Multi-operator Temporal Decision Trees

Vera Shalaeva

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Vera SHALAEVA

Thèse dirigée par **Ahlame DOUZAL**
et codirigée par **Cécile AMBLARD**
préparée au sein du **Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Arbre de Décision Temporel Multi-opérateur

Multi-operator Temporal Decision Trees

Thèse soutenue publiquement le **30 novembre 2018**,
devant le jury composé de :

Monsieur Pierre-François MARTEAU
Professeur, Université Bretagne Sud, Rapporteur
Monsieur Christophe MARSALA
Professeur, Sorbonne Université , Rapporteur
Monsieur Romain TAVENARD
Maître de Conférences, Université Rennes 2, Examinateur
Monsieur Massih-Reza AMINI
Professeur, Université Grenoble Alpes, Président
Madame Ahlame DOUZAL
Maître de Conférences, Université Grenoble Alpes, Directeur de thèse
Monsieur Gilles BISSON
Chargé de Recherche, CNRS, Co-directeur de thèse

Contents

List of Figures

Table 1: List of frequently used notations

$(\mathbf{TS}, \mathcal{Y})$	$\triangleq$	training set
$\mathbf{TS} = \{ts_1, \ldots, ts_N\}$	$\triangleq$	the set of N time series
$\mathcal{Y} \in \{1, \ldots, K\}$	$\triangleq$	the set of k classes
$\mathcal{G}$	$\triangleq$	the set of nodes in a tree
g_L	$\triangleq$	left node descending from a parent node g
g_R	$\triangleq$	right node descending from a parent node g
$\mathbf{TS}_L$	$\triangleq$	set of time series assigned to g_L
$\mathbf{TS}_R$	$\triangleq$	set of time series assigned to g_R
N^L, N^R	$\triangleq$	the number of time series in nodes g_L and g_R
p^L, p^R	$\triangleq$	time series proportion into g_L and g_R with respect to N
$\{N_1, \ldots, N_K\}$	$\triangleq$	the number of observations each class contains
$\{p_1, \ldots, p_K\}$	$\triangleq$	the proportion of each class
$\lvert ts \rvert = m$	$\triangleq$	length of time series
t	$\triangleq$	timestamp of one observable time series value
v_i^t	$\triangleq$	value of an observable time series ts_i at a timestamp t
D	$\triangleq$	matrix of size $N \times N$ of pairwise distances between time series from a training set
$d(ts_i, ts_j)$	$\triangleq$	distance values between time series ts_i and ts_j computed with similarity measure d
$\mathbf{d}_{i,r}^t = \{d_{(i,r)}^t\}_{t=1}^m$	$\triangleq$	the set of distances between values of time series at given timestamp t of ts_r
$ts_r^w, r \in 1, \ldots, N$	$\triangleq$	weighted time series from a training set $\mathbf{TS}$
w_r	$\triangleq$	weight vector of a time series ts_r
l_p	$\triangleq$	p norm distance for time series comparison, $p = 2$ corresponds to Euclidean distance
$\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_s\}$	$\triangleq$	warping path of Dynamic Time Warping
$\mathsf{p}(\mathbf{x})$	$\triangleq$	probability distribution
$\mathcal{F}(.) = \{f\}$	$\triangleq$	split operator
$\{(\boldsymbol{p}, \nu)\}$	$\triangleq$	the set of patterns extracted from time series, p stands for a pattern, ν is an attribute
$\mathfrak{l}$	$\triangleq$	lower bound
$\mathfrak{u}$	$\triangleq$	upper bound

Abstract

Rising interest in mining and analyzing time series data in many domains motivates designing machine learning (ML) algorithms that are capable of tackling such complex data. Except of the need in modification, improvement, and creation of novel ML algorithms that initially works with static data, criteria of its interpretability, accuracy and computational efficiency have to be fulfilled. For a domain expert, it becomes crucial to extract knowledge from data and appealing when a yielded model is transparent and interpretable. So that, no preliminary knowledge of ML is required to read and understand results. Indeed, an emphasized by many recent works, it is more and more needed for domain experts to get a transparent and interpretable model from the learning tool, thus allowing them to use it, even if they have few knowledge about ML's theories. Decision Tree is an algorithm that focuses on providing interpretable and quite accurate classification model.

More precisely, in this research, we address the problem of interpretable time series classification by Decision Tree (DT) method. Firstly, we present Temporal Decision Tree, which is the modification of classical DT algorithm. The gist of this change is the definition of a node's split. Secondly, we propose an extension, called Multi-operator Temporal Decision Tree (MTDT), of the modified algorithm for temporal data that is able to capture different geometrical classes structures. The resulting algorithm improves model readability while preserving the classification accuracy.

Furthermore, we explore two complementary issues: computational efficiency of the extended algorithm and its classification accuracy. We suggest that the decreasing of the former is reachable using a Local Search approach to built nodes. And preserving of the latter can be handled by discovering and weighting discriminative time stamps of time series.

Résumé

Aujourd'hui, du fait de la multiplication du nombre des capteurs et, plus généralement, de celle des données issues de dispositifs connectés, de nombreux domaines d'activité s'intéressent à la classification automatique des séries temporelles. Au-delà de la recherche théorique de nouveaux algorithmes d'apprentissage automatique capables de traiter ces données complexes, il est important de fournir aux utilisateurs des méthodes capables de construire efficacement des modèles prédictifs, mais aussi de se focaliser sur l'explicabilité des modèles générés et la transparence des processus mis en oeuvre.

Ainsi, les utilisateurs qui n'ont pas forcément des connaissances en théorie d'apprentissage peuvent prendre en main plus rapidement ces méthodes et surtout valider la qualité des connaissances apprises vis à vis de leur domaine d'expertise.

Dans ce travail de doctorat, nous nous avons cherché à définir un arbre de décision temporel ayant les caractéristiques suivantes:

- une bonne precision,
- des résultats interprétables,
- une complexité controlée.

Ceci nous a conduit à nous focaliser sur les trois aspects suivants: Premièrement, nous avons cherché à definir des opérateurs de séparation adaptés aux series temporelles. Puis nous avons introduit la notion d'arbre de décision temporel multi-opérateur (MTDT) qui consiste à utiliser, en concurrence, plusieurs méthodes pour construire chaque noeud. D'une part cela permet d'améliorer les capacités prédictives des arbres en capturant les meilleures structures géométriques discriminantes pour chaque classe et pour chaque niveau de l'arbre. D'autre part, grâce à cette approche on améliore la lisibilité des modèles en réduisant significativement la taille des arbres qui sont produits. Deuxièmement, nous avons cherché à réduire la complexité des algorithmes en utilisant une recherche locale pour explorer les opérateurs de construction des noeuds. Cette recherche s'appuie sur la définition de bornes dans les métriques utilisées. Elle permet de réduire l'espace des candidats possibles lors de la recherche de la "meilleure" séparation. Enfin, nous avons cherché à introduire la notion de parties

discriminantes locales d'une série temporelle. Chaque instant d'observation de la série est alors pondéré par un poids correspondant au pouvoir discriminant de cet élément de la série. Nous avons développé et comparé différentes méthodes automatiques de pondération des sous-séquences des séries temporelles de manière à maximiser la précision des arbres de décision produits.

Acknowledgements

First and foremost I would like to thank my advisors Gilles Bisson, Cécile Amblard, and Ahlame Douzal for all their contributions of time and ideas. During all these years you provided me assistance to carry this work and inspired me in the challenging moments of procrastination.

This research could not have been completed without the financial support of the IKATS project (an Innovative Toolkit for Analysing Time Series), which was a Research and Development project funded by BPIfrance in the frame of the french national PIA program. I am thankful not only for providing the funding for this study, but also for giving me the opportunity to attend summer schools, conferences and meet so many interesting people.

Besides, I would also like to thank my committee members: Pierre-François Marteau, Christophe Marsala, Roman Tavenard, and Massih-Reza Amini for letting my defense be an enjoyable moment. I am grateful to the jury members for their interesting questions and insightful comments.

I would like to thank all the members of AMA team who have been the source of friendship and collaboration during my PhD studies. Immense appreciation is given to my friends who were always willing to help me and who kept a sense of humour when I had lost mine.

Lastly, I would like to thank my family and express my infinite indebtedness to my parents Antonina Shalaeva and Aleksandr Shalaev for all their love, encouragement, and support in all my pursuits.

Chapter 1

Introduction

1.1 Introduction

Nowadays, vast amount of data has been collecting almost in any domain of interest. They can be static, that is to say, they represent real-world processes at fixed point of time. Or data can be dynamic, if we consider it evolves through time axis, such data can be represented by time series. More concretely, a time series represents an ordered sequence of values, each one being obtained from a measurement at a specified time point. We can observe these type of data in any field because they naturally arise once we are interested in seeing and analyzing any object and its characteristics over time. For example, in biology a time series can be any physiologic signal of a patient, an economic indicator in finance, power consumption in energy, an average temperature in meteorology, etc.

Time series analysis has been taking an important role in many disciplines as economics, medecine, seismology and meteorology, industry, and recently human-machine interfaces. Analysis is rather a general term that includes variety of tasks such as prediction, classification, anomaly detection, patterns/motif discovery, etc. In this research, we focus our interest on time series classification task that has attracted enormous attention in the last two decades. It rises in broad range of application, where a class object is represented by time series and the objective is to assign a class label for a new unlabelled one. For example, in airplane tracking each time series can represent some attribute of a plane's landing process (speed, angle, etc) that states about its quality. Then the goal is to assign a class label indicating quality of landing to a new track. In gesture recognition each time series can represent a movement, and the goal is to recognize which movement was made based on recorded signal. There are many others examples: protein sequences classification [20], disease classification based on recorded data from sensors, in web search distinguishing humans from web

search robots [23], classification of abnormal behaviour [49], traffic classification for the applications associated with traffic flow [10].

As a consequence of such substantial interest there has been a dramatic increase of interest in applying machine learning (ML) algorithms to this task. However, most of conventional machine learning techniques work on static data and not allows straightforward extension to temporal data. The main reason is that time series data are not initially represented by an explicit set of features. As the result, a great deal of research has gone to develop new algorithms that are capable of handling time series data. Lots of them are becoming highly sophisticated, so that it becomes very difficult for a domain expert to interpret results of model without knowledge of machine learning and data mining. It implies restriction on usability of machine learning algorithms in some domains.

This study lies in the scope of IKATS project (an Innovative Tool-Kit for Analysing Time Series). The goal of which is providing an easy-to-use framework for the comprehensive and exploratory analysis of large volumes of time series data. Withing the tool, which is dedicated to a domain data scientist and build in the form of open-source project [2], IKATS supports ML algorithm for clustering, classification and patterns discovery in time series. In addition, it allows visualisation of data and built models.

In this research, we aim to develop time series classification algorithm that will possess properties of transparency and interpretability, while keeping decent results of classification accuracy. By aiming these objectives, we target to have a machine learning algorithm whose output model can be visualised and be easily used by a domain expert.

We define time series classification as a problem of building a classifier from a collection of labelled training time series set. When we have a classifier, its evaluating can be done by answering following questions:

- how does it help us to understand the data and how can we interpret the results?
- what is the level of performance, how accurate results of the classification?
- how well an algorithm scales when data is growing?

Now, based on above-mentioned motivation for tackling the time series classification problem, and estimation strategy of developed classifier, we aim to design the machine learning algorithm that fullfills properties of interpretability, accuracy and computational efficiency. Decision tree algorithm was selected as the most suitable

baseline. Tree classifier has a good accuracy for classification task. It is often selected among other algorithms when it is important to build interpretable model [66]. Decision trees allow to a domain expert see the logic of classification decisions. Also, it is easy to support this classifier by a visualisation tool showing the feedback of classification results.

The outline of this manuscript is following:

- Chapter 2 presents state-of-the-art of research done on classification of time series.
- Chapter 3 describes modified version of the Decision Trees algorithm that is able to deal with time series data.
- Chapter 4 introduces our first contribution: Multi-operator Temporal Decision Trees.
- Chapter 5 provides an approximated algorithm on Multi-operator Temporal Decision Trees.
- Chapter 6 introduces techniques to enhance discriminative and explanatory characteristics of time series by weighting important timestamps.

Each chapter about our research contribution is completed by empirical study and results.

Chapter 2

State-of-the-art

Contents

In this chapter we present the state-of-the-art of the most popular algorithms that have been proposed for Time Series Classification (TSC) problem. In Section 2.2 we discuss the problem of interpretability in Machine Learning (ML) algorithms. Then, in the Section 2.3 we present different techniques and methods on time series representation. In forthcoming Section 2.4 we provide an overview on TSC algorithms that were designed or modified to handle time series data. They employ one or another time series representation for yield classification models and can be separated by two categories: classical and ensemble ML methods. To this end, we come back to the question about interpretability of ML algorithm, however in the scope of TSC task.

2.1 Introduction

Nontriviality of TSC task and its vast range of applications has attracted ML researchers' interest over recent years. However, many researchers from other disciplines, such as statistics and engineering, have been approached resolving this problem earlier. Hence, lots of work that has been done is rather applicative and does not provide us with general algorithms and techniques. ML algorithms are not specified to the particular application domain, but classical classification algorithms are not capable of fitting complex data with time dependant structure. The undeniable importance of a need in time series mining counting as well classification methods was risen in the work of Keogh and Kasetty [42]. It is one of the first studies where the lack of methods able to mine time series data was pointed out. In particular, for classification task the main contributions had focused on developing new similarity measures for k-nearest neighbour classifier [22, 83, 88]. Later on, lots of novel techniques were proposed. In the survey done by Xing et al. [88] the classification methods were reviewed for sequential data including time series. Esling and Agón [25] and Fu [29] provide an overview of mining and classification algorithms for time series. Based on the current state of TSC research, the algorithms developed to classify time series data can be divided into following categories:

1. TSC methods that require to represent time series data by a set of explicit features. Hence, these algorithms include an additional step to transform raw time series data into an appropriate input. Depending on how the data is represented, these algorithms fall into categories:

    - **Similarity-based algorithms**, as the name suggests, use the notion of similarity to make classification of an unlabelled example. That is to say, similarity comparison between time series of each class should be performed in order to assign a class label. A similarity measure has to be defined to compare time series between each other. These algorithms are able to deal with either whole time series or its subintervals. The classification quality is extremely dependant on the choice of similarity measure.
    - **Pattern-based algorithms** are focused on extraction of patterns from time series. To do so, modification and transformation techniques are applied to raw time series to find meaningful patterns. Discovered dictionary of patterns is a representation of time series data. This can be fed as input to any conventional ML classifier. As a consequence of data modification,

the algorithms in this category can lose the class discriminative information and be less interpretable.

2. TSC end-to-end methods, which do not require any modification on input time series data. This category includes deep neural network algorithms and, in particular, Convolutional Neural Network (CNN). Basically, features extraction is embedded in network learning process. These algorithms lie beyond the scope of this research. Even though, we will not focus our attention on them, one can find more details and additional references in the paper of Wang et al. [84].

TSC algorithms can be combined forming *Ensemble TSC learning* in order to boost classification accuracy performance. Typically, the flip side of good accuracy is the lack of interpretability. It becomes difficult or even sometimes impossible (especially when the combination of classifier is of heterogeneous nature [6]) to analyze results of model that have complex structure.

2.2 Interpretability, explainability and transparency in machine learning

Before looking at the details of the TSC algorithms, we discuss the problem of interpretable and comprehensible ML. This question is arised regardless of ML method or application, it is rather a general problem. It also becomes absolutely crucial when an end-user of model is non ML expert. If during a learning process an algorithm loses a link between observed data and obtained outcomes, the trust and understanding of a domain practitioner is lost too. Recently, many workshops on interpretability in ML has been organised at the conferences such as NIPS, ICML, IJCAI, etc. This emphasises not only the interest among the research community to this topic, but also the need of studying this problem.

There are many notions that are usually employed in the literature about ML methods to describe how a model works and what insights about the nature of the problem we can get: interpretability, explainability and explanation, readability and transparency of learning models. These terms are highly ambigious [57] and especially it becomes challenging when we would like to formalize them as properties of ML algorithms. Even though, there are lots of work in psychology and cognitive sciences that discuss their definition and difference, there are much less work done in giving strict definition of these terms when applied to ML algorithms. Miller [60] provides general discussions about how to transfer these notions from philosophy and cognitive

sciences to artificial intelligence (AI) and ML. In the work of Lipton [57], the authors narrow down the problem and attempt to define properties of interpretable ML algorithms. They divide them into two categories: the first relates to transparency of the built model, and the second determines the post-hoc explanations that the model can provide.

Let us define what each of the above-mentioned terms means and how we can apply them to an ML algorithm in order to improve their understanding for users.

- **Readability.** This term points to the representation of a model and addresses the following two questions: What steps of a training process can be easily shown to a user? Can a built model be visualized? Positive answers to these questions give an evidence that the yielded model is structured. It can be explored by a user in order to focus on relevant classification attributes. Decision trees of reasonable size can be taken as a good example of readable model.

- **Interpretability.** The term of interpretability appeals to the idea of assessment the degree to which a domain expert can understand how a model works and how classification decisions are made. For example, in classification task we say that a model is interpretable if steps of classification process made by obtained model are clear for a user.

  Lipton [57] in his work argued about the term ambiguity. He states that for each new developed ML algorithm, one must define what sense of interpretability it possesses. Though, in general we say that a model is interpretable, if it is able to reveal the important information about data which a domain expert can easily understand.

- **Explainability and explanation.** In cognitive sciences, these two terms refer to a cause of a problem we are trying to solve [60]. The questions like *What happened and why?* are of interest when humans are seeking for understanding. An ML algorithm that is capable of giving explicit explanations to a user about learning process and results can be considered as explainable. Certainly, most ML algorithms with their innate complexity of techniques (transformations of original data to other spaces, dimensionality reduction, etc) remain black-boxes for a domain expert. Ribeiro et al. [71] proposed a technique to explain the outcomes of any classifier to a user. Such approach is post-hoc explanations [57] that gives practical information about the classification results. For example, a model can be supported by visualisation of data and/or their representation.

Also, taken classification decisions can be annotated by provided additionally text explanations. However, Miller [60] states that the difference between interpretability and explainability is subtle and it is possible to equate these notions.

- **Transparency.** This term rather refers to the properties of an ML system that provides a model that aims to be readable, interpretable and explainable to its users. Infusion of the notion of *transparency* as characterized property of an ML algorithm is the goal of making a model more understandable.

Speaking about an ML system designed to be transparent for its end-user, we note that it has to involve at least two modes: data preprocessing mode and ML algorithms mode. Each of them can be described in more details as follows:

1. the first mode allows analyzing raw materials, i.e. non-processed data, to get a conceptual understanding of their properties. The importance of this step is clear. By using the knowledge about a field, a domain expert can make the initial hypothesis about the intrinsic regularities the data contains. Then, by looking at the dictionary of patterns, they receive the very first information about discriminating time ranges in time series. And already at this stage, they can correct the prior beliefs about the data.

2. the second mode allows exploring and understanding a model generated by ML tools. The results reveal captured underlying insights from the data. A user must interpret these recommendations about the data with respect to their own knowledge and validate the different parts of the model. This allows an expert to set up some feedback to change the data description or learning parameters.

These models can be served as the fundamental core of any ML system, though, surely, can be extended.

In the forthcoming section we give an overview of the state-of-the-art of TSC algorithms and at the end of the chapter we discuss again the interpretability problem, but in the context of time series classification task.

2.3 Time series data representation

In order to modify classical ML algorithms or design a new one for TSC task, one needs to define how to represent time series data. Due to the time component of time

series, which brings an additional level of complexity, there are many approaches and techniques that have been proposed. Hereafter, we attempt to describe and summarize them in the following three categories:

- preprocessing and discretization methods, which are aimed to compress data and/or get rid of the noise they may contain;
- similarity-based methods, which represent time series as points in high-dimensional space with defined distance between them;
- pattern-based methods, which target to transform time series data in order to obtain their static description.

2.3.1 Preprocessing and discretization methods

Oftentimes, before applying any of data representation tecnhiques, time series has to be preprocessed and/or discretized [25]. The aim of this process is to clean the raw data and/or to find their compressed decomposition by basis functions.

Time series preprocessing. Preprocessing step is mostly needed to clean data [61], for instance when raw time series contain missing values, and/or the scales of values are different, and/or their disturbed by noise occurence. Missing values can be handled by many techniques such as *linear, spline or cubic interpolation*. To have time series values in the range between 0 and 1, the normalization techniques can be applied, such as *min-max* or *z-score* normalization. The latter is particularly useful when the data contains outlier values. *Moving-average smoothing* is the technique allowing to remove noise that can occur in time series data. Sometimes it is required to have time series of the same length, for example, Euclidean distance computation between two time series. To obtain the same length resampling and down-sampling can be done [25, 42].

Time series discretization. High-dimensionality of time series data is well known problem [25, 29, 46]. The goal is to compress raw data in the meaningful way, i.e., minimizing loss of class discriminative information that original data contains. There are methods that reduce data dimension by discretizing it into frequency domain, into frequency and space, or into symbolic strings. Below, we overview briefly some widely used techniques. More detailed description and other methods as well can be found in [18, 25, 30, 61].

- **Discrete Fourier Transfrom (DFT)** decomposes time series by sequence of frequencies represented by complex numbers [4], each of which is computed using the following formula:

$$f_t = \frac{1}{\sqrt{m}} \sum_{n=1}^{m-1} v^n \exp\left(-\frac{2\pi j}{m} tn\right), t \in 1, \cdots, m, \tag{2.1}$$

where $j = \sqrt{-1}$ is imaginary unit. Typically, only the first few lowest frequencies are kept and the rest are discarded. There exist the Fast Fourier Transform [26] algorithm to compute DFT coefficients in $O(m \log m)$ time.

- **Discrete Wavelet Transform (DWT)** represents time series into multi-resolutional way(frequency and space domain) [1]. *Haar* wavelets transform is widely used to discretize time series providing a good approximation. From computational point of view, it requires linear time $O(m)$ with respect to the length of time series. The Haar wavelet function is defined as follows:

$$\psi(t) = \begin{cases} 1 & 0 < t < 0.5 \\ -1 & 0.5 < t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

with the scaling function $\phi(t) = \begin{cases} 1 & t \in (0,1) \\ 0 & \text{otherwise} \end{cases}$

- **Piecewise Aggregate Approximation (PAA)** decomposes time series into predefined number of equal-length segments, each of which being replaced by the average of values within the segment [46]. PAA representation of time series is a competetive technique of two approaches described above. From computational perspective it takes only linear time $O(m)$.

- **Singular Value Decomposition (SVD)** was proposed to reduce dimensionality of time series by finding its principal components [47]. Each time series $ts \in \mathbf{TS}$ is mapped to trajectory matrix composed by lagged vectors. It forms a matrix $\mathsf{TS}_{l \times (m-l+1)}$ to which SVD is applied:

$$\mathsf{TS} = U \times S \times V^T, \tag{2.3}$$

where U is a column orthonormal $l \times r$ matrix (r is the rank of TS), S is a diagonal $r \times r$ matrix, and V^T is a column-orthonormal matrix of size $(m - l + 1) \times r$. This techniques is computationally more expensive, which amounts to $O(\min(l^2(m - l + 1), (m - l + 1)^2 l))$.

- **Symbolic Aggregate ApproXimation (SAX)** discretize time series into symbolic strings [53]. The transformation configured by two parameters: a desired word size w and an alphabet size a. SAX produces a symbolic approximation of time series $ts \in \mathbf{TS}$ of length m by compressing it into a string of length w, whose letters are taken from the alphabet a. The steps of transformations are as follows:

  1. Reduce dimensionality from m to w piecewise aggregate approximation (PAA). All time series are z-score normalized. PAA divides a time series into w segments and mean values of points within each segment are computed. The sequence of these values forms PAA approximation of ts.
  2. Discretize along amplitude axis with chosen alphabet a. A set of breakpoints are computed to divide the normalized time series values distribution space into the $|a|$ equal-sized and -probable regions. Then, each of PAA values is converted into a letter of an alphabet a using the lookup table.
  3. Produce a symbolic word that represents time series.

  The complexity of the algorithm is linear $O(m)$ with respect to the length of time series.

2.3.2 Similarity-based methods

Similarity-based TSC algorithms use a similarity function to measure the proximity between a pair of time series. To measure how similar two time series are, we need to define a similarity measure. The choice of similarity measure is very important to produce high performance classification. Also in this section, we provide the definition of metric and its properties. We review some time series similarity measures and point out the time series characteristics that they are capable of capturing. Then, we describe the classification algorithms that are based on these similarities or could be modified to use them.

2.3.2.1 Time series metrics.

One property of any developed similarity measure is metricity. A distance d is called to be a metric if it satisfies the following properties. Let x, y and z be three time-series.

- Reflexivity $d(x, x) = 0$
- Nonnegativity $d(x, y) \geq 0$

- Symmetry $d(x, y) = d(y, x)$
- Triangle inequality $d(y, z) \leq d(x, y) + d(x, z)$

Metricity allows the use of speed methods and techniques that have been designed to facilitate nearest-neighbour search [38]. Time series similarity measures can include or not an alignment of time stamps between two time series. If measurements are not shifted along time dimension, we can speak about *static alignment*. That means each observation v_i^t of a time series ts_i measured at time stamp t corresponds to observation v_j^t of ts_j, where i and j states for indicator of time series. If we cannot state this and measurements are shifted along time axis, a *dynamic alignment* is required to correctly measure the similarity.

Let $ts_i = \{v_i^1, \ldots, v_i^m\}$ and $ts_j = \{v_j^1, \ldots, v_j^{m'}\}$ be two times series of length m and m' respectively. To align two sequences, we construct an m-by-m' matrix where each entity of the matrix contains the distance (commonly l_1 or l_2) between two time series values at given timestamp t.

A warping path $\boldsymbol{\pi}$ is an alignment between two time series ts_i and ts_j is defined as a sequence of s pairs $\boldsymbol{\pi} = \pi_1, \ldots, \pi_s$. The l^{th} element of a path $\pi_l = (t, t')_l$ is a mapping, where $t \in 1, m$ and $t' \in 1, m'$ are indicators of ts_i and ts_j timestamps respectively.

A valid warping path has to satisfy the following constraints:

- Boundary condition: $\pi_1 = (1, 1), \pi_s = (m, m')$. This condition states that a path has to start and finish in diagonally opposite corner cells of the matrix.
- Monotonicity condition: $\pi_{l+1} \in ((t + 1), t')_{l+1}, (t, (t' + 1))_{l+1}, ((t + 1), (t' + 1))_{l+1}$. This ensures that timestamps in $\boldsymbol{\pi}$ are monotonically spaced in time.
- Step size condition: it requires unitary increments. Allowable steps in a path are adjacent cells.

There exist many possible alignments between two times series that satisfies the above conditions. The interest is to find an optimal path $\boldsymbol{\pi}^*$ which minimizes the cumulative distance value between two time series called *warping cost*. Distance measures that define an alignment path π are called elastic due to their ability of matching two time series points located at different timestamps.

We divide similarity measures into three groups depending on which attributes they are able to capture: value-based, shape-based, and value-shape-based measures. Figure 2.1 shows a graphical example of three time series that are similar with respect

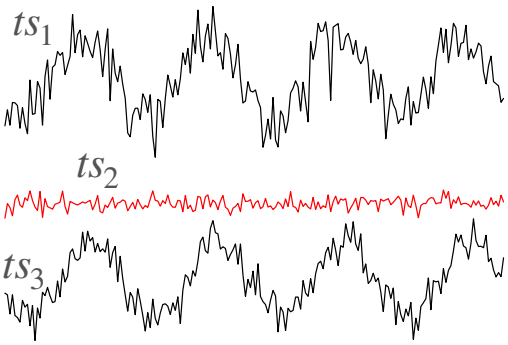

Figure 2.1: Example of three time series to illustrate different types of similarity. Time series ts_3 is more closer to a time series ts_2 with respect to amplitude of values, while it is more similar to ts_1 with respect to shape.

to different domains. Time series ts_3 is more closer to a time series ts_2 with respect to amplitude of values, while it is more similar to ts_1 with respect to shape.

2.3.2.2 Value based similarity measures

Value-based measures focus on the difference of time series amplitudes at each time stamp.

The l_p distances. The l_p distances are the simplest measures of similarities between time series. Given two time series ts_i and ts_j of equal length m, the l_p is defined as the p normed sum of differences between corresponding values of each time-series:

$$\mathrm{l}_p(ts_i, ts_j) = \Big(\sum_{t=1}^{m} |v_i^t - v_j^t|^p \Big)^{\frac{1}{p}} \tag{2.4}$$

The case where the value $p = 1$ corresponds to the Manhattan distance. The Euclidean distance is the l_p norm with $p = 2$, which is the one of the most popular norms for comparing time series [42]. However, Aggarwal et al. [3] has proved that using distances with lower values of p is qualitatively better for high-dimensional data. Thus, for comparing long time series one prefers using the lower value of p. The accuracy performance of a classifier that employs this distance decreases with increasing the value of norm parameter p. There are mainly two types of limitations for this family of distances. l_p distances are sensitive to time shifting, hence it requires a static alignment of time series. Another bottleneck is that time series must have the same length.

Dynamic Time Warping. In practice, oftentimes, time series do not line up in time because of possible poor quality of their recording process. Another reason is that events caracterizing the same class can occur at different time periods, hence, while computing distance these events have to be aligned. To alleviate these problems, DTW was introduced as an elastic measure that tolerates distortion in time dimension [11]. The DTW between two time-series ts_i and ts_j is defined as

$$\mathrm{DTW}(ts_i, ts_j) = \min_{\boldsymbol{\pi}} \Big(\sum_{l=1}^{s} |v_i - v_j|_{\pi_l} \Big). \tag{2.5}$$

DTW aligns two time series and optimize a path by minimazing the distance through an alignment. The DTW optimal path is computed using dynamic programming. The optimal path $\boldsymbol{\pi}^*$ and distance value can be found using dynamic programming as follows:

$$\tilde{d}^{(t,t')} = d^{(t,t')} + \min \Big(\tilde{d}^{(t-1,t'-1)}, \tilde{d}^{(t-1,t')}, \tilde{d}^{(t,t'-1)} \Big),$$

where $\tilde{d}^{(t,t')}$ indicates cumulative distance at time point (t_i, t_j). The cumulative distance is augmented by the minimum of cumulative distances of the adjacent elements. DTW distance is widely used when we need to cope with the problem of time shifting and does not require time series to be of the same length [43].

Although DTW has been successfully used in many domains, due to its not limited elasticity it sometime can output incorrect alignment. For example, a single time stamp on one time series maps into a large timestamp subset of another time series [44]. Even having a large lag between points that have to be aligned, it is unlikely to shift very far from the diagonal of pairwise distance matrix. To improve performance and to accelerate the computation, the constraints on a warping band have been used. The classical approaches [39, 74] limit the warping band to a warping window of size l, directly above and to the right of the diagonal. Ratanamahatana and Keogh [69] proposed constraints based on the shape of warping path.

Weighted Dynamic Time Warping. DTW computes the distance by adding the absolute difference of values along uniformly a warping path, i.e., regardless of the phase difference of matched points. Jeong et al. [40] introduced the penalty-based DTW. When searching an optimal alignment $\boldsymbol{\pi}^*$, WDTW penalizes each value of distance between points in the warping path with high phase difference by a multiplicative weight $w_{|t-t'|}$, where t and t' are timestamps of time series ts_i and ts_j respectively. A larger weight is imposed to mappings with a higher phase difference. Weights are

monotonically increasing for further points. The difference between two points v_t and $v_{t'}$ is calculated as

$$\delta_w(v_i^t, v_j^{t'}) = ||w_{|t-t'|}(v_i^t - v_j^{t'})||_p. \tag{2.6}$$

The distance between two time series is defined by minimizing an alignment path over all possible paths

$$\text{WDTW}(ts_i, ts_j) = \delta_w(v_i^t, v_j^{t'}) + \min(\delta_w(v_i^{(t-1)}, v_j^{(t'-1)}), \delta_w(v_i^{(t-1)}, v_j^{t'}), \delta_w(v_i^t, v_j^{(t'-1)}))$$

To assign weights, the authors proposed to use a modified logistic weights function. Imposed weight is defined as

$$w_t = \frac{w_{max}}{1 + \exp(-\lambda(t - \frac{m}{2}))},$$

where t is a timestamp of a time series of length m, w_{max} is the chosen upper bound of a weight parameter (set to 1 in the paper Jeong et al. [40]) and the constant $\lambda \in [0, +\infty)$ controls the level of phase difference penalization.

Complexity-Invariant Distance. This distance was proposed by Batista et al. [8]. In the paper, they reviewed different types of time series attributes (called invariances) that can be distorted during recording and ways how it can be mitigated. A new type of invariance has been introduced in the paper, which is intuitively based on the number of peaks, valleys and patterns. Basically, defined complexity in the paper tells us how complex an object represented by a time series is. The complexity factor for each time series can be computed as follows:

$$CF(ts) = \sum_{t=1}^{m}(v^{t+1} - v^t)^2 \tag{2.7}$$

The idea behind the formula is to measure the length of a time series if it would be stretched to the straight line. There are many other proposed variations on how the complexity of time series can be measured. The authors proposed a variant of complexity measure that is intuitive and has low computational cost.

