wilcoxon signed ranks test [37] against all baselines and found out that all results are statistically significant at $p < 0.05$, as can be deduced by the p-values of the most bottom row.

### 6.6.3 Competitive Running Time

Since the idea of this chapter is entirely novel, our first priority is to evaluate its prediction accuracy rather than elaborating on speed-up techniques, as in the fast shapelets approach [95]. Nevertheless, we would like to show that our method is indeed feasible and competitive in terms of running time and faster than the exhaustive candidate search approach [62, 80]. We compared the time needed to find the best shapelet of each dataset against the F-Stat metric, which is the fastest quality metric [62, 80]. The best shapelet run-time comparison is advocated by our baseline [62], in order to ensure that methods can process the same number of candidates. As can be seen from Table 6.2, our method can learn the shapelet within a faster time (57 times faster in average) compared to the baseline, which is an indication that our method is practically feasible in terms of running time. Each execution of our method searched over five different shapelet sizes $\{0.025, 0.075, 0.125, 0.175, 0.2\} \times Q$ and the other parameters were set to $\eta = 0.01$, maxIter= 3000 and $\lambda_W = 0.001$.

## 6.7 Conclusion

In this chapter we introduced a novel perspective into learning time-series shapelets. In contrast to related work which searches for top shapelets from a pool of candidates, we propose a novel mathematical formulation of the task via a classification objective function. In addition, we introduced a learning algorithm which *learns* near-to-optimal shapelets by exploring shapelet interactions. An extensive experimentation on 28 time-series datasets and 13 baselines is conducted. Our method outperforms all the baselines with statistically significant margins in terms of both wins and ranks.

# Chapter 7

# Fast Extraction of Target-Driven Features

## Contents

**Highlights:** Whilst the shapelets learning method of the previous chapter is quite accurate, the gradients of its algorithm are computationally demanding. In order to overcome the runtime bottleneck, this chapter will present a fast shapelet discovery technique. The innovation of this method relies on pruning shapelet candidates using both their similarity and prediction accuracy. Overall, this fast shapelet method allows MB-scale datasets to be classified in seconds, while GB-scale datasets to be classified in minutes.

Shapelets are discovered by measuring the prediction accuracy of a set of potential (shapelet) candidates. The candidates typically consist of all the segments of a dataset, therefore, the discovery of shapelets is computationally expensive. This chapter proposes a novel method that avoids measuring the prediction accuracy of similar candidates in Euclidean distance space, through an online clustering/pruning technique. In addition, our algorithm incorporates a supervised shapelet selection that filters out only those candidates that improve classification accuracy. Empirical evidence on 45 univariate datasets from the UCR collection demonstrate that our method is 3-4 orders of magnitudes faster than the fastest existing shapelet-discovery method, while providing better prediction accuracy. In addition, we extended our method to multivariate time-series data, because the technique presented in this chapter finally supports the classification of large datasets. Impressive runtime results over 4 real-life multivariate datasets indicate that our method can classify MB-scale data in a matter of seconds and GB-scale data in a matter of minutes. The achievements do not compromise quality, on the contrary, our method is even superior to the multivariate baseline in terms of classification accuracy.

## 7.1 Introduction

In contrast to the high classification accuracy, discovering shapelets remains challenging in terms of runtime. The current discovery methods need to search for the

most predictive shapelets from all the possible segments of a time-series dataset [86, 126]. Since the number of possible candidates is high, the required time for evaluating the prediction quality of each candidate is prohibitive for large datasets. Therefore, the time-series research community has proposed several speed-up techniques [86, 95, 126], aiming at making shapelet discovery **feasible** in terms of time.

This chapter proposes a novel method that discovers time-series shapelets considerably faster than the fastest existing method. Our method follows the knowledge that time-series instances contain lots of similar segments. Often inter-class variations of time series depend on differences within small segments, with the remaining parts of the series being similar. Therefore, we hypothesize that the time needed to discover shapelets can be **scaled**-up by pruning candidate segments that are similar in Euclidean distance space. We introduce a fast distance-based clustering approach to prune future segments that are similar to previously considered ones. In addition, we propose a fast supervised selection of shapelets that filters out the qualitative shapelets using an incremental nearest-neighbor classifier. Extensive experiments conducted on real-life data demonstrate a large reduction (3-4 orders of magnitude) of the discovery time, by even gaining prediction accuracy with respect to baselines. The contributions of this paper can be short-listed as follows:

1. A fast pruning strategy for similar shapelets in Euclidean space involving a distance-based clustering approach;

2. A fast supervised selection of qualitative shapelets using an incremental nearest-neighbor classifier, conducted jointly with the pruning;

3. Extensive experimental results against the fastest existing univariate shapelet discovery methods on a large set of 45 time-series datasets.

4. Extension to multivariate time-series datasets showing that our method scales to GB-sized data

## 7.2 Chapter Notations

This chapter will adhere to the notational conventions of Table 7.1 and the forthcoming brief explanation.

| Symbol $\in$ Domain | Description |
|---|---|
| $R \in \mathbb{N}$ | Number of different candidate lengths |
| $\Phi \in \mathbb{N}^R$ | Set of shapelet lengths |
| $s \in \mathbb{R}^{\Phi_*}$ | A shapelet candidate |
| $\mathcal{D} : \left(\mathbb{R}^L \times \mathbb{R}^Q\right) \to \mathbb{R}_+$ | Distance between a shapelet and a series |
| $p \in [1, \ldots, 100]$ | Pruning distance percentile |
| $r \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8} \ldots\}$ | Dimensionality reduction ratio |
| $X \in \mathbb{R}^{N \times N}$ | Pairwise distances between series |
| $Y \in \mathbb{N}^N$ | Labels of the series |
| $\mathcal{A} \in \mathbb{R}^{*\times *}$ | Accepted shapelet candidates |
| $\mathcal{R} \in \mathbb{R}^{*\times *}$ | Rejected shapelet candidates |
| $V \in \mathbb{N}$ | Number of time-series dimensions |
| $\lvert s \rvert \in \mathbb{N}$ | Length (cardinality) of shapelet $s$ |
| $\sigma \in \mathbb{R} \to \mathbb{R}$ | The sigmoid logistic function |

Table 7.1: Notational Conventions of Chapter 7

## 7.3 Scalable Shapelet Discovery

### 7.3.1 Distances of Shapelets to Series as Classification Features

Throughout this paper we denote a time-series dataset having $N$ series of $Q$ points each, as $T \in \mathbb{R}^{N \times Q}$. While our method can work with series of arbitrary lengths, we define a single length $Q$ for ease of mathematical formalism. The distances of shapelets to series can be used as classification features, also known as shapelet-transformed features [62]. The distance of a candidate shapelet to the closest segment of a series can be perceived as a membership degree for that particular shapelet. Equations 7.1 and 7.2 formalize the minimum distances between a shapelet $s$ and the dataset $T$ as a vector of the Euclidean distances ($\mathcal{D}$) between the shapelet and the closest segment of each series. (The notation $T_{i,a:b}$ denotes a sub-sequence of series $T_i$ from the $a$-th element to the $b$-th element.)

$$\mathbf{MinDist}(s, T) := \begin{bmatrix} \mathcal{D}(s, T_1) \\ \mathcal{D}(s, T_2) \\ \vdots \\ \mathcal{D}(s, T_N) \end{bmatrix} \tag{7.1}$$

$$\mathcal{D}(s, T_i) := \min_{j=1,\ldots,Q-|s|+1} \left\| T_{i,j:j+|s|-1} - s \right\|^2 \tag{7.2}$$

An illustration of the minimum distances between shapelets and series is shown in Figure 7.1 for the TwoLeadECG dataset. Two shapelets (purple) are matched to four time series of two different classes (red and blue). Following the principle that Equation 7.2 states, the distance of a shapelet is computed to the closest series segment. The distances between training time series and the two shapelets can project the dataset to a 2-dimensional shapelet-transformed space, as shown on the right sub-plot. A nearest neighbor classifier and the corresponding classification decision boundary is also illustrated.

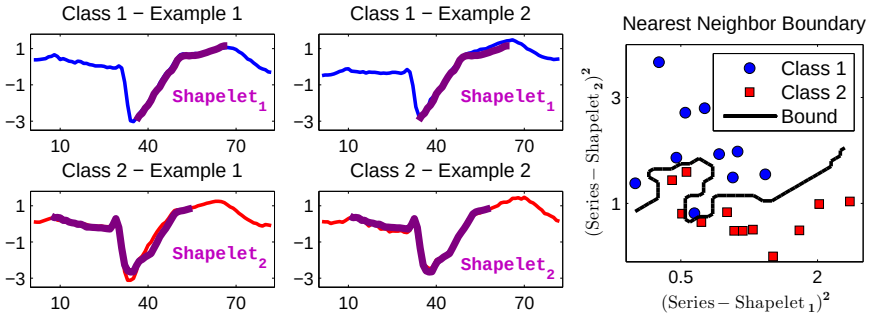



Figure 7.1: TwoLeadECG dataset: Aligning shapelets to the closest series segments, and on right the resulting 2-dimensional shapelet-transformed training data.

### 7.3.2 Quantification of Similarity Using a Distance Threshold

A time-series dataset generally contains lots of similar patterns spread over various instances. Since series from the same class generally follow a similar structure, similar patterns repeat over time series of the same class. Similarities can also be observed among time series of different classes, because often classes are discriminated by differences in small sub-sequences rather than the global structure. As

a result, we raise the hypothesis that existing state-of-the-art techniques, which exhaustively search all candidates, inefficiently consider lots of very similar patterns.

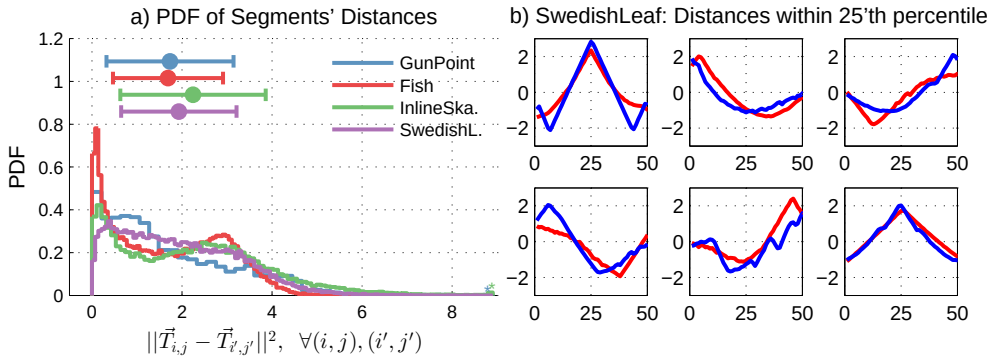



Figure 7.2: **a)**Distribution of distances among random pairs of candidates; **b)** Illustration of similar segments from the SwedishLeaf dataset with pairwise distances less than the 25−th percentile of the distribution in **a)**.

Figure 7.2 illustrates the distribution of distances among arbitrary pairs of candidate segments from various time series of the UCR collection of datasets [67]. As can be seen from sub-figure *a)*, the distribution of distances is highly skewed towards zero, which indicate that most candidates are very similar to each other. However, a threshold separation on the similarity distance is required to judge segments as being similar or not. We propose to use a threshold over the percentile on the distribution of distances. For instance, Figure 7.2.b) displays pairs of similar segments whose pairwise distances are within the 25-th percentile of the distance distribution.

The procedure of determining a distance threshold value, denoted $\epsilon$ and belonging to the $p$-th percentile of the distance distribution, is described in Algorithm 3. The algorithm selects a pair of random segments starting at indices $(i, j), (i', j')$ and having random shapelet lengths $\Phi_*$. Then a distribution is built by accumulating the distances of random pairs of segments and the distance value that corresponds to the desired percentile $p$ is computed from the sorted list of distance values. For instance, in case all the distance values are sorted from smallest to largest, then the 25-th percentile is the value at the index that belongs to 25% of the total indices.

In total, there are $\mathcal{O}(NQR)$ segments in a time-series dataset and the total number of pairs is $\frac{1}{2}(NQR)(NQR-1)$. However, in order to estimate the distribution of a set of values (here distances), one does not need to have access to the full population of values. On the contrary, a sample of values are sufficient for

**Algorithm 3**: **ComputeThreshold**: Compute the pruning similarity distance threshold $\epsilon$.

**Data**: Time series data $T \in \mathbb{R}^{N \times Q}$, Percentile $p \in [1, \ldots, 100]$, Shapelet Lengths $\Phi \in \mathbb{N}^R$
**Result**: Threshold distance $\epsilon \in \mathbb{R}$

$Z \leftarrow \emptyset$;
**for** $1, \ldots, NQ$ **do**
  Draw random shapelet length $\Phi_* \sim \mathcal{U}(\Phi_1, \ldots, \Phi_R)$ ;
  Draw segment indices $(i, j) \sim (\mathcal{U}(1, \ldots, N), \mathcal{U}(1, \ldots, Q - \Phi_* + 1))$ ;
  Draw segment indices $(i', j') \sim (\mathcal{U}(1, \ldots, N), \mathcal{U}(1, \ldots, Q - \Phi_* + 1))$ ;
  $Z \leftarrow Z \cup \left\{ \frac{1}{\Phi_*} ||T_{i,j:j+\Phi * R-1} - T_{i',j':j'+\Phi_*-1}||^2 \right\}$;
**end**
$Z \leftarrow \text{sort}(Z)$;
$\epsilon \leftarrow Z_{\lceil \frac{p}{100} N Q \rceil}$;
**return** $\epsilon$

estimating the distribution. In order to balance between a fast and accurate compromise we choose to select $NQ$-many random segment pairs for estimating the distance distributions. The runtime speed up success of Section 7.4.3 indicates that the distance threshold estimation is accurate.

### 7.3.3 Main Method: Scalable Discovery of Time-series Shapelets

The scalable discovery of time-series shapelets follows the two primary principles of this paper: *i)* Pruning of similar candidates, and *ii)* on-the-fly supervised selection of shapelets. The rationale of these principles is based on the knowledge that the majority of patterns from any specific time series are similar to patterns in other series of the same dataset. Therefore, it is computationally non-optimal to measure the quality of lots of very similar candidates. Instead, we aim at considering only a small nucleus of non-redundant candidates.

#### 7.3.3.1 Taxonomy of The Terms

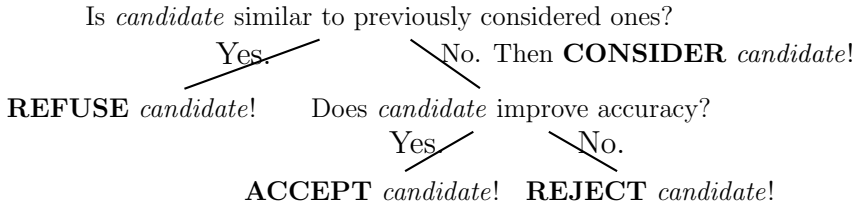
The fate of any candidate shapelet will be one of **refused**, **considered**, **accepted** and **rejected**. The decision tree below helps clarifying those terms.

**Algorithm 4**: **DiscoverShapelets**: Scalable discovery of shapelets

**Data**: Time series data $T \in \mathbb{R}^{N \times Q}$, Labels $Y \in \mathbb{N}^N$ Distance Threshold Percentile $p \in [1, \dots, 100]$, Piecewise Aggregate Approximation ratio $r \in \{\frac{1}{2}, \frac{1}{4}, \dots\}$, Shapelet lengths $\Phi \in \mathbb{N}^R$

**Result**: Accepted shapelets list $\mathcal{A} \in \mathbb{R}^{*\times *}$, Minimum Distances $D \in \mathbb{R}^{*\times *}$

$\epsilon \leftarrow \text{ComputeThreshold}(T, p, \Phi)$;
$\mathcal{A} \leftarrow \emptyset, \mathcal{R} \leftarrow \emptyset, D \leftarrow \emptyset, X \leftarrow \mathbf{0}_{N\times N}, \text{prevAccuracy} \leftarrow -\infty$;
**for** $1, \dots, NML$ **do**
  Draw random series: $i \sim \mathcal{U}\{1, \dots, N\}$;
  Draw random shapelet length: $\Phi_* \sim \mathcal{U}\{\Phi_1, \dots, \Phi_R\}$;
  Draw random segment start: $j \sim \mathcal{U}\{1, \dots, Q - \Phi_* + 1\}$;
  Selected random candidate: $s \leftarrow T_{i,j:j+\Phi_*-1}$;
  **if** $\neg \mathit{LookUp}(s, \mathcal{A}, \epsilon) \wedge \neg \mathit{LookUp}(s, \mathcal{R}, \epsilon)$ **then**
    $d^s \leftarrow \text{MinDist}(s, T)$ ;
    **for** $i = 1, \dots, N;\ m = i + 1, \dots, N$ **do**
      $X_{i,m} \leftarrow X_{i,m} + (d_i^s - d_m^s)^2$;
    **end**
    $\alpha \leftarrow \text{Accuracy}(X, Y)$;
    **if** $\alpha > \mathit{prevAccuracy}$ **then**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{s\}$;
      $D \leftarrow D \cup \{d^s\}$;
      $\text{prevAccuracy} \leftarrow \alpha$;
    **else**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{s\}$;
      **for** $i = 1, \dots, N;\ m = i + 1, \dots, N$ **do**
        $X_{i,m} \leftarrow X_{i,m} - (d_i^s - d_m^s)^2$;
      **end**
    **end**
  **end**
**end**
**return** $\mathcal{A}, D$



The similarity of a candidate is first evaluated by looking up whether a close candidate has been previously considered, i.e has been previously flagged as either accepted or rejected. The considered non-redundant (non-similar to previous) candidates are subsequently checked on whether they improve the classification accuracy of previously selected candidates, and are either marked as accepted or rejected.

We are presenting our method as Algorithm 4 and incrementally walking the reader through the steps. The algorithm is started by compressing the time series via the Piecewise Aggregate Approximation technique, to be detailed in Section 7.3.4. In order to prune similar candidates, the threshold distance $\epsilon$ is computed using Algorithm 3. Our method operates by populating two lists of accepted and rejected shapelets, denoted as $\mathcal{A}$ and $\mathcal{R}$, and storing a distance matrix $X$ for distances between series in the shapelet-transformed space.

#### 7.3.3.2 Pruning Similar Candidates

Random shapelet candidates, denoted $s$, are drawn from the training time series and a similarity search is conducted by looking up whether similar candidates have been previously considered (lines 4-8). Equation 7.3 formalizes the procedure as a similarity search over a list $\mathcal{L}$ (e.g., $\mathcal{A}$ or $\mathcal{R}$), considering candidates having same length ($length()$). Please note that in the concrete implementation we use a pruning of the Euclidean distance computations, by stopping comparisons exceeding the threshold $\epsilon$.

$$\textbf{LookUp}(s, \mathcal{L}, \epsilon) \;:=\; \exists q \in \mathcal{L} \;\;|\;\; ||s-q||^2 < \epsilon \;\wedge\; |s| = |q| \tag{7.3}$$

#### 7.3.3.3 Incremental Nearest Neighbor Distances

In case a candidate is found to be novel (not similar to previously considered), then the distance of the candidate to training series are computed using Equation 7.1 and stored as $d^s$. Our approach evaluates the **joint** accuracy of accepted shapelets, so far, using a nearest neighbor classifier over the shapelet-transformed data, i.e. distances of series to accepted shapelets.

When checking how does a new $(|\mathcal{A}|+1)$-th candidate influence the accuracy of $|\mathcal{A}|$ currently accepted candidates, an important speed-up trick can be used. We can pre-compute the distances among shapelet-transformed features in an incremental fashion. The distances among series in the shapelet-transformed space are stored in a distance matrix, denoted $X$, as shown in Equation 7.4.

$$X_{i,m}(D) = \sum_{j=1}^{|\mathcal{A}|} (D_{i,j} - D_{m,j})^2, \quad \forall i \in \{1,\ldots,N\}, \forall m \in \{1,\ldots,N\} \quad (7.4)$$

We propose a novel trick, which can add the distance contribution of a new candidate to the distance matrix in an incremental manner. When adding one more attribute $d^s$ to the shapelet-transformed data $D$, we can use the previously computed pair-wise distances to incrementally update the new pair-wise distances as shown in Equation 7.5.

$$\begin{aligned} X_{i,m}(D \cup d^s) &= \sum_{j=1}^{|\mathcal{A}|} (D_{i,j} - D_{m,j})^2 + (d_i^s - d_m^s)^2 \\ &= X_{i,m}(D) + (d_i^s - d_m^s)^2 \end{aligned} \quad (7.5)$$

Those steps correspond to lines 10-12 and 19-21 in Algorithm 4. It is trivial to verify that this technique can improve the runtime of a nearest neighbor from $\mathcal{O}(N^2|\mathcal{A}|)$ to $\mathcal{O}(N^2)$, which means that we can avoid recomputing distances among previously accepted $|\mathcal{A}|$-many shapelets, yielding a speed-up factor $|\mathcal{A}|$ for every *considered* shapelet candidate.

#### 7.3.3.4 Supervised Shapelet Selection

In case the contribution of a unique candidate improves the classification accuracy of a nearest neighbor classifier, then the shapelet is added to the accepted list and the distance vector is stored in a shapelet-transformed data representation $\mathcal{D}$, in order to be later on used for classifying the test instances. Otherwise, the shapelet is inserted to the rejected list and the contribution of the candidate to the distance matrix $X$ is rolled back. The classification accuracy of the distances between series and a set of shapelets is measured by the nearest neighbor accuracy of the cumulative distance matrix $X$. The accuracy over the training data is formalized in Equation 7.6.

$$\textbf{Accuracy}(X,Y) := \frac{1}{N} \left| \left\{ i \in \{1,\ldots,N\} \mid Y_i = Y_{\text{argmin}_{m,m\neq i} X_{i,m}} \right\} \right| \quad (7.6)$$

The mechanism described in Section 7.3.3.3 and Section 7.3.3.4 consists of a supervised variable selection for shapelet-transformed features [58]. The strategy is a "Forward greedy selection" where shapelets are *Accepted* incrementally if they improve the accuracy [58].