Employment of computed complexity to a Euclidean distance computation is straightforward and defined as

$$l_2^{CF}(ts_i, ts_j) = CF(ts_i, ts_j) \times l_2 \tag{2.8}$$

where the complexity factor between two time series is defined as

$$CF(ts_i, ts_j) = \frac{\max\left(CF(ts_i)CF(ts_j)\right)}{\min\left(CF(ts_i)CF(ts_j)\right)} \tag{2.9}$$

It corrects a distance value between two time series by multiplying it by the complexity factor. The authors of [8] empirically showed that in the same way it also can be applied to DTW. Experimental results have shown positive impact on classification accuracy because of its ability to correct similarity by avoiding matching complex time series to a simpler one.

We finish the description of value-based measures with two empirical observations on comparison of efficiency of Euclidean and DTW distances. Xi et al. [87] has shown that on small datasets, elastic measures (DTW) are more efficient in terms of accuracy than Euclidean distance. Whereas Ding et al. [21] empirically proved that on large datasets euclidean distance is almost as accurate as elastic. These results can serve as the guideline for a user which similarity function to select based on a dataset under analysis.

2.3.2.3 Shape-based similarity measures.

When discriminativeness of time series belonging to different classes is defined by their overall shapes rather than difference in amplitudes, the value-based metrics cannot serve anymore as an efficient time series comparison method. Distance measures aiming to capture changes in shape of time series, typically, include in their computation differences between adjacent values of each time series. So, a simple method is to first compute time series derivatives and, then, apply one of the value-based metrics. The derivative shows what happens in the neighbourhood of a given timestamp t of an observation $ts \in \mathbf{TS}$.

Derivative Dynamic Time Warping. A derivative version of DTW was introduced by Keogh and Pazzani [44]. The distance called as DDTW is able to alleviate the problem of incorrect matching by considering local time series derivatives instead of using raw data. It appears when a single time stamp on one series may map into multiple timestamps of another series. The algorithm transforms time series by computing first order differences. Given $ts_i = \{v_i^1, \ldots, v_j^m\}$ the modified version of a time series is $\tilde{ts}_i = \{\tilde{v_i}^1, \ldots, \tilde{v_i}^n\}$ where each value $\tilde{v_i}^t$ is computed as

$$\tilde{v}_{it} = \frac{((v_i^t - v_i^{(t-1)}) + (v_i^{(t+1)} - v_{(t-1)}))/2}{2}, 1 < t < m. \tag{2.10}$$

This estimation of derivative takes into account three consecutive timestamps and computes an average of the slope between v_i^t and v_i^{t-1} and the slope between $v_i^{(t-1)}$ and $v_i^{(t+1)}$. The similarity between two time series computed in the same way as standard DTW. Using DDTW Luan et al. [58], Mokhtar et al. [62] showed good performance in practical applications and more robust to ouliers. .

Cort. Chouakria and Amblard [16] proposed Cort distance to measure similarity between time series capturing shape characteristics by computing the correlation of its shape changing. Two time series are similar if the growth rate of its co-increasing or co-decreasing behaviour is the same. Cort distance is defined as modification of the Pearson correlation that keeps temporal order. Cort can be formulated as static distance as well as elastic with a dynamic alignment path.

$$\mathrm{Cort}(ts_i, ts_j) = \frac{\sum_{t=1}^{m}(v_i^{t+1} - v_i^t)(v_j^{(t+1)} - v_j^t)}{\sqrt{\sum_t (v_i^{t+1} - v_i^t)^2}\sqrt{\sum_t (v_j^{(t+1)} - v_j^t)^2}}. \tag{2.11}$$

$$\mathrm{Cort}_\pi(ts_i, ts_j) = \min_{\boldsymbol{\pi}}\Big(\frac{\sum_{l=1}^{s}(xy)_{\pi_l}}{\sqrt{\sum_{l=1}^{s} x_{\pi_l}^2}\sqrt{\sum_{l_1}^{s} y_{\pi_l}^2}}\Big), \tag{2.12}$$

where $\pi_l = (t, t')_l$, $x_{\pi_l} = v_i^{(t+1)} - v_i^{t'}$, $y_{\pi_l} = v_j^{(t+1)} - v_j^{t'}$. Equation 2.11 is applicable for time series of the same length and without discord in time axis. Whereas, Equation 2.12 is able to compensate both problems with a dynamic alignment path. The value of Cort belongs to $[-1, 1]$, where values 1 and -1 indicate similar and opposite shape between two time series respectively. The value of 0 means two time series are neither similar nor opposite shape.

2.3.2.4 Value-shape-based similarity measures

In practice, we cannot expect that it is always sufficient to use just shape-based or value-based metrics to compare time series. For example, for multiclass classification it can be the case that one pair of classes is discriminated by difference in amplitude values, while another distinct pair of classes is separable by time series shape behaviour. However, employing only value- or only shape-based similarity function within a classifier will not be capable of carring high performance classification. The value-shape similarity measures were designed to consider values of time series and its general shape. Oftentimes, to regulate intensity of each component for different datasets, these distances are parametrized.

DTW-Cort. Introduced in Chouakria and Amblard [16], this similarity measure aims to cover both amplitude and shape attributes.

$$\text{DTW-Cort}_{\pi}(ts_i, ts_j) = \min_{\boldsymbol{\pi}}\Big(\frac{2}{(1+\exp\left(\lambda * \text{Cort}\right))}\sum_{l=1}^{s}|v_i - v_j|_{\pi_l}\Big) \tag{2.13}$$

The parameter $\lambda \in [0, 6]$ modulates the importance of value ($\lambda = 0$) and shape ($\lambda = 6$) components.

Distance based on derivative. The similarity function proposed by Górecki and Luczak [32] is based on the mixture of value- and derivative-based distances. They state that the measure is capable of tuning the derivative component for each dataset during a model training. Computationally, the method does not have overhead upon other elastic distance measure. The derivative-based distance measure (DD) is defined as

$$DD(ts_i, ts_j) := \alpha * d(ts_i, ts_j) + \beta * d(\tilde{ts_i}, \tilde{ts_j}), \tag{2.14}$$

where d is the base distance function. In Górecki and Luczak [32] they propose using Euclidean as the most straightforward measure and DTW distances as the most effective. The parameters $\alpha, \beta \in [0, 1]$ are chosen in the learning phase through cross-validation technique. The time series $\tilde{ts_i}, \tilde{ts_j}$ are first order derivatives of time series ts_i, ts_j correspondingly. To estimate derivatives of a time series, the authors use value difference of two consecutive time stamps defined as $\tilde{v_i^t} = v_i^{t+1} - v_i^{(t)}, i \in [1, m-1]$. Though, they picked two points derivative formula, experiments include the results with employment three point derivative formula. However, they did not concluded that the final distance DD is sensitive to the choice of derivative formula. Note, that in the formula it is possible to use combination of different functions for distance computation between time series $\text{d}(ts_i, ts_j)$ and between derivatives $\text{d}(\tilde{ts_i}, \tilde{ts_j})$. The extension of the distance for the second derivative was proposed in [33].

Time Warp Edit Distances. TWED is an elastic function of similarity proposed by Marteau [59]. It satisfies metric properties in contrast to previously mentioned time series similarity measures. The proximity between two time series is measured as the minimum cost sequence of "edit operations" needed to transform one time series into another. It controls stiffness along the time axis by a warping parameter $\lambda > 0$. Unlike a warping window in DTW, which also limits an alignment by using timestamp

differences threshold, it enforces penalty on the distance between matched points. Setting λ to ∞ results to have the Euclidean distance, while fixing $\lambda = 0$ give us original DTW distance with no stiffness at all. TWED uses elementary operations called del_{ts_i}, del_{ts_j}, *match*. While searching for an alignment, match operation corresponds to the time axis direction with simultanious increment of timestamps t and t' of time series ts_i and ts_j by one step $(t + 1 \leftarrow t' + 1)$. Applying del_{ts_i} will increment only t, accordingly, the new mapping will be $(t + 1 \leftarrow t')$ and for del_{ts_j} it is vice versa $(t \leftarrow t' + 1)$. Once a mapping is found, an accumulated distance is augmented by l_p with spatial and temporal penalties. Computation of TWED distance can be done through dynamic programming [59].

Move-Split-Merge Metric. Move-Split-Merge Metric (MSM) proposed in [82] is a distance that measures the cost of transformation of one time series into another. MSM uses the set of basic operations that can transform any time series to any other time series. It has three fundamental operations, which are employed to perform a transformation: Move, Split and Merge. *Move* changes the values of a single element, *Split* converts a single element into two consecutive elements, and *Merge* joins two consecutive element into one. Each operation has an associated cost and MSM distance between two time series ts_i and ts_j is defined as the cost of the cheapest sequence of operations that transforms a time series ts_i into a second one ts_j. The foremost motivation in formulating MSM is to satisfy a single distance measure using a set of certain desirable properties: robustness to temporal misalignments, metricity, quadratic time complexity, translation invariance and treating all values equally.

2.3.2.5 Similarity function learning

As we have seen, there are lots of measures that have been proposed to quantify the similarity between pairs of time series. They capture different time series class discriminative characteristics. Table 2.1 reveals their arrangement according measured time series properties.

Selection of proper similarity measure is a fundamental issue. Moreover, in general there is no such thing as 'best' similarity measure, it directly depends on the data nature and questions being posed. On one hand, lack of prior knowledge may lead to an inappropriate measure for a given dataset. On other hand, even if we have some prior knowledge about data attributes, which are important to discriminate classes, employing only one of them may be uncapable of faithfully capturing all important information that data contains.

Table 2.1: Similarity measures arranged by their categories and by the type of time stamps alignment.

Alignment	Value	Shape	Value-Shape
Static	l_p, CID	Cort	l_p-Cort
Dynamic	DTW, WDTW	DD, Cort$_\pi$	DTW-Cort$_\pi$, TWED, MSM

An alternative to mitigate the problem of the similarity measure selection is to learn a combination of distances capturing different modalities presented in data. Do et al. [22] proposed an algorithm of Multi-modal and Multi-scale Temporal Metric Learning (M^2TML). The learnt measure is a weighted combination of metrics that capture amplitude, shape, and frequency charachteristics in the data (Multi-modal) on different time intervals (Multi-Scale). Metric learning is done by SVM algorithm in dissimilarity space where training time series have been transformed. Each point in new representation space is a high dimensional vector, each entity is dissimilarity between a pair of time series (ts_i, ts_j) given a dissimilarity measure, d_c^I, where c denotes category of measure (amplitude, shape, frequency) and $I \in [0, m]$ is a time series segment. Hence, each time series ts_i forms $N-1$ points, where N is the size of training set. The authors proposed to shorten the training set in a new space by selecting only $k\alpha, \alpha > 1, \alpha \in \mathbb{N}$ nearest points of the same class ($y_i = y_j$) and the same number of nearest point of distinct class labels ($y_i \neq y_j$)). The parameter k defines the number scaled by α of nearest neighbour points.

New meta metric d^{new} is computed by performing binary SVM in dissimilarity space. To satisfy desirable metric properties of positivity, symmetry, reflexivity and local order ($d^{new}(ts_i, ts_j) < D(ts_i, ts_r), \forall y_i = y_j, y_i \neq y_r$) the authors suggest using nonlinear combination of learnt solution with exponential term. To conduct classification of new time series, the learnt distance is combined with 1-NN classifier.

The clear advantage of metric learning is the ability to automatically assign weights to the top discriminative measures and its accuracy performance in comparison with uni-model distances. The flip side of this approach is that one needs to define which distances to include in the representation vector and how many segments of a time series should be considered.

2.3.3 Feature-based methods

The second category of algorithms for TSC problem involves some preprocessing steps of deriving robust representation for time series. In this section, we provide

an overview of the latest algorithms that discover explicit discriminatory features of varying size time series intervals.

The goal of the transformation process is to generate a set of features minimazing loss of discriminative information from the original data. Large number of high level representations [31, 64, 72] have been proposed by the research community. Still, the task of finding interpretable and discriminative patterns is complex, and by using any transformation in a classfier, we increase the risk of loosing insightful connection between data and classification results. An overview of different feature extraction approaches have been done by Fulcher [30] and Daw et al. [18]. In this section, we describe some widely used feature extraction methods which are categorized in three groups: statistical, dictionary, and shapelets approaches. Most of them build upon discretized time series by one of the method mentioned above.

2.3.3.1 Statistical methods

BOSS: Bag-Of-SFA-Symbols. One of feature-based TSC algorithms BOSS proposed by Schäfer [78] which uses sliding window to form words over time series. BOSS algorithm uses a Discrete Fourier Transform (DFT) to discretize time series into frequency domain. The obtained segments are quantized along amplitude axis through a technique called Multiple Coefficient Binning (MCB), rather than using fixed intervals. MCB finds the disretizing break points as a preprocessing step by estimating the distribution of the Fourier coefficients. Oftentimes, adjacent sliding window can have the same Symbolic Fourier Approximation (SFA) words, thus only the first occurence of a SFA word is counted to reduce words numerosity. Though, employing different pattern extraction techniques, BOSS involves similar stages as BOP and SAX-VSM algorithms. For each class, it computes tf-idf vector representation. Cosine similarity metric with 1-NN classifier combination is used to assign a class label for a new time series.

BOP, SAX-VSM, and BOSS can perform noise reduction and are efficient for long time series. Similar to SAX-VSM, BOSS uses representation over classes instead of time series. While outperforming in accuracy its rival methods (BOP and SAX-SVM) it has the same limitation when it comes to conceptual understanding of classification results.

TSBF: Time series Bag of Features. Time series Bag of Features (TSBF) is another approach that represents time series through statistical features on intervals. The algorithm was proposed by Baydogan et al. [9] and includes random intervals

step generation and the creation of statistical features representation. Over each interval of time series, it calculates the mean, variance and slope. The start and end point of the interval are also included as features in order to retain the possibility of detecting temporal similarity. To this end, the new input set is a set of generated subsequences and their features. Then, SVM classifier is applied to get a table of class probability estimates for each subsequence. Next, for each time series the algorithm builds a histogram of obtained estimates over each class. The resulting histograms concatenated for all training time series form the feature space.

Although the algorithm is capable of capturing global and local characteristics of time series by employing heterogeneous set of features, it evidently lacks interpretability. For a domain expert as well as an ML expert, reconstructing a link between results of classification and original data is challenging.

IF: Interval-based Features. Deng et al. [19] proposed statistical approach to extract a set of interval features to capture temporal characteristics of time series. For a time series $ts = (v_1, \ldots, v_m)$ and interval $[t_{start}, t_{end}]$ three summary statistics are computed as follows:

$$f_\mu = \frac{\sum_{i=t_{start}}^{t_{end}} v_i}{t_{end} - t_{start} + 1}; \tag{2.15}$$

$$f_\sigma = \sqrt{\frac{\sum_{i=t_{start}}^{t_{end}} (v_i - f_\mu)^2}{t_{end} - t_{start}}}; \tag{2.16}$$

$$f_\beta = \beta, \tag{2.17}$$

where β is the regression line slope of the time series segment between t_{start} and t_{end}. Each time series is represented by the set of these three values computed at each interval.

2.3.3.2 Pattern-based methods

BOP: Bag-of-patterns. Lin et al. [54] proposed feature-based time series representation based on SAX transformation. The bag-of-patterns approach first converts a time series into a symbolic set of series, each of which is extracted from a time series segment obtained by a sliding window (SAX step). Then, a numerical feature set is collected via a frequency count of extracted words in a series.

Bag-of-patterns [54] is similar to an Information Retrieval (IR) "bag of words" concept where each "bag" is represented by a set of symbolic strings (SAX words) that

a time series contains and its occurence frequency vector. These vectors are classified with 1-NN classifier built with Euclidean distance or Cosine similarity applied to raw frequencies or their *tf*idf* weighting. The BOP has several advantages: it has linear complexity in terms of the number of time series, it is rotation-invariant. Representing each time series as a bag of SAX words allows considering local and global structures simultaneously.

SAX-VSM: SAX Vector Space Model. SAX-VSM algorithm proposed by Senin and Malinchik [79] is a generalization BOP that forms word distributions over classes rather than series and weights. Using a sliding window technique, each time series of a training set is converted into bags of SAX words. All retrieved words placed in unordered collection that is a representation of the original time series ts. Then, instead of building N bag of words for each of the training series, the algorithm builds a single bag of words for each of the classes, that effectively provides a compact solution of K weight vectors, where K is the number of classes (typically $K << N$). To achieve this, tf-*idf* weighting [75] is used.

To perform classification, two steps have to be done. First, unlabeled time series, taken from a test set, are transformed into a SAX words frequency vector by the same procedure and with the same parameters that were selected on the training step. Second, cosine similarity is computed between the representative vector of a test series and K tf-*idf* weight vectors and, then, the class with the maximum similarity is assigned to a test time series.

Among feature based classifiers, SAX-SVM provides more interpretable insights on multiclass classification since it builds weights over classes. Hence, each classified time series is rather associated with an assigned class rather than with a time series and its class from the training set.

SVM-DTW: DTW featured Support Vector Machines. Different approaches have been proposed in [41], their algorithm uses DTW pairwise distances to obtain the set of feature vectors and then incorporates them into SVM algorithm to classify time series data. The idea is to use the strength of DTW distance [88] to find good features of time series and then, employ the power of SVM algorithm to learn accurate classifier. The authors emphasize that getting good accuracy comes with good time series representation and strong classifier. Each time series is represented in terms of its DTW distances from each training series, and each distance is considered as a feature. In addition to features formed by DTW, the authors use Euclidean distance,

elasticity constrained DTW (DTW-p, where p states for penalty imposed on warping path), SAX time series representation and their various combinations. Total number of features is up to $k(N-1)$ for a dataset of size N, where k is the number of used time series transformations.

2.3.3.3 Segment-based methods

Sometimes not all time series timestamps are informative, and class discriminative information is conveyed by sub-intervals of the entire time series. Removing non-informative part of time series can improve classification accuracy. Therefore, in this case extraction of meaningful subsequences from raw time series becomes necessary.

LTP: Local Temporal Patterns. Geurts [31] proposed patterns extraction technique. Each pattern $\boldsymbol{p}$ represented by a segment of discretized time series and a distance (Euclidean) value threshold. For each time series $ts \in \mathbf{TS}$, we say that its associated pattern $\boldsymbol{p}$ of length l is detected on a timestamp t' if:

$$d(\boldsymbol{p}, ts) = \frac{1}{|\boldsymbol{p}|} \int_{t'-|\boldsymbol{p}|}^{t'} (\boldsymbol{p}(t) - ts(t))^2 dt \leq \theta, \tag{2.18}$$

where θ is a threshold of minimal allowed distance to the pattern $\boldsymbol{p}$. The number of split pattern candidates would be intractable if one consider the full training set. In order to overcome this, a set of referenced time series for each distinct class is subsampled.

Shapelets. *Time Series Shapelets* were proposed by Ye and Keogh [91]. They introduced an algorithm that finds the time series snapshot called shapelets which can represent a class. *Shapelet* is a class discriminative subsequence of time series. For binary classification problem, shapelets can be used to separate the training data into two parts according to a distance threshold. A distance value between a time series ts from the training set and a shapelet $\tilde{ts}, |\tilde{ts}| \ll m$ is computed as a distance between the best matching location of $|\tilde{ts}|$ in ts and $|\tilde{ts}|$. Once all pairwise distances are computed for a given shapelet, they are sorted, and a threshold value which provides the best data separation is defined as the optimal.

To find the set of shapeletes a brute force method is used. There are two major parameters of the algorithm: the number of shapelets and their length. It generates all subsequences from the time series training set. Each candidate from generated set is evaluated on its ability to separate the data by optimizing information gain.

Since there is a large number of candidates, the brute force search has high runtime complexity. To alleviate this, the authors proposed some techniques such as early abandoning of distance computations and entropy pruning of the information gain.

LS: Logical Shapelets. Due to interpretability and good accuracy performance, the research interest has increased and there are many studies that attempt to improve and develop the original method. [63] proposed *logical shapelets* as generalization of shapelets approach. The idea is to use conjunctions and disjunctions of shapelets in order to improve impurity of the data partition. First, the top k shapelets are extracted from time series. Second, they are combined by using either $\wedge$ or $\vee$ logical operations. To avoid overfitting, a single distance threshold is considered for all shapelets in a combination.

In order to make a partition for a given pair of shapeletes, one need to make an ordeline of pairwise distances. In fact, we have two distance vectors, one per shapelet in the combined pair. Authors proposed the following tecnhique to define an unique distance orderline: if conjunction was used then, the maximum of distances is taken; if shapelets were combined by disjunction, then the minimum is considered.

While this approach is able to provide purer separation of data, it has a flaw in data overfitting. It comes from the possibility to combine the number of shapelets equal to the number of classes. To overcome this problem, a hard bound on the cardinality of the set of shapelets is used in [63].

The crucial concern about shapelets use is that its discovery process is computationally expensive, namely $O(N^2m^3)$, which brings limitations for large datasets. To overcome this, Keogh and Rakthanmanon [45] proposed an extension of shapelet based decision tree to speed up its finding process. By using SAX, the dictionary of words is generated for each possible shapelet. By using random projection the dimensionality of the SAX dictionary is reduced. Once the k-best SAX words are selected they are mapped back to the original space of shapelets and used in the same way as in *Time Series Shapelets*. The algorithm preserves comprehensibility of the original algorithm while accelerates the training time. Random-based approaches to accelerate shapelet discovery process was proposed in [70, 86].

Shapelets learning. Novel perpective on discovering shapelets was proposed by Grabocka et al. [34] who formulated this task as an optimization of classification objective function. The solution provides a near-optimal top-k shapelets and avoids exhaustive search. The k shapelets are initialized through k-means clustering of all

candidates from a training data. The objective function for the optimization process is a logistic loss function with a regularization term for each class. The algorithm jointly learns the weights for the regression and the shapelets in a two-stage iterative process to produce a final logistic regression model.

To summarize, shapelets are good in capturing the local properties of time series and they are interpretable. Due to the requirement of input parameters, such as the number of top shapelets, there is a risk of loosing the important information from original data while it performs its search process.

2.4 Time series learning algorithms

With a representation of time series obtained by one of above mentioned methods, different ML algorithm can be used to perform the classification task. In this section, we provide the description of classical and ensemble method that were combined with transformed input set of time series.

2.4.1 Classical methods

The k-NN, Decision Trees and Support Vector Machines (SVM) algorithms are widely exploited for TSC in the conjunction with one or few similarity measures. Below, we are briefly explain these techniques.

k-Nearest Neighbour (k-NN) Classifier. One of the most commonly used algorithms in TSC task is the k-NN classifier [85, 90]. It is a simple, robust and accurate classifier, which depends on very few parameters and requires no training [76].

Given a training set of labelled time series $(\mathbf{TS}, \mathcal{Y})$ and a new series ts_{test} with an unknown class, the objective is to predict a class for ts_{test}. The algorithm finds k most similar time series in $\mathbf{TS}$ to ts_{test} according to pre-selected distance. Predicted class $y* \in \mathcal{Y}$ is the modal class of the k nearest neighbours. To avoid ties, the number of k is usually selected to be odd with respect to the number of classes. Nevertheless, in cases where there is no majority, ties are split arbitraly.

This algorithm with one of the previously mentioned similarities has been shown to be efficient to perform well in classification of time series. [53] embed SAX time series representation with the defined distance on the symbolic space to 1-NN classifier. Classification results are accurate, however, employing transformed time series worsen the connection between original data and results.

Empirical studies have demonstrated that the choice of the parameter $k = 1$ is quite efficient for time series classification. It has been shown the obtained accuracy of 1-NN with DTW are difficult to beat [21, 55, 83, 87, 88]. However, it has two shortcomings: high computational cost for DTW ($O(Nm^2)$) distance and the high variance.

Decision Trees. If providing comprehensible classification results is of interest, Decision Trees are usually recommended. This method uses a top-down approach in order to recursively build a tree. At each internal node, there is a split test evaluation and each leaf node contains information about a class to be assigned to the new test instance. While the classical Decision Trees introduced by Breiman et al. [14] performs data partition based on nominal attributes of the input data, Temporal Decision Tree [16, 89] were developed as a modification that allows tackling time series data as it is, e.g., without extracting explicit statistical features. Input instances are split based on similarity between time series or their segments.

There are two main approaches of modified versions of the Decision Tree algorithm. One aims to handle the entire time series and another uses segments of time series as input examples. The former keeps the link between the input data and output results. The latter modifies data by extracting class discriminative segments of time series and then compares them, using a similarity measure, with input instances in order to define a split.

1. **Whole time series.** Yamada et al. [89] proposed the algorithm of learning Time-Series Tree based on a similarity measure. Each internal node contains reference series which exists in training data. Partitioning of input time series into two descendant nodes is done by its similarity to reference series. Each time series from the training set can be considered as a candidate to be a split series. For a given measure of similarity, [1] its value between a split candidate and input series is computed and a threshold that provides the best data separation is defined. Assignment of series to child nodes is done by comparing the distance between an input series and a split candidate with the threshold. The best reference series is the one that splits data in the best possible way. In order to evaluate the goodness of separation, gain ratio criterion was selected [67]. If there are ties in the obtained gain for few time series, the algorithm choses a series with the largest distance gap of the closest point to the threshold.

[1] the authors proposed to use DTW as a robust measure to the distortions along time axis

In addition, the authors [89] proposed a cluster-based split test to partition a node. Two time series are then taken from the training set as split representatives. Next, a distance value is measured between the split series and time series that a node contains. The latter is associated with those to which it is closer according to selected distance measure. A similar approach was proposed in [7]. While Euclidean and/or DTW similarity measures were used in previous works, Chouakria and Amblard [16] elaborated cluster-based split test by introducing adaptive similarity measure DTW-Cort$_\pi$. It enables one to capture different discriminative attributes of time series and obtain a purer partition. Split based on pair of original time series is quite interpretable since split decisions are based on a visual percept of analogy. These algorithms are considered as the baseline of the conducted research in the thesis; their detailed description will be introduced in the following chapter.

2. **Segments of time series.** Decision Trees cannot only be built based on the entire time series, but also on the segments extracted from time series. In Section 2.3 we reviewed different methods of generating time series subsequences from the training data. Output of each of them can be used as input for Decision Trees. For example, shapelets within a decision tree are widely used to classify time series [63, 91]. Geurts [31] proposed using time series segmental patterns as training set to build a tree. The classifier built on time series segments shows high classification accuracy, if class discriminative information are concentrated in time series sub-interval and not in the entire time series. Also, using time series segments allows preserving interpretability of the yielded models [91], [88], [56].

Support vector machines (SVM). SVM classifier can be applied once time series represented as a matrix of examples described by a set of features. For example, BOP time series representation proposed by Senin and Malinchik [79] is fed to SVM algorithm with polynomial kernel in order to learn a classification model.

There have been attempts to apply kernel SVM to perform classification. Kernel functions can be viewed as the similarity between time series. Leslie et al. [51] proposed k-spectrum kernel for protein classification and, later, an extended approach to deal with mismatching of time series [52]. Applying SVM to time series classification involves the question of how to define a kernel and how to speed up the computation of kernel matrices. There have been several attempt to apply DTW directly as similarity measure in a kernel-based classification model [35, 80]. However, obtained results are

inferior than 1-NN DTW. The problem is that DTW distance does not correspond to a valid positive kernel. Thus, direct use of DTW leads to an indefinite kernel matrix that neither corresponds to a loss minimization problem nor giving a convex optimization problem. Kate [41] proposed to consider DTW pairwise distances as a set of feature vectors. To learn a classification model, they use SVM algorithm with polynomial kernel. The bottleneck of performing TSC task with SVM is that it is hardly interpretable for a domain expert to gain knowledge besides classification results.

2.4.2 Ensemble methods

Ensemble methods integrate various time series representations and classification techniques. When the accuracy is the foremost goal and criterion for performance evaluation of TSC algorithm, ensemble learning is an appropriate technique. An ensemble of classifiers is a set of base classifiers whose individual decisions are combined to classify new examples. One key concept in ensemble learning is diversity of approaches, which enables one to capture different discriminative characteristics of data [48]. This diversity can be achieved by employing different classification algorithms. Often, different features are selected randomly to train each classifier. Below, we will explain some popular ensemble methods in TSC.

Ensemble built on shapelet transformation. It has been shown that improved classification accuracy can be achieved through simple ensemble schemes. The algorithms described in previous section were grouped up to enpower accuracy of the results. Bostrom and Bagnall [12] employ binary shapelet transformation in the ensemble of 8 heterogeneous classifers that use probabilistic (k-NN, Naive Bayes, Bayesian Network), tree based (C4.5 decision tree Quinlan [67], Random Forest [13], Rotation Forest (10 trees) [73]) and kernel based models (SVM with linear and basis function kernels [17]).

Another ensemble classification is *Shapelet Transform* and was proposed by [36]. The authors have proposed a shapelet transformation on a single run instead of using a decision tree and search the best shapelet in a node. Then, a new dataset is represented by an instance (a time series shapelet) and its attributes (distance values between a shapelet and time series of the training set). In the procedure of shapelets learning transformation, they are evaluated based on how well they discriminate just one class and how they are balanced per class. The input is then feed to an ensemble classifier.

Hills et al. [36] proposed a shapelet transformation that separates the shapelet discovery from training of the classifier by finding the top k shapelets on a single run. Then, shapelet data is transformed to the alternative data space where each feature represents the distance between a shapelet and each series from the training set. The new feature set is an input for each classifier in the ensemble group.

Random Forest. TSC classifier based on a Random Forest ensemble method [13] has been used to boost classification accuracy in TSC. Baydogan et al. [9] used the representation of time series training set via class probability histograms to train a model with a Random Forest classifier. In work of Deng et al. [19], the authors proposed to employ a set of interval features to split data at each node of the learnt tree. The interval features target to capture temporal characteristics of time series. However, obtained set of features is large ($O(m^2)$) for one time series of the length m. To keep the time complexity with respect to time series length, random sampling is used at each node [13]. In each split node, they generate two random samples, each of which of size ($O(\sqrt{m})$) over interval length and its starting position. This leads to a reduced feature space size with respect to $O(m)$. In the same paper, the authors address the problem of how to get interpretable results, since a forest contains multiple trees. To provide a good understanding on features distinguishing time series from different classes, they proposed a tool called the temporal importance curve. For each time point t of a time series the importance computed as the sum of entropy gain over features used for splitting. If there is no split feature $f_{(.)}(t_{start}, t_{end})$ such that $t \in [t_{start}, t_{end}]$, it has zero importance value. Importance is computed for each feature type (one of the Equation 2.15,2.16,2.17) separately using the following equation:

$$f_{(.)}^{importance}(t) = \sum_{t \in [t_s, t_e], \nu} Entropy(f_{(.)}(t_{start}, t_{end}), \nu), \tag{2.19}$$

where ν is a set of split nodes in time series forest. Evidently, due to the random sampling over time interval, the results of importance curve are biased and in order to tackle this issue the number of trees in a forest should be large. In the performed experiments, the authors built 500 trees.

Ensemble of Elastic Distance Measures (EE). Lines and Bagnall [55] used ensemble of recently proposed elastic measures (WDTD[40], TWED[59], MSE [82]) to incorporate them into heterogeneous elastic ensemble (EE). Driven by the fact that there is no significant difference between the elastic distance measures in terms

of accuracy, the authors proposed to combine several elastic measures with simple ensemble classifiers.

EE is a mixture of 11 elastic distance measures on the whole time series combined with 1-NN classifier. The measures are Euclidean distance, DTW and derivative DTW (DDTW) with full warping window, DTW and DDTW with warping path constraints, whose size is defined through cross validation (DTW-CV and DDTW-CV), weighted DTW and DDTW (WDTW, W-DDTW), Longest Common Subsequence (LCSS), introduced in Hirschberg [37], ERP, TWED, MSM. The authors experimentally show that none of individual distances combined with 1-NN classifiers significantly outperforms DTW-CV.

Then, they introduce ensemble learning with 11 weighted 1-NN classifiers, the class prediction is done by voting. They used 4 different schemes to weight each base classifier, where a weight is obtained through cross validation. The experimental tests showed that the improved classification accuracy of EE classifier is significantly better than each of the component separately.

COTE: the collective of transformation based ensembles. The feature-based algorithms are useful to yield accurate classification results, however, one can reinforce them by employing ensemble learning models build upon homogeneous or heterogeneous transformations of time series. The former tranforms time series into class discriminative features using a single approach and then embeds this representation to a set of simple or complex classifiers. The latter uses a collection of different techniques to obtain data transformation that is complete to capture a variety of characheristics of original data. Then, in the same way, a batch of classifiers uses them as the input set for the training set.

For the purpose of classification accuracy improvement, [6] proposes to form the collective of ensemble classifiers built on different data transformations named as COTE. Based on the previously studied ensembled schemes [5, 36, 55], the authors state the need of creating heterogeneous ensemble classifiers, rather than employing ensemble of weak classifier schemes.