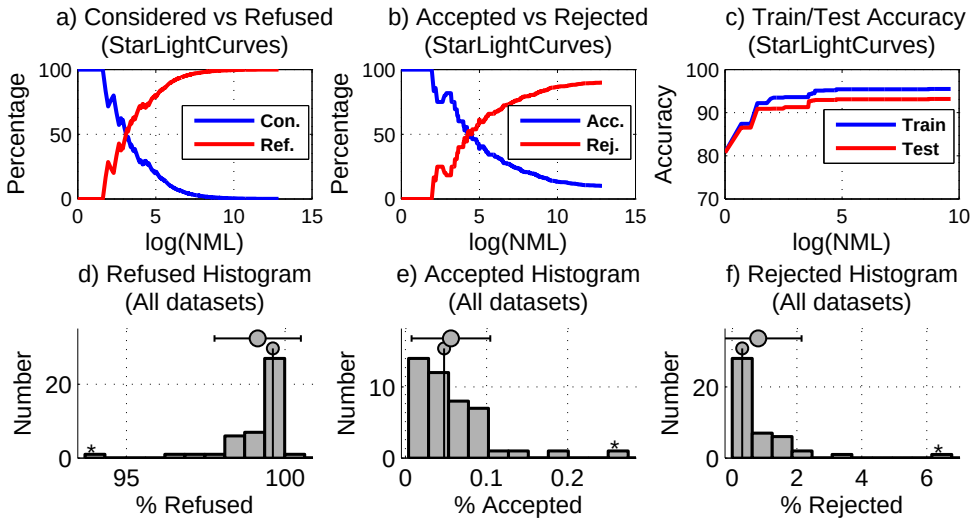



Figure 7.3: **a,b,c)** Relations of refused, rejected and accepted candidate shapelets, and the resulting accuracy, for the Starlight dataset; **d,e,f)** Histograms of refused, accepted and rejected candidate percentages over all 45 UCR datasets.

#### 7.3.3.5 Number of Sampled Candidates

Algorithm 4 samples shapelet candidates randomly, however the total number of sampled candidates is $NQR$, that upper bounds the total possible series segments of a dataset. Our method could perform competitively even if we would sample a subset of the total possible candidates, as indicated by Figure 7.3 plot c). That plot illustrates that the train and test accuracy on the StarLightCurves dataset converges well before trying out all the candidates. However, since the state of the art methods try out all the series segments as candidates, we also opted for the same approach. In that way, the runtime comparison against the baselines provides an isolated hint on the impact of the pruning strategy.

#### 7.3.3.6 An Illustration of The Process

We present the main idea of our method with the aid of Figure 7.3. Sub-figures *a), b), c)* display the progress of the method on the StarLightCurves dataset, the largest dataset from the UCR collection [67]. The fraction of considered (accepted+rejected) shapelets are shown in *a)* with respect to the total candidates in the X-axis. As can be seen, the first few candidates are considered until the accepted and rejected lists are populated with patterns from the dataset. Afterwards, the algorithm starts refusing (pruning/not considering) previously considered candidates within the 25-th percentile threshold, while in the end, an

impressive 99.97% of candidates are pruned. In fact this behavior is not special to the StarLightCurves dataset. We ran the algorithm over all the 45 datasets of the UCR collection and measured the fraction of refused candidates as displayed in the histogram of sub-figure *d)*. In average, 99.14% of candidates can be pruned, with cross-validated values $p, r$ on the training data for each dataset.

Among the considered candidates, a supervised selection of shapelets is carried on by accepting only those candidates that improve the classification accuracy. Sub-figure *b)* shows that the number of rejections overcomes the number of acceptances as candidates are evaluated, which validates the current belief that very few shapelets can accurately classify a dataset [126]. As a consequence of the accepted shapelets, the train and test accuracy of the method on the dataset is improved as testified by sub-figure *c)*. With respect to all datasets of the UCR collection, histograms of sub-figures *d), e)* show that on average **only** 0.06% of candidates are accepted and 0.81% are rejected.

#### 7.3.3.7 A further intuition

The similarity based pruning of candidates can be compared to a particular type of clustering where the considered candidates represent centroids. In principle, the mechanism resembles fast online clustering methods [1]. Figure 7.4 illustrates how the considered shapelets (blue) can be perceived as an $\epsilon$ threshold clustering of the refused candidates (gray). Each cluster is represented by a hyper-ball of radius $\epsilon$ in a $\Phi_*$-dimensional space, for $\Phi_*$ being the shapelet length. For the sake of illustration we selected random points of the shapelets and printed 2-dimensional plots of the 6 considered candidates and 7036 refused candidates from the MALLAT dataset.

The threshold distance used for pruning similar candidates has a significant effect on the quantity of refused candidates. Figure 7.5 shows that an increase of the percentile parameter both deteriorates the classification accuracy (sub-figure *a)*) and significantly shortens the running time (sub-figure *b)*). The higher the distance threshold percentile, the more distant segments will be considered similar and subsequently more candidates will be refused. In order to avoid a severe accuracy deterioration, the percentile parameter $p$ needs to be fixed by cross-validating over the training accuracy.

### 7.3.4 Piecewise Aggregate Approximation (PAA)

The Piecewise Aggregate Approximation (PAA) is a dimensionality reduction technique that shortens time series by averaging consecutive values [22]. An defi-

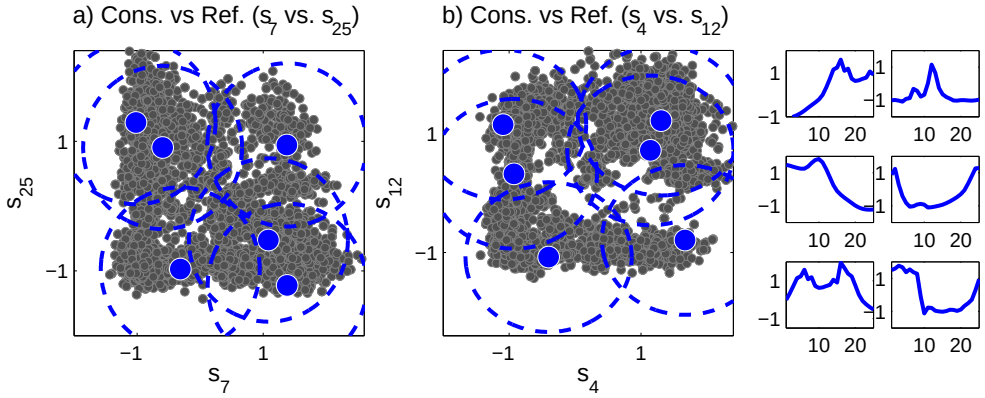



Figure 7.4: Refused (gray) candidates versus Considered ones (blue), together with the distance threshold circles, are shown for MALLAT dataset. Considered shapelets are displayed on the right. Parameters: $r = 0.125$, $p = 25$, (i.e. radius is $\epsilon = 1.26$), $\Phi_* = 25$.

nition of PAA was previously described in Section 3.3.2. In addition , Algorithm 5 illustrates how the time series of a dataset can be compressed by a specified ratio $r$. For instance, if $r = \frac{1}{4}$ then every four consecutive points are replaced by their average values.

PAA significantly reduces the discovery time of shapelets as shown in Figure 7.6.b for selected datasets. Moreover, subfigure a) shows that the classification accuracy does not deteriorate significantly because time-series data can be compressed without undermining the series pattern.

The exact amount of PAA reduction and the percentile of the pruning similarity threshold are hyper-parameters that need to be fixed per each dataset using the training data. For instance, Figure 7.6.c illustrates the accuracy heatmap on the 50words dataset as a result of alternating both parameters. As shown,

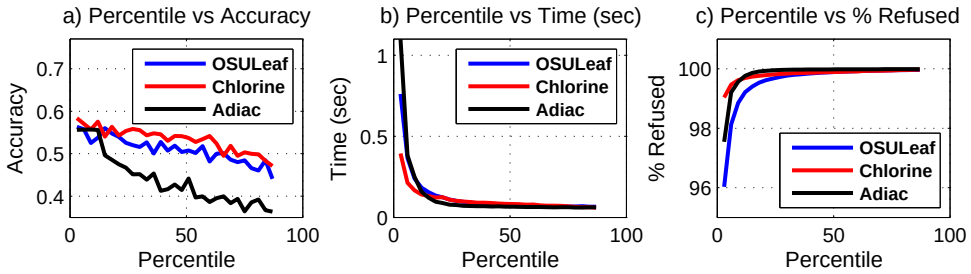



Figure 7.5: Impact of alternating the distance threshold's **percentile** ($p$) value on accuracy, discovery time and the fraction of refused candidates.

**Algorithm 5**: **PiecewiseAggregateApproximation**: Compress every series by a ratio $r$.

**Data**: Time series data $T \in \mathbb{R}^{N \times Q}$, PAA ratio $r \in \left\{\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots\right\}$
**Result**: $T^{\text{PAA}} \in \mathbb{R}^{N \times \lceil Q\, r \rceil}$

$T \leftarrow \mathbf{0}_{N \times \lceil Q\, r \rceil}$;
**for** $i = 1, \dots, N$, $j = 1, \dots, \lceil Q\, r \rceil$ **do**
  **for** $k = \lceil \frac{1}{r}(j-1)+1 \rceil, \dots, \lceil \frac{j}{r} \rceil$ **do**
    $T^{\text{PAA}}_{i,j} \leftarrow T^{\text{PAA}}_{i,j} + T_{i,k}$;
  **end**
  $T^{\text{PAA}}_{i,j} \leftarrow T^{\text{PAA}}_{i,j}\ r$;
**end**
**return** $T^{PAA}$

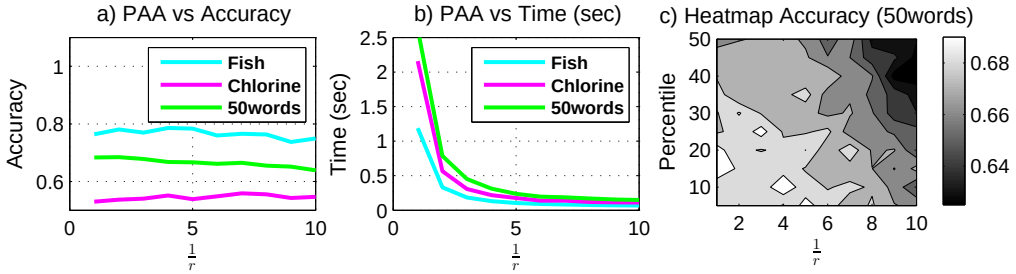



Figure 7.6: **a,b)** Consequence of PAA into accuracy and running time; **c)** Grid sensitivity of the impact of PAA and the percentile distance threshold over accuracy.

the best accuracy is achieved for moderate values of percentile threshold and compression. In contrast, (i) excessive compression and (ii) high threshold percentiles can deteriorate accuracy by (i) destroying informative local patterns by compression and (ii) pruning qualitative variations of shapelet candidates.

### 7.3.5 Algorithmic Analysis of the Runtime Speed-Up

The runtime of shapelet discovery algorithms, which explore candidates among series segments, is upper bounded by the number of candidates in a dataset. Given $N$-many training series of length $Q$, the total number of shapelet candidates has an order of $\mathcal{O}(NQ^2)$, while the time needed to find the best shapelet is $\mathcal{O}(N^2Q^4)$. Please note that the discovery time is quadratic in terms of the number of candidates. Applying Piecewise Aggregate Approximation (PAA), in order to reduce the length of time-series by a ratio $r \in \{\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots, ...\}$, does alter

the runtime complexity into $\mathcal{O}\left(N^2\left(r\,Q\right)^4\right)$ translated to $\mathcal{O}\left(\mathbf{r^4}N^2Q^4\right)$. In other words, PAA reduces the running time by a factor of $\mathbf{r^4}$. Furthermore, similarity pruning of candidates has a determinant role in reducing the runtime complexity. Let us denote the fraction of considered candidates as $f = \frac{\#\text{accepted}+\#\text{rejected}}{NQ^2}$. Therefore, if executed after a PAA reduction, our algorithm reduces the number of candidates to $\mathcal{O}\left(fN\left(rQ\right)^2\right)$ and impacts the total runtime complexity by $\mathcal{O}\left(fN\left(rQ\right)^2 \times \left(N\left(rQ\right)^2 + 2N^2\right)\right)$, which is upper bounded by $\mathcal{O}\left(fr^4N^2Q^4\right)$, since usually $\left(rQ\right)^2 \gg 2N$. Ultimately, the expected runtime reduction factor achieved by this paper is upper-bounded by $\mathbf{fr^4}$.

There is an additional term that adds up into the runtime complexity: the time needed to check whether any sampled candidate has been previously considered. Such a complexity is $\mathcal{O}\left(N(rQ^2) \times f|r\Phi_*|\right)$, in other words, all candidates times the time needed to search for $\epsilon$ similarity on the accepted and rejected lists ($f$-considered candidates having length $|r\Phi_*|$). Since $|r\Phi_*| \sim \mathcal{O}\left(rQ\right)$, then the whole operation has a final complexity of $\mathcal{O}\left(fr^3NQ^3\right)$. Such a complexity is smaller than the time needed to evaluate the accuracy of the candidates ($\mathcal{O}\left(fr^4N^2Q^4\right)$), therefore does not alter the big-O complexity.

Let us illustrate the theoretically expected speed-up via an example. Assume we compress time-series into a quarter of the original lengths, i.e. $r = \frac{1}{4}$. The average fraction of considered shapelets in the UCR datasets is $f = 0.0086$, as previously displayed in Figure 7.3. Therefore, a run-time reduction factor of $fr^4 = (0.0086)\,(0.065) \approx 5.3 \times 10^{-4}$ is expected. As shown, the expected theoretic runtime speedup can be 4 orders of magnitude compared to the exhaustive shapelet discovery. A detailed analysis of the effects of the dimensionality reduction (PAA compression) and pruning on the runtime performance is provided in Section 7.4.6. Furthermore, in Section 7.4.3 we will empirically demonstrate that our method is faster than existing shapelet discovery methods.

### 7.3.6 Effect Analysis of Supervised Shapelet Selection

In this subsection we analyze the effects of the supervised shapelet selection mechanism. In particular, one could ask whether the incremental Nearest Neighbor (NN) method of Section 7.3.3.3 is better than not pruning based on accuracy. Stated alternatively, would *accepting* all *considered* candidates (no *rejection* as per the taxonomy of Section 7.3.3.1) be equally preferable?

There are two primary reasons why an incremental NN is needed: *interpretability* and *classification time*. Meanwhile, Figure 7.7 helps clarifying both points. One of the motivations for shapelets is interpretability, therefore visual

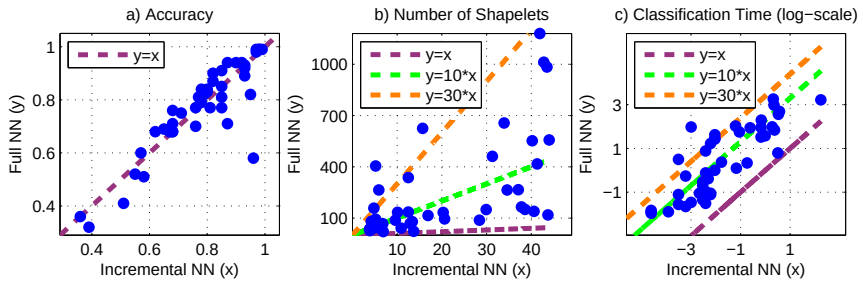



Figure 7.7: Comparing an Incremental NN against a Full NN in terms of accuracy, model complexity and classification time on 45 datasets from the UCR collection

comprehension demands a small set of shapelets [126]. As is seen in Figure 7.7.b) a full NN (no *rejected* candidates) ends up having on average 1477% more accepted candidates than our incremental approach. As a form of Variable Subset Selection, our incremental NN is expected to achieve comparable accuracy compared to a NN with a full set of features. As Figure 7.7.a) indicates, the full NN has slightly higher accuracy values, however, the differences are way insignificant according to a Wilcoxon signed rank test indicating a p-value of $p = 0.65272$ with a significance level of $p \leq 0.05$. The last argument in favor of an incremental NN approach is the classification time, which is a trivial consequence of having more features (i.e. more accepted shapelets). Figure 7.7.c) shows the comparisons of classification times between the two approaches, with the full NN being on average 1566% slower.

## 7.4 Experimental Results

### 7.4.1 Baselines

In order to evaluate the efficiency of the proposed method Scalable Shapelet Discovery (denoted by **SD**), the fastest state-of-the-art shapelet discovery methods were selected, being:

1. **Logical Shapelet [86] (denoted as LS)**: advances the original shapelet discovery method [126] by one order of magnitude, via: (i) caching and reusing computations, and (ii) applying an admissible pruning of the search space [86].

2. **Fast Shapelet [95] (denoted as FS)**: is a recent state-of-the-art method that proposes a random projection technique on the SAX representation by filtering potential candidates [95]. FS has been shown to reduce the shapelet discovery time of LS by two to three orders of magnitude [95].

3. **Improved Fast Shapelet (denoted as FS++)**: is a variation of FS that we created for the sake of being fair to the FS baseline. The original FS paper iterates through all the shapelet lengths from one to the length of the series. In comparison, our method SD iterates through a subset of the possible lengths ($\Phi$) as mentioned in Section 7.4.2. In order to be fair (with respect to runtime), we created a variant of the FS, named FS++, that also iterates through the same subsets of shapelet lengths that SD does.

The comparison against the listed state-of-the-art methods will show the efficiency of our method in terms of runtime scalability. When proposing a faster solution to a supervised learning task, it is crucial to also demonstrate that the speed-up does not deteriorate the prediction accuracy. For this reason, we payed attention to additionally compare the classification accuracy against the baselines.

### 7.4.2 Setup and Reproducibility

In order to demonstrate the speed-up achievements of the proposed shapelet discovery method, we use the popular collection of time-series datasets from the UCR collection [67]. The collection includes 45 univariate time-series datasets of different number of instances, different number of classes and lengths, found on [67].

Our Scalable Shapelet Discovery method, denoted as SD, requires the tuning of two parameters, the aggregation ratio $r$ and the threshold percentile $p$. The parameters were searched for each dataset via cross-validation using only the training data. The combination $(r, p)$ that yielded the highest accuracy on the training set was selected. A grid search was conducted with parameter ranges being $r \in \left\{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\right\}$ and $p \in \{15, 25, 35\}$. We start with the fastest configuration $r = \frac{1}{8}$ and $p = 35$. Subsequently we increase $r$ and decrease $p$ with the values of the range, one at a time. The selection stops when there is no more increase in accuracy as a result of relaxing the dimensionality reduction $r$ and threshold $p$. Finally, the winning combination of parameters was applied over the test data. We would like to note that we used three shapelet lengths for all our experiments, i.e. $L = 3$ and $\Phi = \{0.2Q, 0.4Q, 0.6Q\}$. In order to neutralize the randomness

effect (lines 4-6 of Algorithm 4), all the results of our method represent the averages over five different repetitions.

We used the Java programming language to implement our method (SD), while the other baselines (LS, FS, FS++) are implemented in C++. We decided to use the C++ source codes provided and optimized by the respective baseline paper authors [86, 95], in order to avoid typical allegations on inefficient re-implementations. Finally, we are presenting the exact number of accepted shapelets per each dataset and the respective percentages of the accepted, rejected and refused candidates in the columns merged under "SD Performance". **All** experiments (both our method and the baselines) were conducted in a *Sun Grid Engine* distributed cluster with 40 node processors, each being *Intel Xeon* E5-2670v2 with speed 2.50GHz and 64GB of shared RAM for all nodes. The operating system was *Linux CentOS* 6.3. All the experiments were launched using the same cluster parameters.

### 7.4.3 Highly Qualitative Runtime Results

The empirical results include both the discovery time and the classification accuracy of our method SD against baselines for 45 UCR datasets. Table 7.2 contains a list of results per dataset, where the discovery time is measured in seconds. A time-out threshold of 24 hours was set for the discovery of shapelets of a single dataset. As can be seen, the Logical Shapelet (LS) exceeded the time-out threshold in a considerable number of datasets. The reader is invited to notice that 24 hours (86400 seconds) is a very large threshold, given that our method SD often finds the shapelets within a fraction of one second, as for instance in the 50words dataset.

It can be clearly deduced that our method SD is faster than the fastest existing baselines LS [86] and FS [95]. There is no dataset where any of the baselines is faster. Even, our modification of FS, i.e. the FS++, is considerably slower than SD. For instance, it took only 3.19 seconds for our method to find the shapelets of the StarLightCurves dataset, which has 1000 training instances each having 1024 points. The high-level conclusion from the presented discovery time results is: **Since the introduction of shapelets in 2009, time-series community believed shapelets are very useful classification patterns, but finding them is slow. This paper demonstrates that shapelets can be discovered very fast.**

The discovery time measurements do not include the time needed by a practitioner to tune the parameters of the methods. While our method has two

Table 7.2: **Parameters** of SD and **Runtime Results** of SD and State-of-the-art baselines over 45 UCR datasets (n/a denotes a 24h time-out)

| No | Dataset | SD Params. | | Discovery Time (seconds) | | | |
|---|---|---|---|---|---|---|---|
| | | r | p | LS | FS | FS++ | SD |
| 1 | 50words | 0.250 | 35 | n/a | 2198.1 | 35.2 | **0.36** |
| 2 | Adiac | 0.500 | 15 | 12683.2 | 332.6 | 6.4 | **0.25** |
| 3 | Beef | 0.125 | 35 | 242.3 | 194.9 | 1.9 | **0.03** |
| 4 | CBF | 0.500 | 35 | 66.9 | 10.9 | 0.4 | **0.03** |
| 5 | Chlorine. | 0.125 | 15 | 36402.3 | 760.3 | 13.9 | **0.17** |
| 6 | CinC_ECG. | 0.125 | 25 | 2150.0 | 4398.9 | 9.9 | **0.34** |
| 7 | Coffee | 0.250 | 35 | 621.9 | 22.5 | 0.2 | **0.03** |
| 8 | Cricket_X | 0.250 | 35 | n/a | 3756.0 | 47.9 | **0.63** |
| 9 | Cricket_Y | 0.250 | 35 | n/a | 3605.7 | 45.7 | **0.52** |
| 10 | Cricket_Z | 0.250 | 35 | n/a | 4679.2 | 46.2 | **0.67** |
| 11 | Diatom. | 0.125 | 15 | 184.3 | 15.6 | 0.2 | **0.02** |
| 12 | ECG200 | 0.125 | 15 | 618.8 | 16.3 | 0.9 | **0.04** |
| 13 | ECGFive. | 0.500 | 15 | 47.6 | 3.6 | 0.1 | **0.03** |
| 14 | FaceAll | 0.500 | 35 | 16255.5 | 757.5 | 27.0 | **1.25** |
| 15 | FaceFour | 0.500 | 35 | 561.2 | 102.9 | 1.0 | **0.11** |
| 16 | FacesUCR | 0.500 | 35 | 2528.5 | 280.3 | 8.7 | **0.33** |
| 17 | Fish | 0.250 | 25 | 11153.0 | 935.6 | 6.7 | **0.16** |
| 18 | Gun_Point | 0.500 | 25 | 266.1 | 9.5 | 0.3 | **0.04** |
| 19 | Haptics | 0.500 | 25 | n/a | 12491.0 | 31.1 | **1.78** |
| 20 | InlineSkate | 0.125 | 15 | n/a | 22677.2 | 42.6 | **0.61** |
| 21 | ItalyPower. | 1.000 | 25 | 4.9 | 0.4 | 0.1 | **0.02** |
| 22 | Lighting2 | 0.500 | 35 | 5297.6 | 1131.3 | 5.0 | **1.89** |
| 23 | Lighting7 | 0.500 | 35 | 8619.3 | 322.8 | 3.7 | **0.43** |
| 24 | MALLAT | 0.125 | 35 | 1254.9 | 1736.5 | 6.2 | **0.08** |
| 25 | MedicalImages | 0.500 | 35 | 19325.2 | 371.5 | 8.5 | **0.60** |
| 26 | MoteStrain | 1.000 | 15 | 6.9 | 3.1 | 0.1 | **0.05** |
| 27 | Non.Fat.ECG.1 | 0.250 | 25 | n/a | 70970.6 | 254.2 | **7.03** |
| 28 | Non.Fat.ECG.2 | 0.125 | 25 | n/a | 50898.0 | 232.8 | **4.99** |
| 29 | OliveOil | 0.125 | 15 | 502.3 | 107.2 | 0.8 | **0.05** |
| 30 | OSULeaf | 0.125 | 25 | 14186.5 | 1629.7 | 20.0 | **0.15** |
| 31 | Sony.I | 1.000 | 35 | 4.6 | 1.1 | 0.1 | **0.02** |
| 32 | Sony.II | 1.000 | 35 | 9.8 | 1.3 | 0.1 | **0.03** |
| 33 | StarLight. | 0.125 | 25 | n/a | 21473.5 | 78.5 | **3.19** |
| 34 | SwedishLeaf | 0.500 | 25 | 11953.6 | 451.7 | 12.9 | **0.36** |
| 35 | Symbols | 0.250 | 25 | 894.3 | 93.0 | 0.6 | **0.04** |
| 36 | synthetic. | 0.250 | 35 | 3667.4 | 63.9 | 3.6 | **0.07** |
| 37 | Trace | 0.500 | 35 | 4626.9 | 181.0 | 1.7 | **0.13** |
| 38 | Two_Patterns | 0.500 | 35 | 65783.1 | 957.2 | 37.7 | **1.71** |
| 39 | TwoLeadECG | 1.000 | 25 | 14.3 | 1.3 | 0.03 | **0.02** |
| 40 | uWave.X | 0.250 | 25 | n/a | 4827.5 | 54.1 | **4.94** |
| 41 | uWave.Y | 0.250 | 25 | n/a | 4379.6 | 56.6 | **3.69** |
| 42 | uWave.Z | 0.125 | 25 | n/a | 5215.9 | 50.9 | **1.83** |
| 43 | wafer | 0.500 | 35 | 34653.1 | 190.5 | 5.0 | **1.39** |
| 44 | WordsS. | 0.250 | 25 | n/a | 1140.0 | 18.7 | **0.31** |
| 45 | yoga | 0.250 | 15 | 11389.0 | 1711.6 | 11.2 | **0.34** |
| **Total Wins** | | | | **0** | **0** | **0** | **45** |
| **Average Rank** | | | | **0.000** | **0.000** | **0.000** | **1.000** |

Table 7.3: **Parameters** of SD and **Classification Accuracy Results** of SD and SOTA baselines over 45 UCR datasets (n/a denotes a 24h time-out)

| No | Dataset | #Acc | Classification Accuracy | | | |
|---|---|---|---|---|---|---|
| | | | LS | FS | FS++ | SD |
| 1 | 50words | 39 | n/a | 0.511 | 0.446 | **0.680** |
| 2 | Adiac | 28 | **0.586** | 0.574 | 0.486 | 0.583 |
| 3 | Beef | 5 | **0.567** | 0.513 | 0.503 | 0.507 |
| 4 | CBF | 5 | 0.886 | 0.935 | 0.907 | **0.975** |
| 5 | Chlorine. | 13 | **0.618** | 0.579 | 0.558 | 0.553 |
| 6 | CinC_ECG. | 13 | 0.699 | 0.751 | 0.656 | **0.773** |
| 7 | Coffee | 4 | **0.964** | 0.921 | 0.907 | 0.961 |
| 8 | Cricket_X | 43 | n/a | 0.472 | 0.368 | **0.672** |
| 9 | Cricket_Y | 42 | n/a | 0.480 | 0.464 | **0.675** |
| 10 | Cricket_Z | 44 | n/a | 0.438 | 0.376 | **0.673** |
| 11 | Diatom. | 4 | 0.801 | 0.886 | **0.928** | 0.896 |
| 12 | ECG200 | 10 | **0.870** | 0.766 | 0.786 | 0.818 |
| 13 | ECGFive. | 5 | 0.994 | **0.995** | 0.994 | 0.953 |
| 14 | FaceAll | 40 | 0.659 | 0.631 | 0.571 | **0.714** |
| 15 | FaceFour | 6 | 0.489 | **0.917** | 0.881 | 0.820 |
| 16 | FacesUCR | 31 | 0.662 | 0.703 | 0.654 | **0.847** |
| 17 | Fish | 14 | 0.777 | **0.809** | 0.785 | 0.755 |
| 18 | Gun_Point | 6 | 0.893 | **0.933** | 0.915 | 0.931 |
| 19 | Haptics | 13 | n/a | **0.376** | 0.347 | 0.356 |
| 20 | InlineSkate | 13 | n/a | 0.266 | 0.282 | **0.385** |
| 21 | ItalyPower. | 6 | **0.936** | 0.877 | 0.796 | 0.920 |
| 22 | Lighting2 | 9 | 0.426 | 0.707 | 0.698 | **0.795** |
| 23 | Lighting7 | 16 | 0.548 | 0.630 | 0.485 | **0.652** |
| 24 | MALLAT | 7 | 0.656 | **0.939** | 0.926 | 0.926 |
| 25 | MedicalImages | 34 | 0.587 | 0.596 | 0.494 | **0.676** |
| 26 | MoteStrain | 5 | **0.832** | 0.783 | 0.767 | 0.783 |
| 27 | Non.Fat.ECG.1 | 41 | n/a | 0.766 | 0.622 | **0.814** |
| 28 | Non.Fat.ECG.2 | 44 | n/a | 0.802 | 0.635 | **0.855** |
| 29 | OliveOil | 5 | **0.833** | 0.723 | 0.773 | 0.790 |
| 30 | OSULeaf | 21 | **0.686** | 0.680 | 0.555 | 0.566 |
| 31 | Sony.I | 4 | **0.860** | 0.686 | 0.802 | 0.850 |
| 32 | Sony.II | 5 | 0.846 | 0.792 | **0.945** | 0.780 |
| 33 | StarLight. | 20 | n/a | **0.942** | 0.932 | 0.933 |
| 34 | SwedishLeaf | 30 | 0.813 | 0.779 | 0.725 | **0.849** |
| 35 | Symbols | 4 | 0.643 | **0.933** | 0.756 | 0.865 |
| 36 | synthetic. | 11 | 0.470 | 0.922 | 0.870 | **0.983** |
| 37 | Trace | 7 | **1.000** | 0.994 | 0.999 | 0.965 |
| 38 | Two_Patterns | 38 | 0.539 | 0.310 | 0.753 | **0.981** |
| 39 | TwoLeadECG | 4 | 0.856 | **0.928** | 0.798 | 0.867 |
| 40 | uWave.X | 44 | n/a | 0.707 | 0.580 | **0.761** |
| 41 | uWave.Y | 41 | n/a | 0.608 | 0.466 | **0.671** |
| 42 | uWave.Z | 37 | n/a | 0.627 | 0.565 | **0.676** |
| 43 | wafer | 10 | **0.999** | 0.998 | 0.949 | 0.993 |
| 44 | WordsS. | 35 | n/a | 0.437 | 0.389 | **0.625** |
| 45 | yoga | 17 | **0.740** | 0.705 | 0.697 | 0.625 |
| **Total Wins** | | | **13** | **9** | **2** | **21** |
| **Average Rank** | | | **2.313** | **2.178** | **3.089** | **1.889** |

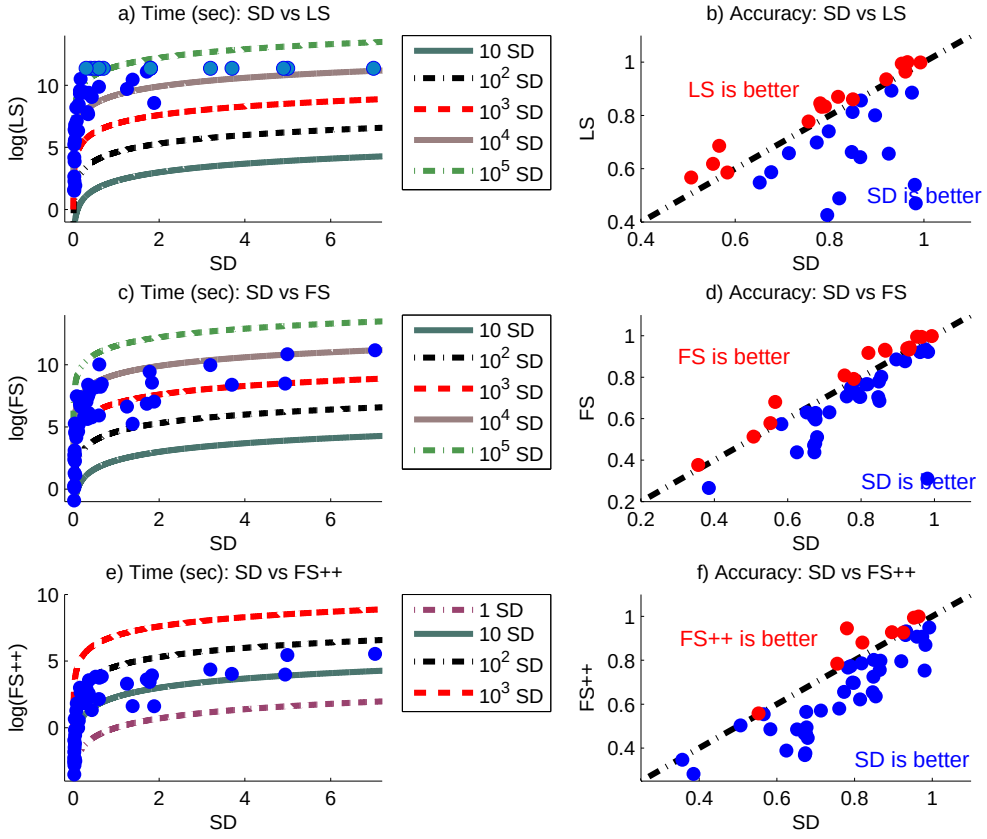



Figure 7.8: Time and accuracy comparison of our method (denoted **SD**) against state-of-the-art methods both in terms of discovery time and classification accuracy for all the 45 UCR datasets.

parameters ($p$ and $r$, totaling $3 \times 4 = 12$ combinations, see Section 7.4.2), the strongest baseline (Fast Shapelet) has more parameters, concretely four: the reduced dimensionality and cardinality of SAX, the random projection iterations and the number of SAX candidates (denoted d,c,r,k in the original paper [95]).

### 7.4.4 Competitive Prediction Accuracy

In addition, our results are atypical in another positive aspect. Most scalability papers propose speed-ups of the learning time by sacrificing a certain fraction of the prediction accuracy. In contrast, our results show that our method is both faster and more accurate than the baselines. The winning method that achieves the highest accuracy on each dataset (on each row) is distinguished in bold. Our

method has more wins than the baselines (21 wins against 13 of the second best method) and also a better rank (1.889 against 2.178 of the second best method). The accuracy improvement arises from the joint interaction of accepted shapelets as predictors (distance matrix X in Algorithm 4), while the baselines measure the quality of each shapelet separately, without considering their interactions during the discovery phase [86, 95, 126]. Incorporating the interactions among shapelets into the prediction model has been recently shown to achieve high classification accuracy [54].

### 7.4.5 Speed-Up Analysis

In order to show the speed-up factor of our method with respect to the (former) state-of-the-art, we provide another presentation of the results in Figure 7.8. The three plots on the left side show the discovery time of SD in x-axis and the logarithm of the discovery time of each baseline as the y-axis. As can be easily observed from the illustrative order lines, SD is 4 to 5 orders of magnitude faster than the Logical Shapelet (LS) and 3 to 4 orders of magnitude faster than the Fast Shapelet (FS). The datasets where LS exceeds the 24 hour threshold are depicted in light blue. In addition, FS++ is faster than FS because it iterates over less shapelet length sizes, yet it is still 1 to 2 orders of magnitude slower than SD.

The plots on the right represent scatter plots of the classification accuracy of SD against the baselines. While generally better than LS and FS, our method SD is largely superior to FS++. Such a finding indicates that the accuracy of the Fast Shapelet (FS) is dependent on trying shapelet candidates from a fine-grained set of lengths, while our method is very accurate even though it iterates over few shapelet lengths.

### 7.4.6 A Modular Decomposition of the Performance

We have already seen that our proposed method, SD, outperforms significantly the state-of-the-art in terms of runtime and produces even better prediction accuracy. Nevertheless, there are a couple of questions that can be addressed to our method, such as:

1. What fraction of SD's runtime reduction is attributed to the novel candidate pruning and what fraction to the PAA compression?

2. To what extent does pruning deteriorate the prediction accuracy?

Table 7.4: Modular Decomposition of The Performance of Our Method (SD), N/A Denotes a 24H time-out

| Dataset | Discovery Time (seconds) | | | | Classification Accuracy | | | |
|---|---|---|---|---|---|---|---|---|
| | ✘PAA ✘prun. | ✔PAA ✘prun. | ✘PAA ✔prun. | ✔PAA ✔prun. | ✘PAA ✘prun. | ✔PAA ✘prun. | ✘PAA ✔prun. | ✔PAA ✔prun. |
| 50words | 4028.85 | 154.74 | 5.24 | 0.36 | 0.684 | **0.701** | 0.679 | 0.680 |
| Adiac | 799.06 | 153.21 | 0.89 | 0.25 | **0.624** | 0.555 | 0.604 | 0.583 |
| Beef | 61.35 | 0.65 | 0.54 | 0.03 | 0.533 | **0.600** | 0.500 | 0.507 |
| CBF | 2.22 | 0.57 | 0.37 | 0.03 | **0.992** | 0.964 | 0.929 | 0.975 |
| Chlorine. | 1598.05 | 30.14 | 2.40 | 0.17 | 0.527 | **0.596** | 0.539 | 0.553 |
| CinC_ECG. | 3718.48 | 11.74 | 12.71 | 0.34 | **0.809** | 0.768 | 0.776 | 0.773 |
| Coffee | 11.79 | 1.27 | 0.39 | 0.03 | **0.964** | 0.893 | 0.893 | 0.961 |
| Cricket_X | 4218.58 | 141.80 | 23.63 | 0.63 | **0.697** | **0.697** | 0.669 | 0.672 |
| Cricket_Y | 3953.86 | 137.75 | 14.20 | 0.52 | **0.715** | 0.687 | 0.677 | 0.675 |
| Cricket_Z | 5313.96 | 132.06 | 40.17 | 0.67 | 0.700 | 0.682 | **0.726** | 0.673 |
| Diatom. | 5.00 | 0.50 | 0.50 | 0.02 | 0.915 | **0.948** | 0.827 | 0.896 |
| ECG200 | 14.59 | 0.70 | 0.42 | 0.04 | 0.820 | 0.800 | **0.830** | 0.818 |
| ECGFive. | 1.41 | 0.40 | 0.36 | 0.03 | **0.999** | 0.945 | 0.981 | 0.953 |
| FaceAll | 1276.24 | 297.23 | 4.87 | 1.25 | 0.720 | **0.731** | 0.724 | 0.714 |
| FaceFour | 18.04 | 3.10 | 1.21 | 0.11 | 0.852 | 0.898 | **0.943** | 0.820 |
| FacesUCR | 107.35 | 24.74 | 2.70 | 0.33 | **0.871** | 0.868 | 0.841 | 0.847 |
| Fish | 1808.85 | 46.46 | 1.61 | 0.16 | 0.817 | **0.846** | 0.800 | 0.755 |
| Gun_Point | 7.69 | 1.55 | 0.60 | 0.04 | 0.900 | 0.913 | **0.953** | 0.931 |
| Haptics | 17273.44 | 2634.99 | 6.59 | 1.78 | 0.354 | **0.373** | 0.321 | 0.356 |
| InlineSkate | 34776.14 | 99.61 | 19.82 | 0.61 | **0.411** | 0.342 | 0.313 | 0.385 |
| ItalyPower. | 0.77 | 0.49 | 0.62 | 0.02 | **0.936** | 0.925 | 0.915 | 0.920 |
| Lighting2 | 843.42 | 90.84 | 12.23 | 1.89 | **0.852** | 0.836 | 0.836 | 0.795 |
| Lighting7 | 120.39 | 20.15 | 4.89 | 0.43 | 0.699 | **0.740** | 0.685 | 0.652 |
| MALLAT | 2295.97 | 6.25 | 1.99 | 0.08 | 0.909 | 0.938 | **0.941** | 0.926 |
| MedicalI. | 349.15 | 57.75 | 1.76 | 0.60 | 0.625 | 0.658 | 0.668 | **0.676** |
| MoteStrain | 0.91 | 0.62 | 0.21 | 0.05 | 0.734 | **0.815** | 0.777 | 0.783 |
| Non.ECG.1 | n/a | 35833.59 | 36.79 | 7.03 | n/a | **0.840** | 0.795 | 0.814 |
| Non.ECG.2 | n/a | 11086.13 | 58.18 | 4.99 | n/a | 0.852 | **0.858** | 0.855 |
| OliveOil | 75.17 | 0.90 | 1.12 | 0.05 | **0.900** | 0.800 | 0.700 | 0.790 |
| OSULeaf | 2379.27 | 16.64 | 3.18 | 0.15 | 0.570 | 0.541 | **0.583** | 0.566 |
| Sony.I | 1.28 | 0.62 | 0.76 | 0.02 | 0.829 | **0.902** | 0.792 | 0.850 |
| Sony.II | 0.46 | 0.47 | 0.86 | 0.03 | 0.727 | 0.774 | 0.742 | **0.780** |
| StarLight. | n/a | 4673.16 | 74.14 | 3.19 | n/a | **0.933** | 0.929 | **0.933** |
| SwedishLeaf | 830.60 | 301.63 | 1.24 | 0.36 | **0.869** | 0.856 | 0.856 | 0.849 |
| Symbols | 27.58 | 1.64 | 0.58 | 0.04 | 0.805 | 0.787 | 0.819 | **0.865** |
| synthetic. | 51.03 | 6.01 | 0.56 | 0.07 | 0.980 | **0.993** | 0.980 | 0.983 |
| Trace | 138.09 | 25.15 | 0.60 | 0.13 | 0.950 | **0.990** | 0.960 | 0.965 |
| Two_Patterns | 4572.63 | 1216.45 | 2.78 | 1.71 | 0.985 | 0.984 | **0.986** | 0.981 |
| TwoLead. | 0.54 | 0.88 | 0.41 | 0.02 | **0.932** | 0.774 | **0.932** | 0.867 |
| uWave.X | 27142.53 | 1565.73 | 19.46 | 4.94 | 0.757 | 0.745 | **0.762** | 0.761 |
| uWave.Y | 25276.28 | 1385.23 | 16.74 | 3.69 | 0.647 | 0.643 | **0.671** | **0.671** |
| uWave.Z | 24532.05 | 513.11 | 14.09 | 1.83 | 0.662 | 0.668 | **0.681** | 0.676 |
| wafer | 6352.87 | 1750.96 | 3.31 | 1.39 | 0.994 | 0.994 | **0.995** | 0.993 |
| WordsS. | 1220.31 | 44.25 | 3.13 | 0.31 | 0.627 | **0.639** | 0.607 | 0.625 |
| yoga | 5098.73 | 254.54 | 3.05 | 0.34 | **0.812** | 0.802 | 0.799 | 0.625 |
| **Total Wins** | | | | | **14.0** | **15.0** | **12.0** | **4.0** |
| **Average Ranks** | | | | | **2.2** | **2.3** | **2.5** | **2.7** |

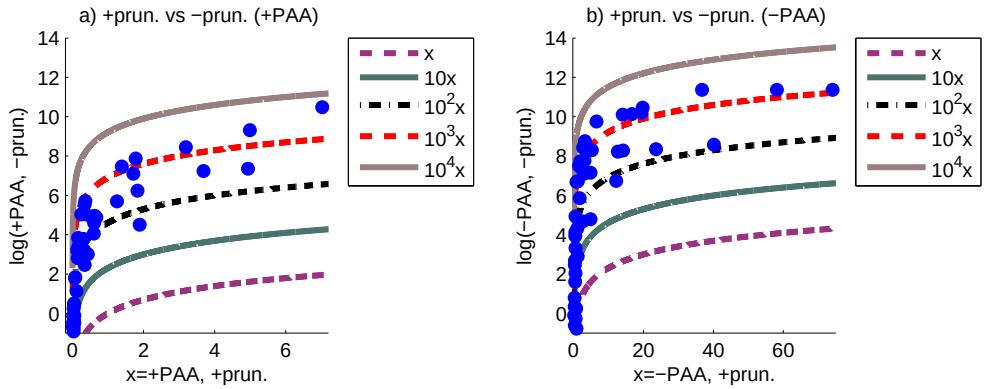



Figure 7.9: Runtime comparison (seconds) plots among variants of SD with and without pruning

In order to address those analytic questions we will decompose our method in a modular fashion. Our method, SD, conducts both a PAA approximation and a pruning by the parameters $r, p$ provided in Table 7.2. In order to isolate the effect of compression and pruning we are creating four variants of our method, namely all the permutations "With/Without PAA compression" and "With/Without Pruning" (w.r.t. to $p, r$ from Table 7.2). All the decomposed results of the SD variants are shown in Table 7.4. Note that "No pruning" means $p = 0$, while "no PAA" means $r = 1$. The variant with both pruning and PAA is the same as SD from Section 7.4.3, which already was shown to be superior to the state of the art.

Looking into the results of Table 7.4, it is important to observe that the variant with PAA compression alone is significantly faster than the variant without compression (columns 4 vs column 3). However, using pruning without compression is much faster than the exhaustive approach and also much faster than compression alone (column 5 vs. columns 3,4). When pruning and compression are combined (column 6), then the runtime reduction effect multiplies. More concretely, Figure 7.9 analyses the runtime reduction of SD variants: those that use pruning (X-axis) against variants without pruning (Y-axis) for both scenarios with PAA (plot a)) or without PAA (plot b)) compression. As can be clearly deduced, pruning alone has a significant effect on the runtime reduction by 3 to 4 orders of magnitude, compared to the cases where no pruning is employed. While PAA helps our method to be even faster, it is clear that the lion's share of the speedup arises from the proposed pruning mechanism.