The main motivation is to capture heterogenueous discriminatory features of time series that can be found in different domains as values, shape, frequency or complexity difference. A unique transformation of the data is not able to detect all variety of features. The authors define a massive feature space through heterogeneous data transformations including shapelets, auto-correlation and periodogram based

features. The batch of 8 heteregoneous classifiers forms the collective ensemble (k-Nearest Neighbour, Naive Bayes, C4.5 decision tree [66], support vector machines [17] with linear and quadratic basis function kernels, Random Forest [13] (100 trees), Rotation Forest [73] (10 trees), Bayesian network. Since two elements are also ensemble classifiers, COTE is called meta-classifier of time series data. An assignment of class value to a new time series is done by weighted vote of each component in the collection.

The set of experiments conducted on 72 time series data datasets showed that COTE is significantly more accurate than any classifier separatety. Based on the achieved results, the authors state that while finding a good data representation is crucial, employing unique data transformation in a classfier does not guarantee high accuracy performance. One reason is that the complex data such as time series requires having class discriminants in multiple representations. The second reason is that a set of heterogeneous classifiers need to be applied to deal with redundant features.

In the conclusion of [6], it is pointed out that the interaction between different classifiers, used transformations, and its impact on the results is the direction of further research. Besides the improved classification accuracy, the method has some drawbacks in results comprehensibility. Due to use of wide range of transformation techniques and classifiers, the algorithm has high time complexity ($O(N^2m^4)$).

2.5 On interpretability in TSC algorithms

Time series data in their nature is way more complex than static data, therefore it is more challenging to get insightful knowledge from them. As a consequence, ability of TSC algorithms to provide comprehensible classification results becomes extremely important. In addition to accuracy performance, for a practitioner in any domain, understanding of complex class distinguishing patterns in time series data is the most important objective. To make the process of data analysis all-encompassing, it is crucial to provide the user a set of results alongside the information why and where the model succeeded or failed. The significance of the question about interpretability, explainability and readability in ML algorithms as significant as in algorithms dedicated to time series classification. Here, we would like to narrow down the problem of interpretability for TSC models and attempt to give more concrete examples.

In TSC domain, there are studies that target yielding interpretable time series classification models [16, 34, 53, 63, 77, 79, 89, 91]. Regardless of absence of the rigorous definition of interpretability in TSC algorithms, the ability to preserve the

link with original time series data during the learning process is considered as one of the grounded properties of an interpretable model. It becomes even more important when a domain expert do not have knowledge about ML. In this case, the link with raw time series will serve as a tool to analyze classification results. For example, we say that Decision Tree model can be interpretable for non ML experts because it is capable of rendering the logic of taken classification decisions. If a model missclassifies a test example, then a user can track the path of prediction process in order to find the explanation of getting this result.

Surely, even a classifier is naturally more interpretable, it will not be enough if data representation is not. As we saw, there are lots of techniques for time series representation that can be applied before we feed the data to a classifier. One one hand we desire to extract non-trivial patterns from time series, that can explain what the class discriminative attributes are. On the other hand, the chosen representation should be comprehensible for a user in order for them to undertand the data.

Thus, we can see that the question of interpretability of a model is two level:

1. getting interpretable representation of time series
2. building interpretable TSC model.

2.6 Conclusion

As we have seen, there are lots of different ML algorithms proposed for TSC problem. From a domain expert point of view, we can divide them on two big categories. One category comprises more interpretable algorithms, but not always yields high accuracy. However, the algorithms in the second category are in general more accurate, but less interpretable. Ensemble learning models in either case improve accuracy, but interpretability and comprehensibility of the results will drastically drop.

To summarize, many questions on the step of classification tool selection are risen for a domain expert. Which method shoud be used by a practitioner who is keen to obtain clear, comprehensible, and accurate classification results? The ideal method would satisfy these criteria simultaneously. In this research, we attempt to design TSC algorithm that yields a readable model with interpretable classification results. We selected as the baseline algorithm, which we aim to develop, Temporal Decision Trees [16], that is intrinsically interpretable and provide good classification results. We will address the following aspects of the algorithm:

- reinforcing of interpretability power,

- preserving and improvement of classification accuracy,
- reducing training time complexity.

In the forthcoming chapter, we provide detailed description of the Decision Tree algorithms capable of tackling time series data. We show few similarity- and pattern-based approaches that have been proposed to modify classical Decision Tree method for static data. We will discuss the limitations of it and directions for improvement that we address in later chapters.

Chapter 3

Mono-operator Temporal Decision Trees: TDT

Contents

The Decision Trees machine learning methods are generally recommended when providing interpretable models and results are crucial. In terms of the classification accuracy they are able to provide good results, even though they are not the most competitive algorithms. Complexity of temporal data reinforces motivation of developing algorithms that bring up its conceptual understanding. This chapter describes Temporal Decision Trees (TDT) algorithm that was designed for time series classification task (TSC). Several authors [7, 16, 31, 89] proposed to modify the Decision Trees algorithm for time series data. We dedicate this chapter to provide detailed explanation of these algorithms on which we elaborate in this work and contribute in forthcoming chapters. In the first section, we give detailed description of learning TDT based on global properties of time series Balakrishnan and Madigan [7], Geurts [31], Yamada et al. [89]. The second section introduces embedding of dichotomy search into a tree to find the most discriminative time intervals Chouakria and Amblard [16].

3.1 Temporal Decision Trees (TDT) learning

3.1.1 Introduction

Before providing formalization of the classification problem, let us give the definition of time series.

Definition. *A time series* $ts = (v^1, \ldots, v^m)$ *is an ordered sequence of real values* $v^t \in \mathcal{R}$*, each value measured at specified time point* $t \in 1, \cdot, m$*. The set* $t = (t_1, \cdots, t_m), t \in \mathcal{N}$ *is set of timestamps of* ts*, with the condition* $t_i < t_j$ *whenever* $i < j$*.*

Let $(\mathbf{TS}, \mathcal{Y})$ be a set of labelled time series, $\mathbf{TS} = \{ts_1, \ldots, ts_N\}$, where N is the number of time series and a set of class labels $\mathcal{Y} = 1, \ldots, J$. Given an unlabeled time series $ts \notin \mathbf{TS}$ the classification task is to assign it to one of classes of a set $\mathcal{Y}$. In other words, the objective is to learn a classifier function that maps from the space of time series to the space of class values.

Given training set of labelled time series $(\mathbf{TS}, \mathcal{Y})$ the goal is to build a classification binary tree. Beginning from the root that contains all training data, the model is obtained by recursively partitioning data at each *internal* node. A node without children is called a *leaf* node. A class label to which the majority of time series in a *leaf* node belong is associated with it. It is possible to have few leaves with the same label. The binary partition involves evaluation of split candidates from training set. Once tree is built, classification procedure is quite simple. A new test instance is placed in the tree root, then it traverses through the tree by passing split tests in internal nodes. It is classified by a class label attached to a leaf into which an instance falls.

From now we introduce an additional terminology that we need for a tree. Let $G = \{g^R\} \cup G^I \cup G^T$ be a set of tree's nodes, denoted the root node as g^R , $G^I \subset G$ is a set of internal nodes, G^T is a set of terminal (leaf) nodes. Any non-terminal node g^I has two descendant nodes g_L and g_R.

The whole process of decision tree construction involves defining three main points that are :

1. Split node selection.

2. Decision to declare a node as terminal or to continue partitioning.

3. Class label assignment for each leaf node.

Last two stages remain the same as for classical decision trees. The second step is quite simple, decision is done automaticaly if we build the tree until a node consist time series of only one class. In other cases, a percentage threshold of a major class is required to declare the node as a leaf.

The whole story about modification of the Decision Trees (DT) algorithm working initially on static data to time series is in defining what is a split of a node and how to find a "good" one which is able to separate data minimizing class impurity level.

3.1.2 Split operators

The crux of the problem with time series data is that they cannot be described naturally by the set of explicit features as any static dataset. Two main approaches, similarity-based and feature-based are used in classification to overcome this difficulty. While the later one requires explicit local patterns extraction pre-processing step, the former one classifies objects by its similarity. Both ways of tackling time series can be incorporated into the DT algorithm, and define different nature of data split. Whatever is a choice, we say that generated split candidates are *homogeneous* if each of them provide a split which defines the same geometry of data partition.

Definition. *A node split operator* $\mathcal{F}(.) = \{f\}$ *is defined as the set of homogenious split candidates of a node.*

Let us describe different proposals of split operators for TDT algorithm.

Hyperplane split operator.

Definition. $\mathcal{F}(H) = ((ts_L, y_L), (ts_R, y_R))^d$ *is a hyperplane similarity based split operator defined by a pair of time series* (ts_L, ts_R) *from a training set* **TS** *belonging to distinct classes* $y_L \neq y_R$ *and a dissimilarity measure* d *to compare time series.*

For any split candidate in $\mathcal{F}(H)$ the data is separated by using its similarity to ts_L and ts_R. Each time series in a node that is closer to ts_L according to selected dissimilarity measure d ($d(ts_L, ts) \leq d(ts_R, ts)$) falls into the left descendant node. In the opposite case, it goes to the right descendant node. Assume that $\mathbf{TS}_L$ is the set of time series in g_L, whereas $\mathbf{TS}_R$ are time series in g_R. Thus, we say that time series in $\mathbf{TS}_L$ is associated with ts_L and those that are in $\mathbf{TS}_R$ with a split time series ts_R. The best split pair (ts_L, ts_R) is the one providing the purest data partition according to Gini impurity index. This way of partitioning resembles clustering [7, 16, 89] in the sense that we have one representative of each group of separated data. On the

step of the model analysis and interpretation, one can make a detailed exploration of split time series in order to get insights about data they are associated with. There are two nice properties of this split operator which make it highly interpretable :

- a split of data is done by analogy with one of split time series;
- it allows to a user to inspect visually split time series, which are characterized each divided subset of data.

The limitation of having distinct class labels of time series in a split candidate is done for the purpose of clarity. In general case, we assume that the data within one class is homogeneous, which means most of time series are highly similar to each other. Therefore, generating split pair of time series belonging to the same class does not guarantee to have good discriminative abilities. One can relax the assumption of strong class homogeneity saying that one class can be composed by several sub-classes. Then, by allowing generating pairs of time series with the same label we can enlarge the set of split candidates. By doing this, we increase the probability to find the discriminative split. However, without assessing if class homogeneous assumption is hold or not, we impose additional computational requirement on split candidate examination. In this research, we suppose that classes are homogeneous.

From geometrical point of view, a split by a pair of time series defines hyperplane in time series space. Figure 3.1, *a*) illustrates a partition obtained for a time series pair (ts_L, ts_R). Each point schematically represents a time series. Figure 3.2 shows an illustration of a tree node with a split done by a pair of time series. On this example, the hyperplane split is depicted by two time series of class "-1" and "1". The node contains 4 time series of class "-1" and 3 time series of class "1". Class labels are rendered visually by different colors. The split time series ts_L has the class label "-1" and ts_R has the class label "1". Each descendant branch of the node is marked (obs_000 and obs_070) with the name of split time series in the training set. The split is based on visual percept of analogy. Each split series is the representative of descendant node which is associated to it.

The complexity of search for the best split pair in the space of all candidates is quadratical ($O(N^2)$) with respect to the number of time series in the training set **TS**. While the complexity of each split candidate assessment is $O(N)$. Hence, total complexity of employing this split operator is $O(N^3)$.

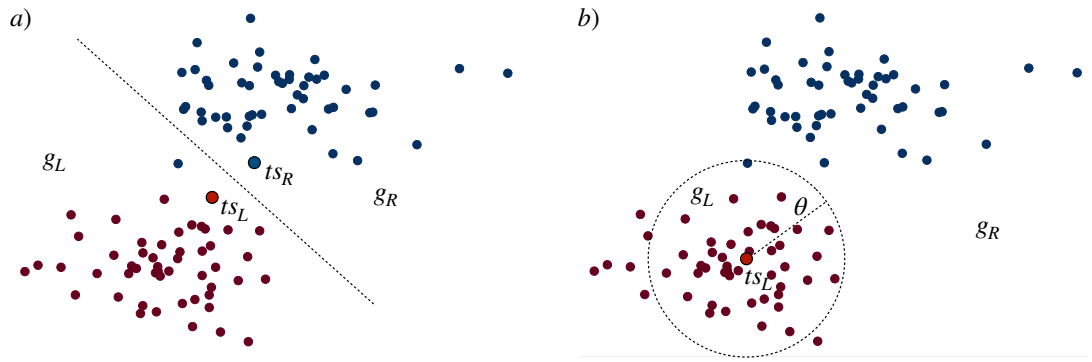

Figure 3.1: Illustration of a partition's geometry done by hyperplane (plot a)) and hypersphere (plot b)) split operator. Each point schematically represents a time series. a). Two bold points are split time series ts_L and ts_R that define hyperplane. Points on each side of hyperplane constitute the left (g_L) and the right (g_R) sub-nodes respectively. b). A node's split is presented by a time series ts_L and a distance threshold θ that forms hypershere. Points inside the hypersphere are assigned to g_L, points outside the hypersphere to g_R.

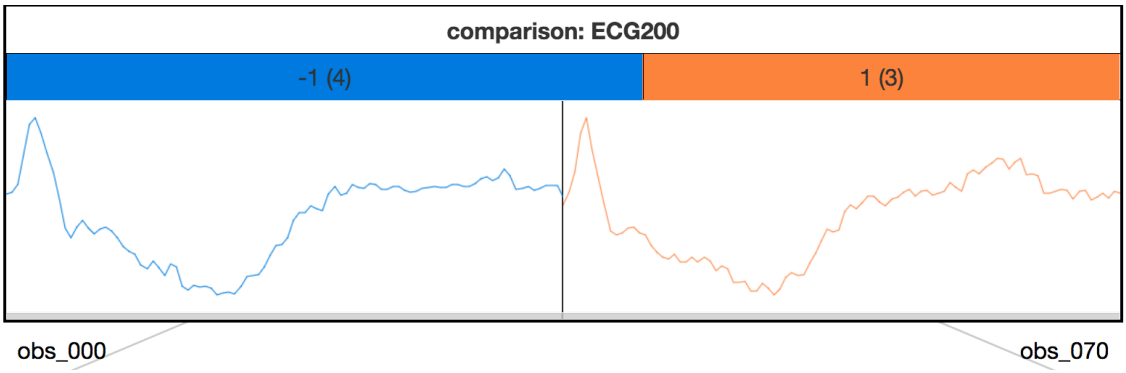

Figure 3.2: Example of employing hyperplane split operator in an internal node of the tree built on ECG200 UCR dataset displayed by the graphical interface developed in the IKATS project. Horizontal color bars indicate class labels and the value in parenthesis is the number of time series the node contains of.

Hypersphere split operator.

Definition. $\mathcal{F}(S) = ((ts_L, y_L), \theta)^d$ *is a hypersphere similarity based split operator defined by a time series from a training set* **TS** *and a dissimilarity measure* d *to compare time series,* θ *is a distance threshold.*

The operator is represented by one time series ts_L and the distance threshold θ. Each time series of the training set is considered as a split candidate [89]. Each time series ts for which $d(ts_L, ts) \leq \theta$ holds falls into the left descendant node. Those time series for which $d(ts_L, ts) > \theta$, i.e., dissimilar with ts_L according to the threshold θ, fall into the right descendant node. Evaluation of each split candidate involves the step of finding the optimal distance threshold, which leads to the best data separation. In order to find such threshold, all pairwise distances are computed between a split time series and all the rest time series in the training set. Then, all of them are sorted to obtain an orderline of distances. Each middle value between two contiguous values of distances is a threshold candidate. We select the optimal distance threshold by maximizing a Gini impurity gain on partitions obtained for each split threshold candidate. Then, among all split candidates $((ts_L, y_L), \theta)^d$, the one with the maximum Gini gain is picked to make a node partition.

Geometry of this split type is shown on Figure 3.1,*b*), each point refers to a time series. Once a threshold θ is found, it forms a hypersphere. Points inside the hypersphere form the left descendant node, the right one contains all points that are outside the hypersphere. Visualisation of the hypersphere split operator is shown on the Figure 3.3. The node contains 2 time series of class "1", 13 time series of class "2", 12 time series of class "3", and 5 time series of class "4". Class labels are rendered visually by different colors. The split time series ts_L has the class label "2". The value of the threshold θ is equal to 17.231, this value is marked on each descendant branch. Similar to the hyperplane split operator, we are still based on visual concept of similarity. Even though, if the notion of threshold here is more difficult to assess for a user, due to the lack of referential.

Complexity of the best split candidate search is linear with respect to the number of time series in the learning set, however for each split candidate ts_L we need to assess $N - 1$ threshold candidates θ in order to find the optimal one. Thus, the overall complexity to find the best hypersphere split candidate in $O(N^2)$.

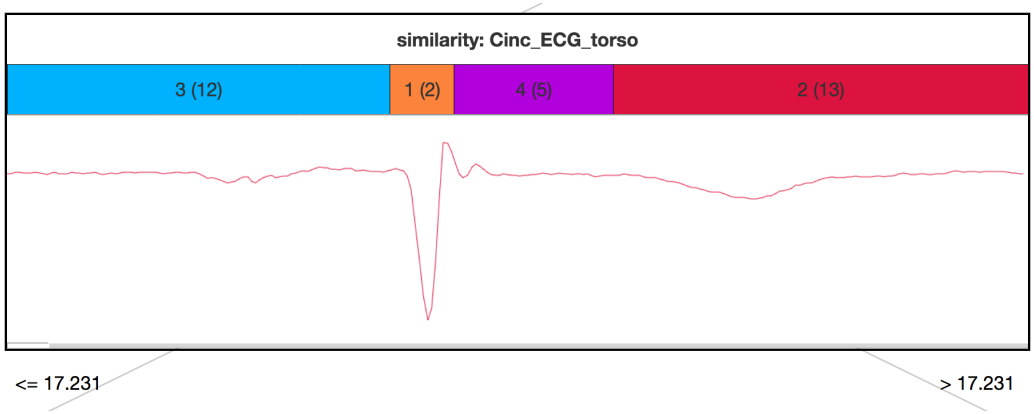

Figure 3.3: Example of employing hyperplane split operator in an internal node of the tree built on CinCECGtorso UCR dataset displayed by the graphical interface developed in the IKATS project. Horizontal color bars indicate class labels and the value in parenthesis is the number of time series the node contains of.

Patterns split operator.

Definition. $\mathcal{F}(\mathbf{P}) = (\mathbf{p}, \nu)$ *is a feature based split operator defined by a pattern* $\mathbf{p}$ *extracted from a time series from a training set* $\mathbf{TS}$. *Its corresponding attribute variable* ν *can be represented by numerical or categorical values.*

In a pair $(\boldsymbol{p}, \nu)$ the term $\boldsymbol{p}$ points on time series representation, and ν quantatively or qualitatively describes this representation. Patterns can have different nature, i.e., it can be class discriminative segment of time series, it might be some summary statistics (mean, variance, etc) collected over set of time series belonging to the same class, or even some transformation can be applied to obtain high level of representation. An attribute ν can be represented by numerical values as well as categorical ones. For example, a pattern $\boldsymbol{p}$ can stand for a time series representation in perceptually important points - landmarks (method proposed in Perng et al. [65]). Its corresponding attribute ν will be the set of important landmark points. Another example, a pattern $\boldsymbol{p}$ stands for collected statistics over time series such as mean, and ν represents its value [64].

Patterns discover techniques suppress a noise that original data can contain reinforcing class distinguishing properties. At the same time, it may happen that we get rid off some useful information to perform accurate classification. Even though, a local pattern can be just a sub-interval of time series, if split is done by employing a

similarity measure, we equal it to the one of two operators defined above, depending on the way of split candidate generating.

Before embedding patterns into TDT, one should include pre-processing step to extract the set of patterns from the training data. There are immense variety of methods to extract local patterns from time series data [18, 30]. Patterns can have a high discriminative power leading to increase the classification accuracy, however they are not often easily interpretable. Lots of transformations on the input data can be used on the step of searching a patterns set.

Once the set of pattern is discovered, we can consider them as variables (as in static data) described by numerical or categorical attributes. Then, data partitioning can be done using standard Decision Trees algorithm. Let ν_{ts} be a value of a pattern $\boldsymbol{p}$ presented in a time series ts. The test questions at an internal node are defined for any pattern $\boldsymbol{p}$ as follows:

- if ν is an attribute variable of categorical type, i.e., ν_{ts} takes values in the finite set of ν. Then, for any subset s of power set of ν, denoted as $\mathcal{P}(\nu)$, the test question has the form: *does ν_{ts} belong to a subset s?*

  $$\nu_{ts} \in s? \tag{3.1}$$

  In the case of positive answer, a time series goes to the left descendant node, otherwise to the right descendant node.

- if $\nu \in \mathbb{R}$ is an attribute variable of numerical type, the test question has the form: *is a value ν_{ts} is less than a value θ?*

  $$\nu_{ts} \leq \theta? \tag{3.2}$$

  where θ is an optimal threshold value associated with a pattern $\boldsymbol{p}$.

The training set **TS** is represented by the set of patterns, that are considered as features. Hence, each time series ts can be represented as a point in $|\boldsymbol{P}|$-dimensional space, each $(\boldsymbol{p}, \nu)$ is a coordinate axis. Once the best pattern $\boldsymbol{p}$ to make a split in a node is found, input space is partitioned by a hyperplane that is orthogonal to the coordinate axis $\boldsymbol{p}$. The geometrical perspective on a partition is shown on the Figure 3.4. In this simplified example, there are two patterns $(\boldsymbol{p}_1, \nu_1), \nu_1 \in \mathbb{R}$ and $(\boldsymbol{p}_2, \nu_2), \nu_2 \in \mathbb{R}$ that represent time series input set in a node g. The values θ_1 and θ_2 are the optimal threshold values to make a partition for patterns $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$

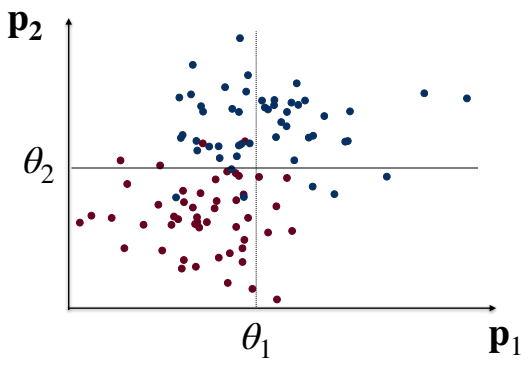

Figure 3.4: Illustration of a partition's geometry done by pattern split operator. Each point is a time series representation through two extracted patterns $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. Each pattern and its threshold divides data by orthogonal hyperplane.

respectively. In given example, the best split will be represented by a pair $(\boldsymbol{p}_2, \nu_2 = \theta_2)$, its corresponding partition shown on the Figure 3.4 by solid black line.

Complexity of split candidates search space is linear to the number of extracted patterns. However, the evaluation complexity of a partition for each split candidate depends on the domain of its attribute variable ν. If it is numerical, i.e., measured by real numbers, then the complexity is $O(N^2 logN)$ for one split candidate $(\boldsymbol{p}, \nu)$ and becomes $O(|\boldsymbol{P}|N^2 logN)$ for a pattern-based operator with numerical attribute ν. If ν is categorical, then the complexity depends on the cardinality of the set ν. For $|\nu| = q$, there are $2^{q-1} - 1$ possible ways to make a binary partition, for each partition it costs $O(N)$ to assess its quality by Gini impurity index. There are different techniques of encoding categorical feature to the numerical one, it lies beyond this work, and hereafter, we employ patterns with numerical attributes.

In this research, we will focus mainly on the interpretable properties of an approach to be used as one to discover the set of local patterns. In this chapter, we consider only the general methodology of using patterns in TDT without refering to any particular method. However, in forthcoming chapter, we introduce the novel heuristic approach of pattern extraction that we employ in the developed framework. Then, we will characterize this method by its complexity.

3.1.3 Splitting evaluation criteria

To make a binary split of data at each node, the tree algorithm generates a set of split candidates. Next, fundamental question is how to select the "best" split on a node such that class labels in descendant nodes are less mixed. The estimate of goodness of each split candidate is based on an impurity function.

Definition. *An impurity function is a function ϕ defined on the set of all K-tuples of class label proportion $\{\frac{N_1}{N}, \ldots, \frac{N_K}{N}\}$ satisfying the following properties:*

- *ϕ is a symmetric function of $\{\frac{N_1}{N}, \ldots, \frac{N_K}{N}\}$*
- *ϕ is a maximum only at the point $(\frac{1}{K}, \ldots, \frac{1}{K})$.*
- *ϕ amounts to its minimum only at the points $(1, 0, \ldots, 0), (0, 1, \ldots, 0), (0, 0, \ldots, 1)$*

Definition. *The impurity measure $i(g)$ of a node $g \in G$ given an impurity function ϕ is defined as follows:*

$$i(g) = \phi(\mathsf{p}(1|g), \mathsf{p}(2|g), \ldots, \mathsf{p}(K|g)),$$

where $\mathsf{p}(j|g)$ is the estimated posterior probability of class j.

We are interested to select the split candidate as a splitter that maximally decreases impurity in descendant nodes. That is to say, class labels of time series within each node become more and more class homogeneous. Hence, the goodness of a split $s(f, g), f \in \mathcal{F}$ with proportion of the data $p^L = \frac{N^L}{N}$ in the left child node g^L and $p^R = \frac{N^R}{N}$ in the right child node g^R can be formalized in general as:

$$\Delta i(s(f, g)) = i(g) - p^L \cdot i(g_L) - p^R \cdot i(g_R), \tag{3.3}$$

Many impurity measures were proposed for selecting the best candidate. Two splitting criteria are widely used in trees construction: *Gini impurity index* and *Entropy*, which are defined as:

- *Gini impurity index*:

$$i_{\text{Gini}}(g) = 1 - \sum_{j=1}^{K} p_j^2 \tag{3.4}$$

- *Entropy*:

$$i_{\text{Entropy}}(g) = 1 - \sum_{j=1}^{K} p_j \log p_j \tag{3.5}$$

It is possible that few split candidates share the same quality of split, i.e., values of optimized impurity criterion are equal. Typically, there are two ways to deal with that, one is to simply pick "the best of the best" splitter arbitrarily. The other one is to optimize an additional criterion such as, for example, the level of separability. For similarity based split, operator separability is measured in terms of distance margin between the n closest elements belonging to the left and right sub-nodes [92]. For feature based split operator, separability is measured by *twoing* criterion [14].

- *Separation Margin*:

$$i_{\text{Gap}}(g) = \frac{1}{N^L} \sum_{ts \in g_L} d(ts_L, ts) - \frac{1}{N^R} \sum_{ts \in g_R} d(ts_R, ts), \tag{3.6}$$

  $ts_L = ts_R$ in the case of use hypersphere split operator $\mathcal{F}(S)$. Note that in defined criterion the average distance is taken over all points on the left sub-node and the right sub-node, alternatively it can be replaced by k-nearest neigbour time series to a decision boundary.

- *Twoing*:

$$i_{\text{Twoing}} = \frac{p^L p^R}{4} \Big(\sum_{j \in \mathcal{Y}} [p(j|g_L) - p(j|g_R)] \Big), \tag{3.7}$$

  where $p(j|g_L)$ and $p(j|g_R)$ are proportions of class label j in nodes g_L and g_R respectively.

In theory and in practice [68],[14], there is no intrinsic reason for preference of one criterion over another. The misclassification rate of yielded trees is not much sensitive to splitting criterion. Hence, we can rely on consistency of experimental results with any choice of splitting criterion. Only for the purpose of computational efficiency, in this research we selected to employ *Gini impurity index* in all experiments.

3.2 Mono-TDT split node algorithms

3.2.1 Hyperplane TDT with adaptive metric.

We have seen the methodology of tree "growing" in general case. Now, we focus on the algorithm of Temporal Decision Trees ([16], [89], [7]) that employs one of the abovementioned split operators. While all steps are remained as we have described before, the main interest lies in a node's split procedure. The discriminative charateristics of time series at different levels of tree can vary from amplitude to shape differences. By using an adaptive metric DTW-Cort$_\pi$ introduced in [16] within TDT, the algorithm is capable of changing a metric from one internal node to another. This allows on the one side to capture alteration in discriminative properties, on the other side, to understand underlying time series attributes that are important for classification. As an alternative to DTW-Cort$_\pi$, the range of heterogeneous metrics can be used encompassing those that are able to capture different time series percularities. In the algorithms description, we fix the usage of DTW-Cort$_\pi$ as presented in Formula 2.13. The values of the parameter λ lies in the range $[0, 6]$, as

Chouakria and Amblard [16] have shown that it suffices to work out all modularities of this distance. In the experiments, that we present in the forthcoming chapter we test as well an option with the metric range usage.

We list three algorithms of a node split based on hyperplane, hypersphere and pattern split operators.

Hyperplane node split. Algorithm 3.1 shows how to find the best hyperplane split of the data **TS** belonging to a node g. With pre-defined metric DTW-Cort$_\pi$ and split operator $\mathcal{F}(H)$, it returns the best splitter f_H^* of a node g. First, all split candidates, which are represented by a pair of time series with distinct class labels, are generated (line 3). Then, for each value of amplitude-shape tuning parameter λ, each candidate $f_H \in \mathcal{F}(H)$ produces a binary partition of a node g based on distance comparison (line 4 - 7). Time series satisfying the condition $d(ts_L, ts) <= d(ts_R, ts)$ are sent to the left descendant node, otherwise to the right descendant node. Once an assignment of all **TS** $\in g$ to g_L and g_R is done, *Gini impurity gain* is computed. A split candidate with maximum value of gain, i.e., the one that induce the "purer" data partition, is chosen as the best split of a node g.

Hypersphere node split. Given a node g split candidates of hypersphere split operator is represented by a time series $ts \in$ **TS** and a threshold. The search of the best split of data is twofold. First, for each split candidate time series an optimal separating distance threshold must be found. Then, among all split candidates, the one leading to the purest data bipartion is selected. Algorithm 3.2 shows how to find the best hyperplane split of the data **TS** belonging to the node g. Each split candidate $f = (ts_L, \theta)$ produces a partition based on corresponding optimal threshold. For each split candidate ts_L, the best threshold is the value that maximize class labels gomogeneity in the partition.

An optimal split threshold is one from the middle points between adjacent sorted distance values (line 8-9) with the maximum impurity gain. The left descendant node contains time series with a distance to split time series that is less than defined threshold θ, while the right descendant node is composed of remained time series (line 12-13). The best split time series is the one that maximizes Gini impurity gain (line 16-18).

Pattern node split. A node split with a pattern split operator looks like to a split done by features in the classical Decision Trees algorithm. In order to use the pattern

ALGORITHM 3.1: Algorithm of Adaptive Node Split with Hyperplane Split Operator.

1 **Function** *HyperplaneNodeSplit*:
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g; d = \text{DTW-Cort}_\pi$; `// Set of labelled time series in a node` $g \in G$
  Result: $s^*(g, f_H) := \left(g_L^*, g_R^*, (ts_L^*, ts_R^*), \lambda^*\right), \Delta i_{\text{Gini}}(s^*(g, f_H))$
2 $\Delta i_{\text{Gini}}(s^*(g, f_H)) \leftarrow \infty$;
3 $\mathcal{F}(H) \leftarrow \{f_H : ((ts_i, y_i), (ts_j, y_j)); \forall ts_i, ts_j \in \mathbf{TS}; y_i \neq y_j\}$;
4 **foreach** $\lambda \in [0, 1, 2, 3, 4, 5, 6]$ **do**
5 **foreach** $(f_H = (ts_L, ts_R)) \in \mathcal{F}(H)$ **do**
6 $g_L \leftarrow \{ts : d(ts_L, ts) <= d(ts_R, ts)\}$;
7 $g_R \leftarrow \{ts : d(ts_L, ts) > d(ts_R, ts)\}$;
8 $i_{\text{Gini}}(g_L), i_{\text{Gini}}(g_R) \leftarrow$ compute with Formula 3.4 ;
9 $\Delta i_{\text{Gini}}(s^*(g, f_H)) \leftarrow$ compute Gini gain with use of Formula 3.3 ;
10 **if** $\Delta i_{Gini}(s(g, f_H)) > \Delta i_{Gini}(s^*(g, f_H))$ **then**
11 $\Delta i_{\text{Gini}}(s^*(g, f_H)) \leftarrow \Delta i_{\text{Gini}}(s(g, f_H))$;
12 $s^*(g, f_H) \leftarrow s(g, f_H)$;
13 **end if**
14 **end foreach**
15 **end foreach**
16 **end**

split operator within TDT, one should include the step of feature extraction from time series data [2]. These features must capture some discriminative properties that underly in training data. Variations of approaches Fulcher [30] can be used to get a dictionary of patterns. At this step we describe the Algorithm 3.3 of a node's split assuming that patterns were extracted a priori to this step (line 4). Then each pattern is represented by a pair of $(\boldsymbol{p}, \nu)$ where $\boldsymbol{p}$ is a pattern/feature that can be a segment of original time series, symbolic representation, statistics collected over whole time series, etc. And a value of ν is a characteristics of each pattern, for example, it can describe frequency with which a pattern occurs in a class, etc. To make a bipartition by a pattern split candidate $(\boldsymbol{p}, \nu)$ the presence of a pattern $\boldsymbol{p}$ within each time series in a node should be evaluated (typically, we already have this information once the process of patterns extraction is done). A value ν works here as a threshold; time series contain a pattern $\boldsymbol{p}$ with a value less than ν are assigned to the left sub-node, otherwise to the right one (line 6-9). The goodness of each split is evaluated by Gini impurity index and the pattern that provides the purest partition is selected as the best one (line 10-12).

[2] We use terms feature and pattern of time series interchangeably in this work.

ALGORITHM 3.2: Algorithm of Adaptive Node Split with Hypersphere Split Operator.

Function *HypersphereNodeSplit*:
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g; d = \text{DTW-Cort}_\pi$; `// Set of labelled time series in a node` $g \in G$
  Result: $s^*(g, f_S) := \left(g_L^*, g_R^*, (ts_L^*, \theta^*), \lambda^*\right), \Delta i_{\text{Gini}}(s^*(g, f_S))$
  $\Delta i_{\text{Gini}}(s^*(g, f_S)) \leftarrow \infty$;
  $\vec{\theta} \leftarrow \emptyset$;
  $\mathcal{F}(S) \leftarrow \{f_S : (ts_i, \vec{\theta}), \forall ts_i \in \mathbf{TS}\}$;
  foreach $\lambda \in [0, 1, 2, 3, 4, 5, 6]$ **do**
    foreach $f_S = (ts_L, \vec{\theta}) \in \mathcal{F}(S)$ **do**
      $\vec{d} \leftarrow$ sort distances $d(ts_L, ts)$;
      $\vec{\theta} \leftarrow \{(d_i + d_{i+1})/2, d_i \in \vec{d}, i \in 1, \ldots, N-1\}$;
      foreach $\theta \in \vec{\theta}$ **do**
        $f_S \leftarrow (ts_L, \theta)$;
        $g_L \leftarrow \{ts : d(ts_L, ts) <= \theta\}$;
        $g_R \leftarrow \{ts : d(ts_L, ts) > \theta\}$;
        $i_{\text{Gini}}(g_L), i_{\text{Gini}}(g_R) \leftarrow$ compute with Formula 3.4[1] ;
        $\Delta i_{\text{Gini}}(s(g, f_S)) \leftarrow$ compute Gini gain with use of Formula 3.3 ;
        if $\Delta i_{Gini}(s(g, f_S)) > \Delta i_{Gini}(s^*(g, f_S))$ **then**
          $\Delta i_{\text{Gini}}(s^*(g, f_S)) \leftarrow \Delta i_{\text{Gini}}(s(g, f_S))$;
          $s^*(g, f_S) \leftarrow s(g, f_S)$;
        end if
      end foreach
    end foreach
  end foreach
end

The complexity to build one node of a tree depends on an employed split operator. The total complexity is $O(\lambda N^2 \times X)$. The term $O(N^2)$ comes from all pairwise distance computations $\frac{N(N-1)}{2} \sim O(N^2)$, and X is the complexity of a split operator, which is equal to

1. $O(N^3)$ for hyperplane split operator $\mathcal{F}(H)$;
2. $O(N^2 \log N)$ for hypersphere split operator $\mathcal{F}(S)$;
3. $O(|\mathbf{P}|N^2 \log N)$ for pattern-based split operator $\mathcal{F}(\mathbf{P})$.

ALGORITHM 3.3: Algorithm of Adaptive Node Split with Patterns Split Operator.

Function *PatternNodeSplit*:
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g, g \in \mathcal{G}$; set of time series features $\mathbf{P}$
  Result: $s^*(g, f_{\boldsymbol{p}}) := \left(g_L^*, g_R^*, (p^*, \nu^*), \lambda^*\right), \Delta i_{\text{Gini}}(s^*(g, f_{\boldsymbol{p}}))$
  $\Delta i_{\text{Gini}}(s^*(g, f_{\boldsymbol{p}})) \leftarrow \infty$;
  $\mathcal{F}(\mathbf{P}) \leftarrow \{f_{\boldsymbol{p}} : (p, \nu), \forall p \in \mathbf{P}\}$;
  foreach $\lambda \in [0, 1, 2, 3, 4, 5, 6]$ **do**
    foreach $f_{\mathbf{p}} = (p, \nu) \in \mathcal{F}(\mathbf{P})$ **do**
      $g_L \leftarrow \{ts : \nu_{ts} <= \theta\}$; // ν_{ts} is a value of pattern associated with $ts \in \mathbf{TS}$
      $g_R \leftarrow \{ts : \nu_{ts} > \theta\}$;
      $i_{\text{Gini}}(g_L), i_{\text{Gini}}(g_R) \leftarrow$ compute with Formula 3.4 ;
      $\Delta i_{\text{Gini}}(s(g, f_{\boldsymbol{p}})) \leftarrow$ compute Gini gain with use of Formula 3.3 ;
      if $\Delta i_{Gini}(s(g, f_{\mathbf{p}})) > \Delta i_{Gini}(s^*(g, f_{\mathbf{p}}))$ **then**
        $\Delta i_{\text{Gini}}(s^*(g, f_{\boldsymbol{p}})) \leftarrow \Delta i_{\text{Gini}}(s(g, f_{\boldsymbol{p}}))$;
        $s^*(g, f_{\boldsymbol{p}}) \leftarrow s(g, f_{\boldsymbol{p}})$;
      end if
    end foreach
  end foreach
end

3.2.2 TDT with dichotomy search

Whereas the whole time series retains the complete information about the original data, one can ask if all observational values along time axis are uniformly discriminative. Clearly, it depends on a dataset. The positive answer would points on global discriminative properties of a time series. Negative answers, however, means existence of local subintervals that have more discriminative power than the whole time interval. In the case of presence in time series local class distinguishing properties, its capturing can improve both perfomance criteria of algorithm: accuracy and interpretability. Increasing performance comes out from removing values from the distance computation that are actually noisy. By emphasizing a time series segment that has high discriminative power rather showing uniformly distributed values along a whole time axis, a domain expert can easily link this information with an undelying physical process that was measured.

To find the most discriminative sub-interval of a split candidate, *dichotomy search* for hyperplane split operator approach was proposed in the work of Chouakria and Amblard [16]. In order to assess whether the subsequence of a time series is more dis-

criminative than the whole one, one can rely on Gini impurity index estimation. Given the best split defined by a pair of the entire time series, by its consequently bisection, we generate a new split candidate defined by a segment of time series. We can compute impurity of the new partition to assess if Gini gain increases. Algorithm 3.4 reveals a node split process described previously, enriched by dichotomy search approach to find the most discriminative subsequences. As input, it takes time series set $\mathbf{TS} \in g$ and the best found split candidate $\Delta i_{\text{Gini}}(s^*(g, f_H)), I^* = [1, m])$ obtained on the whole interval of time series. The algorithm divides a given interval I^* into two parts (line 3 - 4), whose overlap proportion is defined by a parameter $\alpha \in [0, 1]$, and generates two new input sets $(\mathbf{TS}, \mathcal{Y})_{I_L}$ and $(\mathbf{TS}, \mathcal{Y})_{I_R}$ comprising of left and right time series sub-intervals accordingly (line 5 - 6). For each set, Algorithm 3.4 returns the best possible split $s^*_{I_L}(g, f_H)$ and $s^*_{I_R}(g, f_H)$ (line 7 - 8).

The algorithm checks whether at least one of the "optimal" splits obtained on the sub-intervals I_L and I_R improves Gini gain. If it is not the case, it returns the best split on the entire time interval (line 10). Otherwise, the algorithm continues recursive dichotomous search on the most prominent sub-intervals (line 12 - 16).

Unless we do not have any prior information about the time interval where we can probably find class discriminative segment, it is reasonable to fix parameter $\alpha = 0.6$ [16]. Dichotomous search is completely heuristic and does not guarantee to find optimal discriminative sub-interval. It highly depends on the choice of parameter α. The drawback of the default choice is that it can shatter dicriminative segment on two parts. Therefore yielded split will actually contain only one piece of important information. As a consequence, the learned tree model will loose its interpretability and classification performance.

Adding the option of dichotomy search for discriminative time sub-interval option increases the complexity to $O(\log_{\frac{1}{\alpha}}(m)\lambda N^3)$, since the maximum number of recursive calls *HyperplaneDichoSplit* algorithm 3.4 is $O(\log_{\frac{1}{\alpha}}(m))$ for each sub-interval. By using hyperplane dichotomous mono-operator TDT the total computational complexity is $O(\lambda N^3 + 2\log_{\frac{1}{\alpha}}(m)\lambda N^3)$, that can be approximated by $O(\log_{\frac{1}{\alpha}}(m)\lambda N^3)$.

Surely, the same algorithm can be naturally extended for hypersphere split operator and be employed in TDT. The complexity would be $O(\log_{\frac{1}{\alpha}}(m)\lambda N^2 \log N)$.

ALGORITHM 3.4: Algorithm of Hyperplane Dichotomous Node Split in TDT.

Function *HyperplaneDichoSplit*:
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g, \Delta i_{\text{Gini}}(s^*(g, f_H)), I^*, \alpha$
  Result: $s^*_{I^*}(g, f_H), \Delta i_{\text{Gini}}(s^*_{I^*}(g, f_H)), I^*$
  $[a, b] \leftarrow I^*$;
  $I_L = [a, a + \alpha(b - a)]$;
  $I_R = [b - \alpha(b - a), b]$;
  $(\mathbf{TS}, \mathcal{Y})_{I_L} \leftarrow \{ts_i : \{v_i^t\}, t \in I_L\}$;
  $(\mathbf{TS}, \mathcal{Y})_{I_R} \leftarrow \{ts_i : \{v_i^t\}, t \in I_R\}$;
  $s^*_{I_L}(g, f_H), \Delta i_{\text{Gini}}(s^*_{I_L}(g, f_H)) \leftarrow HyperplaneNodeSplit\big((\mathbf{TS}, \mathcal{Y})_{I_L}\big)$;
  $s^*_{I_R}(g, f_H), \Delta i_{\text{Gini}}(s^*_{I_R}(g, f_H)) \leftarrow HyperplaneNodeSplit\big((\mathbf{TS}, \mathcal{Y})_{I_R}\big)$;
  if $\Delta i_{Gini}(s^*(g, f_H)) \geq \min[\Delta i_{Gini}(s^*_{I_L}(g, f_H)), \Delta i_{Gini}(s^*_{I_R}(g, f_H))]$ **then**
    Return $s^*(g, f_H), \Delta i_{\text{Gini}}(s^*(g, f_H)), I^*$;
    else
      if $\Delta i_{Gini}(s^*_{I_L}(g, f_H)) > \Delta i_{Gini}(s^*_{I_R}(g, f_H))$ **then**
        $HyperplaneDichoSplit\big((\mathbf{TS}, \mathcal{Y}), \Delta i_{\text{Gini}}(s^*_{I_L}(g, f_H)), I_L), \alpha\big)$;
      end if
      else
        $HyperplaneDichoSplit\big((\mathbf{TS}, \mathcal{Y}), \Delta i_{\text{Gini}}(s^*_{I_R}(g, f_H)), I_R), \alpha\big)$;
      end if
    end if
  end if
end

3.3 Conclusion

We have seen how to learn Temporal Decision Trees on the time series dataset and what kind of split operators can be employed to make a data partition in a node. Although algorithms considered in Section 3.2 yield a model with decent interpretable properties, it has few shortcomings. A split operator has to be fixed before we train the model. It leads to testing goodness of partitions made by a set of homogeneous split candidates. Each split operator is capable of capturing only one particular geometrical data structure. Surely, we rarely can have any prior information about class distributions before training process. Thus, oftentimes the prior choice of split operator is made arbitrary or from the perspective of a split explainability. The latter aim rather favors hyperplane split operator over others because a domain expert has referential time series of training set for each subset of binary partition. The main flaw of this method is that it does not reveal true information about the real structure of the data, and if our choice of split operator does not coincide with the data properties, we will get a deep tree that will be difficult to read and analyze. Therefore, the level

of results interpretability and model readability will drop significantly.

Taking into the consideration that the foremost objective is *interpretable* time series classification and the requirements of the algorithm's computational resources to build a model, we do not rely on ensemble classification method such as random forest. Even it is possible to build Random Forest model with Temporal Decision Trees. Clearly, this limitation to build a single tree may lead to the issue of having lower accuracy than if we would have employed ensemble methods. From this perspective, the question is that can we get improved resultant accuracy on one learned tree? We tackle this problem in the current research as well.

The next chapter is dedicated to the problem of interpretability and explainability of yielded mono-operator TDT trees, we propose the solution that gives us an algorithm with improved characteristics of model interpretability and readability.

Chapter 4

Multi-operator Temporal Decision Trees: MTDT

Contents

Previously, we have considered time series classification by using mono-operator Temporal Decision Tree (TDT). In this chapter, we will integrate several split operators within one learned tree in the order to adaptively capture geometrical structure of data class labels distribution within a node. We propose learning Multi-operator Temporal Decision Tree (MTDT) that includes evaluation of a few split operator at each internal node. In addition to similarity-based split operators, we add a feature-based operator obtained by the algorithm we introduce in this chapter. It is called Heuristic Approach of Recognizing Patterns (HARP). Then, we discuss advantages and shortcomings of using each operator separately and in combination within one growing tree. To this end, we provide the empirical study of impact of the multi-operator approach on model readability, interpretability, and performance.

4.1 Multi-operator TDT

4.1.1 Heuristic approach of recognizing patterns: HARP

The interest of employing feature based split operator within TDT mainly comes from the fact that it allows accuracy improvement and concise time series representation. The latter one is appealing particulary in case of sparse signal data, for example, a physical process can be measured during a long time period, where a class defining event appears within a very short time interval. In such cases, first, using similarity based techniques will blur a discriminative short time interval since a distance measure sums up amplitude differences along the whole time series. Secondly, visual rendering of the entire time series, which is sparse, to a domain expert does not facilitate its analysis and does not contribute to interpretability of classification model.

To be able to face and process such data, in general classification framework one should include in learning model the possibility of extraction of time series local patterns and its exploitation. Within our main objective to design interpretable framework of time series classification, we impose two main requirements on the algorithm of pattern finding. The first one is an interpretability of extracted time series patterns. Even though we use transformation technique on time series, we should be able to not completely lose the connection with original data. The second one is a computational time needed to find relevant patterns. This is extremely important because we want to employ extracted patterns within TDT, which is already a computationally costly algorithm, thus adding preprocessing option should not be a deadweight on it.

Pursuing to fit abovementioned characteristics for time series pattern recognition approach, we propose the heuristic algorithm of recognizing patterns called shortly HARP, which we further employ in generalized version of TDT. It is a supervised algorithm that uses the one-against-all method to learn distinctive patterns. Each pattern is an existing segment of length q of a time series from a training set represented by symbols from a given alphabet A. As an output of the algorithm, we get a set of discovered patterns for each time series if any exists.

A pattern $\boldsymbol{p}$ is described by a set of its attributes $\{\nu\}$. It includes following measured statistical entities:

- **Class frequency** is the number of a pattern's occurences within a corresponding class.

- **Position** represented by three values that are the *first*, the *last* and the *average* time position of a pattern within a time series.

- **Gap** is the *minimum*, *maximum* and *average* length of time difference between two sequential occurrences of a pattern.

A pair of a pattern $\boldsymbol{p}$ and one of aboved attributes ν constitutes a split candidate for a pattern split operator $\mathcal{F}(\boldsymbol{P})$.

The HARP algorithm is composed of three main stages:

1. **Patterns generation.** It is the initial step of the HARP algorithm to get a bag of patterns set. In the forthcoming stages the most important patterns for classification process will be kept. This stage includes two steps that are discretization by amplitude and time axes. Borrowing an idea from SAX method [53], we represent time series amplitude values by symbols of a given alphabet A. The procedure of generating a set of all possible patterns is shown in Algortithm 4.2. It starts from generation of a bag of patterns for each time series by consequently discretizing time axis by a sliding window q. Simultaneously, all patterns are grouped up by class labels (line 5 - 9). Once all possible patterns for given parameters are generated, the algorithm gets rid of ineffective patterns by applying double filter on two parameters:

   (a) *coverage* rate e^{cov} within a class label

   $$e^{cov} = \frac{n(\boldsymbol{p}|k)}{N_k}, \tag{4.1}$$

   where $n(\boldsymbol{p}|k)$ is the number of times a pattern $\boldsymbol{p}$ occurs in time series of a given class $k \in \mathcal{Y}$ and N_k is the cardinality of time series belonging to class $k \in \mathcal{Y}$ (line 15);

   (b) *purity* which measures the ratio between the number of occurences of $\boldsymbol{p}$ within class k and total number of occurences of $\boldsymbol{p}$ in the whole dataset (line 15 - 16).

   $$e^{pur} = \frac{n(\boldsymbol{p}|k)}{\sum_{k \in \mathcal{Y}} n(\boldsymbol{p}|k)}. \tag{4.2}$$

   Eventually, only the class discriminative patterns that have coverage and purity above fixed thresholds η^{cov}, η^{pur} (typically both greater than 90%) are kept for the next processing step.

2. **Patterns aggregation.** At this stage, we aim to find more general patterns, i.e., a combination of patterns with similar shape and coverage. Hereafter, we use a heuristic searching approach: given a symbolic pattern $\{a_1, a_2 \dots\}, a_i \in A$, we do the generalization of each symbol a_i. Consequently, by allowing changing it on one level up and down of an alphabet A. For example, given an alphabet $A = \{ABCDEF\}$ and obtained pattern $\boldsymbol{p} = \{BE\}$, the aggregated patterns to check are

   $[AB]E, [BC]E, [AC]E, B[DE], B[EF], B[DF], [AB][DE], [AB][EF], [AB][DF],$
   $[BC][DE], [BC][EF], [BC][DF], [AC][DE], [AC][EF], [AC][DF].$

   We fix all symbols except for one, which we generalize. In our example, we start from B and aggregate it to $[AB]$, $[BC]$, and to wider possible range $[AC]$. Then, the process is continued for all other symbols of a pattern. At the end, the process is repeated with the previously generalized symbols. One should note that meaning of $[AC]E$ is the possibility that a time series have the following patterns AE, BE, or CE. For a pattern of size $|\boldsymbol{p}|$, the maximum number of generalized candidates to check is $4^{|\boldsymbol{p}|} - 1$. For all generated patterns, we test their purity on the training set and if the values are greater or equal to a threshold, we add a new generalized pattern to the set of patterns. Algorithm 4.3 and Algorithm 4.4 reveal the pseudocode of this stage.

3. **Tailoring.** This is the final step, the algorithm aims to choose a reduced subset of the most representative patterns that covers the same amount of time series as the full set. We use a greedy approach to successively look up and keep the most strong patterns. Algorithm 4.5 shows the steps of this technique to get compact pattern set. It lists all time series that comprise at least one pattern $\mathbf{TS_P}$ from aggregated set obtained on the previous stage (line 2). Then, by looping over patterns it selects the one $\hat{\boldsymbol{p}}$ that is contained in the most of time series. The subset of time series covered by $\hat{\boldsymbol{p}}$ are subtracted from $\mathbf{TS_P}$ (line 3 - 6). The procedure continues until we get an empty set of $\mathbf{TS_P}$.

The final set of tailored patterns are the feature based representation of the training set of time series data. Now, this set can be used as a pattern split operator within TDT. Each split candidate in a node is a pair that contains a pattern $\boldsymbol{p}$ and an attribute ν taken from the set $\{\nu\}$ based on the collected statistics. The complexity of a bag of patterns generating is $O(Nm)$, the cost of the patterns aggregation is $O(m(4^{|p|}))$ and the tailoring step takes $O(m^2)$. Hence the *HARP* algorithm complexity is $O(m(N + 4^{|p|}) + m^2)$.

ALGORITHM 4.1: HARP algorithm.

Function *HARP*:
  Input: $(\mathbf{TS}, \mathcal{Y})$, an alphabet a, a size of sliding window $1 \leq q \leq m-1$, coverage and purity thresholds $\eta^{cov}, \eta^{purity}$
  Result: $\mathbf{P} := \{(\boldsymbol{p}, \{\nu\})\}$
  $\mathbf{P} \leftarrow PatternsGenerator((\mathbf{TS}, \mathcal{Y}), a, q, \eta^{cov}, \eta^{purity})$; `// getting bag of patterns`
  $\mathbf{P}^{agg} \leftarrow PatternsAggregation(\mathbf{P}, \eta^{purity})$; `// getting extended set with aggregated patterns`
  $\mathbf{P} \leftarrow PatternsTiling((\mathbf{TS}, \mathcal{Y}); \mathbf{P}^{agg})$; `// getting tailored pattern set`
end

4.1.2 On combination of mono-split TDT operators

As we mentioned in previous chapter, a prior selection of a split operator imposes the assumption of underlying data classes structure. However, without a prior knowledge on how class labels at a given node are distributed, it is difficult to make the right choice which split operator to employ. This problem results in large trees as the algorithm with the pre-selected type of split operator attempts to approximate regions that are not naturally separable by it. On the first part of the Figure 4.1, it is shown an example of distribution of class labels on an orderline of distances between split time series and a time series from the training set. On the second part, training time series are represented by their labels as distance-coordinates to two time series ts_L and ts_R. By choosing a hypersphere manner of a split represented by a time series ts_L and the threshold θ, we can end up with purer data partition than if a hyperplane split operator would be chosen. To cope with such situations, we have found useful to suggest examination set of all split operators instead of a priori fixing the particular one.

We propose the evaluation of split candidates generated from the following set of operators: hyperplane split operator $\mathcal{F}(H)$, hypersphere operator $\mathcal{F}(S)$, and feature-based operator $\mathcal{F}(\mathbf{P})$. The last one is based on exploitation the *HARP* algorithm introduced in Section 4.1.1. Since each pattern discovered by *HARP* is a segment of real time series from the training set, we claim that using this designed feature based approach is the most appropriate for approaching the goal of interpretable classification.

Algorithm 4.6 introduces steps of a node split with generalized Multi-operator Temporal Decision Tree. Here, among three types of split operators, the algorithm selects the best one.

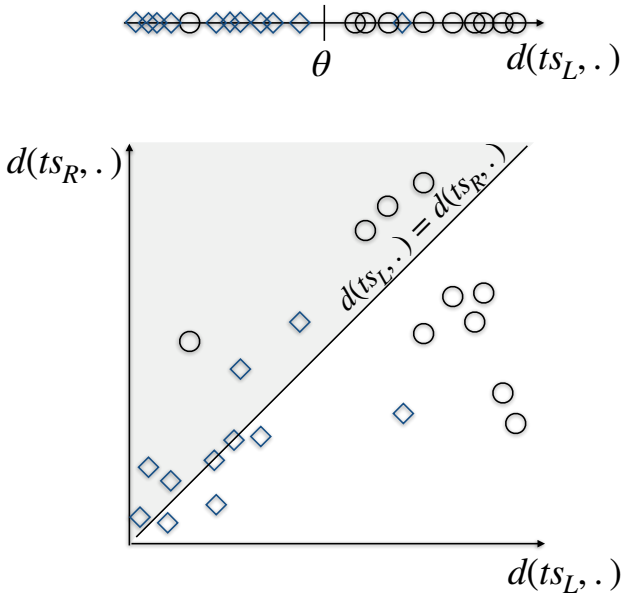

Figure 4.1: Illustration of the case when class labels distribution is such that it can be easily separated by split of a time series and the threshold (ts_L, θ) (hypersphere split operator) but not by a pair of time series (ts_L, ts_R). Each point represents a distance between split time series and a time series from the training set.

Allowing examination of three heterogeneous split operators widen the range of data geometrical properties that the algortihm is capable of capturing. At the same time, we exclude some limitations in the learning process. By increasing the number of split options we increase the complexity of the algorithm with respect to the number of evaluated split candidates. However, assessment of all split candidates generated from hyperplane and hypersphere split operators requires computation of only one pairwise distance matrix. Therefore, MTDT has the same complexity as TDT with respect to the number of distance computations. MTDT with patterns, hyperplane and hypersphere split operators has the complexity $O(\lambda \log_{\frac{1}{\alpha}}(m)(|\boldsymbol{P}|N \log N + N^3 + N^2 \log N))$, that is equal to $O(\lambda \log_{\frac{1}{\alpha}}(m)N^3 \log N)$.

4.2 Empirical study

4.2.1 Experimental settings

We carried a set of experiments to evaluate whether Multi-operator Temporal Decision Trees (MTDT) improves output readability in the terms of size of trees when compared with TDT while keeping the same accuracy. In our experiments we used 46 univariate time series datasets provided by the UCR benchmark [15]. The datasets originate from different sources and include transformed to time series images, measurements

from sensor data, spectrographs, and simulated datasets. We sorted all UCR datasets by increasing order of complexity computed as the number of training time series multiplyed by its length($O(Nm)$) and selected top 46 datasets. Characteristics of these datasets are revealed in Table 4.1.

The first series of the experiments have been done on the split of data on train and test sets provided by UCR. The measured characteristics of learned trees are the number of nodes and the classification accuracy. Then, to work out the sustainability of obtained results (Table 4.4 and 4.5), we run 10 folds on each of dataset keeping the first fold the same as UCR split on training and test sets. The rest 9 folds are obtained by bootstrap method and kept the class labels distribution of the original UCR split. We compare mono-operator TDT with different possible split operators, which are hyperplane, hypersphere and patterns, with Multi-operator TDT. We compare our results with the baseline algorithm Hyperplane TDT, proposed by Chouakria and Amblard [16] with and without dichotomy search (Section 3.2.2), denoted in the tables as H_b, H_b^d accordingly. We also kept the adaptive metric DTW-Cort_π (Section 2.3.2.4) as was proposed in [16] with three possible values of the parameter λ. The value $\lambda = 0$ corresponds to value-based distance DTW, $\lambda = 6$ to shape-based distance, while the value $\lambda = 3$ takes into account both notions. As an alternative to adaptive metric, we test Hyperplane TDT with the range of distance measures that are the l_p (Section 2.3.2.2) norm distance with p being 1 or 2, DTW, $1 - \mathrm{Cort}_\pi$ (Section 2.3.2.3), and value-shape-based $(1-\mathrm{Cort}_\pi)*$DTW. The latest one is comparable to DTW-Cort_π with $\lambda = 3$, and selected for the purpose of avoiding computation of the exponential component. For MTDT, we examine the learned trees readability and accuracy performance by adding hypersphere and pattern split operators to the hyperplane operator.

4.2.2 Results and Discussion

Results on accuracy and the size of learned trees on the unique train and test split are shown in Table 4.2 and Table 4.3. Summary of experimental results conducted on 10 times resampled datasets are revealed in Table 4.4 and Table 4.5. Histograms with summarized results over 46 datasets obtained by TDT and MTDT are shown on Figure 4.2 and 4.3. In the resulting tables, the name of dataset is shown in bold if *HARP* did not detect any patterns. Therefore, for these datasets results of the experiment with use hyperplane and hypersphere split operator, $H + S$, are equals to those obtained with three split operators $H + S + P$.

In the results comparison of mono-operator Hyperplane TDT with adaptive metrics versus the range of metric, one should note that the average total number of non-terminal nodes is reduced by 23,9% (1238 versus 941) on unit train/test split data. Looking at the resampled results, we can see that the tendency of trees size reduction is consistent. In the average, we improved on 24,9% (1233,9 versus 926,5). The main difference between the adaptive similarity measure and the used range of distances is inclusion of l_p distance, which has a positive impact on trees readability for 41 out of 46 datasets. The baseline Hyperplane TDT with dichotomy search is more effective and improves results in the both terms: accuracy and readability. However, it brings significant increase in terms of computational complexity since for each sliced time series interval we need to recompute a distance matrix, which has the complexity of $O\Big((\alpha m N)^2\Big)$.

We observe much higher decrease in trees size by adding a hypersphere split operator: on 44,3% (690 over 1238) and on 43,3% (699,6 over 1233,9) for the unit train test data split and resampled results respectively. This empirical evidence highlights the importance to capture different underlying data structures. However, the average accuracy is slightly decreased in compare with hyperplane TDT that employs the range of similarity measures rather than the adaptive DTW-Cort$_\pi$ (72,4% TDT($\mathcal{F}$(S)) and 73,6% TDT($\mathcal{F}$(H)) Table 4.3; 72,8% TDT($\mathcal{F}$(S)) and 73,6% TDT($\mathcal{F}$(H)) Table 4.5).

By using the combination of spherical and hyperplane split operators ($\mathcal{F}$(H) + $\mathcal{F}$(S)), the total number of non-terminal nodes amount to the average value of 656,4 nodes, that is two times less than for baseline algorithm H$_b$ with 1233,9. Additionaly, we also see the improvement in terms of accuracy with the average 75,3% for $\mathcal{F}$(H) + $\mathcal{F}$(S) over 73,1% for H$_b$. As a result of adding patterns extracted by *HARP* to MTDT as a split operator, the total number of nodes drops to 529,2, i.e., MTDT built trees of the size on 57,1% less than the baseline TDT (H$_b$). Clearly, this shows strong positive influence on tree's readability as well as on the algorithm classification performance (76,8% for $\mathcal{F}$(H) + $\mathcal{F}$(S)+$\mathcal{F}(P)$).

By visualizing constructed tree, a user is able to analyze the decision process of data separation and interpret a class prediction of a new time series. Graphical representation of each non-terminal node includes an interactive view on a selected split operator allowing quick understanding of the most discriminating time series or pattern. Figure 4.4 reveals the model of learned tree on "Car" UCR dataset by MTDT algorithm. The tree is composed by 6 internal nodes:split tests of 3 nodes are represented by hyperplane split operator and three others are represented by

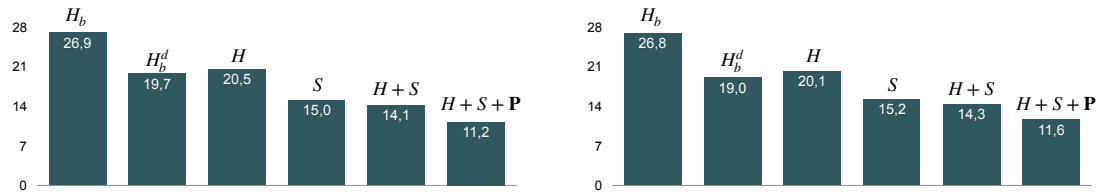

Figure 4.2: Histogram represents the average number of nodes for trees obtained on 46 UCR datasets by TDT and MTDT. The left plot: results on unit train/test data partition. The right plot: results on 10 bootstrap cross-validation. Each bar corresponds to employed split operators: hyperplane (H, H_b - the baseline algorithm, d stands for dichotomy search), hypersphere (S), and patterns (**P**).

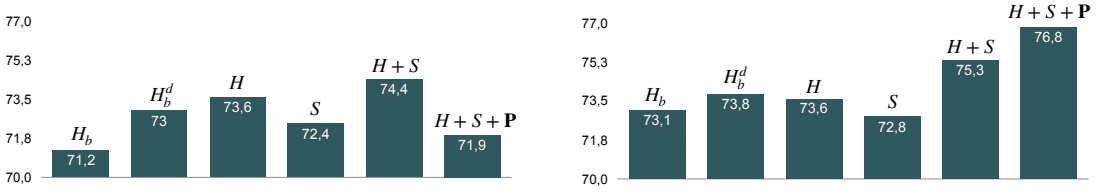

Figure 4.3: Histograms represents the average accuracy for trees obtained on 46 UCR datasets by TDT and MTDT. The left plot: results on unit train/test data partition. The right plot: results on 10 bootstrap cross-validation. Each bar corresponds to employed split operators: hyperplane (H, H_b - the baseline algorithm, d stands for dichotomy search), hypersphere (S), and patterns (**P**).