There is still a concern on how does pruning affect the classification accuracy. The prediction accuracy results are demonstrated in Table 7.4 for all the datasets, with the winning variant emphasized in bold. The total wins and the ranks of the variants indicate that the best prediction performance is attributed to the exhaustive methods (no pruning, columns 7,8). Such a finding is natural because exhaustive approaches consider all the candidate variants and can extract more qualitative minimum distance features. Yet, are the results of the exhaustive variants better with a **statistical significance** margin? Table 7.5 illustrates the p-values of a Wilcoxon Signed Rank test of statistical significance, for a two-tailed hypothesis with a significance level of 5% ($\alpha = 0.05$).

Table 7.5: Wilcoxon Statistical Significance Test: p-values (Significance Level 5%, Two-Tailed Hypothesis)

| | ✘ PAA ✘ prun. | ✔ PAA ✘ prun. | ✘ PAA ✔ prun. | ✔ PAA ✔ prun. |
|---|---|---|---|---|
| ✘ PAA, ✘ prun. | - | 0.904 | **0.119** | 0.046 |
| ✔ PAA, ✘ prun. | 0.904 | - | 0.153 | **0.112** |
| ✘ PAA, ✔ prun. | **0.119** | 0.153 | - | 0.873 |
| ✔ PAA, ✔ prun. | 0.046 | **0.112** | 0.873 | - |

The p-values which compare variants that use pruning against variants that do not use pruning are shown in bold and correspond to $p = 0.119, p = 0.112$. Therefore, the prediction quality using pruning is not significantly (significance means $p < 0.05$) worse than the exhaustive approach. The final message of this section is: "Pruning of candidates provides 3 to 4 orders of runtime speedup without any statistically significant deterioration in terms of classification accuracy.".

### 7.4.7 Comparison To Other State-of-the-art Shapelet Discovery Methods

One would categorize the methods focusing on shapelet discovery into "speed-oriented" and "accuracy-oriented" approaches. The method proposed in this paper SD and the baselines LS, FS were focused on reducing the runtime of shapelet discovery. On the other hand, there are other methods which prioritize on achieving the highest classification accuracy. The most prominent methods on accurate shapelet discovery are "Shapelet Transformation" [62] (denoted as ST) and the shapelet learning method of Chapter 6 (denoted as LTS).

Table 7.6: Comparison of the proposed method SD against LTS [54] and ST [62]

| Dataset | Accuracies | | | Running Times (sec) | | |
|---|---|---|---|---|---|---|
| | LTS | ST | SD | LTS | ST | SD |
| StarLightCurves | 0.964 | - | 0.933 | 65657.07 | 1728000.00* | 3.19 |
| Non.Fat.ECG.1 | 0.865 | - | 0.814 | 1448862.51 | 1728000.00* | 7.03 |
| Non.Fat.ECG.2 | 0.897 | - | 0.855 | 1528267.41 | 1728000.00* | 4.99 |

In this section we aim at showing that while those methods are more accurate, their runtime is much slower than the proposed method SD. For this reason we selected the three largest datasets of the UCR collection as shown in Table 7.6 and ran SD, ST and LTS on those datasets. In order to be fair to the baselines, LTS and ST were also run on a subset of shapelet lengths $\{0.2Q, 0.4Q, 0.6Q\}$. For both LTS and ST we used the source code provided by the authors. Since those methods are known to be slow we violated the 24 hours time-out of Section 7.4.2 and instead gave the methods a very large time-out deadline of 20 days to complete the execution over the three datasets. The results of Table 7.6 indicate that LTS is more accurate than SD in all the dataset, however it took LTS from 18.2 hours to 17.7 days to compute. Furthermore, ST could not finish learning on any of the three datasets within 20 days (time-out denoted by $^*$). On the other hand, SD needs 3.19 to 7.03 seconds to compute the shapelets of those datasets, for a speed-up of up to 346293 times faster. On the other hand, the deterioration in accuracy varies only between 3.3% and 6.2% worse than LTS.

## 7.5 Extension to Multivariate Time Series

Multivariate time series has become increasingly popular in the data mining research community. Part of the popularity is attributed to the widespread of affordable motion sensor devices. In fact, multivariate (*synonym:* multidimensional) time series are a generalization of univariate series. In the multivariate case a single time-series instance is composed of different streams measured at the same time. An example of multivariate series are recordings of wearable body sensors, where signal measuring devices are positioned at different parts of the body [6, 7].

We can formalize a time-series dataset having $N$ instances and $V$ many dimensions as $T \in \mathbb{R}^{N \times V \times Q*}$, where each series has a different length $T_{i,:,:} \in \mathbb{R}^{Q_i}, \forall i \in \{1, \ldots, N\}$. Whilst the lengths of different instances vary, we assume that the different dimensions within one instance have the same length. In this section we

will demonstrate that it is trivial to extend the method proposed in this paper to multivariate time-series datasets. All is needed is to sample shapelet candidates from random dimensions and accept them based on their joint accuracy.

### 7.5.1 Addressing Challenges of Multivariate Series

**Different series lengths** is a common reality for multivariate series. The time-series research community has worked extensively on the UCR collection of datasets, where the time-series instances were preprocessed to have the same lengths. In reality this is rarely the case, however different lengths pose no concrete problem for shapelet-based methods. The minimum distance between a candidate and various series segments is independent on the number of segments. There is however a small problem, the case when a series is shorter than a shapelet. In order to overcome this concern we propose to slid the series over the shapelet, i.e. to measure the minimum distance of a series to all the segments of the shapelet candidate. Equation 7.7 formalizes the distance between the $v$-th dimension of the $i$-th series $T_{i,v,:}$ and a shapelet candidate $s$.

$$\mathcal{D}(s, T_{i,v,:}) := \begin{cases} \min\limits_{j=1,\dots,|T_{i,v,:}|-|s|+1} \left\| T_{i,v,j:j+|s|-1} - s \right\|^2 & |T_{i,v,:}| \geq |s| \\ \min\limits_{j=1,\dots,|s|-|T_{i,v,:}|+1} \left\| s_{j:j+|T_{i,v,:}|-1} - T \right\|^2 & |s| > |T_{i,v,:}| \end{cases} \tag{7.7}$$

**Features from different dimensions** are known to improve the classification accuracy, however related work takes diverse approaches in how series of different dimensions are incorporated. In terms of shapelets, an early approach extended the concept of univariate shapelets into multi-variate shapelets [47]. However, a label might not be associated with certain dimensions or there might be shifts of the starting time of a pattern across dimensions. As a result, a recent work [21] proposed to learn a shapelet-based classifier on each dimension and use a majority voting over the predictions of the per-dimension models.

In contrast, we propose a simple and novel technique to incorporate features from different dimensions. The principle relies on sampling random candidates from random dimensions. Roughly speaking we will harvest accepted and rejected candidates per each dimensions. Distance features from each dimensions will be **jointly integrated** into the same incremental nearest neighbor and filtered by classification accuracy. This mechanism will allow to fuse features of candidates from different dimensions into a joint feature set.

**Algorithm 6**: **DiscoverShapeletsMultivariate**: Scalable discovery of shapelets from multivariate series

**Data**: Multivariate time-series data $T \in \mathbb{R}^{N \times V \times Q*}$, Labels $Y \in \mathbb{N}^N$ Distance, Threshold Percentile $p \in [1, \dots, 100]$, Piecewise Aggregate Approximation ratio: $r \in \{1, \frac{1}{2}, \frac{1}{4}, \dots\}$, Shapelet lengths: $\Phi \in \mathbb{N}^R$

**Result**: Accepted shapelets list $\mathcal{A} \in \mathbb{R}^{V \times * \times *}$, Minimum Distances $D \in \mathbb{R}^{* \times *}$

$\epsilon_v \leftarrow \text{ComputeThreshold}(T_{:,v,:}, p, \Phi),\ v \in \{1, \dots, V\}$;
$\mathcal{A} \leftarrow \emptyset^V, \mathcal{R} \leftarrow \emptyset^V, D \leftarrow \emptyset, X \leftarrow \mathbf{0}_{N \times N}, \text{prevAccuracy} \leftarrow -\infty$;
**for** $1, \dots, NMLV$ **do**
  Draw random series: $i \sim \mathcal{U}\{1, \dots, N\}$;
  Draw random dimension: $v \sim \mathcal{U}\{1, \dots, V\}$;
  Draw random shapelet length: $\Phi_* \sim \mathcal{U}\{\Phi_1, \dots, \Phi_R\}$;
  Draw random segment start: $j \sim \mathcal{U}\{1, \dots, Q_i - \Phi_* + 1\}$;
  Selected random candidate: $s \leftarrow T_{i,v,j:j+\Phi_*-1}$;
  **if** $\neg \mathit{LookUp}(s, \mathcal{A}_v, \epsilon_v) \wedge \neg \mathit{LookUp}(s, \mathcal{R}_v, \epsilon_v)$ **then**
    $d^s \leftarrow \text{MinDist}(s, T_{:,v,:})$ ;
    **for** $i = 1, \dots, N;\ m = i + 1, \dots, N$ **do**
      $X_{i,m} \leftarrow X_{i,m} + (d_i^s - d_m^s)^2$;
    **end**
    $\alpha \leftarrow \text{Accuracy}(X, Y)$;
    **if** $\alpha > \mathit{prevAccuracy}$ **then**
      $\mathcal{A}_v \leftarrow \mathcal{A}_v \cup \{s\}$;
      $D \leftarrow D \cup \{d^s\}$;
      $\text{prevAccuracy} \leftarrow \alpha$;
    **else**
      $\mathcal{R}_v \leftarrow \mathcal{R}_v \cup \{s\}$;
      **for** $i = 1, \dots, N;\ m = i + 1, \dots, N$ **do**
        $X_{i,m} \leftarrow X_{i,m} - (d_i^s - d_m^s)^2$;
      **end**
    **end**
  **end**
**end**
**return** $\mathcal{A}, D$

### 7.5.2 Algorithm for Multivariate Shapelet Discovery

The concrete implementation of our multivariate method is described in Algorithm 6. Our method selects $NMLV$-many random candidates, from random series $i$, random dimension $v$, random length $\Phi_*$ and starting at random time index $j$ (lines 4-8). Each random candidate is looked up for similarity to previously considered (accepted or rejected) candidates within that dimension (line 9). If a shapelet candidate is not found to be similar to previous candidates, then its feature vector is computed (line 10) and the pair-wise distance matrix $X$ is updated (line 11-13). In case the candidate improves the overall accuracy (line 14), then it is accepted (lines 15-17) otherwise it is rejected (lines 19-23). In the end we are going to have lists of accepted shapelets for each dimension, such that the features of those accepted shapelets achieve the highest classification accuracy.

### 7.5.3 Experimental Results

In order to test our method we compared against the most recent and relevant method which elaborates shapelets for multivariate classification [21]. Furthermore, we are going to experiment on four multivariate datasets, whose statistics are displayed in Table 7.7. Three of them ('HMP', 'M-Health', 'REALDISP') are related to human action recognition using wearable sensors, while 'Characters' represents pen tip trajectories of handwritten characters. The instances of all datasets were randomly divided into train and test sets. It is interesting to note that those datasets are diverse in terms of number of dimensions ($V$ from 3 to 117), number of instances (63 to 1429), number of classes ('Cls.' from 12 to 33) and lengths (109 to 5643).

Table 7.7: Results of Scalable Shapelet Discovery on Multivariate Datasets, n/a denotes a 24h timeout

| **Dataset** | **Data Statistics** | | | | | **Accuracy** | | **Runtime (sec)** | |
|---|---|---|---|---|---|---|---|---|---|
| | **Train/Te.** | $V$ | **Cls.** | **Length** | **Size** | **LS** | **SD** | **LS** | **SD** |
| Characters [121] | 1429/1429 | 3 | 20 | 109-205 | 8.9mb | 0.722 | **0.980** | 779.76 | **3.05** |
| HMP [16] | 487/492 | 3 | 21 | 125-9318 | 11.5mb | 0.104 | **0.707** | 8857.01 | **2.87** |
| M-Health [6] | 63/63 | 23 | 12 | 513-3431 | 60.3mb | 0.762 | **0.813** | 33781.44 | **13.54** |
| REALDISP [7] | 749/749 | 117 | 33 | 318-5643 | 1.75gb | n/a | **0.723** | n/a | **289.40** |

Another aspect worth consideration is the size of the datasets. For instance REALDISP has a training set size of 889 mb and a total size of 1.75 gb, which is considerably large for labeled time-series data. As a comparison, the largest

univariate dataset from the UCR collection is 'StarLightCurves', which has a train set of 16 mb and a total size of 144 mb. In order to address this size challenge we opted for the single fastest parameter configuration for all datasets, with respect to the ranges of Section 7.4.2, concretely $r = \frac{1}{8},\ p = 35$. In order to aggregate the randomness effect we present the average figures of five different executions of our method. The runtimes of Table 7.7 include the classification time.

The results of Table 7.7 indicate great achievements in terms of runtime. **Our method can classify the mb-scale datasets in matters of seconds and the gb-scale dataset in matter of minutes.** Concretely, our method needs less than 5 minutes to classify the 1.75gb dataset. Compared to the baseline method [21], our method is 255.7 to 2494.93 times faster on the mb-scale datasets. Unfortunately, the baseline could not complete on the gb-scale dataset under the 24h timeout specified in Section 7.4.3. On the other hand, our method is more accurate than the baseline on all the datasets. We believe that such a superiority comes from the joint interactions of features from different dimensions, as opposed to learning isolated per-dimension classifiers.

The achievements of the proposed method are **game-changing** in the domain of shapelet discovery and time-series classification. For instance the "Fast Shapelet" method [95], which was priorly considered the fastest shapelet discovery technique, needs 280 seconds to find the shapelets of the 'FacesUCR' univariate dataset from the UCR collection, where 'FacesUCR' has a training size of 413 kb. In contrast, our method needs 289 seconds to learn shapelets out of the multivariate REALDISP dataset having a 889 mb training set and 1.75 gb total size. In the light of such achievements, we can safely conclude that the method proposed in this paper is the first successful attempt to transport shapelet discovery to the *Big Data* era. Having achieved the gb-landmark, the future work relies on transporting time-series classification to tb-scale datasets.

### 7.5.4 A Further Discussion on Scalability

Scalability has become one the leading buzzwords in Data Mining, due to the need for models that can cope with constantly increasing data sizes. Without diverting the focus from the realm of the problem, our understanding of scalability follows directly the stance taken by the closely related works in terms of shapelets discovery [87, 95]. **A scalability method should significantly reduce the time required to find time-series shapelets [87, 95], yet offering an "accuracy that is not perceptibly different [w.r.t. baselines]" [95].** In this chapter we showed that both the runtime of our method was several orders

of magnitude faster than the baselines. Yet, we also indicated that our scalable approach was not worse than the baselines with a statistically significant margin, in terms of classification accuracy.

## 7.6 Conclusion

Shapelets represent discriminative segments of a time-series dataset and the distances of time-series to shapelets are shown to be successful features for classification. The discovery of shapelets is currently conducted by trying out candidates from the segments (sub-sequences) of the time-series. Since the number of candidate segments is large, the time-series community has spent efforts on speeding up the discovery time of shapelets. This chapter proposed a novel method that prunes the candidates based on a distance threshold to previously considered other similar candidates. In a joint fashion, a novel supervised selection filters those shapelets that boost classification accuracy. We empirically showed that our method is 3-4 orders of magnitude faster than the fastest existing shapelet discovery methods, while providing a better prediction accuracy. In addition, we extended our method to multivariate datasets. Results indicate that our approach is able to classify Mb-scale datasets in a matter of seconds and Gb-datasets in a matter of minutes, therefore transporting shapelet discovery to the *Big Data* era.

# Chapter 8

# Scalable Classification of Repetitive Time-Series

## Contents

**Highlights:** This chapter focuses on classifying time series that contain repetitive patterns, instead of a single structure. Shapelet-based techniques are not suited for repetitive series, because the discriminative factor is not only the presence of a pattern, but also how often it occurs (frequency). In order to extract features that capture frequencies of repetitive patterns, we propose a bag-of-words technique that aggregates local segments. The segments are represented as strings using a novel polynomial discretization mechanism.

Traditionally, the focus of time-series classification has been on short time-series data composed of a few patterns exhibiting variabilities, while recently there have been attempts to focus on longer series composed of multiple local patterns repeating with an arbitrary irregularity. The primary contribution of this chapter relies on presenting a method which can detect local patterns in repetitive time-series via fitting local polynomial functions of a specified degree. We capture the repetitiveness degrees of time-series datasets via a new measure. Furthermore, our method approximates local polynomials in linear time and ensures an overall linear running time complexity. The coefficients of the polynomial functions are converted to symbolic words via equi-area discretizations of the coefficients' distributions. The symbolic polynomial words enable the detection of similar local patterns by assigning the same word to similar polynomials. Moreover, a histogram of the frequencies of the words is constructed from each time-series' bag of words. Each row of the histogram enables a new representation for the series and symbolizes the occurrence of local patterns and their frequencies. In an experimental comparison against state-of-the-art baselines on repetitive datasets, our method demonstrates significant improvements in terms of prediction accuracy.

## 8.1 Introduction

Most of the existing literature on time-series classification focuses on classifying short time series, that is series which mainly incorporate a single pattern. Nev-

ertheless, few studies [17, 77, 78] have been focusing towards the classification of time-series data which is repetitive and composed of many repeating local patterns. The degree of time-series repetitiveness in a dataset has been identified by common sense and visual inspections. We propose a novel repetitiveness measure that can objectively identify the repetitiveness score of a dataset. Using such an objective repetitiveness measure, we can successfully switch traditional methods for non-repetitive datasets and our method for repetitive data.

Furthermore, this chapter presents a method that classifies repetitive time series composed of local patterns occurring in an unordered fashion and by varying frequencies. In this chapter we extract the frequencies of repeating patterns as a new series representation. Figure 8.1 provides a toy clustering illustration, in order to demonstrate the efficiency of the similarity using frequency representation.

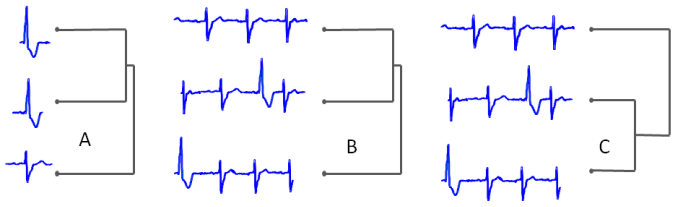



Figure 8.1: Three non-repetitive patterns in A and three repetitive series in B and C. A and B use the Euclidean measure and C our proposed method.

For instance, series in sub-plot A are non-repetitive and therefore similarity measures like Euclidean distance are accurate in matching the patterns. However, in the B and C sub-plots, the series are repetitive and composed of three normal heart beats (top) or two normal beats plus one Premature Ventricular Contractions (PVC) (middle, bottom). Euclidean distance fails to detect similar series, as shown in B, because (i) the position of the PVC pattern varies and (ii) the number of beats varies. On the other hand, we can define a new representation as the frequencies of local patterns and represent the series as frequencies $\{(3\ 0), (3\ 1), (3\ 1)\}$, where the first index denotes the frequency of normal beats and the second the frequency of PVC. An L2 distance over the frequency representation yields the correct similarity pairings in C.

As will be detailed in Section 8.4.2 we propose a fast technique to process sliding window content which has a linear run-time complexity. Our principle relies on detecting local polynomial patterns which are extracted with a sliding window approach, hence fitting one polynomial to each sliding window segment. Once the

polynomial coefficients of each sliding window content are computed, we convert those coefficients into symbolic forms (i.e. alphabet words). The motivation for calling the method Symbolic Polynomial arises from that procedure. Such a discretization of polynomial coefficients, in the form of words, allows the detection of similar patterns by converting similar coefficient values into the same word. In addition, the words computed from the time series allow the construction of a dictionary and a histogram of word frequencies, which produces an efficient representation of local patterns.

We utilize an equi-area discretization of the distributions of the polynomial coefficients to compute the symbolic words, as will be explained in Section 8.4.3. Threshold values separate the distribution into equal volumes and each volume is assigned one alphabet letter. Consequently, each polynomial coefficient is assigned to the area its value belongs to, and is replaced by the area's character. Ultimately, the word of a polynomial is the concatenation of the characters of each polynomial coefficient merged together. The words of each time series are then stored in a separate 'bag'. A dictionary is constructed from each word appearing at least once in the dataset and a histogram is initialized with each row representing a time series and each column one of the words in the dictionary. Finally, the respective frequencies of words are updated for each time series and the rows of the histogram are the new representation of the original time series. Such a representation offers a powerful mean to reflect which patterns (i.e. symbolic polynomial words) and how often they occur in a series (i.e. the frequency value in each histogram cell).

The technical novelty of our method, compared to state-of-art approaches [77, 78] which utilize constant functions to express local patterns of series, relies on offering an expressive technique to represent patterns as polynomials of arbitrary degrees. Furthermore, we present a fitting algorithm which can compute the polynomial coefficients for a sliding window segment in linear time, therefore our method offers superior expressiveness without compromising run-time complexity.

Our experimental evaluation is composed of two parts and detailed in Section 8.5.5. Initially we analyze a large pool of 47 time-series datasets for identifying data having highly repetitive characteristics. Then we conduct experiments on prediction accuracy and run-time against state of the art baselines in the repetitive datasets. Our method outperforms the state of the art in most repetitive datasets with a statistically significance margin in the majority of cases. We add experiments regarding the running time of the method and we show that our linear running time method is practically fast and feasible.

## 8.2 Chapter Notations

This chapter will adhere to the notational conventions of Table 8.1.

| Symbol | Explanation |
|---|---|
| $\Sigma = \{A, B, \ldots, Z\}$ | Alphabet |
| $\alpha = \lvert\Sigma\rvert$ | Alphabet size |
| $w \in \Sigma^*$ | Word |
| $d \in \mathbb{N}$ | Polynomial degree |
| $\beta \in \mathbb{R}^{d+1}$ | Polynomial coefficients |
| $X \in \mathbb{N}^L$ | Segment-relative time indices |
| $Z \in \mathbb{N}^{L \times (d+1)}$ | Vandermonde matrix |
| $S_* \in S$ | A series segment |
| $\Phi \in \mathbb{N}^{N \times (Q-L) \times (d+1)}$ | Polynomial coefficients of all segments |
| $W \in \mathbb{N}^{N \times (Q-L) \times (d+1)}$ | Words of all segments |
| $H \in \mathbb{N}^{N \times \lvert W\rvert}$ | Histogram |

Table 8.1: Notational Conventions of Chapter 8

In order to increase the readibility of the symbols we are describing them in more depth below.