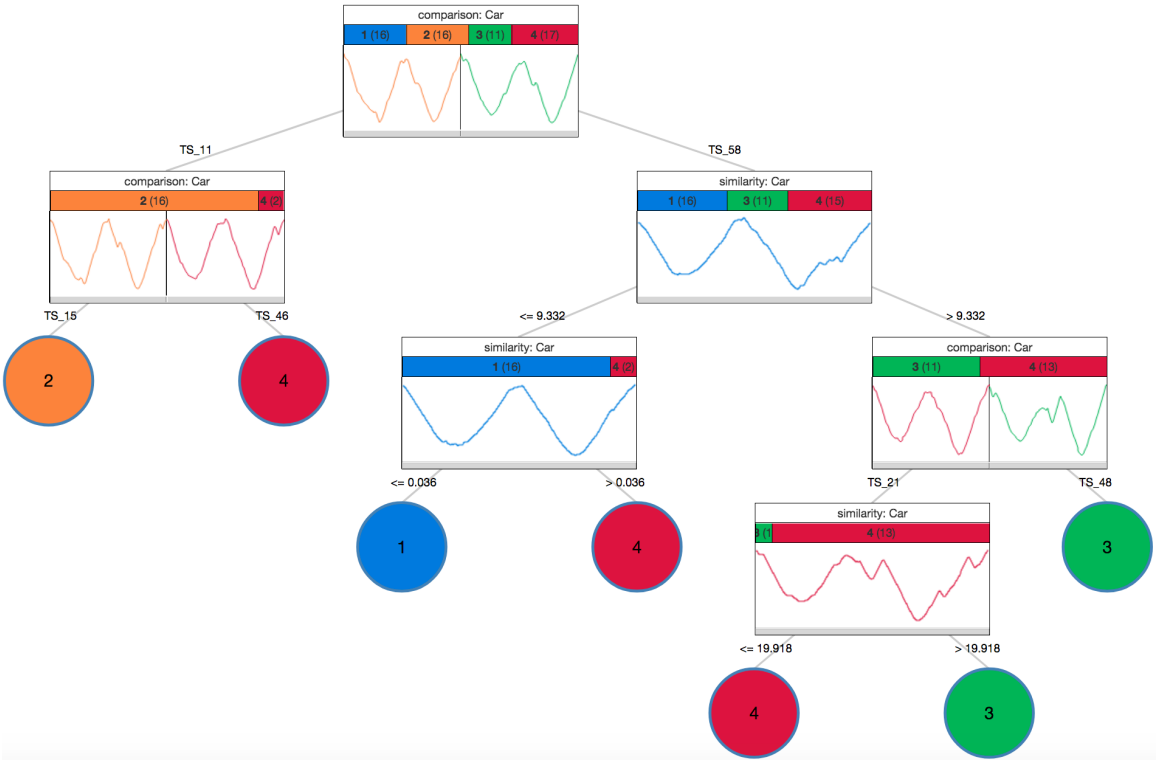

Figure 4.4: Learned tree on Car UCR dataset by MTDT algortihm with split operators configuration $\mathcal{F}(\mathrm{H}) + \mathcal{F}(\mathrm{S}) + \mathcal{F}(P)$. Each non-terminal node depicts the selected split operator and the distribution of class labels highlighted by colors. A split time series are rendered in the color of corresponding class label. Circles represents the leaves, its color and the number in the middle indicates a class label. Visualization displayed by the graphical interface developed in the IKATS project.

hypersphere split. We can notice that 3 leaves contain a class label "4". This observation points out that some time series within one class are less similar between them, than with time series belonging to different classes. Though, majority of time series of the class "4" (13 out of 17) lies in the same leave. Looking at the tree we also can conclude that data has complex geometrical structure. Indeed, we see that selected by learning the best split operators vary from one node to another. In average, for these dataset we obtained the reduction of trees size by factor of 2 (14,5 over 6,6). While for this particular dataset the tree size yielded by the baseline algorithm has the decent quantity of the number of nodes, doubtless it is easier to analyse the concise model. Furthemore, such reduction is more significant for datasets on which large trees were learned ("DistalPhalanxOutputAge", "DistalPhalanxOutputCorrect", "DistalPhalanxTW", "InsectWingbeatSound", "MiddlePhalanxOutlineAgeGroup", "MiddlePhalanxOutlineCorrect", "MiddlePhalanxTW").

4.3 Conclusion

In this chapter, we proposed a generalized algorithm to build the temporal decision tree with multiple split operators. We have obtained a significant reduction of constructed trees size. That confers compactness of our model making its analysis easier and its visual representation better. MTDT is able to capture different geometrical structures of the data class distribution. Besides, it makes split decisions by operators that permit an intuitive explanation of a split and facilitates interpretability of classification process. Similarity-based data separation allows an expert to understand the splitting process at each node by inspecting selected split operator, all of which keeps the link with original training set. It preserves a better connection between initial data and learned model implying easy comprehension of new data classification and validation of obtained results. Visualization of a model along with the interpretable learning algorithm supports transparent framework contributing to a user's understanding how a classification results were obtained.

In forthcoming chapter, we address the problem of reducing the number of examinated split candidates for hyperplane and hypersphere split operators. By pursuing this direction, we aim to reduce computational complexity yet to preserving the same level of interpretability and accuracy.

ALGORITHM 4.2: Algorithm of a pattern set generation.

Function *PatternsGenerator*:
  Input: $(\mathbf{TS}, \mathcal{Y})$, an alphabet a, a size of sliding window $1 \leq q \leq m-1$, coverage and purity thresholds η^{cov}, η^{pur}
  Result: $\mathbf{P} := \{(\boldsymbol{p}, \{\nu\})\}$
  $\mathbf{TS}_a \leftarrow \{ts_i : \{s_i^t : 1 \leq t \leq m\}, i = 1, \ldots, N, s_i \in a\}$// `symbolic representation of time series values`
  $\mathbf{P} \leftarrow \emptyset$; // `set of patterns`
  $\mathbf{H} \leftarrow \{k : \emptyset\}$; // `dictionary of patterns for each class` $k \in \mathcal{Y}$
  for $(ts, k) \in (\mathbf{TS}_a, \mathcal{Y})$ **do**
    for $j \in (1, m-q)$ **do**
      $\boldsymbol{p}_a^j \leftarrow \{s^t : j \leq t \leq j+q-1\}$;
      $\mathbf{P} \leftarrow \{(\boldsymbol{p}_a^j, \{\nu\})\}$;
      $\mathbf{H}[k] \leftarrow (\boldsymbol{p}_a^j, \{\nu\})$;
    end for
  end for
  // `refinement of a pattern set`
  for $(k \in \mathcal{Y})$ **do**
    for $\mathbf{p} \in \mathbf{H}[k]$ **do**
      $e^{cov}(\boldsymbol{p}) \leftarrow$ compute with Formula 4.1 ;
      if $e^{cov}(\mathbf{p}) \leq \eta^{cov}$ **then**
        $\mathbf{P} \leftarrow \mathbf{P} \backslash \boldsymbol{p}$;
        else
          $e^{pur}(\boldsymbol{p}) \leftarrow$ compute by Formula 4.2 ;
          if $e^{pur}(\mathbf{p}) \leq \eta^{pur}$ **then**
            $\mathbf{P} \leftarrow \mathbf{P} \backslash \boldsymbol{p}$;
          end if
      end if
    end for
  end for
end

ALGORITHM 4.3: Algorithm of aggregation the set of patterns.

Function *PatternsAggregation*:
  Input: $\mathbf{P} := \{(\boldsymbol{p}, \{\nu\})\}$, a purity threshold η^{purity}
  Result: $\mathbf{P}^{agg} := \{(\boldsymbol{p}, \{\nu\})\}$
  // extended set with generalized patterns
  $\mathbf{P}^{agg} \leftarrow \mathbf{P}$;
  $\tilde{P} \leftarrow GeneralizePattern(\boldsymbol{p})$;
  for $\tilde{\mathbf{p}} \in \tilde{P}$ **do**
    $e^{pur} \leftarrow$ compute with Formula 4.2 ;
    if $e^{pur} \leq \eta^{pur}$ **then**
      $\tilde{P} \leftarrow \tilde{P} \backslash \tilde{\boldsymbol{p}}$;
    end if
    $\mathbf{P}^{agg} \leftarrow \tilde{\boldsymbol{p}}$;
  end for
end

ALGORITHM 4.4: Algorithm to get generalized patterns.

Function *GeneralizePattern*:
  Input: a pattern $\boldsymbol{p}$
  Result: the set of generalized patterns $\tilde{P}$
  $\tilde{P} \leftarrow \boldsymbol{p}$;
  foreach $a_i \in \mathbf{p}$ **do**
    $a' \leftarrow a_i \downarrow$; // one symbol down from a_i in alphabet A
    $a'' \leftarrow a_i \uparrow$; // one symbol up from a_i in alphabet A
    foreach $\tilde{\mathbf{p}} \in \tilde{P}$ **do**
      $\tilde{P} \leftarrow \tilde{\boldsymbol{p}}[i]$.insert($[a'a_i]$) ;
      $\tilde{P} \leftarrow \tilde{\boldsymbol{p}}[i]$.insert($[a_i a'']$) ;
      $\tilde{P} \leftarrow \tilde{\boldsymbol{p}}[i]$.insert($[a'a'']$) ;
    end foreach
    $\tilde{P} \leftarrow \tilde{P} \backslash \boldsymbol{p}$;
  end foreach
end

ALGORITHM 4.5: Algorithm of tailoring set of patterns to data

Function *PatternsTailoring*:
  Input: $\mathbf{P} := \{(\boldsymbol{p}, \{\nu\})\}$
  Result: $\mathbf{P}^{red} := \{(\boldsymbol{p}, \{\nu\})\}$
  $\mathbf{TS_P} \leftarrow \{ts \in \mathbf{TS} : \exists \boldsymbol{p} \in ts, \boldsymbol{p} \in \mathbf{P}\}$;
  while $\mathbf{TS}_{\boldsymbol{P}} \neq \emptyset$ **do**
    $\hat{p} \leftarrow \underset{\boldsymbol{p}}{\text{argmax}}\{|\mathbf{TS}_{\boldsymbol{p}}| : \{\mathbf{TS}_{\boldsymbol{p}} := \{ts \in \mathbf{TS} \wedge \boldsymbol{p} \in ts\}\}$; `// a pattern covering the maximum number of time series`
    $\mathbf{TS_P} \leftarrow \mathbf{TS_P} \backslash \mathbf{TS}_{\boldsymbol{p}}$;
    $\mathbf{P}^{red} \leftarrow \mathbf{P} \backslash \boldsymbol{p}$;
  end while
end

ALGORITHM 4.6: Algorithm of MTDT Node Split.

Function *MTDTNodeSplit*:
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g, g \in \mathcal{G}; d = \text{DTW-Cort}_{\pi}$
  Result: $s^*(g, f_{(\cdot)}) := (g_L^*, g_R^*, f_{(\cdot)}^*, \lambda^*), \Delta i_{\text{Gini}}(s(g, f_{(\cdot)}))$
  $\Delta i_{\text{Gini}}(s^*(g, f_{(\cdot)})) \leftarrow \infty$;
  $s^*(g, f_H) \leftarrow HyperplaneNodeSplit(\mathbf{TS}, \mathcal{Y})$; ; `// Algorithm 3.1`
  $s^*(g, f_S) \leftarrow HypersphereNodeSplit(\mathbf{TS}, \mathcal{Y})$; ; `// Algorithm 3.2`
  $s^*(g, f_{\boldsymbol{p}}) \leftarrow PatternNodeSplit(\mathbf{TS}, \mathcal{Y})$; ; `// Algorithm 3.3`
  $s^*(g, f_{(\cdot)}) \leftarrow \underset{f_{(\cdot)}}{\text{argmax}}\{\Delta i_{\text{Gini}}(s(g, f(.))), f_{(.)} \in \{f_H, f_S, f_{\boldsymbol{p}}\}\}$;
end

Table 4.1: Summary of datasets.

#	Datasets	Train	Test	Length	Nb classes	Type
1	Adiac	390	391	176	37	image
2	ArrowHead	36	175	251	3	image
3	Beef	30	30	470	5	spectro
4	BeetleFly	20	20	512	2	image
5	BirdChicken	20	20	512	2	image
6	Car	60	60	577	4	sensor
7	CBF	30	900	128	3	simulated
8	CincECGtorso	40	1380	1639	4	sensor
9	DisPhOutAge	400	139	80	3	image
10	DisPhOutCor	600	276	80	2	image
11	DisPhTW	400	139	80	6	image
12	Earthquakes	322	139	512	2	sensor
13	ECG200	100	100	96	2	sensor
14	ECG5000	500	4500	140	5	sensor
15	ECGFiveDs	23	861	136	2	sensor
16	FaceFour	24	88	350	4	image
17	FacesUCR	200	2050	131	14	image
18	GunPoint	50	150	150	2	motion
19	Ham	109	105	431	2	spectro
20	Herring	64	64	512	2	image
21	InsectWngS	220	1980	256	11	sensor
22	ItalyPowDem	67	1029	24	2	sensor
23	Lightning2	60	61	637	2	sensor
24	Lightning7	70	73	319	7	sensor
25	MALLAT	55	2345	1024	8	simulated
26	Meat	60	60	448	3	spectro
27	MedicalImag	381	760	99	10	image
28	MidPhAge	400	154	80	3	image
29	MidPhCor	600	291	80	2	image
30	MidPhTW	399	154	80	6	image
31	MoteStrain	20	1252	84	2	sensor
32	OliveOil	30	30	570	4	sensor
33	ProxPhAge	400	205	80	3	image
34	ProxPhCor	600	291	80	2	image
35	ProxPhTW	400	205	80	6	image
36	ShapeletSim	20	180	500	2	simulated
37	SonyAIBO1	20	601	70	2	sensor
38	SwedishLeaf	500	625	128	15	image
39	SynControl	300	300	60	6	simulated
40	ToeSegm1	40	228	277	2	motion
41	ToeSegm2	36	130	343	2	motion
42	Trace	100	100	275	4	sensor
43	TwoLdECG	23	1139	82	2	sensor
44	Wine	57	54	234	2	spectro
45	Worms	181	77	900	5	motion
46	Worms2Class	181	77	900	2	motion

Table 4.2: The number of non-terminal nodes for 46 UCR datasets for mono-operator and Multi-operator Temporal Decision Trees (TDT and MTDT). Split operators are hyperplane (H, H_b - the baseline algorithm, d stands for dichotomy search), hypersphere (S), and patterns(**P**). Training and models evaluation are done on unit train and test UCR split. For datasets shown in bold results for H+S equals to H+S+P.

		TDT				MTDT	
#	Datasets	H_b	H_b^d	H	S	H+S	H+S+**P**
TOTAL		1238	904	941	690	648	**516**
AVERAGE		26.9	19.7	20.5	15.0	14.1	**11.2**
1	**Adiac**	145	122	102	83	**78**	**78**
2	ArrowHead	11	6	5	**4**	**4**	**4**
3	Beef	13	8	12	8	7	**6**
4	BeetleFly	6	3	3	**2**	**2**	**2**
5	BirdChicken	2	2	3	2	2	**1**
6	Car	10	8	7	10	6	6
7	CBF	**2**	**2**	**2**	3	**2**	**2**
8	**CincECGtor**	18	**7**	8	**7**	**7**	**7**
9	DistPhOutAge	29	19	26	15	15	**8**
10	DistPhCor	61	40	43	**25**	**25**	**25**
11	DistPhTW	36	30	30	22	22	**9**
12	Earthquakes	27	20	22	14	12	**7**
13	ECG200	17	7	8	7	6	**5**
14	ECG5000	44	26	29	20	**19**	**19**
15	ECGFiveDs	3	3	3	3	3	**2**
16	FaceFour	4	**3**	4	**3**	**3**	**3**
17	FacesUCR	20	28	24	24	23	**12**
18	GunPoint	6	3	3	4	3	**2**
19	**Ham**	17	15	21	13	**12**	**12**
20	**Herring**	20	12	21	10	**8**	**8**
21	**InsectWngS**	96	56	65	**47**	51	51
22	ItalyPowerDem	3	4	4	3	**2**	**2**
23	**Lighting2**	7	9	8	**5**	**5**	**5**
24	Lighting7	13	9	11	10	10	**7**
25	MALLAT	8	**7**	**7**	**7**	**7**	**7**
26	Meat	**2**	**2**	**2**	3	**2**	**2**
27	MedicalImag	88	72	84	59	**56**	**56**
28	MidPhAge	58	28	46	31	27	**12**
29	MidPhCor	67	48	34	26	**23**	**23**
30	MidPhTW	55	45	47	33	32	**14**
31	**MoteStrain**	2	**1**	4	2	2	2
32	OliveOil	5	4	**3**	**3**	**3**	**3**
33	ProxPhAge	63	49	48	28	30	**20**
34	ProxPhCor	99	62	61	42	40	**14**
35	ProxPhTW	45	28	36	21	21	**10**
36	ShapeletSim	2	2	3	2	3	**1**
37	SonyAIBO1	2	**1**	**1**	**1**	**1**	**1**
38	**SwedishLeaf**	55	47	44	38	**33**	**33**
39	SynControl	**5**	6	**5**	8	**5**	**5**
40	ToeSegm1	10	**4**	9	**4**	**4**	**4**
41	ToeSegm2	**2**	5	3	**2**	**2**	**2**
42	Trace	**4**	**4**	**4**	5	**4**	**4**
43	TwoLdECG	3	**1**	**1**	2	**1**	**1**
44	**Wine**	10	12	5	6	**4**	**4**
45	Worms	26	21	21	16	14	**12**
46	Worms2Class	17	13	9	7	7	**6**

Table 4.3: The classification accuracy (%) of trees for 46 UCR datasets for mono-operator and Multi-operator Temporal Decision Tree (TDT and MTDT). The split operators are hyperplane (H, H_b - the baseline algorithm, d stands for dichotomy search), hypersphere (S), and patterns (**P**). For datasets shown in bold results for H+S equals to H+S+P. Training and models evaluation are done on unit train and test UCR split.

		TDT				MTDT	
#	Datasets	H_b	H_b^d	H	S	H+S	H+S+**P**
AVERAGE		0.712	0.730	0.736	0.724	**0.744**	0.719
1	**Adiac**	0.49	0.45	0.54	0.60	**0.64**	**0.64**
2	ArrowHead	0.64	0.67	**0.71**	0.62	0.62	0.62
3	Beef	0.53	0.53	0.50	0.57	**0.60**	0.53
4	BeetleFly	0.65	0.65	0.85	0.75	**0.95**	0.90
5	BirdChicken	0.80	0.80	**0.85**	0.65	0.80	0.80
6	Car	0.57	0.57	0.65	0.65	0.67	**0.70**
7	CBF	**0.98**	0.94	0.95	0.91	0.95	0.96
8	**CincECGtor**	0.57	**0.66**	**0.66**	0.57	0.58	0.58
9	DistPhOutAge	0.73	0.72	**0.78**	**0.78**	0.77	0.65
10	DistPhOutCor	0.67	0.72	0.68	**0.81**	**0.81**	**0.81**
11	DistPhTW	0.68	0.68	**0.76**	0.72	0.73	0.74
12	Earthquakes	**0.71**	0.66	0.64	0.68	0.67	0.47
13	ECG200	0.75	0.76	**0.82**	0.77	0.77	0.80
14	ECG5000	0.90	0.90	0.91	0.92	**0.93**	**0.93**
15	ECGFiveDays	0.68	0.67	**0.78**	0.66	0.64	0.53
16	FaceFour	0.69	**0.84**	0.65	0.78	0.65	0.65
17	FacesUCR	**0.76**	0.74	**0.76**	0.71	0.75	0.54
18	GunPoint	0.86	0.86	**0.94**	0.87	0.81	0.73
19	Ham	0.60	**0.67**	0.65	0.63	0.61	0.61
20	Herring	0.56	0.55	0.44	**0.59**	0.52	0.52
21	InsectWngS	0.41	0.45	0.53	0.55	**0.57**	**0.57**
22	ItalyPowDem	0.93	0.87	**0.97**	0.93	**0.97**	0.95
23	**Lighting2**	0.77	**0.87**	0.69	0.70	0.70	0.70
24	Lighting7	**0.67**	0.55	0.62	0.56	0.63	0.51
25	MALLAT	0.80	0.86	**0.96**	0.80	0.88	0.88
26	Meat	0.83	**0.95**	0.92	0.70	0.92	0.85
27	MedicalImag	0.64	**0.68**	0.53	0.63	0.62	0.62
28	MidPhAge	0.71	0.62	0.71	**0.72**	**0.72**	0.57
29	MidPhCor	0.70	0.69	0.68	0.72	**0.76**	**0.76**
30	MidPhTW	**0.61**	0.51	0.57	0.57	0.58	**0.61**
31	**MoteStrain**	0.86	**0.91**	0.85	0.83	0.81	0.81
32	OliveOil	0.77	**0.83**	0.70	0.77	0.77	0.70
33	ProxPhAge	**0.80**	0.74	0.79	0.77	0.77	0.69
34	ProxPhCor	0.76	0.72	0.82	0.82	**0.83**	0.74
35	ProxPhTW	0.70	0.69	0.68	**0.74**	0.73	**0.74**
36	ShapeletSim	0.59	0.62	0.64	0.64	0.58	**0.94**
37	SonyAIBO1	0.76	**0.85**	0.82	0.71	0.71	0.82
38	**SwedishLeaf**	0.80	0.8	0.83	0.83	**0.86**	**0.86**
39	SynControl	**0.96**	**0.96**	0.94	0.94	**0.96**	**0.96**
40	ToeSegm1	0.72	0.75	0.73	0.78	**0.80**	0.60
41	ToeSegm2	0.74	**0.82**	0.65	**0.82**	**0.82**	**0.82**
42	Trace	0.97	**0.99**	0.98	**0.99**	0.98	**0.99**
43	TwoLdECG	0.90	**0.96**	0.94	0.81	0.94	0.93
44	**Wine**	0.50	**0.85**	0.76	0.69	0.78	0.78
45	Worms	0.38	**0.45**	0.38	0.37	0.42	0.38
46	Worms2Class	0.65	0.56	**0.66**	**0.66**	**0.66**	0.60

Table 4.4: The tree size in terms of the number non-terminal nodes for 46 UCR dataset for mono-operator and Multi-operator Temporal Decision Tree (TDT and MTDT). The split operators are hyperplane (H, H_b - the baseline algorithm, d stands for dichotomy search), hypersphere (S), and patterns (**P**). For datasets shown in bold results for H+S equals to H+S+**P**. Training and models evaluation is done on 10 bootstrap cross-validation.

		TDT				MTDT	
#	Datasets	H_b	H_b^d	H	S	H+S	H+S+**P**
Total		1233.9	874.4	926.5	699.6	656.4	529.2
Average		26.8	19.0	20.1	15.2	14.3	11.6
1	Adiac	144.3± 6.9	114.0± 6.2	92.8± 5.8	81.0± 4.5	76.7± 4.0	76.7± 4.0
2	ArrowHead	7.9± 2.1	5.6± 1.3	5.5± 1.0	4.9± 0.8	4.4± 0.7	4.1± 0.7
3	Beef	12.8± 2.3	9.5± 1.6	9.2± 1.6	9.0± 1.5	7.4± 1.1	7.6± 1.1
4	BeetleFly	4.3± 1.6	2.8± 0.9	2.6± 1.0	2.0± 0.6	2.0± 0.6	1.7± 0.5
5	BirdChicken	3.2± 1.0	2.3± 0.8	2.9± 0.8	2.7± 0.8	2.4± 0.7	1.5± 0.5
6	Car	14.5± 2.9	8.8± 1.3	10.1± 2.1	9.7± 1.5	7.8± 1.5	6.6± 1.4
7	CBF	2.1± 0.3	2.0± 0.0	2.1± 0.3	2.5± 0.7	2.0± 0.0	2.1± 0.3
8	**CincECGtor**	13.7± 2.4	5.9± 1.5	9.5± 1.9	7.5± 1.6	7.5± 1.3	7.5± 1.3
9	DistPhOutAge	26.3± 4.6	18.9± 3.8	21.2± 3.3	13.8± 1.3	13.0± 1.5	8.3± 1.6
10	DistPhOutCor	53.5± 4.6	33.5± 4.2	41.4± 2.5	23.1± 4.0	23.6± 3.2	14.1± 4.0
11	DistPhTW	37.2± 5.4	26.8± 3.2	28.5± 2.7	22.4± 1.4	20.9± 1.4	17.0± 3.8
12	Earthquakes	28.4± 5.1	20.1± 2.0	21.9± 2.5	14.0± 1.1	13.8± 1.9	10.0± 2.0
13	ECG200	14.0± 2.4	9.5± 2.3	10.8± 1.7	6.6± 1.2	6.5± 1.5	4.5± 1.0
14	ECG5000	38.0± 6.1	24.9± 4.7	27.8± 5.1	21.6± 3.3	19.9± 4.1	7.9± 4.0
15	ECGFiveDs	4.5± 1.6	2.3± 0.9	2.7± 0.9	3.0± 0.8	2.5± 0.8	2.5± 0.5
16	FaceFour	3.3± 0.5	3.1± 0.3	3.1± 0.3	3.2± 0.4	3.0± 0.0	3.0± 0.0
17	FacesUCR	27.5± 3.8	26.4± 2.5	29.3± 3.7	27.8± 3.1	24.4± 2.1	22.0± 3.9
18	GunPoint	4.1± 1.4	2.9± 0.9	3.4± 1.3	3.1± 0.7	2.6± 0.7	1.8± 0.9
19	**Ham**	21.0± 3.2	14.0± 2.4	17.5± 2.8	12.6± 1.1	11.3± 0.8	11.3± 0.8
20	**Herring**	18.3± 2.2	12.2± 2.6	12.5± 3.7	8.7± 0.6	8.8± 0.6	8.8± 0.6
21	**InsectWngS**	93.5± 4.2	56.0± 3.2	59.8± 4.4	48.9± 4.3	47.6± 2.7	47.6± 2.7
22	ItalyPowDem	3.1± 1.7	2.5± 1.2	1.9± 1.3	2.7± 0.9	1.4± 0.5	1.2± 0.4
23	**Lighting2**	8.5± 1.9	7.0± 1.5	8.2± 2.7	5.4± 1.2	5.3± 1.3	5.3± 1.3
24	Lighting7	12.5± 1.3	10.1± 1.6	12.1± 1.4	10.9± 0.8	10.3± 1.1	9.3± 1.4
25	MALLAT	7.5± 0.9	7.3± 0.5	7.2± 0.4	7.2± 0.4	7.2± 0.4	7.4± 0.5
26	Meat	3.5± 1.2	2.6± 0.9	2.0± 0.0	4.0± 0.8	2.0± 0.0	2.0± 0.0
27	MedicalImag	96.8± 8.4	69.5± 6.1	84.7± 7.4	59.5± 4.4	58.6± 2.6	47.3± 3.8
28	MidPhAge	41.4± 6.4	29.1± 2.4	31.1± 5.9	22.0± 3.4	21.2± 2.6	14.1± 2.9
29	**MidPhCor**	66.4± 7.4	41.3± 5.6	44.5± 6.1	29.4± 3.4	27.0± 3.0	27.0± 3.0
30	MiddPhTW	55.5± 4.4	42.6± 2.6	45.5± 3.0	34.5± 1.6	33.2± 1.5	25.8± 4.5
31	**MoteStrain**	1.6± 0.7	1.5± 0.5	1.7± 1.0	1.7± 0.6	1.5± 0.5	1.5± 0.5
32	OliveOil	5.6± 1.4	3.7± 0.8	4.8± 1.5	3.6± 0.7	3.6± 0.7	3.5± 0.5
33	ProxPhAge	75.0± 7.0	49.5± 4.9	52.5± 5.5	31.4± 3.7	31.1± 2.9	10.5± 3.5
34	ProxPhCor	93.3± 7.1	64.7± 6.4	68.5± 7.4	41.9± 3.9	40.5± 2.7	21.6± 3.7
35	ProxPhTW	46.6± 5.4	30.6± 4.1	33.5± 3.5	22.1± 2.1	21.1± 1.4	13.3± 2.1
36	ShapeletSim	3.3± 1.5	2.8± 0.8	3.4± 0.5	2.2± 0.4	2.4± 0.5	1.0± 0.0
37	SonyAIBO1	1.2± 0.4	1.0± 0.0	1.3± 0.6	1.3± 0.5	1.2± 0.4	1.0± 0.0
38	**SwedishLeaf**	57.4± 4.2	45.1± 5.1	42.5± 4.8	36.6± 3.1	32.0± 1.3	32.0± 1.3
39	SynControl	5.8± 0.9	5.6± 0.7	5.6± 0.7	8.5± 0.9	5.6± 0.7	5.0± 0.0
40	ToeSegm1	6.1± 2.4	4.3± 1.0	5.9± 1.8	3.4± 1.0	3.3± 1.0	3.2± 0.9
41	ToeSegm2	5.4± 1.9	4.1± 1.3	4.6± 2.0	3.2± 1.0	3.2± 1.0	3.2± 1.2
42	Trace	3.5± 0.5	3.5± 0.5	3.5± 0.5	4.2± 1.0	3.5± 0.5	3.1± 0.3
43	TwoLdECG	1.5± 1.0	1.3± 0.6	1.0± 0.0	2.1± 0.7	1.0± 0.0	1.0± 0.0
44	**Wine**	14.6± 2.5	10.5± 2.6	10.0± 2.6	7.5± 0.7	6.9± 1.4	6.9± 1.4
45	Worms	27.6± 2.9	20.3± 1.6	23.1± 2.3	17.5± 1.2	16.7± 1.4	14.5± 1.3
46	Worms2Class	18.0± 3.1	12.5± 1.4	12.7± 3.4	8.5± 1.1	8.5± 1.0	6.9± 1.7

Table 4.5: The classification accuracy (%) of trees obtained on the test set of 46 UCR datasets for mono-operator and multi-operator temporal decision tree (TDT and MTDT). The split operators are hyperplane (H, H_b - the baseline algorithm, d stands for dichotomy search), hypersphere (S), and patterns (**P**). For highlighted dataset results for H+S equals to H+S+**P**. Training and models evaluation is done on 10 bootstrap cross-validation.

		TDT				MTDT	
#	Datasets	H_b	H_b^d	H	S	H+S	H+S+**P**
Average		0.731	0.738	0.736	0.728	0.753	0.768
1	**Adiac**	0.47± 0.02	0.47± 0.02	0.53± 0.04	0.57± 0.02	0.58± 0.03	0.58± 0.03
2	ArrowHead	0.69± 0.04	0.69± 0.04	0.74± 0.05	0.65± 0.04	0.70± 0.06	0.73± 0.08
3	Beef	0.49± 0.10	0.53± 0.12	0.53± 0.07	0.48± 0.10	0.60± 0.10	0.56± 0.06
4	BeetleFly	0.68± 0.14	0.68± 0.13	0.73± 0.14	0.71± 0.12	0.71± 0.12	0.76± 0.06
5	BirdChicken	0.76± 0.10	0.79± 0.08	0.75± 0.10	0.70± 0.07	0.74± 0.11	0.83± 0.08
6	Car	0.65± 0.07	0.67± 0.06	0.64± 0.09	0.64± 0.04	0.68± 0.07	0.68± 0.07
7	CBF	0.94± 0.03	0.94± 0.02	0.92± 0.05	0.85± 0.04	0.91± 0.04	0.92± 0.02
8	**CincECGtor**	0.53± 0.05	0.70± 0.05	0.62± 0.04	0.62± 0.04	0.60± 0.07	0.60± 0.07
9	DistPhOutAge	0.74± 0.03	0.74± 0.02	0.75± 0.03	0.77± 0.02	0.77± 0.02	0.78± 0.05
10	DistPhOutCor	0.72± 0.03	0.73± 0.03	0.74± 0.02	0.77± 0.03	0.77± 0.02	0.81± 0.02
11	DistPhTW	0.70± 0.03	0.70± 0.03	0.71± 0.03	0.71± 0.02	0.72± 0.02	0.74± 0.02
12	Earthquakes	0.68± 0.04	0.68± 0.04	0.68± 0.04	0.68± 0.04	0.68± 0.03	0.71± 0.09
13	ECG200	0.76± 0.04	0.76± 0.04	0.78± 0.04	0.80± 0.03	0.82± 0.04	0.80± 0.04
14	ECG5000	0.89± 0.01	0.89± 0.01	0.90± 0.01	0.92± 0.01	0.92± 0.01	0.92± 0.01
15	ECGFiveDs	0.71± 0.04	0.79± 0.08	0.82± 0.06	0.64± 0.07	0.81± 0.08	0.75± 0.10
16	FaceFour	0.79± 0.06	0.79± 0.06	0.75± 0.08	0.69± 0.11	0.74± 0.10	0.81± 0.07
17	FacesUCR	0.76± 0.02	0.76± 0.02	0.74± 0.02	0.71± 0.03	0.74± 0.03	0.70± 0.07
18	GunPoint	0.91± 0.03	0.92± 0.03	0.87± 0.05	0.83± 0.06	0.85± 0.04	0.86± 0.06
19	**Ham**	0.68± 0.07	0.70± 0.05	0.69± 0.04	0.64± 0.05	0.73± 0.06	0.73± 0.06
20	**Herring**	0.61± 0.07	0.58± 0.05	0.54± 0.06	0.59± 0.04	0.61± 0.05	0.61± 0.05
21	**InsectWngS**	0.39± 0.01	0.47± 0.02	0.51± 0.02	0.52± 0.02	0.55± 0.02	0.55± 0.02
22	ItalyPowDem	0.90± 0.03	0.90± 0.03	0.95± 0.02	0.92± 0.02	0.95± 0.01	0.95± 0.02
23	**Lighting2**	0.76± 0.06	0.75± 0.09	0.73± 0.06	0.76± 0.03	0.74± 0.04	0.74± 0.04
24	Lighting7	0.68± 0.06	0.66± 0.06	0.62± 0.07	0.64± 0.09	0.62± 0.06	0.62± 0.06
25	MALLAT	0.87± 0.03	0.85± 0.06	0.90± 0.04	0.88± 0.04	0.90± 0.03	0.85± 0.04
26	Meat	0.94± 0.05	0.93± 0.04	0.96± 0.02	0.87± 0.07	0.96± 0.02	0.92± 0.05
27	MedicalImag	0.66± 0.02	0.65± 0.02	0.61± 0.04	0.65± 0.02	0.67± 0.03	0.64± 0.03
28	MidPhAge	0.65± 0.04	0.65± 0.03	0.65± 0.05	0.69± 0.03	0.69± 0.03	0.71± 0.06
29	**MidPhCor**	0.68± 0.03	0.68± 0.04	0.71± 0.02	0.73± 0.02	0.76± 0.03	0.76± 0.03
30	MiddPhTW	0.52± 0.04	0.53± 0.02	0.53± 0.02	0.56± 0.03	0.56± 0.04	0.59± 0.02
31	**MoteStrain**	0.85± 0.05	0.84± 0.05	0.85± 0.03	0.78± 0.10	0.84± 0.03	0.84± 0.03
32	OliveOil	0.81± 0.05	0.83± 0.07	0.77± 0.07	0.78± 0.10	0.81± 0.05	0.78± 0.07
33	ProxPhAge	0.75± 0.03	0.74± 0.02	0.76± 0.03	0.81± 0.03	0.80± 0.04	0.89± 0.06
34	ProxPhCor	0.78± 0.03	0.77± 0.03	0.80± 0.03	0.82± 0.03	0.82± 0.03	0.84± 0.04
35	ProxPhTW	0.69± 0.02	0.70± 0.03	0.70± 0.03	0.74± 0.03	0.75± 0.03	0.79± 0.04
36	ShapeletSim	0.63± 0.05	0.62± 0.04	0.57± 0.06	0.64± 0.07	0.58± 0.09	0.99± 0.02
37	SonyAIBO1	0.81± 0.07	0.83± 0.07	0.80± 0.04	0.85± 0.08	0.80± 0.06	0.80± 0.05
38	**SwedishLeaf**	0.81± 0.01	0.82± 0.02	0.82± 0.03	0.82± 0.02	0.84± 0.01	0.84± 0.01
39	SynControl	0.96± 0.01	0.96± 0.01	0.96± 0.01	0.95± 0.01	0.97± 0.01	0.94± 0.01
40	ToeSegm1	0.75± 0.03	0.73± 0.03	0.74± 0.04	0.78± 0.06	0.80± 0.05	0.79± 0.07
41	ToeSegm2	0.78± 0.06	0.76± 0.05	0.76± 0.07	0.73± 0.08	0.72± 0.10	0.75± 0.06
42	Trace	0.98± 0.02	0.97± 0.02	0.95± 0.02	0.96± 0.02	0.95± 0.02	0.96± 0.03
43	TwoLdECG	0.91± 0.03	0.91± 0.03	0.90± 0.06	0.77± 0.06	0.90± 0.05	0.93± 0.02
44	**Wine**	0.83± 0.13	0.80± 0.10	0.79± 0.07	0.76± 0.06	0.79± 0.07	0.79± 0.07
45	Worms	0.46± 0.05	0.47± 0.03	0.46± 0.05	0.49± 0.05	0.49± 0.05	0.48± 0.05
46	Worms2Class	0.62± 0.04	0.61± 0.05	0.62± 0.03	0.65± 0.03	0.66± 0.03	0.65± 0.04

Chapter 5

Local Search Temporal Decision Trees

.

Contents

Examination of all split candidates in order to find the best one in terms of ability to split the data is computationally costly. It requires all pairwise distance computations between candidate series and all the other time series. The problem appears when either the employed distance has high runtime complexity or the number of split candidates grows rapidly. The latter is either due to increasing the size of dataset or due to using different split operators. In this chapter, we propose a method of approximated search on the space of split candidates called Local Search (LS). It allows us to reduce both terms: the number of distance computations and the number of split candidates. The Section 5.2 presents the Local Search strategy for Euclidean distance. Section 5.3 extends the algorithm to non static distances. In Section 5.4, we carry a set of experiments to show the effectiveness of local search speed up technique. We empirically prove that our method yields models with approximately the same

performance as the full search. However, it requires less computational resources to build the tree.

5.1 Introduction

In order to improve the efficiency of the method, we introduce Local Search algorithm. At each node, only a subset of all possible candidates is examined on order to find the best split candidate. The cardinality of this subset is determined by a user. Initial subset of candidates is initialized randomly and then extended by including their nearest neighbours.

The idea originates from the algorithm to accelerate k-means, proposed by Elkan [24], by using the triangle inequality. It is based on the fact that most distance calculations in a standard k-means are redundant. The majority of data points keeps their assignment to the same closest center during the centroids update step. Following the same logic, we make the assumption that if we exchange one split candidate by another taken from its neighbourhood, the partition induced by the new split candidate is similar to the partition done by the initial split candidate. However, *this similar partition will have different Gini impurity index.* Clearly, the new obtained partition can be better, with purer class distribution in the descendant nodes, or worse than the initial one. In case of getting the positive change, we can continue exploration of neighbour time series of previously examined split candidate, otherwise we stop. The main advantage of doing this is that we save computational resources because we do not completely explore space of split candidates completely, and thus we omit computation of all distances.

In the forthcoming section, we provide detailed explanation of the Local Search approach within Temporal Decision Tree algorithm (LSTDT). We introduce the algorithm for hyperplane split operator, because the geometry of data split resembles clustering. Each set of time series in descendant nodes is represented by a split time series from a pair (ts_L, ts_R). Further, we elaborate *LSTDT* for hypersphere split operator with some changes, since the geometry of split is different. The whole algorithm is based on the triangle inequality, hence we have a constraint in our choice of similarity measures. The majority of widely used time series dissimilarities do not satisfy triangle inequality, and DTW is one of them. We start with Euclidean dissimilarity measure employment in the algorithm, which satisfies all metric properties. Then, we generalize the algorithm for two important dissimilarities, namely DTW and $(1 - \mathrm{Cort}_\pi)$, by using their lower bounds that satisfy the property of weak triangle

inequality. Moreover, we suggest that it is up to the user to decide whenever they want to employ a complete or an approximated search on the space of split candidates. This provided by including into the algorithm parameters that depend on allocated computational resources which allows tuning the level of local search.

5.2 Local Search Temporal Decision Trees for Euclidean Distance.

In this section, we use the Euclidean distance to present the Local Search algorithm. Generalization of results to other similarity measures will be presented in the next section.

5.2.1 Local Search for Hyperplane Split Operator.

The idea of the Local Search Temporal Decision Tree algorithm (*LSTDT*) is to obtain the approximated version of the Full Search Temporal Decision Tree algorithm (*FSTDT*), which could be Mono-operator Temporal Decision Tree or Multi-operator Temporal Decision Tree. By employing the triangle inequality and introducing a set of lower and upper bounds on distances between split candidate series and time series in a node, *LSTDT* can avoid unnecessary distance calculations. With minimum number of additional distance computations, one can evaluate 'goodness' of pairs that lie in the neighbourhood of the initial split pair. The main advantage of this approach is that it guarantees yielding the exact partition induced by the split candidate under assessment. That is to say, the partition obtained using bounds is the same as the one with real distance values.

Given a split candidate $(ts_L, ts_R) \in \mathcal{F}(H)$ and a time series ts, we are searching for the best split pair $(ts_L, ts_R) \in \mathcal{F}(H)$, which maximizes the Gini impurity gain. For each split candidate and a set of time series **TS** belonging to a node g, we obtain its bipartition on two descendent nodes g_L and g_R. Each time series in **TS** is assigned to its closest split series according to a dissimilarity measure d. Here, we consider d to be Euclidean distance between two time series:

$$d(ts_i, ts_j) = \Big(\sum_{t=1}^{m} (v_i^t - v_j^t)^2 \Big)^{\frac{1}{2}}.$$

Therefore, time series satisfying the condition $d(ts_L, ts) \leq d(ts_R, ts)$ are assigned to the left node g_L and others to the right one g_R. We need to compute two distances $d(ts_L, ts)$ and $d(ts_R, ts)$ for each time series ts in order to assign them to one of the

descendent nodes. However, by applying the triangle inequality one can eliminate some distance computations.

Into *LSTDT* algorithm a set of split candidates, which will be examined, is divided into two subsets: seed split and neighbour split candidates. The latter one is created from the nearest neigbours of split candidates from the former one. Skipping computation of real distance values by using triangle inequality can be done in two different steps of split candidates assessment, which are

- time series assignment to the initial split pair (ts_L, ts_R) taken from seed list of candidates;
- time series assignment a new split pair, which is created by moving the initial split pair to its neighbour split candidate (ts'_L, ts_R) or (ts_L, ts'_R).

Below, we introduce upper and lower bounds on distances between time series and explain how to make the data partition based on them.

Triangle inequality. Let ts, ts_L, ts_R be three time series and d is the Euclidean distance, then the triangle inequality is defined as:

$$d(ts_L, ts_R) \leq d(ts_L, ts) + d(ts_R, ts) \tag{5.1}$$

This inequality allows us to build some bounds on $d(ts_L, ts)$ and $d(ts_R, ts)$, and to make the partition, without calculating the exact distance value. From the following corollaries proposed in [24] we can conclude how to get useful bounds. And, then, how to employ triangle inequality to reduce the number of distance computations.

Corollary 5.2.1. *Let* $ts, ts_L, ts_R \in \mathbf{TS} \times \mathbf{TS} \times \mathbf{TS}$ *are three time series taken from the training set* **TS**. *If* $d(ts_L, ts_R) \geq 2d(ts_L, ts)$ *then* $d(ts_L, ts) \leq d(ts_R, ts)$.

Proof. We know that $d(ts_L, ts_R) \leq d(ts_L, ts) + d(ts_R, ts)$, that is equivalent to $d(ts_L, ts_R) - d(ts_L, ts) \leq d(ts_R, ts)$. Putting the condition $d(ts_L, ts_R) \geq 2d(ts_L, ts)$ to the left hand part of the latest inequality, we have $d(ts_L, ts_R) - d(ts_L, ts) \geq 2d(ts_L, ts) - d(ts_L, ts) = d(ts_L, ts)$. Hence $d(ts_L, ts) \leq d(ts_R, ts)$. ■

Accelerated Initial partition. By Corollary 5.2.1 the condition

$$d(ts_L, ts) \leq \frac{1}{2} d(ts_L, ts_R)$$

leads to $d(ts_L, ts) \leq d(ts_R, ts)$. Then, if $d(ts_L, ts) \leq \frac{1}{2} d(ts_L, ts_R)$, we say that ts is associated with ts_L and assigned to g_L without computing $d(ts_R, ts)$.

Suppose we have an upper bound $\mathfrak{u}_{ts_L}(ts)$ on the distance $d(ts_L, ts)$, such that $\mathfrak{u}_{ts_L}(ts) \geq d(ts_L, ts)$. Corollary 5.2.1 says that we need to compute the distance $d(ts_R, ts)$ only if

$$\mathfrak{u}_{ts_L}(ts) \geq \frac{1}{2} d(ts_L, ts_R). \tag{5.2}$$

Application to a split candidate examination is straightforward. For any split pair $(ts_L, ts_R) \in \mathcal{F}(H)$ compute $d(ts_L, ts)$ and check if inequality 5.2 holds, then skip calculation of $d(ts_R, ts)$.

Corollary 5.2.2. *Let $ts, ts_L, ts_R \in \mathbf{TS} \times \mathbf{TS} \times \mathbf{TS}$ are three time series taken from the training set* **TS**. *Then, $d(ts_R, ts) \geq \max\{0, d(ts_L, ts) - d(ts_L, ts_R)\}$.*

Proof. We have $d(ts_L, ts) \leq d(ts_R, ts) + d(ts_L, ts_R)$, so $d(ts_L, ts) - d(ts_L, ts_R) \leq d(ts_R, ts)$. And $d(ts_R, ts) \geq 0$ is always true. ■

Accelerated exploration of a new neighbour split. Corollary 5.2.2 can be employed to explore efficiently nearest neigbour pairs of a split candidate (ts_L, ts_R). Let move ts_L to its neighbour time series $ts'_L = \underset{ts}{\operatorname{argmin}}(d(ts_L, ts))$. The new split candidate to assess is (ts'_L, ts_R). To figure out the data partition made by this new split, for each ts we need to check either inequality $d(ts'_L, ts) < d(ts_R, ts)$ holds or not. Hence, we have to compute the distance value $d(ts'_L, ts)$. Instead, one can employ bounds on the distance $d(ts'_L, ts)$ to induce the new partition g'_L and g_R.

Let us consider two cases:

1. Suppose that ts was associated with ts_L and assigned to g_L, and we have

   - a valid upper bound $\mathfrak{u}_{ts_L}(ts)$ on the distance $d(ts_L, ts) : \mathfrak{u}_{ts_L}(ts) \geq d(ts_L, ts)$,
   - a valid lower bound $\mathfrak{l}_{ts_R}(ts)$ on the distance $d(ts_R, ts) : \mathfrak{l}_{ts_R}(ts) \leq d(ts_R, ts)$.

   Then, we update an upper bound

$$\mathfrak{u}_{ts'_L}(ts) = d(ts_L, ts) + d(ts'_L, ts_L), \tag{5.3}$$

the lower bound $\mathfrak{l}_{ts_R}(ts)$ will not change. If $\mathfrak{u}_{ts'_L}(ts) \leq \mathfrak{l}_{ts_R}(ts)$, then $d(ts'_L, ts) \leq \mathfrak{u}_{ts'_L}(ts) \leq \mathfrak{l}_{ts_R}(ts) \leq d(ts_R, ts)$. And a time series ts will be assigned to the node g'_L.

2. Suppose that ts was associated with ts_R and assigned to g_R, and we also have

   - a valid upper bound $\mathfrak{u}_{ts_R}(ts)$ on the distance $d(ts_R, ts) : \mathfrak{u}_{ts_R}(ts) \geq d(ts_R, ts)$,
   - a valid lower bound $\mathfrak{l}_{ts_L}(ts)$ on the distance $d(ts_L, ts) : \mathfrak{l}_{ts_L}(ts) \leq d(ts_L, ts)$.

   Then, we can derive an upper and a lower bounds on the distance between ts and a time series ts'_L:

$$d(ts'_L, ts) \geq \max\{0, d(ts_L, ts) - d(ts'_L, ts_L)\} \tag{5.4}$$
$$\geq \max\{0, \mathfrak{l}_{ts_L}(ts) - d(ts'_L, ts_L)\} \tag{5.5}$$

   Thus, $\mathfrak{l}_{ts'_L}(ts) = \max\{0, \mathfrak{l}_{ts_L}(ts) - d(ts'_L, ts_L)\}$, upper bound $\mathfrak{u}_{ts_R}(ts)$ will not change. If $\mathfrak{u}_{ts_R}(ts) \leq \mathfrak{l}_{ts'_L}(ts)$, then $d(ts_R, ts) \leq \mathfrak{u}_{ts_R}(ts) \leq \mathfrak{l}_{ts'_L}(ts) \leq d(ts'_L, ts)$. And a time series ts will keep its assignment to the node g_R.

One should note that for each time series an upper bound is a bound on the distance to its closest split time series. While a lower bound is a bound on the distance to its farthest split time series. If derived bounds are good approximators and the shift distance $d(ts'_L, ts_L)$ is small enough, then updated bounds are good approximators for a distance $d(ts'_L, ts)$.

Knowing upper and lower bounds on distances $d(ts_L, ts)$ and $d(ts_R, ts)$ of the initial split candidate (ts_L, ts_R), it is not necessary to work out their exact values to make an assignment of ts to the correct descendant node. However, in practice, to obtain bounds and maintain their update, it will be necessary to compute either $d(ts_L, ts)$ or $d(ts_R, ts)$.

Figure 5.1 shows the change in data partition if a time series ts_L from a split pair (ts_L, ts_R) will be replaced by ts'_L. Each point schematically represents a time series of the training set. The value δ stands for the shift distance $d(ts'_L, ts_L)$. Dashed lines correspond to the split done by a pair (ts_L, ts_R). For points that lie inside the circle, computation of the distance value $d(ts_R, ts)$ will be skipped, since for each of them Condition 5.2 holds. A hyperplane drawn by solid line represents the data partition for the new split pair (ts'_L, ts_R).

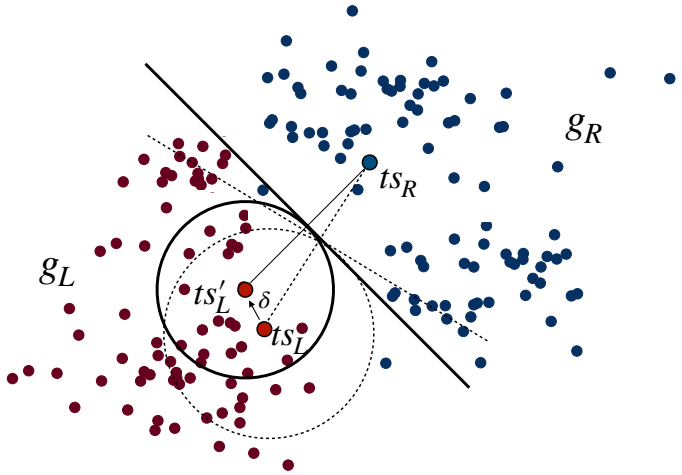

Figure 5.1: Illustration of change in data partition if a time series ts_L from a split pair (ts_L, ts_R) will be replaced by ts'_L. Each point schematically represents a time series of the training set. The value δ stands for the shift distance $d(ts'_L, ts_L)$.

Local search algorithm. As we saw, by using derivations from triangle inequality, one can apply them to assess partitions induced by hyperplane split candidates. By reducing the number of distance computations on the step of the data partition by initial split candidate as well as on the step of the partition update for neighbour split candidate, we can save computational resources. We assume, that we can find a better split quality candidate in the vicinity of the initial one. If this assumption holds, it will allows to improve a node's partition class purity and at the same time, we can avoid exhaustive search over complete split candidate space. Actually, as we will see in this section with empirical results it is the case, and we can obtain the tree of the size almost the same as the one obtained by exhaustive search.

A node's split in *LSTDT* is done in two stages, which are, first, *initial split candidates evaluation* and, second, *exploration of neigbour split pairs*. The former is a preliminary step to be able initialize the further exploration and guarantee to have a valid partition. The latter is heuristical strategy of informed exploration search for split candidates. In order to perform search, we need an evaluation function and stopping criterion. Since we target to find the best possible data partition, the Gini impurity gain seems an appropriate function to evaluate success of search process. However, the next question is how far from the initial split candidate should we go? Efficiency of the algorithm fundamentally depends on tightness of upper and lower bounds. They are tight, when a new split candidate lies not far from the initial pair of time series. The farther we move from the initial split candidate, the looser bounds

we have. It implies that, more distance computations have to be done. This is the fundamental reason to generate a new split candidate that is only slightly different from the previous one.

We introduce three input parameters given by a user:

1. r is the number of reference time series. The algorithm subsamples them from **TS** to generate the initial split candidates (seeds);

2. $LSDist$ is the maximum number of distance computations that a user allocates to the local search procedure;

3. NN is the number of nearest neighbour split candidates for each seed to explore .

We introduced the parameter NN to prevent spending all $LSDist$ distances on exploration of vicinity of only one seed pair. As alternative of NN, we may employ Gini impurity index as the criterion to stop exploration,i.e., if a partition induced by a neibour split candidate does not improve its impurity, the algorithm moves to another seed. However, not having an instantaneous improvement of split quality in the first movement does not necessarily indicate impossibility to reach it, for example, in the second iteration. Therefore, by introducing and determining small reasonable value of NN (can be fixed to 3 or 5 as typical choice in k-Nearest Neighbour) the algorithm can explore vicinity of different split candidates.

We would like to emphasize that the total number of distance computations, denoted as $TDist$ in $FSTDT$ can be factorized by two terms in $LSTDT$:

1. $INDist$ is the number of distance computations between each seed pair and the other time series of the dataset. It is the minimum number of distance computations required to initialize search process.

2. $LSDist$ is the number of additional distances to do the local search.

The value $INDist$ depends on the number of time series r picked to initialize the procedure of search and defined as:

$$INDist = \frac{(2N - r - 1)}{2} r. \tag{5.6}$$

This value corresponds to the minimum number of distances required to examine split quality of seed pairs generated from r time series. How many seeds r, represented by time series of the training set, one should pick to initialize local search? To make a

partition in a node by using hyperplane split operator, the algorithm needs at least one pair of time series. That is to say, the minimum number is equal to two, so $r_{min} \geq 2$. By setting the minimum value for initial number of seeds, we guarantee to have a valid, however far from the optimal, data split in a node.

The choice of the r value is a trade-off between minimizing the cost of $INDist$ term and having a varied set of seeds for the local search procedure. On one hand, if the number r is small, then $INDist$ is small too, and we do not spend many distance computations. However, we may pick seeds that lie close to each other, and as a consequence, we will be able to explore just one particular region of split candidate space. In this case, if split candidates, which provide purer data partition, are far from initial and $LSDist$ value is smal too, we have the risk to not reach them. On the other hand, a large value of r increases chances to explore different regions of split candidate space, but with the high cost of $INDist$ term. If $r = N$, then we finish up with the full search algorithm. Therefore, the value of r can be determined via a slowly increasing function dependent on the number of input size N.

Complementary notations. Before giving the $LSTDT$ description, let us introduce some additional notations that we employ in this chapter. A split pair $f_H \in \mathcal{F}(H)$ is a pair of time series (ts_L, ts_R). Let an operator $a(ts)$ returns the split time series from a split pair $f_H \in \mathcal{F}(H)$ belonging to the same node as a time series ts. And $\bar{a}(ts)$ is its complementary operator.

$$a(ts) = \begin{cases} ts_L & \text{if} \quad ts \in g_L \\ ts_R & \text{if} \quad ts \in g_R \end{cases} \tag{5.7}$$

$$\bar{a}(ts) = \begin{cases} ts_L & \text{if} \quad ts \notin g_L \\ ts_R & \text{if} \quad ts \notin g_R \end{cases} \tag{5.8}$$

To avoid confusion in notation, we will define a distance between left and right split time series as $d(f)$. We denote ts_{nn} be a time series belonging to the neighbourhood of ts_L or ts_R by which on of them will be replace to form new split candidate. The operator $\delta(\cdot)$ returns the distance value between any time series and ts_{nn} if they belong to the same sub-node, otherwise it returns value of zero.

$$\delta(\cdot) = \begin{cases} d(\cdot, ts_{nn}), & \text{if} \quad \cdot = a(ts_{nn}) \\ 0, & \text{otherwise} \end{cases} \tag{5.9}$$

Note that $\delta(\cdot)$ is only used for $\cdot = ts_R$ or $\cdot = ts_L$.

Initial search. Algorithm 5.1 gives the description of making an *initial* partition of a node $g \in \mathcal{G}$ by a set of split candidates generated from referenced time series r. The algorithm exploits triangle inequality properties to make a data partition and minimizes the number of required distance computations. At lines 7-9, it introduces the upper and lower bounds dedicated for each split candidate. Lines 11-22 reveal the use of triangle inequality to avoid redundant distance computation while making a data partition. For each generated split candidate, the algorithm evaluates level of class distribution impurity in each descendant node (line 21-22). As a result, we obtain a set of split candidates $\mathcal{S}$ with the following components:

- partition g_L, g_R,
- *Gini* impurity gain Δi_{Gini},
- upper bounds $\mathfrak{u}_{a(ts)}$,
- lower bounds $\mathfrak{l}_{a(ts)}, \mathfrak{l}_{\bar{a}(ts)}$

Real pairwise distance values between each time series and its closest split series are assigned as initial values of upper bounds. Lower bounds are bounds on the distance from a time series to its farthest split series from a pair (ts_L, ts_R). Note, that if we are able to skip a distance computation $d(ts_R, ts)$, which implies holding $d(ts_L, ts) \leq \frac{1}{2} d(ts_L, ts_R)$, lower bound value is equal to zero. Once a new neighbour split candidate is examined, it will be considered as one which vicinity can explored

as well.

ALGORITHM 5.1: Algorithm of getting Initial Hyperplane split candidate partition.

Function *LSInitialPartition*:
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g$; the number of time series to pick as seeds r; similarity measure d
  Result: $\mathcal{S} := \{s(g, f_H) := \{g_L, g_R, (ts_L, ts_R), \Delta i_{\text{Gini}}, \mathfrak{u}_{a(ts)}, \mathfrak{l}_{a(ts)}, \mathfrak{l}_{\bar{a}(ts)}\}\}$
  $\mathcal{S} \leftarrow \emptyset$;
  $\mathbf{TS}_r \leftarrow$randomly pick r time series from $\mathbf{TS}$;
  $\mathcal{F}(H)_{seed} \leftarrow \{f_H : ((ts_i, y_i), (ts_j, y_j)), \forall ts_i \neq ts_j \in \mathbf{TS}_r \times \mathbf{TS}_r, y_i \neq y_j\}$;
  foreach $f_H \in \mathcal{F}(H)_{seed}$ **do**
    $g_L, g_R \leftarrow \emptyset$;
    $\mathfrak{u}_{a(ts)} \leftarrow \{\emptyset\}_N$;
    `// upper bound of distance to closest split series`
    $\mathfrak{l}_{a(ts)}(ts) \leftarrow \{\emptyset\}_N$;
    $\mathfrak{l}_{\bar{a}(ts)}(ts) \leftarrow \{\emptyset\}_N$; `// lower bounds on the distance to each of split series`
    foreach $ts \in \mathbf{TS}$ **do**
      $g_L \leftarrow g_L \cup \{ts\}$;
      $a(ts) \leftarrow$ Formula 5.7 ;
      $d \leftarrow d(a(ts), ts)$;
      $\mathfrak{l}_{a(ts)}(ts) \leftarrow d$;
      if $d > \frac{1}{2} d(f_H)$ **then**
        $\bar{a}(ts) \leftarrow$ Formula 5.8 ;
        $\mathfrak{l}_{\bar{a}(ts)}(ts) \leftarrow d(\bar{a}(ts), ts)$;
        if $\mathfrak{l}_{\bar{a}(ts)}(ts) < d$ **then**
          $g_L \leftarrow g_L \backslash \{ts\}$;
          $g_R \leftarrow g_R \cup ts$;
        end if
      end if
      $\mathfrak{u}_{a(ts)}(ts) \leftarrow d$;
    end foreach
    $i_{\text{Gini}}(g_L), i_{\text{Gini}}(g_R) \leftarrow$ compute with Formula 3.4 ;
    $\Delta i_{\text{Gini}}(s(g, f)) \leftarrow$ compute Gini gain with use of Formula 3.3 ;
    $\mathcal{S} \leftarrow \mathcal{S} \cup s(g, f_H)$;
  end foreach
end

Local search. Steps of the local search over split candidates are shown in Algorithm 5.2. Exploration of split candidate space continues while *LSDist* is greater than zero (line 4). Local search algorithm includes three main blocks.

1. *Defining nearest split candidates in vicinity of a seed pair.* It starts with the exploration of split pairs from seeds set $\mathcal{S}$. A pair that maximizes Gini impurity

gain is selected to be the first one whose neighbourhood will be explored (line 2). At each iteration of update a seed pair by its neighbour, only one time series in a split pair is replaced, either ts_L or ts_R. A time series ts_{nn} is the one that has minimum value of upper bound. Recall that for each time series ts, we have an upper bound $u_{a(ts)}(ts)$ on the distance between ts and its closest series $a(ts)$. A series ts_{nn} is the one that minimizes this upper bound, thus a split time series $a(ts_{nn})$ in a pair (ts_L, ts_R) will be replaced by ts_{nn}.

For selected split seed $s(g, f_H)$, the algorithm gets NN nearest neighbours split candidates (line 6-7). We change one time series in a split pair seed per iteration. We sort upper bounds in ascending order and pick time series ts_{nn} that has the minimum value. We replace corresponding split time series by a new one and update f_H with ts_{nn}.

2. *Upper and lower bounds updates.* With each update of a split f_H by a new split candidate f_{nn}, the corresponding bounds of each time series ts has to be updated by a value of shift distance $\delta(a(ts))$ using Equation 5.9. Each upper bound value is increased by shift distance $\delta(a(ts))$, while s lower bound value is decreased by the same value (line 10-12).

3. *A partition update.* It includes a preliminary step of estimation of the number of distances that has to be computed in order to update a partition. This step is required to prevent exploration of computationally greedy split candidates. Skipping the assessment of such $(f_H)_{nn}$ pairs works for purpose of saving computational resources and allows exploration other split candidates. Also by doing this, we guarantee that the number of allocated distance computations will be enough to complete the partition's update for a split candidate under examination. We need to evaluate the number of time series that may change its initial assignment to a sub-node, when a split seed f_H moves to its neighbour $(f_H)_{nn}$. We say that, a time series ts tends to change its previously assigned sub-node if Conditions 5.10 and 5.11 are satisfied. Both of them depend on updated bounds and the distance between split time series of the new split candidate $d((f_H)_{nn})$.

$$\mathfrak{u}_{a(ts)}(ts) \geq \frac{1}{2} d((f_H)_{nn}) \tag{5.10}$$

$$\mathfrak{u}_{a(ts)}(ts) \geq \mathfrak{l}(ts, \bar{a}(ts)) \tag{5.11}$$

Each potential swap of a time series can, at the maximum, lead to two distance value computations. Thus, $\max(|TS_{swap}|) = 2N$, that is the upper estimate of the number of computational resources the algorithm will spend to update a partition. In fact, it is the worst case scenario when we compute all pairwise distances between a split candidate and the other time series. In practice, the value of real distance computations $Dist \leq 2N$, because of the distance bounds employment and examination only of the nearest neighbours of initial split candidate. Heuristically, we choose to explore the split candidate $(f_H)_{nn}$ once the value $|TS_{swap}|$ is two times less than the number of resources $LSDist$ (line 13). If the case, then the algorithm updates partition subsets g_L and g_R (line 14). It can happen at some iteration that $Dist \geq LSDist$ (line 15 of Algorithm 5.2 and line 17 of Algorithm 5.4), however the maximum overtaken value is equal to N, which is $\frac{2}{N-1}\%$ of all distance computations. It is a very small percentage, especially for large datasets.

Each new examined split candidate f_{nn} becomes also a seed split pair (line 17). For new obtained split partition, Gini gain is computed and if a new gain value is higher than the best obtained so far, we make an update of the best split candidate. Details of the partition update for a split f_{nn} are given in Algorithm 5.3. For each time series ts in a node g, it verifies Conditions 5.10 and 5.11. In the case of violation, it computes required distances and exchange a sub-node assignment if necessary (line 4 - 12, Algorithm 5.3).

Due to heuristical origin of the local search procedure, it can be the case that the algorithm get stuck in local maximum gain of Gini that has been already obtained on the step of seeds set initialization. Another caveat can be when classes in a dataset are highly homogeneous, that is to say, lots of split candidates have the same discriminative ability. Local search process is initialized at each internal node of a tree separately. The value of $LSDist$ reduces proportionally to time series set size in descendent nodes. All distances that are computed at each level of a tree are saved during a tree training process. For small nodes ($|\mathbf{TS}_g| \ll |\mathbf{TS}|$), it may be relevant to switch from local search to the full search algorithm, if the number of non-computed

distances is less than allocated number of $LSDist$.

ALGORITHM 5.2: Exploration of neigbour split candidates $\mathcal{F}(H)$.

Function *LSExploration-*$\mathcal{F}(H)$**:**
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g$; similarity measure d; the number on nearest neigbours NN; allocated number of distances $LSDist$; seeds set $\mathcal{S}$
  Result: $s(g, f_H^*) := \{g_L^*, g_R^*, (ts_L^*, ts_R^*), \Delta i_{\text{Gini}}^*, \mathfrak{u}_{a(ts)}, \mathfrak{l}_{a(ts)}, \mathfrak{l}_{\bar{a}(ts)}\}$
  $s(g, f_H^*) \leftarrow \underset{s(g,f_H)\in\mathcal{S}}{\text{argmax}}(\Delta i_{\text{Gini}}(s(g, f_H)))$;
  while $LSDist > 0$ **do**
    $s(g, f_H) \leftarrow \underset{s(g,f_H)\in\mathcal{S}}{\text{argmax}}(\Delta i_{\text{Gini}}(s(g, f_H)))$;
    $\mathcal{S} \leftarrow \mathcal{S} \backslash s(g, f_H)$;
    $\mathbf{TS}_{NN} \leftarrow \{ts : \underset{ts_i\in\mathbf{TS}, i=1:NN}{\text{argmin}} \{\mathfrak{u}_{a(ts_i)}(ts_i)\}\}$;
    $\mathcal{F}(H)_{NN} \leftarrow \{f_H : (ts_{nn}, ts_R)$, **if** $a(ts_{nn}) = ts_L$ **, otherwise** $(ts_L, ts_{nn}), ts_{nn} \in \mathbf{TS}_{NN}\}$;
    foreach $(f_H)_{nn} \in \mathcal{F}(H)_{NN}$ **do**
      $\mathfrak{u}_{a(ts)} \leftarrow \mathfrak{u}_{a(ts)} + \delta(a(ts))$;
      $\mathfrak{l}_{a(ts)} \leftarrow \max((\mathfrak{l}_{a(ts)} - \delta(a(ts))), 0)$;
      $\mathfrak{l}_{\bar{a}(ts)} \leftarrow \max((\mathfrak{l}_{\bar{a}(ts)} - \delta(\bar{a}(ts))), 0)$;
      $\mathbf{TS}_{swap} \leftarrow \{ts : \text{Cond } 5.10 \text{ and Cond } 5.11 \text{ hold}\}$;
      if $\frac{|\mathbf{TS}_{swap}|}{2} \leq LSDist$ **then**
        $g_L, g_R, Dist \leftarrow$ *LSUpdatePartition-*$\mathcal{F}(H)((\mathbf{TS}, \mathcal{Y}), d, s(g, (f_H)_{nn})$;
        $LSDist \leftarrow LSDist - Dist$;
        $i_{\text{Gini}}(g_L), i_{\text{Gini}}(g_R) \leftarrow$ compute with Formula 3.4 ;
        $\Delta i_{\text{Gini}}(s(g, (f_H)_{nn})) \leftarrow$ compute Gini with Formula 3.3 ;
        $\mathcal{S} \leftarrow \mathcal{S} \cup s(g, (f_H)_{nn})$;
        if $\Delta i_{Gini}(s(g, (f_H)_{nn}) > \Delta i_{Gini}(s(g, f_H^*))$ **then**
          $s(g, f_H^*) \leftarrow s(g, (f_H)_{nn})$;
        end if
      end if
    end foreach
  end while
end

The complexity of exhaustive search for the best split candidate over all generated candidates for hyperplane split operator is $O(N^3)$. The term $O(N^2)$ is the number of time series pairs and the term $O(N)$ is the complexity to evaluate the partition induced by one split candidate. The full search also requires computation of complete pairwise distance matrix of the size $N \times N$. By employing local search, we reduce the number of split candidates to be assessed and the number of distance computations. The exact values of both terms depends on the input parameters r and $LSDist$ of the $LSTDT$ algorithm. The total number of computed distances in a node for $LSTDT$

ALGORITHM 5.3: Partition update for neigbour split candidates $\mathcal{F}(H)$.