### Alphabet

An alphabet is an ordered set of distinct symbols and is denoted by $\Sigma$. The number of symbols in an alphabet is called the size of the alphabet and is denoted by $\alpha = |\Sigma|$. For illutration purposes we will utilize the Latin characters for the English language composed of the set of character symbols $\Sigma = \{A, B, C, \ldots, Y, Z\}$.

### Word

A word $w \in \Sigma^*$ from an alphabet is defined as a sequence of symbols, therefore one sequence out of the set of possible sequences of arbitrary length $l$. Such a set is known as the Kleene star $\Sigma^* := \cup_{l=0}^{\infty} \Sigma^l$. For instance CACB is a word from the Latin alphabet having length four.

### Polynomial

A polynomial of degree $d$ with coefficients $\beta \in \mathbb{R}^{d+1}$, is defined as a sum of terms known as monomials. Each monomial is a product of a coefficient with a

particular power of the predictor value $X \in \mathbb{R}^L$, as shown in Equation 8.1. A polynomial can also be written as a linear dot product in case we introduce a new predictor variable $Z \in \mathbb{R}^{L\times(d+1)}$ which is composed of all the powers of the original predictor variable $X$.

$$\begin{aligned} \hat{S}_* = \sum_{j=0}^{d} \beta_j X^j &= Z\beta, \\ \text{where } Z &:= \left(X^0, X^1, X^2, \ldots, X^d\right) \end{aligned} \tag{8.1}$$

## 8.3 A Discussion on Repetitiveness

### 8.3.1 Repetitiveness versus Periodicity

Repetitive time series are characterized by two types of patterns: (i) those which repeat periodically, and (ii) those which occur frequently, but without a strict periodic order. Therefore, the realm of this chapter is different from existing time-series periodicity analysis [41, 113]. The traditional concept of time-series periodicity targets *regularly* repeating patterns and is crucial for tasks such as forecasting the future values of a series or discovering unexpected anomaly segments [96]. However, in case the series patterns occur in no strictly-regular order and at arbitrary frequencies, periodicity is no longer an adequate characterization for that type of data. Moreover, the standard periodicity detection approach using off-the-shelf Fourier Transformation is known to struggle encoding the non-periodic signals patterns [64]. Within the context of this study, we refer to series having patterns appearing at arbitrary order/frequencies as simply *repetitive* time series. Please note that periodic series are repetitive, but not every repetitive series is strictly periodic. We propose a method that can handle generic types of repetitive time-series by segmenting the series into overlapping sliding windows and extracting the patterns therein. In terms of applicability, our method can operate with arbitrary types of repeating time-series data, including (but not limited to) activity recognition, physiological datasets (gait, blood pressure, heartbeat, EEG), etc .... The datasets of Section 8.5.5 provide concrete application examples.

### 8.3.2 A Repetitiveness Measure

Does a dataset have repeating patterns? As a first step, we should be able to objectively identify their degrees of repetitiveness in a time-series. Whilst there exist related measures which detect repetitiveness [104], this section proposes a novel, simple and efficient measure for the repetitiveness of a time-series dataset. Whilst being an added value to the chapter, this measure is not an integral part of the proposed method (to be detailed in Section 8.4).

Repetitive datasets, by definition, contain repeating series sub-sequences. Therefore, if we cut the time series into non-overlapping sliding window segments, then each segment will be similar to many others. On the other hand, non-repetitive time series do not have repeating patterns, which means an arbitrary segment is likely different to the others. Subsequently, we define the repetitiveness measure as the average elastic distance among all pairs of non-overlapping segments of a series. We opted for non-overlapping segments in order to avoid comparing against neighboring segments that are very similar (change by one point) and produce close-to-zero distances. Equation 8.2 presents the repetitiveness measure for a time-series dataset $T$, given a sliding window size $L$. The measure computes the average DTW dissimilarity of each pair of the $\frac{Q}{L}$-many segments, while the per-series (indexed by $i$) scores are aggregated into the dataset repetitiveness measure. The correction factor $c = \frac{1}{N\frac{1}{2}\frac{Q}{L}\left(\frac{Q}{L}-1\right)L} = \frac{2L}{NQ(Q-L)}$, enables the metric to be invariant to the number of time-series, their length (number of segment pairs) and the size of the sliding window. The optimal repetitiveness measure, denoted $D(T)^*$ and defined in Equation 8.3, is the smallest value of $D$ over all possible sliding window sizes $L$. Further discussions on the relation of the measure to real-life datasets are elaborated in the experimental setup, Section 8.5.

$$D(T,L) = c\sum_{i=1}^{N}\sum_{j=0}^{\lfloor\frac{Q}{L}-1\rfloor}\sum_{k=j+1}^{\lfloor\frac{Q}{L}\rfloor} DTW(S_{i,jL+1}, S_{i,kL+1}) \tag{8.2}$$

$$D(T)^* = \min_{L\in\{1,\dots,Q\}} D(T,L) \tag{8.3}$$

Figure 8.2 shows time-series instances from three different datasets having various repetitiveness. As can be observed, the most repetitive series has the lowest D value. For the sake of illustration quality, only the first 1000 points of the RATBP instances are shown.

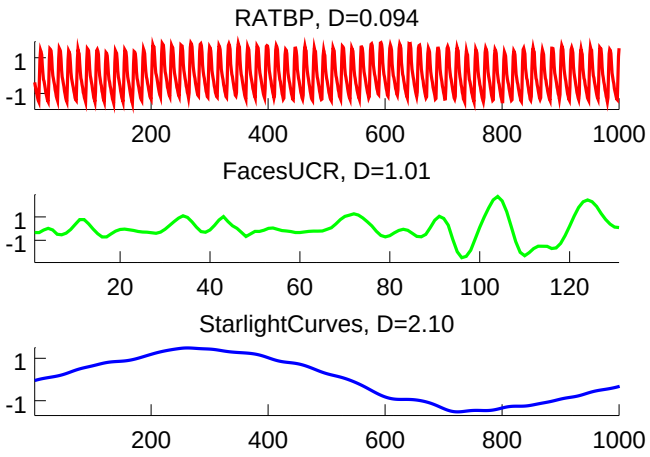



Figure 8.2: Time-series having different repetitiveness

## 8.4 Proposed Method: Frequencies of Local Polynomials

### 8.4.1 Principle

This study proposes to detect local patterns in a repetitive time series by computing local polynomials. Under the scope of bag-of-word models, the polynomials offer a superior tool in detecting local patterns compared to constant or linear models, because they can perceive information like the curvature of a sub-series. Furthermore, in case of reasonably sized sliding windows the polynomials can approximate the underlying series segment without over-fitting. In this chapter, we demonstrate that polynomial fitting for the sliding window scenario can be computed in a linear run-time. Once the local polynomials are computed, we propose a way to utilize the polynomial coefficients for computing the frequencies of the patterns. The polynomial coefficients are converted to alphabet words via an equi-area discretization approach. Such a conversion from real valued coefficients to short symbolic words allows for the translation of similar polynomials to the same word, therefore similar patterns can be detected. We call such words symbolic polynomials. The words of each time series are collected in a separate 'bag' of words, (implemented as a list), then a histogram is created by summing up the frequency of occurrence for each word. Each row of a histogram encapsulates the word frequencies of a particular time series (i.e. frequencies of local patterns). A histogram row is the new representation of the time series and is used as a feature vector for classification.

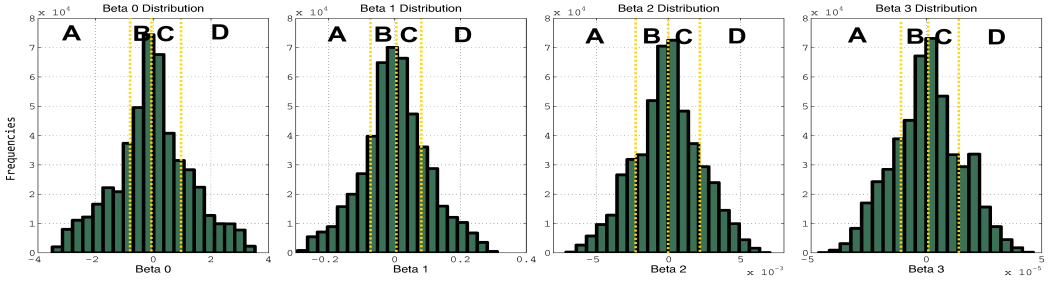



Figure 8.3: Equivolume Discretization of Polynomial Coefficients. The illustration depicts the histogram distributions of a third degree polynomial fit over the sliding windows of the RATBP dataset. Each plot shows the values of a polynomial coefficient versus the frequencies. The alphabet size is four corresponding to the set $\{A, B, C, D\}$. The quantile threshold points are shown by dashed yellow lines.

### 8.4.2 Local Polynomial Fitting

Our method operates by sliding a window throughout a time-series and computing the polynomial coefficients in each sliding window segment. The segment of time series inside the sliding window is normalized before being approximated to a mean 0 and deviation of 1. The incremental step for sliding a window is one, such that every segment is considered. Computing the coefficients of a polynomial regression is conducted by minimizing the least squares error between the polynomial estimates and the true values of the segment. The objective function is denoted by $\mathcal{L}$ and is shown in Equation 8.4. The task is to fit a polynomial that approximates the real values $S_{i,t}$ of the time-series window of length $L$.

$$\mathcal{L}(S_{i,t}, Z\beta) \quad = \quad ||S_{i,t} - Z\beta||^2, \quad , \forall i = 1, \ldots, N, \ \forall t = 1, \ldots, L \tag{8.4}$$

The predictors represent the time indexes $X = [0, 1, \ldots, L-1]$ and are converted to the linear regression form by introducing a Vandermonde matrix $Z \in \mathbb{R}^{L \times (d+1)}$ as shown below in Equation 8.5.

$$Z = \begin{pmatrix} 0^0 & 0^1 & \ldots & 0^d \\ 1^0 & 1^1 & \ldots & 1^d \\ \vdots & \vdots & \ldots & \vdots \\ (L-2)^0 & (L-2)^1 & \ldots & (L-2)^d \\ (L-1)^0 & (L-1)^1 & \ldots & (L-1)^d \end{pmatrix} \tag{8.5}$$

The solution of the least squares system is conducted by solving the first derivative with respect the polynomial coefficients $\beta$ as presented in Equation 8.6.

$$\frac{\mathcal{L}(S_{i,t}, \hat{S}_{i,t})}{\partial \beta} = 0 \quad \text{leads to} \quad \beta = \left(Z^T Z\right)^{-1} Z^T S_{i,t} \tag{8.6}$$

A typical solution of a polynomial fitting is provided in Figure 8.4. On the left plot we see an instantiation of a sliding window fitting. The sliding window of size 120 is shown in the left plot, while the fitting of the segment inside the sliding window segment is scaled up on the right plot. Please note that inside the sliding window the time is set relative to the sliding window frame from 0 to 119. The series of Figure 8.4 is a segment from the GAITPD dataset.

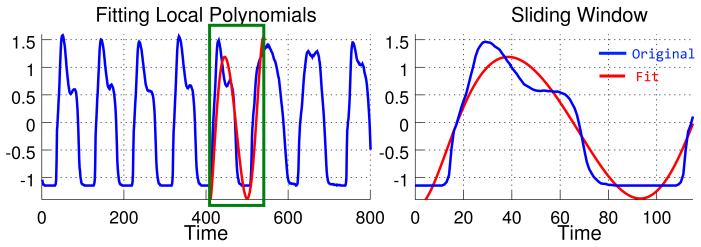



Figure 8.4: Fitting a local polynomial of degree 8 ($\beta = [8.4 \times 10^{-15}, -5.7 \times 10^{-12}, 1.6 \times 10^{-9}, -2.4 \times 10^{-7}, 2.15 \times 10^{-5}, -0.0011, 0.034, -0.36, -2.8]$) to a random sliding window region from GAITPD dataset; Scaled-up fitting illustration on the right plot (relative time axis)

Since the relative time inside each sliding window is between 0 and $L-1$, the predictors $Z$ are the same for all the sliding windows of all time series. Consequently, we can pre-compute the term $P = \left(Z^T Z\right)^{-1} Z^T$ in the beginning of the program and use the projection matrix $P$ to compute the polynomial coefficients $\beta$ of any local segment $S_{i,t}$ as $\beta = PS_{i,t}$. Algorithm 12 describes the steps needed to compute the polynomial coefficients of all sliding windows (starting at $t$) of every time series (indexed by $i$) in the dataset. For every time series we collect all the polynomial coefficients in a bag, denoted as $\Phi^{(i)}$. The outcome of the fitting process are the bags of all time series $\Phi$. Please note that the complexity of fitting a polynomial to a sliding window is linear (with respect to data size ($NQ$) and the overall algorithm has a complexity of $O(N \cdot Q \cdot L \cdot d)$.

In the presence of noise, high degree polynomials can over-fit the content of the data. Over-fitting leads to different polynomial coefficients for similar patterns [100]. Such a problem arises because the complexity of the approximative

model is higher than the bias (informative signal) of the data. There are at least two ways to tackle over-fitting problems, by (i) reducing the complexity of the approximative model, or (ii) strongly regularizing/penalizing the high rank coefficients of the polynomial. In this chapter, we opted for adjusting the complexity of the model in a statistically principled manner. The degree of the polynomial (a.k.a. model complexity) is selected based on the classification accuracy over a validation set (using only training instances). Furthermore, in order to avoid amplitude-based variations we normalized the sliding window content before fitting the polynomial coefficients.

**Algorithm 12** Polynomial Fitting of a Time-Series Dataset

**Require:** Dataset $T \in \mathbb{R}^{N \times Q}$, Sliding window size $L$, Polynomial degree $d$
**Ensure:** Polynomial coefficients of all segments $\Phi \in \mathbb{R}^{N \times (Q-L) \times (d+1)}$

```
P ← (Z^T Z)^{-1} Z^T
for i ∈ {1...N} do
    Φ^(i) ← ∅
    for t ∈ {1...Q − L + 1} do
        β ← PS_{i,t}
        Φ^(i) ← Φ^(i) ∪ {β}
    end for
end for
return (Φ^(i))_{i=1,...,N}
```

### 8.4.3 Converting Coefficients To Symbolic Words

The next step of our study is to convert the computed polynomial coefficients $\Phi$ from Algorithm 12 into words. The aim of the conversion is to transform each of the $d+1$ coefficients of every $\beta$ of $\Phi$ to one symbol. Therefore, the extracted words have lengths of $d+1$ symbols. For each of the $\beta$ values of the polynomial coefficients we construct the histogram distribution and divide it into regions of equal area as shown in Figure 8.3. In the illustration we have divided the histogram into as many regions as the alphabet size ($\alpha = 4$) we would like to utilize. Such a process is called an equi-area discretization. The thresholds between the regions are named quantile points and are illustrated with yellow lines. Dividing the histogram into $\alpha$ many regions is done by sorting the coefficient values and choosing the threshold values corresponding to indexes multiple of $\frac{1}{\alpha}$. For instance, dividing the histogram into 4 regions for an alphabet

of size 4 requires thresholds values corresponding to indexes at $\frac{1}{4}, \frac{2}{4}, \frac{3}{4}$ of the total number of values, which means that the each region has 25% of the values. Formally, let us define a sorted list of the j-th coefficient values regarding all window segments as $B^j \leftarrow \text{sort}\left(\left\{\beta_j \mid \beta \in \Phi^{(i)}, i = 1, \ldots, N\right\}\right)$ and let the size of this sorted list be $s^j \leftarrow |B^j|$. Then the $(\alpha - 1)$ many threshold values are defined as $\mu_k^j \leftarrow B^j_{\left\lfloor s^j \frac{k}{\alpha} \right\rfloor}, \forall\ k \in \{1, \ldots \alpha - 1\}$ and $\mu_\alpha^j \leftarrow \infty$.

**Algorithm 13** Convert Polynomial Coefficients to Words

**Require:** Polynomial Coefficients $\Phi$, Alphabet Size $\alpha$
**Ensure:** $W \in \mathbb{N}^{N \times (Q-L) \times (d+1)}$

{Compute the thresholds}
**for** $j \in \{0 \ldots d\}$ **do**
  $B^j \leftarrow \text{sort}\left(\left\{\beta_j \mid \beta \in \Phi^{(i)}, i = 1, \ldots, N\right\}\right)$
  $s^j \leftarrow |B^j|$
  $\mu_\alpha^j \leftarrow \infty$
  **for** $k \in \{1 \ldots \alpha - 1\}$ **do**
    $\mu_k^j \leftarrow B^j_{\left\lfloor s^j \frac{k}{\alpha} \right\rfloor}$
  **end for**
**end for**
{Convert the coefficients to words}
$\Sigma \leftarrow \{A, B, \ldots, Y, Z\}$
**for** $i \in \{1 \ldots N\}$ **do**
  $W^{(i)} \leftarrow \emptyset$
  **for** $\beta \in \Phi^{(i)}$ **do**
    $w \leftarrow \emptyset$
    **for** $j \in \{0 \ldots d\}$ **do**
      $k \leftarrow \text{argmax}_{k \in \{1, \ldots, \alpha\}}\ \beta_j < \mu_k^j$
      $w \leftarrow w \circ \Sigma_k$
    **end for**
    $W^{(i)} \leftarrow W^{(i)} \cup \{w\}$
  **end for**
**end for**
**return** $(W^{(i)})_{i=1,\ldots,N}$

Algorithm 13 describes the conversion of polynomial coefficients to symbolic form, i.e. words. The first phase computes the threshold values $\mu_k^j$ to discretize the distribution of each coefficient in an equi-area fashion. The second phase processes all the coefficients $\beta$ of time-series sliding windows and converts each

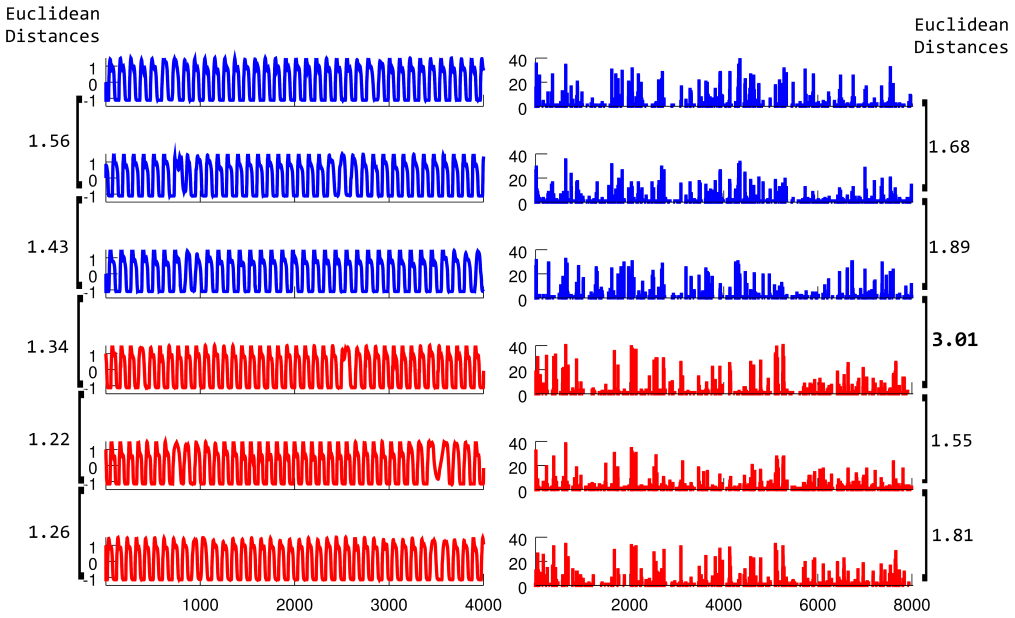



Figure 8.5: GAITPD Dataset: Time series describing the gait in a few, randomly selected, control patients (blue) and Parkinson's disease patients (red). The original time series (x-axis is time and y-axis are normalized values) are displayed on the left column, while the respective histogram of each time series is shown on the right. The x-axis of the right plot represents 8026 words of the dictionary while the y-axis is the frequency of each word. The distances on the right show that histograms among a class have lower Euclidean distances, while the distance between the two histograms belonging to different classes is much higher, at a value of 3.01. See Table 8.2 for all pairwise distances.

individual coefficient to a character $c$, depending on the position of the $\beta$ values with respect to the threshold values. The concatenation operator is denoted by the symbol $\circ$. The characters are concatenated into words $w$ and stored in bags of words $W$. While the size of the sliding window $n$ is a significant value, still it is a constant with respect to the data size $N$ and $Q$. The complexity of this algorithm is also linear in terms of $N$ and $Q$. In case a linear search is used for finding the symbol index $k$, then the complexity is $O(N \cdot Q \cdot \alpha)$, while a binary search reduces the complexity to $O(N \cdot Q \cdot \log(\alpha))$. Please note that Algorithm 1 has a higher complexity and determines the complexity of the proposed method.

The discretization of the polynomial coefficients produces hyperbox clusters, as shown in Figure 8.6. Polynomials of degree $d = 2$ are fit from segments of the ECG2 dataset with length $L = 100$ and plotted in the figure. The lines in

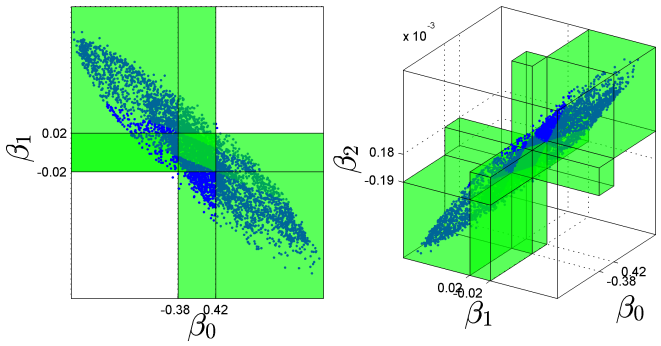



Figure 8.6: Discretizing polynomial coefficients creates hyperbox clusters. Illustrated coefficients are fit from the ECG2 dataset segments with parameters $n = 100, d = 2, \alpha = 3$.

black indicate the quartile values that discretize each coefficient into three areas (alphabet size $\alpha = 3$). Each word correspond to a hyperbox region, shown in green, between the respective quartile lines.

For instance the box in the center of the plot corresponds to the symbolic word 'BBB'. Therefore, our method can be interpreted as a dimensionality reduction technique for the time-series segments. Each segment is represented simply by the word (cluster id) of the hyperbox cluster where the segment's polynomial coefficients are located.