Function *LSUpdatePartition-*$\mathcal{F}(H)$**:**
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g$; similarity measure d; $s(g, (f_H)_{nn})$
  Result: $s(g, (f_H)_{nn}) := \{g_L, g_R, (ts_L, ts_R), \Delta i_{\text{Gini}}, \mathfrak{u}_{a(ts)}, \mathfrak{l}_{a(ts)}, \mathfrak{l}_{\bar{a}(ts)}\}$
  foreach $ts \in \mathbf{TS}$ **do**
    if *Cond 5.10 and Cond 5.11* **then**
      $\mathfrak{u}_{a(ts)}(ts) \leftarrow d(a(ts), ts)$;
      $\mathfrak{l}_{a(ts)}(ts) \leftarrow \mathfrak{u}_{a(ts)}(ts)$;
      if *cond 5.10 and cond 5.11* **then**
        $\mathfrak{l}_{\bar{a}(ts)}(ts) \leftarrow d(\bar{a}(ts), ts)$;
        if $\mathfrak{l}_{\bar{a}(ts)}(ts) \leq \mathfrak{u}_{a(ts)}(ts)$ **then**
          $\mathfrak{u}_{a(ts)}(ts) \leftarrow \mathfrak{l}_{\bar{a}(ts)}(ts)$;
          $a(ts) \leftarrow \bar{a}(ts)$;
          update g_L, g_R ;
        end if
      end if
    end if
  end foreach
end

is approximated by $INDist + LSDist$. The exact number of distances the algorithm needs to calculate for each new neighbour candidate can be approximated by value $\frac{|TS_{swap}|}{2}$. Thus, we can approximate the total number of time series pairs, which $LSTDT$ will assess by $|\mathcal{F}(H)| \approx O(r^2 + \frac{2LSDist}{TS_{swap}})$. The complexity of Algorithm 5.1 is $O(r^2 N)$. Algorithm 5.2 iterates in the worst case $LSDist$. At each iteration, which corresponds to exploration of the vicinity of one split candidate, the seed list is extended, while kept sorted by neighbour candidates. The complexity of Local Search part of $LSTDT$ (Algorithm 5.2) is $O(LSDist \log LSDist)$. However, in practice the algorithm does not iterates $LSDist$ times, because at each iteration this value decrements by $Dist \gg 1$.

5.2.2 Local Search for Hypersphere split operator.

Now, we would like to extend Local Search algorithm to be able to apply it to candidates of hypersphere split operator. Like that, we could use both hyperplane and hypersphere split operators simultaneously while training a tree model. A hypersphere split operator is represented by a pair $(ts_L, \theta), ts_L \in \mathbf{TS}$, where ts_L is associated with the left sub-node, which constitutes a subset of those time series for which the condition $d(ts_L, ts) \leq \theta$ holds.

Assume that we have a split candidate seed $f_S = (ts_L, \theta), ts_L \in \mathbf{TS}$ and we would

like to examine goodness of its neighbour candidate ts_{nn}. The main question is how to update the partition $s(f_S, g)$ induced by f_S when f_S moves to a neighbour split $(f_S)_{nn}$. As before, for each iteration of a split candidate update, we need to figure out which elements in a partition tend to change its assigned sub-node. Then, for each time series with positive swap tendency, we compute a true distance value $d(ts_{nn}, ts)$ and update the partition. The main difference with Local Search algorithm for hyperplane split operator is that we do not have two representatives of a binary partition. Rather, we have elements lying inside hypersphere with a radius θ, which are associated with ts_L, and the rest elements lying outside it. The threshold θ associated to an initial split series ts_L can, surely, be different for t_{nn}. It means that we have to update it as well. Therefore, a partition update for a new split candidate ts_{nn} is two steps iterative procedure:

1. Compute the number of swaps given θ.
2. Update θ given the number of swaps.

It is exactly as an expectation-maximization approach, where, firstly, we compute the expected value of the number time series that can change their initial sub-node assignment. And, secondly, we update θ by maximizing Gini impurity gain of updated partition. We repeat these steps until it converges to the state where there is no more possible swaps. If at some level of iterations, the number of swaps exceeds the remained value of $LSDist$, the algorithms stops. As a consequence, the algorithm does not ensure to find the optimal distance threshold θ_{nn} for ts_{nn}.

To assess the initial number of swaps in a partition for a new split candidate given a threshold θ, we need to derive bounds $\mathfrak{u}(ts)$, $\mathfrak{l}(ts)$ and a margin μ. An upper bound $\mathfrak{u}(ts)$ is a bound on the distance $d(ts_L, ts), ts \in g_L$, and $\mathfrak{l}(ts)$ is a lower bound on $d(ts_L, ts), ts \notin g_L$. A margin μ is defined as follows:

$$\mu = \frac{|d(ts_L, ts_{\theta^-}) - d(ts_L, ts_{\theta^+})|}{2}, \tag{5.12}$$

with

$$ts_{\theta^-} = \underset{ts \in g_L}{\operatorname{argmax}}\, d(ts_L, ts)$$

$$ts_{\theta^+} = \underset{ts \in g_R}{\operatorname{argmin}}\, d(ts_L, ts)$$

If a shift distance $d(ts_L, ts_{nn})$ is less than μ value, it means that a partition made by a split candidate (ts_L, θ) will be the same as the partition made by (ts_{nn}, θ). By

inspecting this simple condition we can get rid of split candidates with the same goodness of Gini gain as for an initial candidate. For cases where this condition does not hold, we employ upper and lower bounds.

For hypersphere split operator, we maintain only one value of lower bound since we have only one center of assignment ts_L. We employ the triangle inequality that provides us with the update on the bounds for each ts as follows:

$$\mathfrak{u}(ts) = d(ts_L, ts) + d(ts_L, ts_{nn}) \tag{5.13}$$

$$\mathfrak{l}(ts) = \max((d(ts_L, ts) - d(ts_L, ts_{nn})); 0) \tag{5.14}$$

For time series that are assigned to the left sub-node, evaluation of swap is based on the upper bound, and for those that lie into the right sub-node, on its lower bound. Indeed, if $\mathfrak{u}(ts) > \theta$ for $ts \in g_L$, then it means that a real distance value $d(ts_{nn}, ts)$ can be greater than θ as well, hence, an assignment of ts may change to g_R. When $ts \in g_R$ and $\mathfrak{l}(ts) < \theta$, ts can swap to g_L, a sub-node, which will be associated with ts_{nn}. Figure 5.2 illustrates this situation for binary classification problem (on the figure class labels are depicted by squares and circles). Each time series ts is represented by a scalar: its distance $d(ts_L, ts)$ to the center of the hypersphere defined by ts_L with a radius θ. Points of time series that will not change their initial assignment to descendent nodes when ts_L moves to ts_{nn} are depicted with filled circles/squares. Equation below summarize conditions of swap:

$$swap(ts) = 1 \Leftrightarrow \begin{cases} ts \in g_L & \text{and} \quad \mathfrak{u}(ts) > \theta \\ ts \in g_R & \text{and} \quad \mathfrak{l}(ts) < \theta \end{cases} \tag{5.15}$$

Once we proceed to update a partition for a neighbour split candidate ts_{nn}, we need to update a threshold θ. Distance values of all time series that could change their initial sub-node assignment lie in the interval $[\theta - d(ts_L, ts_{nn}); \theta + d(ts_L, ts_{nn})]$. Therefore, to update a threshold value, we have to compute the Gini gain for threshold candidates, values of which lie within this interval. Note, that at this step we have time series for which we have computed real distance values with t_{nn}, the rest of them have just an approximation by $\mathfrak{u}$ and $\mathfrak{l}$. Due to this fact, when we update a threshold, we have to check again if there are new time series that tend to swap with respect to updated threshold θ'.

The description of local search for hypersphere split candidate is given in Algorithm 5.4. Initialization steps are the same as for hyperplane split operator (lines 2-7). Bounds updates are done according to the Formula 5.13 (lines 9-10). Steps of

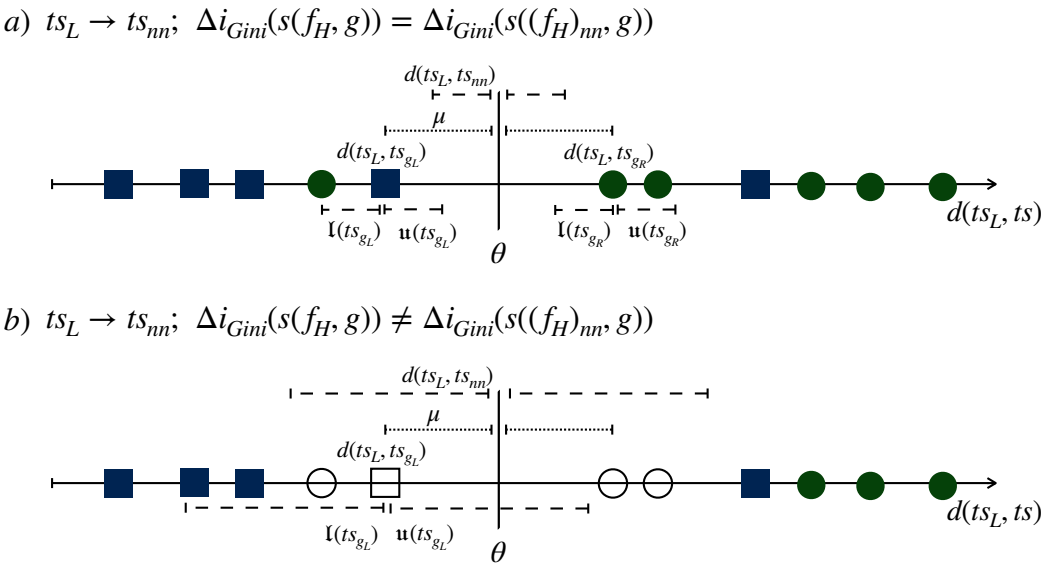

Figure 5.2: Illustration of possible changes in a partition when a split candidate (ts_L) moves to its neighbour (ts_{nn}). Time series represented by a distance value $d(ts_L, ts)$ and depicted as dots. Shape of each point denotes a time series class label. Two cases are shown: *a*) the first, there are no swap possible; a margin μ is greater than $\mathfrak{u}(ts_{g_L})$ and $\mathfrak{l}(ts_{g_R})$; *b*) the second, elements lying within the range $[\theta - d(ts_L, ts_{nn}); \theta + d(ts_L, ts_{nn})]$ require computation of true distance value to update its sub-node assignment.

expectation-maximization approach of a split update are shown in lines 13-26. The algorithm returns the better split in the vicinity of seed split candidates if such exists.

ALGORITHM 5.4: Exploration of neigbour split candidates $\mathcal{F}(S)$.

1 **Function** *LSExploration-*$\mathcal{F}(S)$**:**
  Input: $(\mathbf{TS}, \mathcal{Y}) \in g$; similarity measure d; the number on nearest neigbours NN; allocated number of distances $LSDist$; seeds set $\mathcal{S}$
  Result: $s(g, f_S^*) := (g_L^*, g_R^*, (ts^*, \theta^*), \Delta i_{\text{Gini}}^*, \mathfrak{u}, \mathfrak{l})\}$
2 $s(g, f_S^*) \leftarrow \underset{s(g,f)\in\mathcal{S}}{\text{argmax}}(\Delta i_{\text{Gini}}(s(g, f_S)))$;
3 **while** $LSDist > 0$ **do**
4 $s(g, f_S) \leftarrow \underset{s(g,f_S)\in\mathcal{S}}{\text{argmax}}(\Delta i_{\text{Gini}}(s(g, f_S)))$;
5 $\mathcal{S} \leftarrow \mathcal{S} \backslash s(g, f_S)$;
6 $\mathbf{TS}_{NN} \leftarrow \{ts : \underset{ts_i\in\mathbf{TS}, i=1:NN}{\text{argmin}} \{\mathfrak{u}(ts_i)\}\}$;
7 $\mathcal{F}(S)_{NN} \leftarrow \{f_S : (ts_{nn}, \theta), ts_{nn} \in \mathbf{TS}_{NN}\}$;
8 **foreach** $(f_S)_{nn} \in \mathcal{F}(S)_{NN}$ **do**
9 $\mathfrak{u}(ts) \leftarrow \mathfrak{u}(ts) + d(ts_L, ts_{nn})$;
10 $\mathfrak{l}(ts) \leftarrow \max((d(ts_L, ts) - d(ts_L, ts_{nn})); 0)$;
11 $\mathbf{TS}_{swap} \leftarrow \emptyset$;
12 $\mu \leftarrow$ compute with Formula 5.12 ;
13 **while** $|\mathbf{TS}_{swap}| > 0$ **do**
14 $\mathbf{TS}_{swap} \leftarrow \{ts : Cond.\ 5.15 \text{ holds}\}$;
15 **if** $|\mathbf{TS}_{swap}| \leq LSDist$ **then**
16 $g_L, g_R, Dist \leftarrow$ *LSUpdatePartition-*$\mathcal{F}(S)((\mathbf{TS}, \mathcal{Y}), d, s(g, (f_S)_{nn}))$;
17 $LSDist \leftarrow LSDist - Dist$;
18 update θ ;
19 $i_{\text{Gini}}(g_L), i_{\text{Gini}}(g_R) \leftarrow$ compute with Formula 3.4 ;
20 $\Delta i_{\text{Gini}}(s(g, (f_S)_{nn})) \leftarrow$ compute Gini with Formula 3.3 ;
21 $\mathcal{S} \leftarrow s(g, (f_S)_{nn})$;
22 **if** $\Delta i_{Gini}(s(g, (f_S)_{nn})) > \Delta i_{Gini}(s(g, f^*))$ **then**
23 $s(g, f^*) \leftarrow s(g, (f_S)_{nn})$;
24 **end if**
25 **end if**
26 **else**
27 **break** ;
28 **end while**
29 **end foreach**
30 **end while**
31 **end**

ALGORITHM 5.5: Algorithm of data partition update.

Function *LSUpdatePartition-*$\mathcal{F}(S)$**:**
  Input: $(\mathbf{TS}, \mathcal{Y}), d, s(g, (f_S)_{nn})$
  Result: $s(g, (f_S)_{nn}) := (g_L, g_R, (ts_{nn}, \theta), \Delta i_{\text{Gini}}, \mathfrak{u}, \mathfrak{l})$
  foreach $ts \in \mathbf{TS}$ **do**
    if *Cond 5.15* **then**
      $\mathfrak{u}(ts) \leftarrow d(a(ts), ts)$;
      $\mathfrak{u}(ts) = \mathfrak{l}(ts)$;
      if $\mathfrak{u}(ts) < \theta$ *and* $ts \in g_R$ **then**
        assign ts to g_L ;
      end if
      if $\mathfrak{u}(ts) > \theta$ *and* $ts \in g_L$ **then**
        assign ts to g_R ;
      end if
    end if
  end foreach
end

5.3 Algorithm generalization for non static distances.

Local search technique leverages the nice property of triangle inequality that Euclidean distance possess. However, for time series classification, one of the most usable dissimilarity measures is DTW, which is able to mitigate a problem of distortion along the time axis. Therefore, it would be particularly useful to employ DTW within local search. A primal advantage of this point would be the reduction of DTW distance computations that has quadratic complexity with respect to time series length.

The fact that DTW does not satisfy the triangle inequality prevents us from using it direcly in local search. However, it was proven [50] that DTW satisfies weak triangle inequality. We recall the theorem and its proof below:

Theorem 5.3.1. *(Weak Triangle Inequality for DTW) Given any time series* $ts_x, ts_y, ts_z \in \mathbf{TS}$ *of the same length* m *and* $1 \leq p \leq \infty$*, we have*

$$DTW_p(ts_x, ts_y) + DTW_p(ts_y, ts_z) \geq \frac{DTW_p(ts_x, ts_z)}{\min(2w+1, m)^{1/p}} \tag{5.16}$$

where $DTW_p(ts_x, ts_y)$ *is the* l_p *dynamic time warping distance between* ts_x *and* ts_y*. And* w *is DTW locality constraint on an alignment path* $\boldsymbol{\pi}$*.*

Now, we can modify Lemmas 5.2.1, and 5.2.2 to obtain upper and lower bounds on DTW distance.

Corollary 5.3.1. *Let ts_x, ts_y, ts_z be any three time series in* **TS**. *If $DTW_p(ts_y, ts_z) \geq 2\min(2w+1, m)^{1/p} DTW_p(ts_y, ts_x)$, then $DTW_p(ts_z, ts_x) \geq DTW_p(ts_y, ts_x)$.*

Proof. From the Theorem 5.3

$$\frac{DTW_p(ts_y, ts_z)}{\min(2w+1, m)^{1/p}} \leq DTW_p(ts_y, ts_x) + DTW_p(ts_z, ts_x),$$

hence

$$\frac{DTW_p(ts_y, ts_z)}{\min(2w+1, m)^{1/p}} - DTW_p(ts_y, ts_x) \leq DTW_p(ts_z, ts_x).$$

Therefore, using the assumption of the corollary, the left side of inequality

$$\frac{DTW_p(ts_y, ts_z)}{\min(2w+1, m)^{1/p}} - DTW_p(ts_y, ts_x) \geq 2DTW_p(ts_y, ts_x) - DTW_p(ts_y, ts_x) = DTW_p(ts_y, ts_x).$$

Thus, $DTW_p(ts_z, ts_x) \geq DTW_p(ts_y, ts_x)$. ■

Corollary 5.3.2. *Let ts_x, ts_y, ts_z be time series in* **TS**. *Then*

$$DTW_p(ts_z, ts_x) \geq max\{0, \frac{DTW_p(ts_y, ts_x)}{\min(2w+1, m)^{1/p}} - DTW_p(ts_y, ts_z)\} \tag{5.17}$$

.

Proof. From the Theorem 5.3

$$\frac{DTW_p(ts_y, ts_x)}{\min(2w+1, m)^{1/p}} \leq DTW_p(ts_z, ts_x) + DTW_p(ts_y, ts_z),$$

then

$$DTW_p(ts_z, ts_x) \geq \frac{DTW_p(ts_y, ts_x)}{\min(2w+1, m)^{1/p}} - DTW_p(ts_y, ts_z).$$

And $DTW_p(ts_z, ts_x) \geq 0$ is always true by definition of distance measure. ■

In our case, we fix l_2 norm (denote as DTW) and we do not impose constraints on an alignment path, hence $w = \infty$. Thus, the constant $min(2w+1, m)^{1/p}$ will be replaced by simple term of $\sqrt{m}$ in the Equation 5.3.1. Then, the only thing that we need to embed into DTW for local search algorithm is to introduce a way by which the bounds are updated. We formulate it as follows:

$$\mathfrak{u}_{a(ts)}(ts) = \sqrt{m}\left(\mathfrak{u}_{a(ts)}(ts) + \delta(a(ts))\right) \tag{5.18}$$

$$\mathfrak{l}_{a(ts)}(ts) = \max\left(\frac{\mathfrak{l}_{a(ts)}(ts)}{\sqrt{m}} - \delta(a(ts)); 0\right) \tag{5.19}$$

$$\mathfrak{l}_{\bar{a}(ts)}(ts) = \max\left(\frac{\mathfrak{l}_{\bar{a}(ts)}(ts)}{\sqrt{m}} - \delta(\bar{a}(ts)); 0\right). \tag{5.20}$$

The equations above are dedicated to hyperplane split operator. In the case of hypersphere split operator we have:

$$\mathfrak{u}(ts) = \sqrt{m}\,(\mathfrak{u}(ts) + \mathrm{DTW}(ts_L, ts_{nn})) \tag{5.21}$$

$$\mathfrak{l}(ts) = \max\Big(\frac{\mathfrak{l}(ts)}{\sqrt{m}} - \mathrm{DTW}(ts_L, ts_{nn}); 0\Big) \tag{5.22}$$

By substituting formulas in lines 9 - 11 of the Algorithm 5.2 by 5.18 - 5.20 and formulas in lines 9 - 10 in the Algorithm 5.4 by 5.21, 5.22, we obtain Local Search node split with DTW distance.

On employing $(1 - \mathrm{Cort}_\pi)$ **dissimilarity.** It would be highly desirable to include a behavioural distance measure such as $(1 - \mathrm{Cort}_\pi)$ into *LSTDT*. In this case, we can have a complete approximated algorithm of *FSTDT*. By analogy with Theorem 5.3 for DTW, one can show that the weak triangle inequality holds for $(1 - \mathrm{Cort}_\pi)$ distance.

$$2 - \mathrm{Cort}_\pi(ts_x, ts_y) - \mathrm{Cort}_\pi(ts_y, ts_z) \geq \frac{1 - \mathrm{Cort}_\pi(ts_x, ts_z)}{\min(2w+1, m)^{1/p}} \tag{5.23}$$

To obtain bounds for distances with dynamic path alignment, we used weak triangle inequality, which is loose. Therefore, the bound $u_a(ts)$ is not tight enough, and it grows proportionally with the length of time series m. We can end up with the vacuous bound for long time series.

5.4 Empirical study.

In the empirical study, we pursue to evaluate performance of *LSTDT* algorithm in terms of the size of the tree and classification accuracy. We measure the size of yielded trees in order to check if the approximated algorithm is able to output readable models. We compare results of *LSTDT* with that of *FSTDT*. In all experiments we employ hypersphere or/and hyperplane split operators. Since *LSTDT* is the approximated version of *FSTDT*, we, surely, can not guarantee to have the same performance in terms of classification accuracy. However, we advocate that *LSTDT* is able to find a split pair that is good enough to yield a tree that is close in terms of number of internal nodes and accuracy to the tree grown by full search procedure.

All results that we reveal below were obtained on univariate time series datasets provided by the UCR benchmark [15]. The characteristics of dataset can be found in the Table 4.1.

5.4.1 Triangle inequality violation test.

The first part of our experiments focuses on level of triangle inequality violations when we employ DTW and $(1 - \text{Cort}_\pi)$. To assess, whether there are violations or not, we generated all possible triples $(ts_x, ts_y, ts_z \in \mathbf{TS})$ for each dataset and for each triple, the following value was worked out:

$$V(ts_x, ts_y, ts_z) = \frac{d(ts_x, ts_z)}{(d(ts_x, ts_y) + d(ts_z, ts_y))} \tag{5.24}$$

where $d(\cdot, \cdot)$ is either DTW or $(1 - \text{Cort}_\pi)$. We repeated this procedure to verify if both distances satisfy weak triangle inequality. To do so, we used the relaxed version of triangle inequality to compute a new value $V_{weak}(ts_x, ts_y, ts_z)$:

$$V_{weak}(ts_x, ts_y, ts_z) = \frac{d(ts_x, ts_z)}{\sqrt{m} * (d(ts_x, ts_y) + d(ts_z, ts_y))} \tag{5.25}$$

Table 5.1 provides pivot results across 46 datasets. Actually, we can notice that the level of triangle inequality violations for DTW is not high, only 0.4% from all generated triples. However, the $(1 - \text{Cort}_\pi)$ dissimilarity measure has a more high level of triangle inequality violations, that is 6.6% of total triples quantity. By emploing the weak triangle inequality, we obtain positive result for both disimilarity measures, which shows no violations across all triples generated on 46 UCR datasets.

Table 5.1: Statistics on the level of triangle inequality violations by DTW and $(1 - \text{Cort}_\pi)$ distances on 46 UCR datasets.

Distance measure	TOT # triples	V	V_{weak}	$V, \%$	$V_{weak}, \%$
DTW	126 811 644	476 592	0	0.4	0.0
$(1 - \text{Cort}_\pi)$		8 334 245	0	6.6	0.0

5.4.2 Local Search vs Full Search Temporal Decision Tree

Experiments were designed in order to compare Local Search TDT and Full Search TDT. Both models are built for different distances that are Euclidean, DTW and $(1 - \text{Cort}_\pi)$. For each distance, results are obtained with hyperplane and hypersphere split operators as well as their combination. We picked 21 out of 46 UCR datasets for these experiments with the training set size between 100 and 600 time series. We choose this range to have representative set of datasets, like that each of them is not too small to be able to tune parameter $LSDist$. Since we also perform $FSTDT$ for each dataset, for the sake of keeping reasonable running time, we get rid of some large datasets. There are three input parameters that we need to set in our experiments.

1. The number of referenced time series r that will form initial seeds for local search exploration. We choose to set this value as a function of input set size, $r = \log_2 N$. It grows slowly with respect to the training set size. Hence, it will prevent from taking too much initial time series r in large datasets.

2. The second input parameter is the number of allocated distances for local seach $LSDist$. Since this parameter is defined at each internal node of a tree, it should be reduced proportionally to the number of time series that a sub-node contains. We set $LSDist$ as a function that depends only on r and the number of time series in a node g: $|\mathbf{TS}_g| = N$.

$$LSDist(r, N) = C \times r \times (r-1) \times \sqrt{N}, \tag{5.26}$$

   where C is a constant (we fixed it to be equal 6).

3. The third parameter is the number of nearest neighbours NN of each seed candidate that we will explore. It is fixed to be equal 5 for all carried experiments.

Figure 5.3 shows how the number of total distance computations ($TDist$) grows with respect to the training size if we would employ $FSTDT$ (upper left graph). On the upper right plot, we can see the number of reference time series r for different values of the training set. Finally, the bottom plot of the figure shows the percentage of allocated distance computations $LSDist$ taken from $TDist$ with respect to Formula 5.26. The curve, marked as $INDist$, shows the percentage of distance computations that are carried out to assess split candidates generated from initial time series r.

Tables 5.2 - 5.5 include obtained results summarized across 21 UCR datasets. The number of internal nodes and accuracy are two key performance indicators for comparison of $LSTDT$ with $FSTDT$. At the same time, we are also interested in measuring the total number of computed distances and examined split candidates by the algorithm. It reveals how many computational resources can be saved. For each split operator, results are presented by three columns: FS stands for $FSTDT$, LS stands for $LSTDT$, and LS/FS is the ratio value of results obtained by $LSTDT$ to $FSTDT$ results. Each resultant value represents accumulated value over each yielded model for all 21 UCR datasets. For example, the number of split candidates is the value collected from each internal node of a tree.

Regardless of employed dissimilarity measure and selected split operator, $LSTDT$ outputs tree models which are almost the same, in terms of accuracy and size, to those that are obtained by $FSTDT$. However, the size of decision trees yielded with

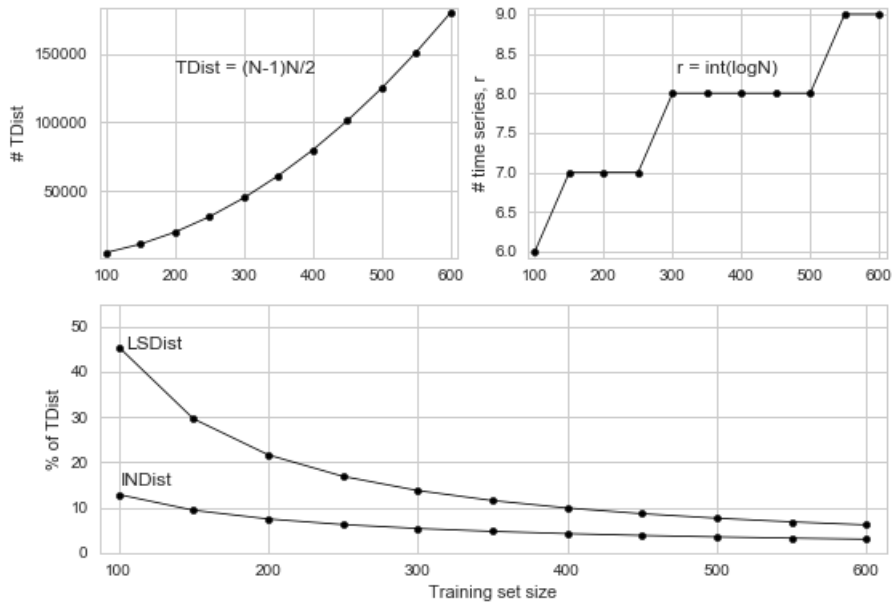

Figure 5.3: The values of input parameters of $LSTDT$ with respect to training set size. Left upper plot illustrates function of total number of distances $TDist$. Right upper plot reveals the number of referenced time series r taken as a logarithmic function (base 2) from the training set. Bottom graph shows change tendency of $INDist$ (Formula 5.6) and $LSDist$ (Formula 5.26).

use of hypersphere split operator $\mathcal{F}(S)$ is less distinct (from the best size, obtained by $FSTDT$) than for those with hyperplane split operator $\mathcal{F}(H)$. For example, trees obtained by $LSTDT$ with $\mathcal{F}(S)$ on $l2$ dissimilarity measure are bigger only by 1%. Though, one can note the algorithm explores the half of the number of explored split candidates (46%). While $LSTDT$ with $\mathcal{F}(H)$ assessed only 6% of split candidates and yielded trees that are bigger by 14%. Again, joining two spit operators shows positive impact on the size of trees, the difference between models of $FSTDT$ and $LSTDT$ is about 5% for all dissimilarity measures.

Performance results of $LSTDT$ models in terms of accuracy are consistently good and do not deteriorate for all experiments. And with less than 50% of all distance computations, the $LSTDT$ algorithm is capable of yielding trees of comparable size with those that were obtained by full search. In addition, since $LSTDT$ is the approximated version of $FSTDT$, which do not compute all distances and do not evaluate all split candidates, it yields models faster than $FSTDT$. Depending on the dissimilarity measure the algorithm works faster than full search by factor from 2 to 4. The smallest running time improvement was obtained for $(1 - \text{Cort}_{\pi})$ dissimilarity measure. It can be explained by the fact that it takes almost 10% more distance computations, which comes together with the fact that this dissimilarity

measure has higher level of triangles inequality violations than DTW. As a consequence, Conditions 5.10 and 5.11 for $\mathcal{F}(H)$ or Condition 5.15 do not hold leading to computation of the exact distance between pair of time series.

To this end, we run experiments of both *FSTDT* and *LSTDT* models with using the complete range of dissimilarity measures and with assessment of split candidates generated from both operators $\mathcal{F}(H)$ and $\mathcal{F}(S)$. Results are provided in the Table 5.5. With 30% of computed distances and only 2% of explored split candidates, we got comparable results with respect to *FSTDT* trees. The size of trees is bigger by 9.8% and accuracy is less than 1%. These results were obtained 9× faster than for *FSTDT*. Note, that we employed the same function for *LSDist* that does not depend on the number of dissimilarity measures that the algorithm uses. That explains the fact of observing less number of distance computations in this experiment than for those obtained previously (Tables 5.2 - 5.4).

The *LSTDT* algorithm provides solution for yielding trees that are close to those built by using exhaustive search over split candidates space at each internal node. At the same time it allows skipping a large amount of distance computations and split candidates assessment. As we observe in the results of local search we examine in total $|\mathcal{F}(H) + \mathcal{F}(S)|$ candidates that is much less than total number, i.e., $|\mathcal{F}(H) + \mathcal{F}(S)| \ll N^2 + N$. Practically, by assessing only 2% (Table 5.5) of total number split candidates, we obtained consistent positive results with compare to the full search.

Table 5.2: Results of *LSTDT* algorithm on 21 UCR datasets with l_2 dissimilarity measure.

	$\mathcal{F}(S)$			$\mathcal{F}(H)$			$\mathcal{F}(H)\&\mathcal{F}(S)$		
	FS	*LS*	*LS/FS*	*FS*	*LS*	*LS/FS*	*FS*	*LS*	*LS/FS*
# Internal nodes	864	870	101%	1321	1512	114%	803	841	105%
# Distances	915 261	382 158	42%	915 261	343 511	38%	915 261	383 582	42%
# Split candidates	46 074	21 033	46%	1 308 914	84 206	6%	1 354 988	112 176	8%
Running time (s)	1397	313	22%	286	135	47%	1350	406	30%
Accuracy (%)	70.9	71.1	100%	69.2	69.2	100%	71.0	71.4	101%

Table 5.3: Results of *LSTDT* algorithm on 21 UCR datasets with DTW dissimilarity measure.