### 8.4.4 Populating the Histogram

Once we have converted our polynomial coefficients and converted them to words, the next step is to convert the words into a histogram of word frequencies, as depicted in Figure 8.7. The first step is to build a dictionary, which is a set of each word that appears in any time series at least once. Then we create a histogram with as many rows as time-series and as many columns as there are words in the dictionary. The initial values of the histogram cells are 0. Each cell indicate a positive integer which semantically represent how many times does a word (column index) appear in a time series (row index). The algorithm iterates over all the words of a series and increases the frequency of that word in the histogram. A formalization of the histogram of each series, denoted $H$, is given in Equation 8.7.

$$H_w^{(i)} \leftarrow \mid \left\{ j \mid W_j^{(i)} = w \right\} \mid, \quad i = 1, \ldots, N \tag{8.7}$$

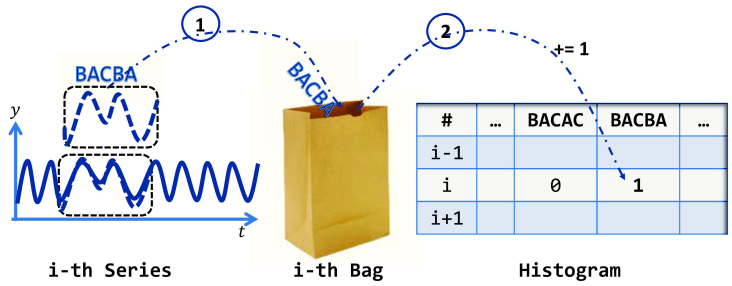



Figure 8.7: Polynomial Words Collection and Histogram Population. In the first step all the words of a series are stored in one 'bag' per each time series. Then a histogram is initialized with a column of zeros for each word that occurs in any bag at least once. During the second step the word frequencies in the histogram are incremented upon each word found in a bag.

Once the histogram is populated, then each row of the histogram denotes a vector containing the frequencies of the dictionary words for that respective time series. A row represents what local patterns (i.e. words) exist in a series and how often they appear. For instance Figure 8.5 presents instances from the GAITPD dataset belonging to two types of patients in a binary classification task, healthy patients (blue) and Parkinson's Disease patients (red). In the left plot we show the original time series while on the right plot the histogram rows containing the polynomial words versus their frequencies. The parameters leading to the histogram for the GAITPD dataset are $L = 100, \alpha = 4, d = 7$. As can be inspected the original time-series offer little direct opportunity to distinguish one class from the other and the series look alike. Moreover the Euclidean distance of adjacent series in the figure show that the Euclidean classifier would mistakenly classify the third instance of the blue class. In contrast the histograms are much more informative and it is possible to observe frequencies of local patterns which allow the discrimination of one class from the other. A complete distance matrix between blue $B$ and red $R$ instances is shown in Table 8.2. As can be seen our histogram representations result in perfect accuracy in terms of nearest neighbor classification (**bold**), while the original series result in 2 errors. As we show later, the high classification accuracy of nearest neighbor using the histogram representation, compared to the original series over multiple datasets, further demonstrates the usefulness of our representation.

Table 8.2: Distances: Time-series (left), Histogram (right)

| | $B_1$ | $B_2$ | $B_3$ | $R_1$ | $R_2$ | $R_3$ |
|---|---|---|---|---|---|---|
| $B_1$ | - | 1.6 | **1.28** | 1.4 | 1.29 | 1.4 |
| $B_2$ | 1.6 | - | 1.4 | 1.5 | 1.4 | **1.31** |
| $B_3$ | 1.28 | 1.4 | - | 1.4 | **1.27** | 1.4 |
| $R_1$ | 1.4 | 1.5 | 1.4 | - | **1.22** | 1.4 |
| $R_2$ | 1.29 | 1.4 | 1.27 | **1.22** | - | 1.26 |
| $R_3$ | 1.4 | 1.31 | 1.4 | 1.4 | **1.26** | - |

| | $B_1$ | $B_2$ | $B_3$ | $R_1$ | $R_2$ | $R_3$ |
|---|---|---|---|---|---|---|
| $B_1$ | - | **1.7** | 2.3 | 2.9 | 2.6 | 2.5 |
| $B_2$ | **1.7** | - | 1.9 | 2.6 | 2.3 | 2.3 |
| $B_3$ | 2.3 | **1.9** | - | 3.0 | 2.6 | 2.8 |
| $R_1$ | 2.9 | 2.6 | 3.0 | - | 1.6 | **1.4** |
| $R_2$ | 2.6 | 2.3 | 2.6 | **1.6** | - | 1.8 |
| $R_3$ | 2.5 | 2.3 | 2.8 | **1.4** | 1.8 | - |

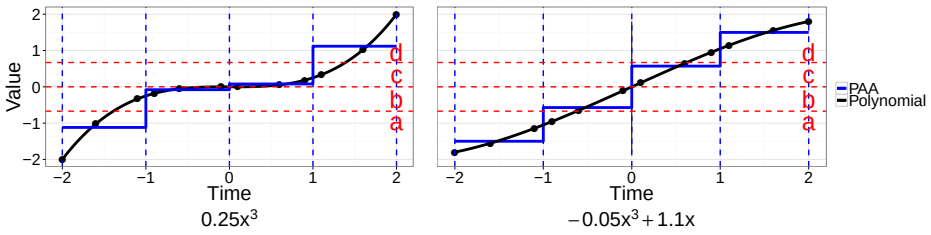



Figure 8.8: Piecewise Aggregate Approximation (PAA) fails to detect the curvature of the patterns and results in the same SAX word 'ABCD' for both series

### 8.4.5 On the Importance of Linear Running Times

Repetitive time series are usually long, as shown in Section 8.5, therefore linear running times of algorithms are a must for computational feasibility. Our method ensures linearity (in terms of $NQ$) in terms of running time by having an algorithmic complexity of $O(NQLd)$. Practically, only two passes over the data are needed, one for computing the coefficients and one for the creation of the histogram. In order to give an illustration on the tyranny of non-linear running times, our method computes in 18.7 mins on the GAITPD dataset, while the full window DTW with nearest neighbor (quadratic complexity) requires 5166.6 mins or 3.9 days.

### 8.4.6 Symbolic Polynomial Words Versus SAX Words

The closest method comparable in nature to ours is the approach which builds histograms from SAX words [77, 78]. However the SAX words are built from locally constant approximations which are generally less expressive than the polynomials of our approach. Figure 8.8 demonstrates the deficiencies of the locally constant approximation in detecting the curvature of a sliding window sub-series. In the experiment of Figure 8.8, we used an alphabet of size four and utilized the classical quantile thresholds for SAX, being values $\{-0.67, 0, 0.67\}$. We have

fit both a constant model and our polynomial model to the series data. Assume we want to have a four character SAX word for each of the sliding windows segment. As can be easily seen the SAX words for both segments are 'ABCD'. On the other hand, referring to the coefficient values of Figure 8.3, we can see that the symbolic polynomial word belonging to polynomial $y(x) = 0.25x^3$ is 'CCBC', while the polynomial word belonging to $y(x) = -0.05x^3 + 1.1x$ is 'BCDC'. As we can see our method can accurately distinguish between those patterns, while the SAX method averages the content and loses information about their curvatures. We would like to point out that our method is more expressive even though it has exactly the same complexity. In this case both methods use four characters. In terms of run-time, SAX needs only one pass through the data, so from the algorithmic complexity point of view both methods have the same algorithmic complexity of $O(N \cdot Q \cdot L)$, i.e. number of series by their length. The complexity of our method has a multiplicative constant, which is the degree of the polynomial as shown in Algorithm 13. Finally, as will be shown in Section 8.5.5, the classification results of our method are significantly better than SAX histograms.

### 8.4.7 Classifier Selection

Our method produces a new time-series representation and is not dependent on any specific classifier. However, the classifier that we are going to use for experimentation is the nearest neighbor method, which is both a strong classifier in time-series classification [39] and is also used by the state-of-the-art methods [77]. After converting the original time series into pattern frequency representations, each row will be treated as a feature vector. The nearest neighbor will utilize the Euclidean distance to compute the difference between histogram rows.

## 8.5 Experimental Setup

### 8.5.1 Selecting Repetitive Datasets

The UCR[1] collection of time-series datasets is a popular data source for the time-series community. However the UCR collection is known to have "atomic" (single non-repetitive pattern) [29] time series datasets and not repetitive ones. In pursue of additional datasets exhibiting repetitiveness, we searched the Physionet [48] repository of medical signals and found seven labeled datasets (ECG2, RATBP,

[1] `www.cs.ucr.edu/~eamonn/time_series_data`

NESFDB, GAITPD, BIDMC, UMW, MVT). Another dataset (PAMAP) on time-series is utilized in a recent study [29]. Table 8.3 represents the repetitiveness scores of all the datasets (the smaller the more repetitive). The scores are computed using the repetitiveness measure of Section 8.3 and shown under the D column. We tried different sliding window sizes ($L \in \{5\%, 10\%, 15\%, 20\%\}$ of $N$) and selected the size which resulted in the best repetitiveness, i.e. smaller D. The time series of each dataset were Z-normalized before computing the repetitiveness score. None of the UCR collection datasets have high repetitiveness according to our objective measure. Such a finding is in accordance to previous beliefs, given that the UCR collection contains "atomic" patterns [29], and further indications that those series are "short" [77]. In addition, we ran an alternative periodicity score [104] which computes the strength of periodic bases. Such a baseline, projects the signal into periodic subspaces, by finding its own set of basis functions, instead of the fixed Fourier terms. The alternate measure is denoted by P and high values denote repetitive time series. The scores of the repetitiveness match very strongly between our measure and the alternative one with the exception of PAMAP, which we included as well to our selection.

Table 8.3: Repetitiveness Scores (D) of 51 Time-Series Datasets

| D | P | Dataset | Len | D | P | Dataset | Len | D | P | Dataset | Len | D | P | Dataset | Len |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2.12 | 0.02 | Two.ECG | 82 | 1.95 | 0.10 | ECGF. | 136 | 1.56 | 0.08 | Cricket_X | 300 | 1.08 | 0.04 | OSULeaf | 427 |
| 2.09 | 0.04 | Trace | 275 | 1.92 | 0.08 | Lighting7 | 319 | 1.56 | 0.08 | Cricket_Z | 300 | 1.02 | 0.08 | FaceAll | 131 |
| 2.09 | 0.02 | StarL. | 1024 | 1.82 | 0.14 | Lighting2 | 637 | 1.53 | 0.05 | Cricket_Y | 300 | 0.98 | 0.17 | FaceFour | 350 |
| 2.08 | 0.02 | Gun. | 150 | 1.79 | 0.03 | yoga | 426 | 1.53 | 0.06 | Haptics | 1092 | 0.93 | **0.24** | **PAMAP** | 2000 |
| 2.08 | 0.03 | Medical. | 99 | 1.77 | 0.04 | Symbols | 398 | 1.52 | 0.09 | CBF | 128 | 0.92 | 0.07 | MALLAT | 1024 |
| 2.07 | 0.03 | uWaveY | 315 | 1.77 | 0.03 | Adiac | 176 | 1.41 | 0.09 | OliveOil | 570 | 0.91 | 0.07 | FacesUCR | 131 |
| 2.06 | 0.01 | Inline. | 1882 | 1.72 | 0.15 | CinC. | 1639 | 1.41 | 0.05 | NESFDB | 1800 | **0.77** | 0.17 | **MVT** | **1021** |
| 2.06 | 0.02 | uWaveZ | 315 | 1.71 | 0.04 | Words. | 270 | 1.38 | 0.05 | Beef | 470 | **0.56** | **0.21** | **ECG2** | **2048** |
| 2.03 | 0.05 | wafer | 152 | 1.71 | 0.04 | Fish | 463 | 1.37 | 0.05 | synthetic | 60 | **0.54** | **0.23** | **UMW** | **1200** |
| 2.01 | 0.02 | uWaveX | 315 | 1.66 | 0.04 | Diatom. | 345 | 1.32 | 0.08 | Two_P. | 128 | **0.19** | **0.33** | **BIDMC** | **15000** |
| 2.00 | 0.01 | ItalyP. | 24 | 1.64 | 0.03 | Sony | 70 | 1.26 | 0.04 | Coffee | 286 | **0.15** | **0.37** | **ratbp** | **2000** |
| 1.99 | 0.03 | ECG200 | 96 | 1.60 | 0.04 | 50words | 270 | 1.24 | 0.10 | Chlorine. | 166 | **0.11** | **0.48** | **gaitpd** | **4000** |
| 1.96 | 0.08 | Mote. | 84 | 1.58 | 0.05 | Swedish. | 128 | 1.22 | 0.05 | SonyII | 65 | | | | |

Therefore, we are finally left with seven highly repetitive datasets for analysis, namely ECG2, GAITPD, RATBP, BIDMC, UMW, MVT and PAMAP. Those datasets will be the testbed of our further experiments. Please note that repetitiveness is not directly related to the length of the time series, but rather to the amount of repetitive patterns inside them.

### 8.5.2 Descriptions of Highly Repetitive Datasets

All the experiments are based on the highly repetitive datasets retrieved from our objective repetitiveness ranking. Three of them are retrieved from Physionet, a repository of complex physiological signals primarily from the health care domain [48]. ECG2, BIDMC and MVT represents ECG recordings. GAITPD and UMV relate to the analysis of gait cycles. RATBP represents the blood pressure recordings of mice, while PAMAP contains human activities.

Table 8.4: **Statistics of Highly Repetitive Datasets**

| Dataset | Instances | Length | Classes |
|---|---|---|---|
| ECG2 | 250 | 2048 | 5 |
| GAITPD | 1552 | 4000 | 2 |
| RATBP | 180 | 2000 | 2 |
| BIDMC | 300 | 15000 | 15 |
| UMW | 60 | 1200 | 6 |
| MVT | 268 | 1021 | 3 |
| PAMAP | 802 | 2000 | 7 |

The statistics of each dataset in terms of the number of instances, the length of each time series and the number of classes are summarized in Table 8.4. Please note that all the instances within one dataset have the same length, for a couple of reasons: (i) respecting the source formats (ECG2, BIDMC, MVT, all UCR collection), (ii) practicality in pre-processing (RATBP, GAITPD, NESFDB, UMW), and (iii) various traditional baselines like Euclidean-based nearest neighbor cannot trivially operate on variable series lengths. However, our histogram representation is very easily extensible to time series of different sizes by normalizing/dividing the pattern frequencies of a particular series by its length.

### 8.5.3 Baselines

For notational sake, let us name our method as SymPol, meaning **Sym**bolic **Pol**ynomials, and refer to our method with the abbreviation form in the remaining sections. In order to evaluate the performance of SymPol, we compare against the following three baselines.

1. **BSAX** (a.k.a. BoW, Bag of Words, in Chapter 5) refers to the method of constructing bags of SAX words from time series through a sliding window approach. The words occurring in the bags are used to populate a histogram of frequencies [77]. A nearest neighbor method is applied to classify

the histogram instances by treating the histogram rows as the new time-series representation. Comparing against this classifier will give a chance to understand the benefit of polynomial approximations compared to constant models and will provide evidences on the state-of-the-art quality of our results.

2. **NN** is the classical nearest neighbor classifier with the Euclidean $L_2$ distance. It operates over the whole time series, without segmenting the series for local patterns. The comparison against the plain nearest neighbor will show whether the detection of local patterns has more advantage than comparing the whole long series.

3. **DTW-NN** differs from the Euclidean nearest neighbor classifier in the distance metric for the comparison of two time series and performs well in time-series classification [39]. Dynamic Time Warping (DTW) operates by creating a matrix with all the possible warping paths, (i.e. alignment of pairs of indexes from two series), and selects the warping alignment with the smallest overall possible distance. DTW compares a full series without segmentations similarly to the Euclidean version of the nearest neighbor. Such comparison will both identify the benefits of the segmentation and also the benefits of local polynomials against global warping alignments.

Table 8.5: **Hyperparameter Search Results**

| **Dataset** | **SymPol** | | | **BSAX** | | |
|---|---|---|---|---|---|---|
| | $L$ | $\alpha$ | $d$ | $L$ | $\|w\|$ | $\alpha$ |
| ECG2 | 100 | 4 | 3,4 | 100 | 4,6 | 4,6 |
| GAITPD | 100 | 4,6 | 7,8 | 100 | 6,7 | 6,8 |
| RATBP | 100 | 4,6 | 4,5,6,7 | 100 | 4,6,7 | 4,6,8 |
| BIDMC | 100 | 4 | 2 | 100 | 4 | 3 |
| UMW | 100,200 | 4,6,8 | 2,5 | 100,200,300 | 3,4,5,6 | 4,6,8 |
| MVT | 50,100,300 | 4,6,8 | 2,3,4 | 200,300 | 3,4,6 | 6,8 |
| PAMAP | 50,100,300 | 4,6,8 | 3,4,6 | 50,100,200 | 6,8 | 4,5,6 |

### 8.5.4 Reproducibility

Two different types of experiments were conducted in our study. The first empirical evidence focuses on the accuracy of our method with respect to the classification of time series. The second experiment analyzes the computational run

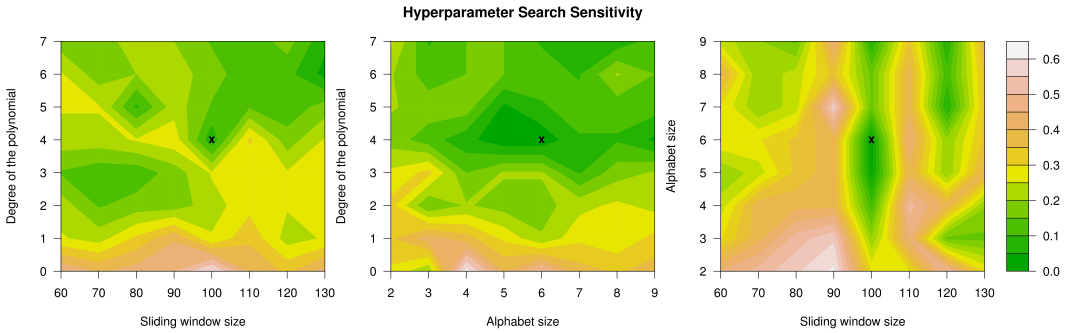



Figure 8.9: Hyperparameter Search Sensitivity. Parameter search for one fold of the RATBP dataset resulting in the optimal values $L = 100, \alpha = 6, d = 4$. In the two dimensional illustration the third parameter (z-axis is invisible) is fixed to the optimal value.

time of the methods. All the experiments were computed in a **five fold cross-validation** experimental setup. The time-series instances of each dataset were divided into 5 folds. In a circular fashion (repeated five times) each different fold was once selected as the testing set, while the remaining four were used for training. Among the four folds used for training, one of them was selected as a validation set and the remaining three left for training. As a summary, all the combination of parameters were evaluated on the validation set and learned on the three training set, while the parameter values giving the smallest errors on the validation were selected. Those parameter values were finally evaluated over the testing set to report the final error rate.

A grid search mechanism was selected for searching the hyperparameter values. Our method SymPol requires the tuning of three parameters, the size of the sliding window $n$, the size of the alphabet $\alpha$ and the degree of the polynomials $d$. The size of the sliding window was selected among the range of $L \in \{50, 100, 200, 300, 400\}$, while the size of the alphabet was picked from $\alpha \in \{4, 6, 8\}$. Lastly the degree of the polynomial was picked to be one of $d \in \{1, 2, 3, 4, 5, 6, 7, 8\}$.

Similarly, the baseline named BSAX also requires the fitting of three hyperparameters. The length of a SAX word, denoted $|w|$, was selected from the range of $\{2, 3, 4, 5, 6, 7, 8, 9\}$, while the size of the alphabet was selected among the values $\{4, 6, 8\}$. The size of the sliding window is selected from a range of $\{50, 100, 200, 300, 400\}$. The datasets are normalized before usage, which is recommended in the realm of time series [94].

Table 8.6: **Error Rate Results**

| Dataset | SymPol | | BSAX | | NN | | DTW-NN | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ (mean) | $\sigma$ (st.dev.) | $\mu$ (mean) | $\sigma$ (st.dev.) | $\mu$ (mean) | $\sigma$ (st.dev.) | $\mu$ (mean) | $\sigma$ (st.dev.) |
| ECG2 | **0.0000** | 0.0000 | 0.0080 | 0.0098 | 0.5480 | 0.0240 | 0.2120 | 0.0160 |
| GAITPD | **0.0238** | 0.0083 | 0.0548 | 0.0120 | 0.3924 | 0.0211 | 0.2468 | 0.0206 |
| RATBP | **0.1333** | 0.0272 | 0.1889 | 0.0111 | 0.4389 | 0.0272 | 0.3333 | 0.0994 |
| BIDMC | **0.0000** | 0.0000 | **0.0000** | 0.0000 | 0.6467 | 0.0356 | 0.2433 | 0.0226 |
| UMW | **0.3167** | 0.0372 | 0.4500 | 0.0745 | 0.7167 | 0.1247 | 0.6333 | 0.0667 |
| MVT | 0.4628 | 0.0535 | 0.4850 | 0.0169 | 0.2724 | 0.0303 | **0.1982** | **0.0513** |
| PAMAP | **0.5325** | 0.0260 | 0.5386 | 0.0269 | 0.7432 | 0.0254 | 0.5948 | 0.0148 |

### 8.5.5 Results

The classification accuracy results of our experiments are presented in Table 8.6. For our method SymPol and all the baselines we show the mean and the standard deviation of the five fold cross-validation experiments as described in the setup section. The smallest error rate is highlighted in bold. The hyper-parameter values, upon which the results are based on, found in our experiments are shown in Table 8.5, with ranges of multiple values due to different parameter searches per each different validation set.

Table 8.7: **Statistical Significance - T-Test (p values)**

| Dataset | SymPol - BSAX | SymPol - NN | SymPol - DTW-NN |
|---|---|---|---|
| ECG2 | 0.1411 | $5.8403 \cdot 10^{-11}$ | $4.4212 \cdot 10^{-9}$ |
| GAITPD | 0.0054 | $2.4784 \cdot 10^{-7}$ | $5.8147 \cdot 10^{-6}$ |
| RATBP | 0.0028 | $8.7033 \cdot 10^{-10}$ | $3.9227 \cdot 10^{-8}$ |
| BIDMC | - | $3.6036 \cdot 10^{-10}$ | $2.2838 \cdot 10^{-8}$ |
| UMW | 0.0072 | $2.6034 \cdot 10^{-4}$ | $2.8225 \cdot 10^{-5}$ |
| MVT | 0.4015 | $1.4826 \cdot 10^{-4}$ | $6.6441 \cdot 10^{-5}$ |
| PAMAP | 0.7205 | $1.8309 \cdot 10^{-6}$ | 0.0019 |

As can be clearly seen our method demonstrates a superiority in the majority of the datasets. SymPol achieves better results in five datasets, namely ECG2, GAITPD, RATBP, UMW, PAMAP while co-sharing a win in BIDMC and losing to DTW once in MVT. Our method performs perfectly in the ECG2 dataset by having 100% classification accuracy. In addition, SymPol reduces the error on the GAITPD dataset by 57% with respect the closest baseline, while on the RATBP dataset the error is reduced by 29%. BIDMC is a trivial dataset where both BSAX and SymPol have 100% accuracies. In addition we ran a statistical significance

test in order to validate the results. Table 8.7 presents the $p$-values of a two tails T-Test, where results are statistically significant with a confidence of 95% for $p < 0.05$. Each cell measures the $p$-value of SymPol against a baseline. The p-value for BIDMC, comparing SymPol vs BSAX, is not defined because of division by zero, since both methods have zero means and zero standard deviations. Our method is superior with a statistically significant margin in 15 out of 21 cases, concretely 3/7 against BSAX, 6/7 against NN, 6/7 against DTW-NN. Please note that ECG2 and BIDMC are trivial datasets, where the baseline (BSAX) has a quasi-perfect score. In a trivial dataset, a method cannot outperform the baseline and the best it can do is to have 0 % error as well, which is what SymPol achieved. Finally we would like to explain the only dataset (MVT) where our method is under-performing to DTW-NN. As Figure 8.10 illustrates MVT series are locally repetitive, but still have a global structure where DTW is hard to beat.

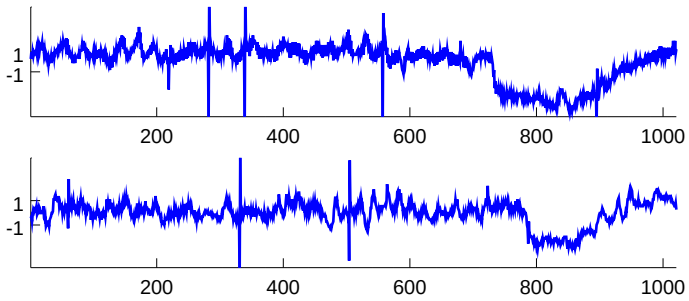



Figure 8.10: Series of MR type (same class) in the MVT dataset

The second type of results represent the running times of the algorithms and is shown in Table 8.8. As can be clearly seen the Euclidean distance on the original dataset is generally the fastest method, which is a natural behavior because no processing is done over series to extract histograms. The BSAX method is the next in terms of speed due to the computational advantage of the constant model. SymPol is positively positioned in terms of run time. As already analyzed before, the algorithmic complexity is comparable to the BSAX except for an additional constant, which is the polynomial degree. In datasets like ECG2 and GAITPD the execution times are bigger by only a small constant factor of two. The runtime constant in the RATBP dataset is higher because the hyperparameter search resulted in a high polynomial degree equal to 7. As a summary, we can clearly see that the method is practically very fast in terms of run time and is close even to techniques that use a constant model to fit local patterns. SymPol approximates polynomials of arbitrary degrees, instead of simple averages as BSAX, yet it does it in a competitive linear running time. The runtime results for both SymPol

Table 8.8: **Run Time Results (seconds)**

| Dataset | SymPol | | BSAX | | NN | | DTW-NN | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ (mean) | $\sigma$ (st.dev.) | $\mu$ (mean) | $\sigma$ (st.dev.) | $\mu$ (mean) | $\sigma$ (st.dev.) | $\mu$ (mean) | $\sigma$ (st.dev.) |
| ECG2 | 4.1 | 0.1 | 3.3 | 3.8 | 2.0 | 0.2 | 1651.5 | 52.9 |
| GAITPD | 1124.0 | 807.4 | 535.0 | 198.2 | 91.7 | 0.9 | 337K | 20K |
| RATBP | 27.2 | 36.7 | 1.8 | 0.8 | 0.7 | 0.0 | 1124.1 | 17.2 |
| BIDMC | 54.4 | 2.5 | 17.8 | 0.8 | 9.5 | 0.4 | 218K | 1.4K |
| UMW | 1.4 | 0.3 | 0.9 | 0.9 | 0.07 | 0.00 | 53.2 | 1.7 |
| MVT | 3.2 | 1.1 | 5.0 | 3.8 | 0.06 | 0.00 | 722.1 | 15.2 |
| PAMAP | 22.4 | 8.5 | 13.6 | 4.6 | 29.2 | 0.9 | 17K | 4K |

and BSAX do not include the time needed to find the hyper-parameters of those models. The approximate total time for the hyper-parameter search is proportional to the number of combinations (Section 8.5.4) multiplied with the average running time of the method.

### 8.5.6 Sensitivity of Parameters

As presented in Section 8.5.4 our hyperparameter search technique is the grid search, where we scan for all the possible combinations of one parameter's values to all the possible values of other parameters. Figure 8.9 shows the sensitivity of SymPol's prediction accuracies against changes in the parameters. As can be easily deduced, the error rate is nonlinear with respect to the parameter values of the method. Therefore, a grid search mechanism is practically suitable, because gradient based methods would have resulted in local optima while nonlinear optimization techniques would require much more computations than the grid.

### 8.5.7 On the Applicability of Previously Covered Methods for Repetitive Datasets

One might rightfully question whether, or not, the methods presented in the previous chapters of this thesis are suitable for repetitive time series. It has been empirically demonstrated that bag-of-words models significantly outperform the state-of-the-art (e.g. DTW) in terms of classifying repetitive time-series [77]. This is an important finding, given that the state-of-the-art methods significantly outperformed the bag of words using the short/non-repetitive datasets of the UCR collection, as detailed in the results of Chapter 6. Such a phenomenon indicates that classes of repetitive time-series are discriminated by the frequency of patterns, rather than the existence/non-existence of patterns. For this reason,

methods that do not count patterns, such as the approaches of Chapters 4, 6 and 7, are implicitly destined to under-perform in the realm of repetitive data. However, the time-series factorization approach of Chapter 5 does count per-segment weights and can be applicable to repetitive data. Unfortunately, the prohibitive aspect of that specific factorization is the runtime, having an asymptotic order of $\mathcal{O}(NQL\mathcal{I}K)$. In contrast, the bag of symbolic polynomial words has a complexity of $\mathcal{O}(NQLd)$, where $d << \mathcal{I}K$. Concretely, we ran the factorization of Chapter 5 on UMW, the smallest repetitive dataset (as detailed in Section 8.5.2), having 60 series with a length of 1200 each. The factorization hyper-parameters were selected following the experimental protocol of Chapter 5 (found as $L = 100, K = 100, \mathcal{I} = 15$). The factorization completed in 1755.81 seconds and yielded a test error of 0.333, which is three orders of magnitude slower than the bag of symbolic polynomials, which completes in 1.4 seconds with an error rate of 0.3167. In this perspective, despite its competitive accuracy, the factorization approach is asymptotically bound to be several orders of magnitude slower than the single-scan bag of words feature extraction.

## 8.6 Conclusion

In this chapter we presented a novel method to classify repetitive time series composed of repeating local patterns. Local polynomial approximations are computed in a sliding window approach for each normalized window segment. The computed polynomial coefficients are converted to symbolic forms (i.e. literal words) via a discretization of the distributions of the coefficients. Thresholds for the distribution of the values of each coefficient are determined to split the coefficient's histogram into equal areas and each area is assigned an alphabet symbol. In a second step all the polynomial coefficients are transformed into characters by locating them within the threshold values of the histogram and assigning the area symbol. The final literal representation of a polynomial is a word composed of the concatenation of each coefficient's character, in the order of the coefficient's monomial degrees. Once the bags of words are computed, a histogram is populated with the frequencies of each word in a time series. We presented a linear time technique to compute the polynomial approximation of a sliding window segment, while the overall method has a run time complexity which is linear in terms of the dataset size.

The classification accuracy of the nearest neighbor method utilizing the histogram rows that our method computed was compared against the performance of three baselines. Our method was the best performer in most experiments,

while achieving a statistically significant margin in the majority of cases. Furthermore, empirical results demonstrate that our method has a fast running time performance.

# Chapter 9

# Thesis Conclusion

## Contents



## 9.1 Summary of Findings

Time series arguably represents the most widely spread type of data in the world. For this reason, mining time series has been a particularly interesting topic for researchers during the last two decades. Specifically, time-series classification is one of the most intriguing tasks for the research community. The problem of classification focuses on predicting the label/category of future time series using a set of recorded instances of expert-labeled series. Unfortunately, time series patterns might be shifted in time, in addition to potentially being locally distorted or scaled. Furthermore, the relevant patterns for classification might be present only on short local segments, rather than on a global structure. Such phenomena are referred to as intra-class variations. Due to those characteristics, a direct usage of time-series values as predictor for off-the-shelf classifiers achieves non-optimal performance [53].

The answer towards resolving the bottlenecks of intra-class variations relies on not using the time-series values as direct features. Instead, the approach of this thesis is to extract a set of features that, on one hand, represent all the variations of the data and, on the other hand, can boost classification accuracy. In other

words, this thesis proposed a list of methods that addressed diverse aspects of intra-class variations.

The first approach that we proposed in Chapter 4 inflates the time series of the training set, in order to create intra-class variations. Virtual instances are created by applying morphing transformations over the values of time series. By adding altered instances, the hyper-plane boundary of off-the-shelf classifiers (in our case SVM) will be updated to accommodate the added variations. The method we described avoids creating variations of all instances, instead we selected only the support vectors, because instances that are close to the decision boundary have a greater chance to influence the hyper-plane. We empirically showed that the performance of an SVM using the inflated set can significantly outperform both an SVM using the original series values and DTW, by yielding higher accuracies (one-vs-all-baselines mode) on 18 datasets against 17 cases where it was inferior to one of the baselines. Whilst successful, inflating the number of training instances creates memory concerns, therefore space-efficient feature extraction methods are demanded.

Our next approach, detailed in Chapter 5, decomposed time-series through a segment-wise convolutional factorization. The strategy involves learning a set of patterns and weights, whose product can approximate each sub-sequence of the time series. The new representation is the summation of the decomposed per-segment weights. By decomposing each segment, it is possible to detect the presence of a pattern independent of its location. On the other side, a low-rank representation reconstructs the segments from a small set of pattern and consecutively avoids reconstructing noisy segments. Such a factorization is learned by optimizing an objective function through coordinate gradient descent. A major advantage of the decomposition technique is the ability to operate in both inductive and transductive modes. The features produces by the factorization helps a SVMs achieve better accuracies against seven baselines over 43 datasets from the UCR collection. For instance, the usage of inductive features helps achieving DTW on 25 datasets, while the transductive features help outperforming DTW on 35 datasets. In addition, our method is more accurate than state-of-the-art semi-supervised methods by a statistically significant margin. Despite the impressive achievements, yet the factorized features are extracted in an unsupervised way, in contrast to the method we elaborate next.

The final solution towards learning a set of patterns, which are useful for classification, is presented in Chapter 6. Shapelets are patterns whose distances to series serve as classification features. The advantage of shapelets relies in utilizing the training labels during the learning process, i.e. the process is supervised.

Since the features are learned on the training labels, there is a higher tendency of performing strongly in terms of predicting the testing labels. Our innovation in terms of shapelets is to propose the first principled shapelet learning method, which can learn patterns directly using the training loss. The results that our method achieves are impressing, it largely outperforms a set of 13 baselines on 28 datasets. The empirical evidences suggest that the shapelet learning classifier is currently the most accurate time-series classifier. For example, it largely outperforms DTW on *all* the 28 datasets.

Even though learning shapelets is very efficient in terms of accuracy, the calculations of shapelet derivatives is computationally expensive. In case a very fast learning time is needed, one can opt for the fast shapelet discovery method presented in Chapter 7. Our strategy is to prune segment candidates using a two step approach. First of all, we prune candidates based on their similarity towards previously considered candidates. Secondly, non-similar (hence diverse) candidates are selected only if the features they produce improve the classification results. Overall, the proposed pruning strategy significantly reduces the time needed to discover shapelets on both univariate and multivariate datasets. Concretely, our method processes MB-scale datasets in seconds and GB-scale datasets in minutes.

The last two chapters of the thesis described two methods that extract features from datasets having special characteristics. Concretely, Chapter 8 presented a feature extraction approach for repetitive time series. Our method is suitable for detecting patterns having different frequencies and occurring in non-periodic orders. From a technical perspective, we computed a polynomial approximation of every segment and converted the polynomial coefficients to symbolic words. The final representation are the frequencies of the symbolic words. An SVM learned over the frequency representation outperforms the state-of-the-art baselines in 6 out of 7 repetitive datasets.

### 9.1.1 Comparison of Thesis Methods

Intuitively, the reader finally expects to learn which of the methods presented throughout the thesis is the best in terms of classification accuracy. Table 9.1 summarizes the comparative accuracy figures of all proposed methods. The datasets used in the analysis belong to the aforementioned UCR collection of datasets [67]. The invariant SVM of Chapter 4 is denoted as ISVM, while the invariant factorization of Chapter 5 is denoted INFA for the transductive mode and INFAI for the inductive mode. In addition, the shapelet learning method of Chapter 6 is denoted as LTS and the fast shapelet discovery as Chapter 7. In addition, we added

two strong baselines for a comparative perspective, specifically DTWNN [39] and TSBF [12]. Meanwhile, except ISVM, all other datasets were experimented using the default training and testing data splits provided by the collection. Even though cross-validation is a better scheme for experimenting, the default splits are an established benchmark and allow a direct comparison to various other published results (e.g. all baselines in Chapters 4-7). As Table 9.1 demonstrates LTS outperforms DTW, TSBF, SD and INFAI with a statistically significant margin, according to the Wilcoxon Signed Rank test. However, the accuracy of the transductive factorization INFA is not significantly worse than LTS. Despite not considering the target variable, INFA has the advantage of exploiting the test predictors in an transductive scheme. Finally, the invariant SVM was experimented using 5-fold cross validation, as detailed in Chapter 4, therefore results show that ISVM is competitive when a sufficient number of labeled instances are present. In contrast, LTS performs qualitatively even with fewer labeled instances. As can be seen from Table 9.1, the default dataset splits have generally fewer training instances than testing ones. Last, it is worth mentioning that the bag-of-polynomials from Chapter 8 focuses only on the scalable classification of highly repetitive (i.e. none of the UCR datasets).

**Conclusive Statements**

- **The most accurate** method of this thesis is the shapelet learning technique of Chapter 6. The classification accuracy of this method is impressive and it currently is the most accurate time-series classification method.

- **The fastest** classifier for time-series data is the supervised pruning technique of Chapter 7. Such a technique enables the classification of MB-scale datasets in matters of seconds and GB-scale datasets in matters of minutes.

- **The most accurate semi-supervised** classifier for time series is the factorization approach of Chapter 5. The transductive decomposition of time-series is significantly more accurate than the other state-of-the-art semi-supervised methods.

## 9.2 Discussions and Future Directions

Time series are becoming increasingly widespread, which indicates that the research community is going to continue dedicating efforts on mining them. In

Table 9.1: Accuracies of Proposed Methods

| Dataset | Cls. | Train | Test | Len. | DTW | TSBF | ISVM* | INFA | INFAI | SD | LS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 50words | 50 | 450 | 455 | 270 | 0.242 | 0.209 | 0.199 | 0.220 | 0.301 | 0.320 | 0.232 |
| Adiac | 37 | 390 | 391 | 176 | 0.391 | 0.245 | 0.202 | 0.322 | 0.435 | 0.417 | 0.437 |
| Beef | 5 | 30 | 30 | 470 | 0.467 | 0.287 | 0.267 | 0.233 | 0.333 | 0.493 | 0.240 |
| CBF | 3 | 30 | 900 | 128 | 0.004 | 0.009 | 0.002 | 0.001 | 0.001 | 0.025 | 0.006 |
| Chlorine | 3 | 467 | 3840 | 166 | 0.350 | 0.336 | - | 0.464 | 0.527 | 0.447 | 0.349 |
| CinCECG | 4 | 40 | 1380 | 1639 | 0.070 | 0.262 | - | 0.138 | 0.291 | 0.227 | 0.167 |
| Coffee | 2 | 28 | 28 | 286 | 0.179 | 0.004 | 0.000 | 0.000 | 0.000 | 0.039 | 0.000 |
| CricketX | 12 | 390 | 390 | 300 | 0.236 | 0.278 | 0.300 | 0.205 | 0.264 | 0.328 | 0.209 |
| CricketY | 12 | 390 | 390 | 300 | 0.197 | 0.259 | 0.271 | 0.197 | 0.354 | 0.325 | 0.249 |
| CricketZ | 12 | 390 | 390 | 300 | 0.180 | 0.263 | 0.306 | 0.192 | 0.274 | 0.327 | 0.201 |
| Diatom | 4 | 16 | 306 | 345 | 0.065 | 0.126 | 0.000 | 0.003 | 0.046 | 0.104 | 0.033 |
| ECG200 | 2 | 100 | 100 | 96 | 0.120 | 0.145 | 0.110 | 0.130 | 0.090 | 0.182 | 0.126 |
| ECGF. | 2 | 23 | 861 | 136 | 0.203 | 0.183 | 0.001 | 0.001 | 0.001 | 0.047 | 0.000 |
| FaceAll | 14 | 560 | 1690 | 131 | 0.192 | 0.234 | 0.020 | 0.238 | 0.380 | 0.286 | 0.218 |
| FaceFour | 4 | 24 | 88 | 350 | 0.114 | 0.051 | 0.036 | 0.000 | 0.011 | 0.180 | 0.048 |
| FacesUCR | 14 | 200 | 2050 | 131 | 0.088 | 0.090 | 0.021 | 0.083 | 0.207 | 0.153 | 0.059 |
| Fish | 7 | 175 | 175 | 463 | 0.160 | 0.080 | 0.094 | 0.023 | 0.051 | 0.245 | 0.066 |
| GunPoint | 2 | 50 | 150 | 150 | 0.087 | 0.011 | 0.030 | 0.007 | 0.007 | 0.069 | 0.000 |
| Haptics | 5 | 155 | 308 | 1092 | 0.588 | 0.488 | - | 0.516 | 0.539 | 0.644 | 0.532 |
| InlineSkate | 7 | 100 | 550 | 1882 | 0.613 | 0.603 | - | 0.636 | 0.640 | 0.615 | 0.573 |
| ItalyPower | 2 | 67 | 1029 | 24 | 0.045 | 0.096 | 0.026 | 0.036 | 0.036 | 0.080 | 0.031 |
| Lighting2 | 2 | 60 | 61 | 637 | 0.131 | 0.257 | 0.289 | 0.180 | 0.213 | 0.205 | 0.177 |
| Lighting7 | 7 | 70 | 73 | 319 | 0.288 | 0.262 | 0.252 | 0.233 | 0.260 | 0.348 | 0.197 |
| MALLAT | 8 | 55 | 2345 | 1024 | 0.086 | 0.037 | - | 0.047 | 0.095 | 0.074 | 0.046 |
| Medical. | 10 | 381 | 760 | 99 | 0.253 | 0.269 | - | 0.299 | 0.416 | 0.324 | 0.271 |
| MoteStrain | 2 | 20 | 1252 | 84 | 0.134 | 0.135 | 0.050 | 0.066 | 0.102 | 0.217 | 0.087 |
| Non.ECG.1 | 42 | 1800 | 1965 | 750 | 0.167 | 0.090 | - | - | - | 0.186 | 0.560 |
| Non.ECG.2 | 42 | 1800 | 1965 | 750 | 0.384 | 0.329 | - | - | - | 0.145 | 0.182 |
| OliveOil | 4 | 30 | 30 | 570 | 0.305 | 0.175 | 0.083 | 0.067 | 0.133 | 0.210 | 0.103 |
| OSULeaf | 6 | 200 | 242 | 427 | 0.141 | 0.196 | 0.296 | 0.095 | 0.190 | 0.434 | 0.082 |
| Sony | 2 | 20 | 601 | 70 | 0.157 | 0.075 | 0.008 | 0.101 | 0.256 | 0.150 | 0.087 |
| SonyII | 2 | 27 | 953 | 65 | 0.062 | 0.034 | 0.004 | 0.054 | 0.105 | 0.220 | 0.036 |
| StarLight. | 3 | 1000 | 8236 | 1024 | 0.017 | 0.008 | - | 0.021 | 0.031 | 0.067 | 0.007 |
| Swedish. | 15 | 500 | 625 | 128 | 0.010 | 0.020 | 0.079 | 0.074 | 0.278 | 0.151 | 0.000 |
| Symbols | 6 | 25 | 995 | 398 | 0.132 | 0.046 | 0.027 | 0.026 | 0.034 | 0.135 | 0.003 |
| synthetic. | 6 | 300 | 300 | 60 | 0.002 | 0.001 | 0.020 | 0.013 | 0.033 | 0.017 | 0.003 |
| Trace | 4 | 100 | 100 | 275 | 0.227 | 0.164 | 0.030 | 0.000 | 0.000 | 0.035 | 0.200 |
| TwoPatt. | 4 | 1000 | 4000 | 128 | 0.301 | 0.249 | 0.001 | 0.003 | 0.016 | 0.019 | 0.287 |
| TwoL. | 2 | 23 | 1139 | 82 | 0.322 | 0.217 | 0.002 | 0.002 | 0.018 | 0.133 | 0.269 |
| uWaveX | 8 | 896 | 3582 | 315 | 0.005 | 0.004 | 0.193 | 0.176 | 0.192 | 0.239 | 0.004 |
| uWaveY | 8 | 896 | 3582 | 315 | 0.252 | 0.302 | 0.253 | 0.237 | 0.303 | 0.329 | 0.340 |
| uWaveZ | 8 | 896 | 3582 | 315 | 0.155 | 0.149 | 0.249 | 0.233 | 0.254 | 0.324 | 0.150 |
| Wafer | 2 | 1000 | 6174 | 152 | 0.095 | 0.022 | 0.002 | 0.002 | 0.003 | 0.007 | 0.034 |
| WordsS. | 25 | 267 | 638 | 270 | 0.185 | 0.138 | 0.200 | 0.307 | 0.386 | 0.375 | 0.131 |
| yoga | 2 | 300 | 3000 | 426 | 0.129 | 0.130 | - | 0.119 | 0.185 | 0.202 | 0.089 |
| **LS-vs-Method, LS Wins** | | | | | **33** | **29** | **-** | **25** | **34** | **37** | **-** |
| **LS-vs-Method, Draw** | | | | | **0** | **0** | **-** | **1** | **1** | **0** | **-** |
| **LS-vs-Method, Method Wins** | | | | | **12** | **16** | **-** | **19** | **10** | **8** | **-** |
| **Wilcoxon Test (p-value)** | | | | | **0.005** | **0.033** | **-** | **0.896** | **0.001** | **0.000** | **-** |

terms of classification, the shapelet-based methods are currently the most successful alternative with respect to accuracy. Keeping the focus of this thesis in mind, there are at least two directions for achieving additional progress. One direction is to develop a GPU implementation of the shapelet learning method from Chapter 6. In that way, a practitioner would be able to use the most accurate shapelet method on truly massive datasets. The other direction can be the detection of sequences of patterns. The shapelet strategy is to detect presences of patterns, not their order. Arguably, in certain datasets, the sequence of patterns is as important as their occurrence. While the detection of such a sequence is non-trivial, it nevertheless opens a door for potential future work. Last, but not least, a prominent direction would be to extend the techniques of time-series classification towards the domain of videos, for instance by treating each pixel as a series dimension.

# References


[1] Allan, J., Papka, R., and Lavrenko, V. (1998). On-line new event detection and tracking. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 37–45, New York, NY, USA. ACM. 130

[2] Andoni, A. (2009). *Nearest neighbor search: the old, the new, and the impossible*. PhD thesis, Massachusetts Institute of Technology. 34

[3] Apté, C., Ghosh, J., and Smyth, P., editors (2011). *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*. ACM. 187, 191

[4] Arthur, D. and Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics. 73

[5] Bahlmann, C., Haasdonk, B., and Burkhardt, H. (2002). On-line handwriting recognition with support vector machines - a kernel approach. In *In Proc. of the 8th IWFHR*, pages 49–54. 37

[6] Banos, O., Garcia, R., Holgado-Terriza, J., Damas, M., Pomares, H., Rojas, I., Saez, A., and Villalonga, C. (2014a). mhealthdroid: A novel framework for agile development of mobile health applications. In Pecchia, L., Chen, L., Nugent, C., and Bravo, J., editors, *Ambient Assisted Living and Daily Activities*, volume 8868 of *Lecture Notes in Computer Science*, pages 91–98. Springer International Publishing. 144, 147

[7] Banos, O., Toth, M. A., Damas, M., Pomares, H., and Rojas, I. (2014b). Dealing with the effects of sensor displacement in wearable activity recognition. *Sensors*, 14(6):9995–10023. 144, 147

[8] Barthelemy, Q., Larue, A., Mayoue, A., Mercier, D., and Mars, J. (2012). Shift and 2d rotation invariant sparse coding for multivariate signals. *Signal Processing, IEEE Transactions on*, 60(4):1597–1611. 43

[9] Batista, G. E., Keogh, E. J., Tataw, O. M., and Souza, V. M. (2013). Cid: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery*, pages 1–36. 23, 80, 83, 85, 87

[10] Batista, G. E. A. P. A., Wang, X., and Keogh, E. J. (2011). A complexity-invariant distance measure for time series. In *SDM*, pages 699–710. SIAM / Omnipress. 23

[11] Baydogan, M. and Runger, G. (2015). Learning a symbolic representation for multivariate time series classification. *Data Mining and Knowledge Discovery*, 29(2):400–422. 27

[12] Baydogan, M., Runger, G., and Tuv, E. (2013). A bag-of-features framework to classify time series. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PP(99):1–1. 31, 80, 83, 180

[13] Berg, I. and Groenen, P. J. (2005). *Modern Multidimensional Scaling*. New York: Springer. 2, 26

[14] Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA. 72

[15] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. 10, 12, 14

[16] Bruno, B., Mastrogiovanni, F., Sgorbissa, A., Vernazza, T., and Zaccaria, R. (2013). Analysis of human behavior recognition algorithms based on acceleration data. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1602–1607. 147

[17] Buza, K. and Schmidt-Thieme, L. (2008). Motif-based classification of time series with bayesian networks and svms. In *GfKl*, pages 105–114. 30, 43, 153

[18] Cadzow, J. A., Baseghi, B., and Hsu, T. (1983). Singular-value decomposition approach to time series modelling. *Communications, Radar and Signal Processing, IEE Proceedings F*, 130(3):202–210. 27

[19] Cai, D., He, X., Han, J., and Huang, T. S. (2011). Graph regularized non-negative matrix factorization for data representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1548–1560. 39

[20] Camerra, A., Palpanas, T., Shieh, J., and Keogh, E. (2010). isax 2.0: Indexing and mining one billion time series. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 58–67, Washington, DC, USA. IEEE Computer Society. 27

[21] Cetin, M. S., Mueen, A., and Calhoun, V. D. (2015). Shapelet ensemble for multi-dimensional time series. In *SDM*. 145, 147, 148

[22] Chakrabarti, K., Keogh, E., Mehrotra, S., and Pazzani, M. (2002). Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228. 28, 130

[23] Chan, K.-P. and Fu, A.-C. (1999). Efficient time series matching by wavelets. In *15th International Conference on Data Engineering, Proceedings 1999*, pages 126–133. 27, 28

[24] Chang, K.-W., Deka, B., Hwu, W.-M. W., and Roth, D. (2012a). Efficient pattern-based time series classification on gpu. In *Proceedings of the 12th IEEE International Conference on Data Mining*. 34

[25] Chang, K.-W., Deka, B., mei W. Hwu, W., and Roth, D. (2012b). Efficient pattern-based time series classification on gpu. In [130], pages 131–140. 97

[26] Chen, L. and Ng, R. (2004). On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 792–803. VLDB Endowment. 23

[27] Chen, L., Özsu, M. T., and Oria, V. (2005). Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 491–502, New York, NY, USA. ACM. 23

[28] Chen, Y., Dong, G., Han, J., Wah, B. W., and Wang, J. (2002). Multi-dimensional regression analysis of time-series data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 323–334. VLDB Endowment. 27

[29] Chen, Y., Hu, B., and Keogh, E. J. (2013). Time series classification under more realistic assumptions. In *SDM*, pages 578–586. SIAM. 167, 168

[30] Chen, Y., Nascimento, M., Ooi, B.-C., and Tung, A. (2007). Spade: On shape-based pattern detection in streaming time series. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 786–795. 23

[31] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297. 35

[32] Cristianini, N. and Shawe-Taylor, J. (2010). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press. 4, 18, 35, 36

[33] Cuturi, M. (2011). Fast global alignment kernels. In et al., G., editor, *Proceedings of the ICML 2011*, ICML 2011, pages 929–936, New York, NY, USA. ACM. 38

[34] Das, A. and Kempe, D. (2008). Algorithms for subset selection in linear regression. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC '08, pages 45–54, New York, NY, USA. ACM. 108

[35] Das Gupta, M. and Xiao, J. (2011). Non-negative matrix factorization as a feature selection tool for maximum margin classifiers. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 2841–2848, Washington, DC, USA. IEEE Computer Society. 41

[36] DeCoste, D. and Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190. 37, 47

[37] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30. 83, 117

[38] Ding, C., He, X., and Simon, H. D. (2005). On the equivalence of nonnegative matrix factorization and spectral clustering. In *in SIAM International Conference on Data Mining*. 68

[39] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. J. (2008). Querying and mining of time series data: experimental comparison of representations and distance measures. volume 1, pages 1542–1552. 22, 23, 26, 115, 167, 170, 180

[40] Duncan, B. and Elkan, C. (2014). Nowcasting with numerous candidate predictors. In Calders, T., Esposito, F., Hüllermeier, E., and Meo, R., editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I*, volume 8724 of *Lecture Notes in Computer Science*, pages 370–385. Springer. 19

[41] Elfeky, M., Aref, W., and Elmagarmid, A. (2005). Periodicity detection in time series databases. *Knowledge and Data Engineering, IEEE Transactions on*, 17(7):875–887. 156

[42] Esling, P. and Agon, C. (2012). Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34. 2

[43] et. al., N. V. C. (2002). Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357. 44, 60

[44] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, SIGMOD '94, pages 419–429, New York, NY, USA. ACM. 27, 28

[45] Fuchs, E., Gruber, T., Nitschke, J., and Sick, B. (2010). Online segmentation of time series based on polynomial least-squares approximations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2232–2245. 27

[46] Gemulla, R., Nijkamp, E., Haas, P. J., and Sismanis, Y. (2011). Large-scale matrix factorization with distributed stochastic gradient descent. In [3], pages 69–77. 38

[47] Ghalwash, M. and Obradovic, Z. (2012). Early classification of multivariate temporal observations by extraction of interpretable shapelets. *BMC Bioinformatics*, 13(1). 145

[48] Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000 (June 13)). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220. Circulation Electronic Pages: http://circ.ahajournals.org/cgi/content/full/101/23/e215 PMID:1085218; doi: 10.1161/01.CIR.101.23.e215. 19, 167, 169

[49] Gooijer, J. G. D. and Hyndman, R. J. (2006). 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443 – 473. Twenty five years of forecasting. 2, 19

[50] Grabocka, J., Drumond, L., and Schmidt-Thieme, L. (2013). Supervised dimensionality reduction via nonlinear target estimation. In *DaWaK*, pages 172–183. 39, 40

[51] Grabocka, J., Nanopoulos, A., and Schmidt-Thieme, L. (2012a). Classification of sparse time series via supervised matrix factorization. In *AAAI*. 27, 43

[52] Grabocka, J., Nanopoulos, A., and Schmidt-Thieme, L. (2012b). Classification of sparse time series via supervised matrix factorization. In *AAAI*. 41

[53] Grabocka, J., Nanopoulos, A., and Schmidt-Thieme, L. (2012c). Invariant time-series classification. In Flach, P. A., Bie, T. D., and Cristianini, N., editors, *ECML/PKDD (2)*, volume 7524 of *Lecture Notes in Computer Science*, pages 725–740. Springer. 2, 19, 177

[54] Grabocka, J., Schilling, N., Wistuba, M., and Schmidt-Thieme, L. (2014a). Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 392–401, New York, NY, USA. ACM. 3, 22, 140, 144

[55] Grabocka, J., Schilling, N., Wistuba, M., and Schmidt-Thieme, L. (2014b). Supporting Website for the Learning Shapelets Method. `http://fs.ismll.de/publicspace/LearningShapelets/`. [Online; accessed 19-April-2015]. 3

[56] Grabocka, J. and Schmidt-Thieme, L. (2015). Learning through non-linearly supervised dimensionality reduction. In Hameurlain, A., Kng, J., Wagner, R., Bellatreche, L., and Mohania, M., editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems XVII*, volume 8970 of *Lecture Notes in Computer Science*, pages 74–96. Springer Berlin Heidelberg. 39, 40

[57] Gudmundsson, S., Runarsson, T. P., and Sigurdsson, S. (2008). Support vector machines and dynamic time warping for time series. In *IJCNN*, pages 2772–2776. IEEE. 38, 46

[58] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182. 128

[59] Hartmann, B. and Link, N. (2010). Gesture recognition with inertial sensors and optimized dtw prototypes. In *IEEE International Conference on Systems Man and Cybernetics*. 33

[60] Hartmann, B., Schwab, I., and Link, N. (2010). Prototype optimization for temporarily and spatially distorted time series. In *the AAAI Spring Symposia*. 33

[61] He, Q., Zhuang, F., Shang, T., Shi, Z., et al. (2012). Fast time series classification based on infrequent shapelets. In *11th IEEE International Conference on Machine Learning and Applications*. 34

[62] Hills, J., Lines, J., Baranauskas, E., Mapp, J., and Bagnall, A. (2013). Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*. 31, 32, 97, 99, 112, 114, 115, 117, 122, 143, 144

[63] Huang, P.-S., Yang, J., Hasegawa-Johnson, M., Liang, F., and Huang, T. S. (2012). Pooling robust shift-invariant sparse representations of acoustic signals. In *INTERSPEECH*. ISCA. 43

[64] Huybrechs, D. (2010). On the fourier extension of nonperiodic functions. *SIAM J. Numer. Anal.*, 47(6):4326–4355. 156

[65] Kate, R. J. (2015). Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, pages 1–30. 26

[66] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. (2001a). Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286. 43

[67] Keogh, E., Zhu, Q., Hu, B., Y., H., Xi, X., Wei, L., and Ratanamahatana, C. A. (2011). The UCR Time Series Classification/Clustering Homepage. `www.cs.ucr.edu/~eamonn/time_series_data/`. [Online; accessed 02-March-2014]. 3, 17, 48, 92, 124, 129, 135, 179

[68] Keogh, E. J., Chakrabarti, K., Pazzani, M. J., and Mehrotra, S. (2001b). Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3(3):263–286. 27

[69] Keogh, E. J. and Pazzani, M. J. (2000). Scaling up dynamic time warping for datamining applications. In *KDD*, pages 285–289. 23, 24, 80

[70] Keogh, E. J. and Ratanamahatana, C. A. (2005). Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386. 24

[71] Koren, Y., Bell, R. M., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37. 38, 39, 40

[72] Krollner, B., Vanstone, B. J., and Finnie, G. R. (2010). Financial time series forecasting with machine learning techniques: a survey. In *ESANN*. 19

[73] Kuksa, P. and Pavlovic, V. (2010). Spatial representation for efficient sequence classification. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3320–3323. 23, 80

[74] Lewicki, M. S. and Sejnowski, T. J. (1999). Coding time-varying signals using sparse, shift-invariant representations. In *Proceedings of NIPS*, pages 730–736, Cambridge, MA, USA. MIT Press. 43

[75] Lin, J., Keogh, E., Wei, L., and Lonardi, S. (2007). Experiencing sax: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144. 27, 30

[76] Lin, J., Keogh, E. J., Lonardi, S., and chi Chiu, B. Y. (2003). A symbolic representation of time series, with implications for streaming algorithms. In Zaki, M. J. and Aggarwal, C. C., editors, *DMKD*, pages 2–11. ACM. 28

[77] Lin, J., Khade, R., and Li, Y. (2012). Rotation-invariant similarity in time series using bag-of-patterns representation. *J. Intell. Inf. Syst.*, 39(2):287–315. 30, 43, 80, 153, 154, 166, 167, 168, 169, 174

[78] Lin, J. and Li, Y. (2009). Finding structural similarity in time series data using bag-of-patterns representation. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, SSDBM 2009, pages 461–477, Berlin, Heidelberg. Springer-Verlag. 30, 43, 80, 153, 154, 166

[79] Lines, J. and Bagnall, A. (2012). Alternative quality measures for time series shapelets. In *Intelligent Data Engineering and Automated Learning*, volume 7435 of *Lecture Notes in Computer Science*, pages 475–483. 97

[80] Lines, J., Davis, L., Hills, J., and Bagnall, A. (2012). A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 31, 97, 98, 99, 107, 112, 114, 115, 117

[81] Loosli, G., Canu, S., and Bottou, L. (2007). Training invariant support vector machines using selective sampling. In *Large Scale Kernel Machines*, pages 301–320. MIT Press. 37

[82] Loosli, G., Canu, S., Vishwanathan, S. V. N., and Smola, A. J. (2005). Invariances in classification : an efficient svm implementation. In *Proceedings of the 11th International Symposium on Applied Stochastic Models and Data Analysis*. 37

[83] Marussy, K. and Buza, K. (2013). Success: A new approach for semi-supervised classification of time-series. In Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., and Zurada, J., editors, *Artificial Intelligence and Soft Computing*, volume 7894 of *Lecture Notes in Computer Science*, pages 437–447. Springer Berlin Heidelberg. 85

[84] Menon, A. K. and Elkan, C. (2010). Predicting labels for dyadic data. *Data Min. Knowl. Discov.*, 21(2):327–343. 38, 41

[85] Moody, G. and Mark, R. (2001). *The impact of the MIT-BIH arrhythmia database*. IEEE Eng in Med and Biol 20(3):45-50. 3

[86] Mueen, A., Keogh, E., and Young, N. (2011a). Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 34, 97, 115, 121, 134, 136, 140

[87] Mueen, A., Keogh, E. J., and Young, N. (2011b). Logical-shapelets: an expressive primitive for time series classification. In [3], pages 1154–1162. 148

[88] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press. 10, 12, 14, 15

[89] Niyogi, P., Girosi, F., and Poggio, T. (1998). Incorporating prior information in machine learning by creating virtual examples. In *Proceedings of the IEEE*. 37

[90] Patel, P., Keogh, E., Lin, J., and Lonardi, S. (2002). Mining motifs in massive time series databases. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 370–, Washington, DC, USA. IEEE Computer Society. 28, 29

[91] Paulin, M., Revaud, J., Harchaoui, Z., Perronnin, F., and Schmid, C. (2014). Transformation pursuit for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 3646–3653. 44

[92] Platt, J. C. (1999). Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA. 71

[93] Rahimi, A., Recht, B., and Darrell, T. (2007). Learning to transform time series with a few examples. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(10):1759–1775. 38

[94] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD*, KDD 2012, pages 262–270, New York, NY, USA. ACM. 23, 80, 171

[95] Rakthanmanon, T. and Keogh, E. (2013). Fast shapelets: A scalable algorithm for discovering time series shapelets. *Proceedings of the 13th SIAM International Conference on Data Mining*. 34, 80, 83, 97, 115, 117, 121, 135, 136, 139, 140, 148

[96] Rasheed, F., Alshalalfa, M., and Alhajj, R. (2011). Efficient periodicity mining in time series databases using suffix trees. *IEEE Transactions on Knowledge and Data Engineering*, 23(1):79–94. 156

[97] Rebbapragada, U., Protopapas, P., Brodley, C. E., and Alcock, C. R. (2009). Finding anomalous periodic time series: An application to catalogs of periodic variable stars. *CoRR*, abs/0905.3428. 11

[98] Rendle, S. and Schmidt-Thieme, L. (2008). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In Pu, P., Bridge, D. G., Mobasher, B., and Ricci, F., editors, *RecSys*, pages 251–258. ACM. 38

[99] Rutkowski, T. M., Zdunek, R., and Cichocki, A. (2007). Multichannel EEG brain activity pattern analysis in time-frequency domain with nonnegative matrix factorization support. *International Congress Series*, 1301:266–269. 38, 39

[100] Ryan, T. P. (2008). *Modern Regression Methods*. Wiley-Interscience, New York, NY, USA, 2 edition. 160

[101] Schaefer, S., McPhail, T., and Warren, J. D. (2006). Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540. 38, 48, 50, 52

[102] Schölkopf, B., Burges, C., and Vapnik, V. (1996). Incorporating invariances in support vector learning machines. In von der Malsburg, C., von Seelen, W., Vorbrüggen, J. C., and Sendhoff, B., editors, *ICANN*, volume 1112 of *Lecture Notes in Computer Science*, pages 47–52. Springer. 37, 47, 56

[103] Scholkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA. 35, 36

[104] Sethares, W. A. and Staley, T. W. (1999). Periodicity transforms. *Signal Processing, IEEE Transactions on*, 47(11):2953–2964. 157, 168

[105] Shieh, J. and Keogh, E. (2008). isax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 623–631, New York, NY, USA. ACM. 27

[106] Shimodaira, H., ichi Noma, K., Nakai, M., and Sagayama, S. (2001). Support vector machine with dynamic time-alignment kernel for speech recognition. In *INTERSPEECH*, pages 1841–1844. ISCA. 37

[107] Singh, A. P. and Gordon, G. J. (2008). A unified view of matrix factorization models. In *ECML/PKDD (2)*, pages 358–373. 38

[108] Sivakumar, P. and Shajina, T. (2012). Human gait recognition and classification using time series shapelets. In *IEEE International Conference on Advances in Computing and Communications*. 33

[109] Smaragdis, P. and Brown, J. C. (2003). Non-negative matrix factorization for polyphonic music transcription. In *In IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 177–180. 38, 39

[110] Srebro, N., Rennie, J. D. M., and Jaakola, T. S. (2005). Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems 17*, pages 1329–1336. MIT Press. 38

[111] Valderrama, M. J., Ocaña, F. A., and Aguilera, A. M. (2002). Forecasting PC–ARIMA models for functional data. In Härdle, W. and Rönz, B., editors, *Proceedings in Computational Statistics*, pages 25–36, Heidelberg. Physica–Verlag. 19

[112] Vlachos, M., Kollios, G., and Gunopulos, D. (2002). Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. 23

[113] Vlachos, M., Meek, C., Vagena, Z., and Gunopulos, D. (2004). Identifying similarities, periodicities and bursts for online search queries. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 131–142, New York, NY, USA. ACM. 156

[114] Wang, F., Lee, N., Hu, J., Sun, J., and Ebadollahi, S. (2012). Towards heterogeneous temporal clinical event pattern discovery: a convolutional approach. In *Proceedings of ACM SIGKDD*, KDD '12, pages 453–461, New York, NY, USA. ACM. 43

[115] Wang, J., Liu, P., She, M. F., Nahavandi, S., and Kouzani, A. (2013a). Bag-of-words representation for biomedical time series classification. *Biomedical Signal Processing and Control*, 8(6):634 – 644. 30

[116] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., and Keogh, E. (2013b). Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309. 26

[117] Wang, Z. and Oates, T. (2015). Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAi. 44

[118] Wei, L. and Keogh, E. (2006). Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 748–753, New York, NY, USA. ACM. 85

[119] Wei, L., Keogh, E., and Xi, X. (2006). Saxually explicit images: Finding unusual shapes. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 711–720, Washington, DC, USA. IEEE Computer Society. 27

[120] Wild, E. W. (2008). *Optimization-based Machine Learning and Data Mining*. ProQuest. 105

[121] Williams, B., Toussaint, M., and Storkey, A. (2006). Extracting motion primitives from natural handwriting data. In Kollias, S., Stafylopatis, A., Duch, W., and Oja, E., editors, *Artificial Neural Networks ICANN 2006*, volume 4132 of *Lecture Notes in Computer Science*, pages 634–643. Springer Berlin Heidelberg. 147

[122] Wistuba, M., Grabocka, J., and Schmidt-Thieme, L. (2015). Ultra-fast shapelets for time series classification. *CoRR*, abs/1503.05018. 34

[123] Xi, X., Keogh, E. J., Shelton, C. R., Wei, L., and Ratanamahatana, C. A. (2006). Fast time series classification using numerosity reduction. In *ICML*. 24, 26, 57

[124] Xing, Z., Pei, J., and Yu, P. (2012). Early classification on time series. *Knowledge and information systems*, 31(1):105–127. 33

[125] Xing, Z., Pei, J., Yu, P., and Wang, K. (2011). Extracting interpretable features for early classification on time series. *Proceedings of the 11th SIAM International Conference on Data Mining*. 33

[126] Ye, L. and Keogh, E. (2009). Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 31, 34, 97, 98, 99, 107, 115, 121, 130, 134, 140

[127] Ye, L. and Keogh, E. (2011). Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data Mining and Knowledge Discovery*, 22(1):149–182. 98

[128] Yu, H.-F., Hsieh, C.-J., Si, S., and Dhillon, I. S. (2012). Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In [130], pages 765–774. 70, 71

[129] Zakaria, J., Mueen, A., and Keogh, E. (2012). Clustering time series using unsupervised-shapelets. In *Proceedings of the 12th IEEE International Conference on Data Mining*. 33, 97

[130] Zaki, M. J., Siebes, A., Yu, J. X., Goethals, B., Webb, G. I., and Wu, X., editors (2012). *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*. IEEE Computer Society. 185, 195

[131] Zhang, D., Zuo, W., Zhang, D., and Zhang, H. (2010). Time series classification using support vector machine with gaussian elastic metric kernel. In *ICPR*, pages 29–32. IEEE. 38