	$\mathcal{F}(S)$			$\mathcal{F}(H)$			$\mathcal{F}(H)\&\mathcal{F}(S)$		
	FS	*LS*	*LS/FS*	*FS*	*LS*	*LS/FS*	*FS*	*LS*	*LS/FS*
# Internal nodes	794	803	101%	1256	1451	116%	745	777	104%
# Distances	915 261	381 085	42%	915 261	333 068	36%	915 261	396 053	43%
# Split candidates	42 415	16 930	40%	1 376 071	68 495	5%	1 702 014	103 463	6%
Running time (s)	1359	284	21%	359	128	36%	1770	438	25%
Accuracy (%)	70.3	71.5	102%	69.9	70.0	100%	71.0	71.9	101%

On tuning *LSDist*. In the previous experiments, we set the parameter *LSDist* as a function depending on the number of referenced time series r and size of input set. In this part of experiments, we modify the equation of *LSDist* by adding two

Table 5.4: Results of *LSTDT* algorithm on 21 UCR datasets with $(1 - \text{Cort}_\pi)$ dissimilarity measure.

	$\mathcal{F}(S)$			$\mathcal{F}(H)$			$\mathcal{F}(H)\&\mathcal{F}(S)$		
	FS	*LS*	*LS/FS*	*FS*	*LS*	*LS/FS*	*FS*	*LS*	*LS/FS*
# Internal nodes	871	904	104%	1440	1660	115%	836	880	105%
# Distances	915 261	457 327	50%	915 261	384 825	42%	915 261	466 024	51%
# Split candidates	44 386	22 900	52%	1 441 314	107 520	7%	1 632 606	145 464	9%
Running time (s)	1635	815	50%	635	364	57%	1789	1023	57%
Accuracy (%)	68.4	67.1	98%	65.7	65.2	99%	69.0	67.8	98%

Table 5.5: Results of *LSTDT* algorithm on 21 UCR datasets with Euclidean, DTW and $(1 - \text{Cort}_\pi)$ dissimilarity measure.

	$\mathcal{F}(H)\&\mathcal{F}(S)$		
	FS	*LS*	*LS/FS*
# Internal nodes	584	641	110%
# Distances	2 745 783	816 459	30%
# Split candidates	5 228 118	104 135	2%
Running time (s)	6222	704	11%
Accuracy (%)	75.2	74.6	99%

new terms. The first term is the number of used dissimilarity measures $|d|$. Each additional distance measure requires distance matrix computation. Each seed split candidate $(ts_L, ts_R) \in \mathcal{F}(H)$ or $(ts_L, \theta) \in \mathcal{F}(S)$ is generated for each distance d, thus we need to allocate resources to explore vicinity of a pair given a distance d. In our experiments this term is equal to 3, because we use Euclidean, DTW, $(1 - \text{Cort}_\pi)$ measures. The second term is the percentage of local search $ps \in [0\%, 100\%]$. We introduce this parameter for two reasons:

1. to get rid of the explicit parameter on the exact number of distance computations *LSDist* and provide a user with a "friendly" parameter of search level over split candidate space;

2. to tune *LSDist* by navigating level of local search between initial search and full search. The former explores only split candidates generated from randomly picked r time series, the latter evaluates all possible split candidates.

The new equation of *LSDist* is defined as follows:

$$LSDist(r, N, ps, |d|) = \frac{1}{2} \times ps \times (N - r) \times (N - 1) \times |d| \tag{5.27}$$

When ps is equal to 0, the algorithm has to examine only split candidates generated from r time series. As we mentioned, we pick them randomly at each node of a tree, so this case corresponds to the Initial search (Algorithm 5.1), denoted as INS. By increasing ps, the value of *LSDist*, which local search algorithm can use to examine

new split candidates, grows as well. Note, that the full search corresponds to the value ps a little bit less than 100%, since there are distances that have been already computed ($INDist$) to assess split candidates in Initial search. The number of referenced series r is the same as in the previous experiments, $r = \log_2(N)$.

Table 5.6: Results of $LSTDT$ algorithm for different values of the percentage parameter ps for $LSDist$. Values in parenthesis show the corresponding ratio value of $LSTDT$ with respect to $FSTDT$. Dissimilarity measures are l_2, DTW, and $(1 - \text{Cort}_\pi)$.

				$\mathcal{F}(H)\&\mathcal{F}(S)$			
	INS	5%	10%	20%	30%	50%	FS
# Internal nodes	806 (138%)	718 (123%)	680 (116%)	637 (109%)	642 (110%)	628 (108%)	584
# Distances	$258K$ (9%)	$729K$ (27%)	$1.08M$ (39%)	$1.59M$ (58%)	$1.85M$ (68%)	$2.2M$ (81%)	$2.7M$
# Split candidates	$23K$ (0.4%)	$32K$ (0.6%)	$90K$ (1.7%)	$303K$ (5.8%)	$585K$ (11.2%)	$925K$ (17.7%)	$5.2M$
Running time (s)	302 (5%)	837 (13%)	1422 (23%)	3017 (48%)	5517 (89%)	9084 (146%)	6222
Accuracy (%)	73.1 (97%)	74.8 (99%)	75.5 (100%)	75.1 (100%)	75.5 (100%)	75.0 (100%)	75.2

Results are presented in the Table 5.6. We fixed parameter ps to be in the set $\{0\%, 5\%, 10\%, 20\%, 30\%, 50\%\}$. As it will be shown shortly, going beyond a certain value of ps, the $LSTDT$ algorithm becomes inneficient. The last column contains results of trees built using exhaustive search over split candidates space (FS). Trees models built with the Initial search has the least accuracy (73.1%) and the number of internal nodes increases by 38% in comparison with the full search. By increasing the value of $LSDist$, results tend to improve. Figure 5.4, illustrates how the performance of yielded trees improves with increase of $LSDist$. We have significant nodes reduction in comparison with only Initial search for $LSDist$ with a parameter ps equals to 5%, 10%, and 20%. The nearly optimal results with respect to the full search are obtained for the values $ps = 20\%$. There is no significant improvement neither in terms of accuracy nor in terms of trees size for the value of $30\% \leq ps \leq 50\%$. However, the number of computed distances increases drastically from 57.8% to 80.5%, and we observe almost three times rise in the number of split candidates (from 5.8% to 17.7%). This point can be seen in the Figure 5.5, where the percentage of computed distances and examined split candidates are illustrated for each setup of ps.

In addition, we observe overhead of the running time for results corresponding to $ps = 50\%$. The algorithm works 1.5 times slower than the full search $FSTDT$. To analyse the obtained results and understand the reasons we factorized running time for each step of local search algorithm. Briefly, they are

1. Initial Search (Algorithm 5.1);

2. Local search split candidates exploration (Algorithms 5.2 and 5.4)

Table 5.7: Running time (in seconds) of each step of *LSTDT* algorithm for different values of the percentage parameter *ps* for *LSDist*. Dissimilarity measures are l_2, DTW, and $(1 - \mathrm{Cort}_\pi)$.

		$\mathcal{F}(H)\&\mathcal{F}(S)$				
	INS	5%	10%	20%	30%	50%
Initial Search	302	312	302	279	260	266
Local Search		525	1120	2738	5257	8817
Seed List Sorting		2	24	395	1335	2532
LS $\mathcal{F}(S)$		415	802	1130	1196	1535
LS $\mathcal{F}(H)$		108	295	1213	2726	4750
Total	302	837	1422	3017	5517	9084

(a) Selection of seed candidate which vicinity will be explored (line 2 in Algorithms 5.2 and 5.4);

(b) Exploration of split candidate (f_H or f_S) from the vicinity of seed candidate.

Running time in seconds of each step is shown in Table 5.7. The complexity of the first step is $O(r^2N)$. The value of r is fixed for all experiments We observe small variations in the execution time; since for each experiment the trees structure is different and, as the result, the number of input time series in a node will be different. It leads to different values of $INDist$ which, in turn, explains the running time variations. If we set $r = N$, the algorithm will generate all split candidates from reference time series, and the Initial search will correspond to the $FSTDT$.

The complexity of the second step is $O(LSDist \log LSDist)$. When we increase *ps*, the value of $LSDist$ increases too and it explores a search space of split candidate which almost the same as that of the full search. The step of local search becomes costly with increase of *ps*, because evaluation of one split candidate is $O(\log LSDist)$, the value that is greater than N. Beyond a certain value of *ps* (20%) employing $LSDist$ becomes inneficient both in terms of running time and the model performance improvement.

Level of search in terms of distance computation can be defined by the user. While in this experimental part we employed the parameter *ps* to regulate local search level, in general it would be recommended to use global parameter of search. The search level $S^{lev} \in [s_{min}\%, 100\%]$ is the percentage of allowed search in the split candidate space, which the algorithm converts to the exact number of distance computations. Therefore, the minimum values of search level s_{min} actually depends on r and equals to $\frac{INDist}{TDist}\%$. The value of $LSDist$ is the number of remaining distances allocated for search after $INDist$ substraction ($LSDist = TDist \times S^{lev} - INDist$). By tuning

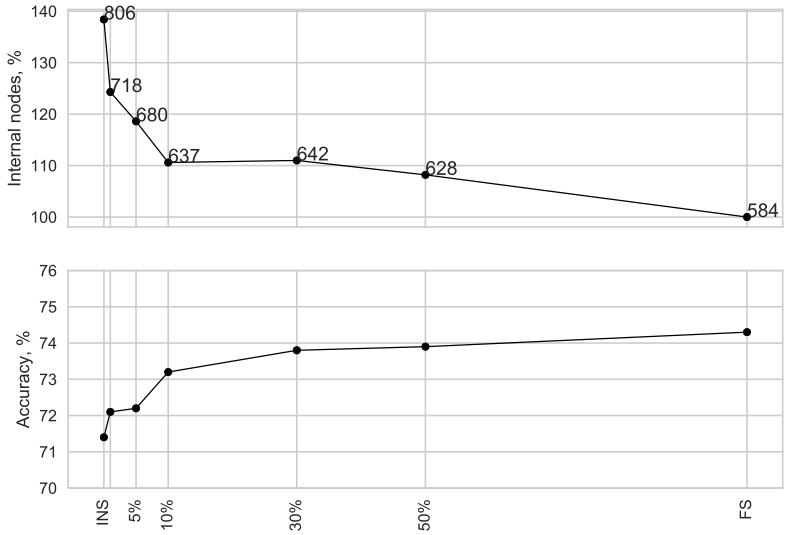

Figure 5.4: Upper plot represents the percentage of internal nodes of trees obtained by *LSTDT* with respect to *FSTDT*. Annotation of points corresponds to the number of internal nodes across trees for 21 UCR dataset. Bottom plot shows results of accuracy obtained for each level of local search. X-axis reveals different setups for input parameter *ps* indicating local search level. *RS* stands for random search, FS stands for full search.

parameter S^{lev}, a user can navigate between *LSTDT* and *FSTDT* algorithms. Notice that when $S^{lev} = s_{min}\%$, we end up with a tree built with Initial Search step of *LSTDT*, because the best split is found from a random subset of split candidates. In that case, it means that no other exploration of split candidates will be done, yet a classification model is guaranteed, possibly without an optimal partition in a node.

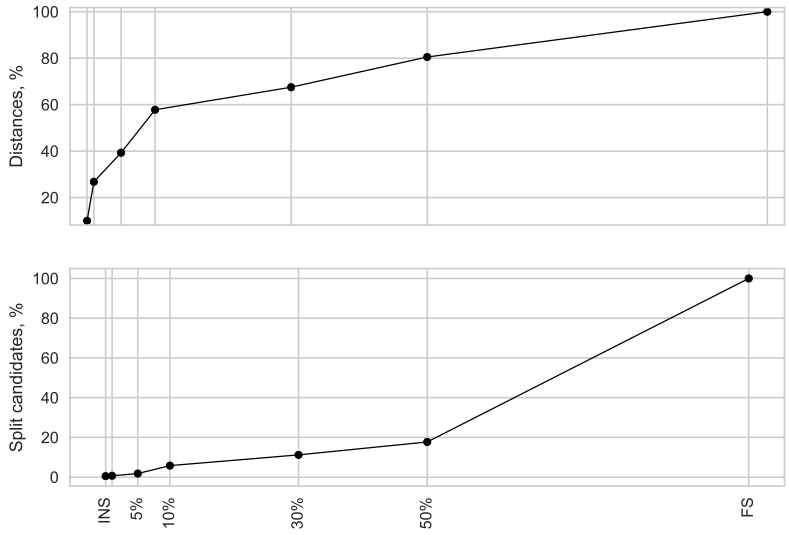

Figure 5.5: Upper plot shows the percentage of computed distances obtained with respect to total number distances in the case of use *FSTDT*. Bottom plot shows the percentage of examined split candidates by *LSTDT* with respect to *FSTDT*. X-axis reveals different setups for input parameter *ps* indicating local search level. *RS* stands for random search, FS stands for full search.

5.5 Conclusion

In this chapter, we developed approximated algorithm of Multi-operator Temporal Decision Trees with use of hyperplane and hypersphere split operators, called Local Search Temporal Decision Trees (*LSTDT*). By employing triangle inequality and bounds on distances between time series, we proposed local search of a node split for hyperplane split operator $\mathcal{F}(H)$ with use of Euclidean distance. Then, we extended the algorithm for hypersphere split operator $\mathcal{F}(S)$. To this end, by using the weak triangle inequality, we showed that the algorithm can be generalized for non static time series dissimilarity measures, such as DTW and $(1-\mathrm{Cort}_{\pi})$. Experimental results show that it is possible to eliminate evaluation of the majority of split candidates, while keeping yielded trees interpretable and accurate.

The proposed algorithm enables a user to maintain the level of search over space of split candidates via an input parameter on the allowed distance computations $LSDist$. It provides flexibility on training time of the model which depends on a domain expert requirements. By setting $LSDist$ to a small value, one can train quickly a tree model in order to get the first classification results and start analyzing them. Then, depending on the further needs, $LSDist$ can be increased, and another more accurate model can be yielded. However, with large values of $LSDist$, the algorithm becomes

inneficient and, in this case, it is more efficient to switch to the original Full Search algorithm. In the next chapter, we address the problem of emphasizing discriminative subinterval in split time series.

Chapter 6

Weighted Temporal Decision Trees

Contents

Trees which are built by MTDT with hyperplane and hypersphere split operators uses the entire time series. Best splits in a node are chosen by optimizing Gini class homogeneity criterion. Assignment of time series to each sub-node is based on its global similarity to a whole split time series. However, it can occur that some parts of time series are more discriminative than others. Similarity computation that is done on the entire time interval is not able to capture this information. In order to emphasize class distinguishing local parts of time series, we propose to weight each timestamp, according to its discriminative power.

In this chapter, we propose three different approaches to find a weight vector for each time series from the training set, computation of each of them depends on

the employed dissimilarity measure. We show, how to compute a weights vector using measures of different originality (static or dynamic, value or shape based). We present how to embed a weight vector to time series similarity computation. Then, we enrich input set of time series by its weighted version and feed it into Multioperator Temporal Decision Trees. The algorithm is called Weighted Multioperator Temporal Decision Trees (WMTDT). To this end, we provide empirical study of weights impact on accuracy and interpretability of yielded models by WMTDT. We conclude this chapter by discussing the results obtained by the proposed methods, explaining the pros and the cons of different weighting approaches.

6.1 Introduction

There are several objectives of finding a weight vector for a time series, which are reinforcing dicriminative features, improving the classification accuracy of a model and providing a user with complementary information about underlying class distinguishing patterns. The latter can be done by visual rendering of a time series weight vector, that aims at facilitating data understanding.

In the literature, there are few works where the authors addressed the problem of learning weights for time series. To the best of our knowledge, in [27, 28], several time series weighting strategies are discussed based on the learnt multiple temporal alignments for time series discrimination. In [93] , a locally weighting scheme is learned to discriminate time series on their neighborhood and enhance kNN classifier, whereas Soheily-Khah et al. [81] propose a generalized k-means-based clustering for temporal data under weighted and kernel time warp measures.

Let $(\mathbf{TS}, \mathcal{Y})$ be the training set of labelled time series and ts_r be a time series for which we aim to find a weight vector. Assume that the length of ts_r is m and its class label is y_r. A weight vector w is defined as $w = \{w_1, \ldots, w_m\}$ with $\sum_{t=1}^{m} w_t = 1$ where each entity $w_t \geq 0$ is a positive value at a given timestamp $t \in 1, m$.

Until generalization to dynamic dissimilarity measures DTW and $(1 - \mathrm{Cort}_\pi)$, we employ the Euclidean distance l_2. From a pairwise distance computation between ts_r and time series from the training set, one can extract distance values at each timestamp $t \in 1, m$. Let us consider for each timestamp t, the set $\{(d^t_{(1,r)}, y_1), \ldots, (d^t_{(N,r)}, y_N)\} \in (\mathbb{R} \times \mathcal{Y})^N$, where $y_i \in \mathcal{Y}$ is a class label of a time series $ts_i \in \mathbf{TS}$ and $d^t_{i,r}$ is the distance between ts_r and ts_i. This set can be divided into K subsets, denoted as D^t_k with $k \in \{1, \ldots, K\}$, labeled according the label values $\mathcal{Y}$. We consider that a

timestamp t has discriminative power if classes represented by values of time series are well separated one from other. To rephrase, given a distance measure d and a pair of classes $j, k \in \mathcal{Y}, j \neq k$, we say that values of D_k^t are different from D_j^t. The more they are different, the greater the importance of this timestamp, and it has to be emphasized during a time series similarity computation. Therefore, we aim to measure the level of classes separability in order to assign a weight to each timestamp t. Then, by incorporating weights to a distance value computation, will allow capturing discriminative local parts of time series.

In forthcoming sections, we address the problem of weight value computation for each timestamp t of an observed time series. In Section 6.2 we introduce three approaches to find a weight vector of a time series ts from **TS**. Section 6.3 provides formalization of weighted distance measures to compute similarity between weighted time series. In Section 6.4 we augment training set of time series by its weighted version and use it within Multi-operator Temporal Decision Trees (MTDT).

6.2 Time series weighting algorithms

6.2.1 Generative weighting algorithm: Between-Within discriminant criterion

Let $p^t(d|k)$ be the class-conditional density of pointwise distance values $d_{(i,r)}^t$ of elements D_k^t. The proposed algorithm of finding a weight vector is based on the statistical assumption that class-conditional distribution law $p^t(d|k)$ follows Gaussian law with parameters $\mu_k^t, (\sigma_k^t)^2$; $\mathcal{N}(\mu_k^t, (\sigma_k^t)^2)$. We can get unbiased estimation of these parameters for each class $k \in \mathcal{Y}$ from the set D_k^t.

We can consider that for each timestamp t the class of label k is represented by the scalar values $\{d_{(i,r)}^t, y_i = k\}_{N_k^t}$. Are these classes confused or well separated? The more separated they are, the more discriminative power a timestamp t has, hence the higher weights value must be assigned to it. In order to assign weights to timestamps, we use ratio of *between* class variance to *within* class variance, which is commonly used to quantify the separability of classes. Recall that at each timestamp t values $d_{(i,r)}^t$ have been divided into K classes D_k^t, following the label of timestamps. Each class contains N_k^t elements that can be described by its density law $p^t(d|k)$. We assume that this law is Gaussian, with mean $\mu_k^t = \frac{1}{N_k} \sum_{i:d_{i,r}^t \in D_k^t} d_{i,r}^t$ and variance $(\sigma_k^t)^2 = \frac{1}{N_k} \sum_{i:d_{i,r}^t \in D_k^t} (d_{i,r}^t - \mu_k^t)^2$.

6.2.1.1 Generic BW

In the following proposed formula for a weight computation, we focus on separability between all classes at once. That is to say, a high weight value must be assigned to a timestamp at which classes are maximally spread out. A weight value w_t can be computed as quotient:

$$w_t^{\mathbf{BW}} = \frac{\sum_{k \in \mathcal{Y}} \Phi_k^t (\mu_k^t - \mu^t)^2}{\sum_{k \in \mathcal{Y}} \Phi_k^t (\sigma_k^t)^2}, \tag{6.1}$$

where

$$\Phi_k^t = \frac{(N_k^t)^{1-\lambda}}{\sum_{k \in \mathcal{Y}} (N_k^t)^{1-\lambda}}, \lambda \in [0, 1] \tag{6.2}$$

with $\sum_{k=1}^{J} \Phi_k^t = 1$ and $\mu^t = \sum_{k \in \mathcal{Y}} \Phi_k^t \mu_k^t$ is a weighted pooled mean of distance values between a time series ts_r and the rest of the training set **TS** at a timestamp t. A constant λ in the class proportion parameter Φ_k^t is devoted to regularization of the class importance. By fixing $\lambda = 0$, the importance of each class in the Equation 6.1 is defined by the proportion of examples $\frac{N_k^t}{N^t}$ comprised in the training set at given timestamp t. Note, that the number of distance values we use to compute a weight t can vary at each timestamp and depends on the dissimilarity measure we employ (static or dynamic), that is why the term Φ_k^t depends on t. The value of $\lambda = 1$ leads to a uniform importance of each class in the computation of weights regardless of the number examples, $\Phi_k^t = \frac{1}{|\mathcal{Y}|}$. In case of highly unbalanced dataset, penalization of the biggest class influence on a weight computation is possible by setting $\lambda = 0.5$. A weight w_t takes its maximum values when between classes variance is high, while within variance is small, indicating class compactness, i.e., classes at timestamp t are well separated from each other. Equation 6.1 can capture the case where classes are diffused but separable. This approach is limited to the cases when noisy data lies far away from a class centroid leading to the increase of the within class variance. When some classes are represented by multimodal distribution, the estimation of its mean and variance will be poor, and the approach will not be capable of measuring separability correctly.

In the proposed formula of weight computation, we do not employ the information about the class label of the time series ts_r. A weight value represents a general level of

class separability. That is to say, it gives no information about which pair of classes are separable or not in the multiclass scenario. For example, in a three-class classification problem $\mathcal{Y} = \{1, 2, 3\}$ and timestamps t and $(t + q), q \in [1, m - t]$ can be weighted by a similar value $w_t \approxeq w_{(t+q)}$. Whereas at t, class 1 is well separated from others and at $(t + q)$ class 2 is well separated from others. Classification of a new time series using split, which contains a time series ts_r, will use simultaneously w_t and $w_{(t+q)}$, even if they do not represent the same separability information. However, using weighted split time series, where weights consistently represent separability of class 1 from others (or class 1 and 2 from others), will orient the partition in a node as follows: one sub-node will contain series of class 1 (or class 1 and 2), and the another node series of other classes. Therefore, we are expecting to get purer node partition and, as the result, better accuracy when a split contains weighted time series, where weight values convey the same discriminative information.

6.2.1.2 Class dependent BW

To cope with the last mentioned problem of possible timestamps class separability clashing in one weight vector, we propose a modification of the previously defined formula. Picking a time series ts_r and its class $y_r \in \mathcal{Y}$, we aim to compute the level of separability of this class from others. This change in the initial formula preserves consistency of information about which classes are discriminated across all timestamps. Basically, a class label of a time series ts_r works as a referenced one. We still measure general level of class label separability, but with respect to y_r. Each value w_t of a weight vector w reveals the cumulative level of separability of class y_r from all the others. We keep focus on aggregated discriminativeness of each class pair $(y_r, k), y_r \neq k$.

Taking into account these factors we modify and simplify the Equation 6.1 into its referenced-versus-one class (RVO) variant, which allows preserving consistency along timestamps of a weighted time series ts_r^w. The formula to compute a weight value is defined as follows:

$$w_t^{\mathbf{BW-RVO}} = \sum_{k \in \mathcal{Y}, k \neq y_r} \Phi_k^t \frac{|\mu_{y_r}^t - \mu_k^t|}{\sigma_{y_r}^t + \sigma_k^t}, \tag{6.3}$$

Equation 6.3 computes accumulated level of separability of class y_r from all other classes $k \in \mathcal{Y}$. Computational complexity of calculating the weight vector of BW approach using Equation 6.1 and 6.3 is $O(mN|\mathcal{Y}|)$ for one time series, where m is the length of time series. Equation 6.3 mitigates the problem when each weight value could convey contradictory information about which classes are separable.

6.2.2 Generative weighting algorithm: MAP classifier

In the previously introduced approach, weights are computed from empirical data **TS** and their values are not bounded above, i.e., $w_t \in (0, +\infty)$. As the result, a weight value measures absolute class separability at given timestamp t and it is difficult to compare importance of different timestamps. That is to say, if $w_t \ll w'_t$, then at most we can conclude that at a timestamp t' classes are more separable than at timestamp t, not being able to say to what extent. To overcome this problem, we propose the second algorithm, which outputs a weight value lying in the range of $(0, 1)$. Computational logic is based on Maximum a Posteriori (MAP) classifier error. We keep the assumption about Gaussianity over class distribution $p^t(d|k) \sim \mathcal{N}(\mu_k^t, (\sigma_k^2)^t), k \in \mathcal{Y}$. From the data $\{(d^t_{(1,r)}, y_1), \ldots, (d^t_{(N,r)}, y_N)\}$, we can estimate μ_k^t and $(\sigma_k^2)^t$ for each class $k \in \mathcal{Y}$. Then, for each time series $ts_i \in \mathbf{TS}$, we have a prior probability $p^t(d^t_{(i,r)}|k)$ at a given timestamp t to be assigned to one of a class $k \in \mathcal{Y}$. Hence, the predicted value is defined as:

$$\hat{y}_i = \operatorname*{argmax}_k p^t(k|d^t_{(i,r)}) = \operatorname*{argmax}_k p^t(k)p^t(d^t_{(i,r)}|k). \tag{6.4}$$

The higher the number of examples that is classified correctly ($\hat{y}_i = y_i$), the higher class separability level at a timestamp t is. Therefore, classification accuracy rate at a given timestamp t can be considered as a weight value. We propose to calculate the error rate of the MAP classifier method by computing the cumulative probability of missclassification. A point $d^t_{(i,r)}$ is correctly classified if its predicted class $\hat{y}_i$ (Equation 6.4) is the true class label of a time series ts_i. The probability to have a correct classifier is $p^t(\hat{y}_i = y_i)$. More precisely, the probability to correctly classify elements of a class $k \in \mathcal{Y}$ is $p^t(\hat{y} = k \text{ and } y = k)$. It is exactly the value

$$\int_{z|\hat{y}=k} p^t(k)p^t(z|k)dz.$$

The probability to missclassify elements of class k is equal to

$$1 - \int_{z|\hat{y}=k} p^t(k)p^t(z|k)dz.$$

While the probability of assigning elements of class k to class j is

$$\int_{z|\hat{y}=j} p^t(k)p^t(z|k)dz.$$

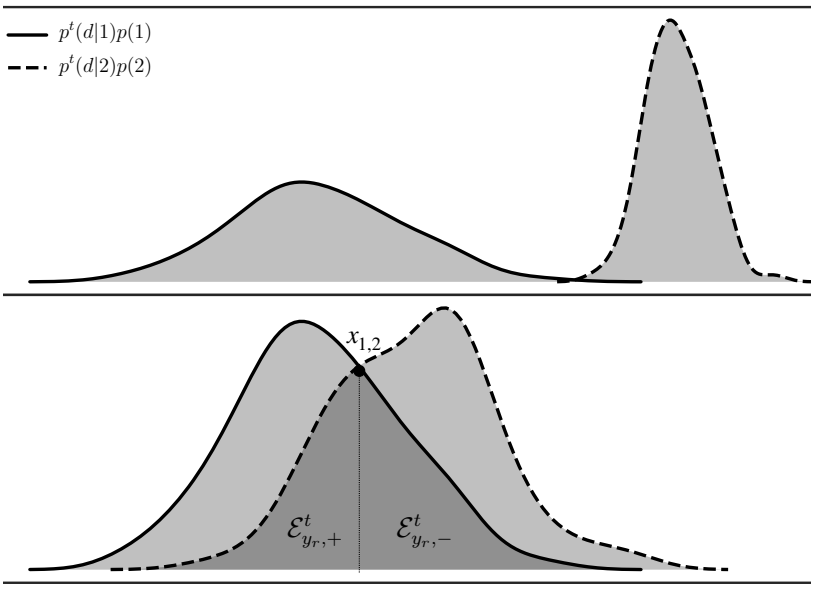

Figure 6.1: Posterior class conditional distributions of class 1 and class 2 at timestamp t. Upper plot illustrates the case where classes are totally separated, as a result a high weight value has to be assigned to a timestamp t. Bottom plot represents the case where classes are confused, shaded area on the left hand side from the point $x_{1,2}$ is the probability to missclassify examples of class 2, and on the right hand side of $x_{1,2}$ is the probability of misclassification error of class 1.

We can write it as follows:

$$p(\hat{y} = j, y = k) = \int_{R_j} p^t(k)p^t(z|k)dz, \tag{6.5}$$

where R_j is the region corresponding to assignment of values $d^t_{(\cdot,r)}$ to class $j \in \mathcal{Y}$. More formally:

$$R_j = \{d^t_{i,r} | \operatorname*{argmax}_k p^t(k)p^t(d^t_{(i,r)}|k = j)\} \tag{6.6}$$

Figure 6.1 illustrates the example of mixture of two Gaussian class conditional distributions for binary classification problem ($\mathcal{Y} = \{1, 2\}$). On the upper plot, posterior distributions of $p^t(1)p^t(d|1)$ and $p^t(2)p^t(d|2)$ do not overlap, signaling good discriminativeness ability of a timestamp t. While the bottom graph shows the case where class label distributions clash. Clearly, the higher level of intersection between $p^t(1)p^t(d|1)$ and $p^t(2)p^t(d|2)$, the lower level of a timestamp separability.

We choose to take into account the class of a split time series ts_r and to quantify the discriminativeness level of a given timestamp t as MAP corrects classification rate of the class y_r with respect to other classes from $\mathcal{Y}$. As shown in Figure 6.1, areas of class-conditional posterior laws intersection will represent the error rate. The less level

of their overlap, the more separable classes at a given timestamp t are. Let $p^t(k) = \Phi_k^t$ (Equation 6.2) be a prior probability of class k. The weight computation involves the following three steps:

1. research of intersection points between all posterior laws $p^t(k)p^t(d|k)$;
2. calculation of the probability of assignment points of a class $k \neq y_r$ to class y_r, $(\hat{y}_k = y_r)$;
3. calculation of the probability of assignment points of a class y_r to other classes $(\hat{y}_r = k)$.

Below, we show how to compute a weight value w_t:

- Find all intersection points $\{x_{y_r,k}\}$ for each $k \in \mathcal{Y}, y_r \neq k$ between $p^t(d|y_r) \sim \mathcal{N}(\mu_{y_r}^t, (\sigma_{y_r}^2)^t)$ and $p^t(d|k) \sim \mathcal{N}(\mu_k^t, (\sigma_k^2)^t$. If $(\sigma_{y_r}^2)^t \neq (\sigma_k^2)^t$, then there are always two intersection points $\{x_{y_r,k}^1, x_{y_r,k}^2\}$, which are the roots of the quadratic equation:

$$\begin{aligned} &Ax^2 + Bx + C = 0 \\ &A = \frac{1}{\sigma_k^2} - \frac{1}{\sigma_{y_r}^2} \\ &B = 2 * \left(\frac{\mu_{y_r}}{\sigma_{y_r}^2} - \frac{\mu_k}{\sigma_k^2}\right) \\ &C = \frac{\mu_k^2}{\sigma_k^2} - \frac{\mu_{y_r}^2}{\sigma_{y_r}^2} + \log\frac{\sigma_k^2}{\sigma_{y_r}^2} + \log\frac{p_{y_r}^2}{p_k^2} \end{aligned} \tag{6.7}$$

- Sort the set of obtained $\{x_{y_r,k}^i\}_{k\in\mathcal{Y}_k}, i \in [1,2]$. Let $\mathbf{R}_r$ be the set of intervals each of which defined by the set of intersection points. All intervals are found, except $]-\infty, x_m]$ and $[x_M, \infty[$, where x_m and x_M are the minimum and the maximum values of $\{x_{y_r,k}^i\}_{k\in\mathcal{Y}_k}, i \in [1,2]$. Each interval defines a region $R_{\hat{k}} \in \mathbf{R}_r$, where $\hat{k}$ is a dominated class in the interval defined as

$$\hat{k} = \underset{k\in\mathcal{Y}}{\operatorname{argmax}}\, p^t(d|k)p^t(k) \tag{6.8}$$

- Compute class conditional distributions overlap. It includes computation of two terms, which are $\mathcal{E}_{y_r,+}^t$ and $\mathcal{E}_{y_r,-}^t$. The former is equal to the probability of assigning elements of class k to the class $y_r \neq k$, while the latter is equal to the probability of assigning elements of the class y_r to a class $\hat{k} \neq y_r$. $\mathcal{E}_{y_r,+}^t$ can be

viewed as the probability with which points of any class distinct from y_r are attracted to it, whereas $\mathcal{E}^t_{y_r,-}$ is the probability with which points of a class y_r are repulsed from its own class towards a class $\hat{k}$. If class y_r is well separated from the others at estimated timestamp t, then both components $\mathcal{E}^t_{y_r,+}$ and $\mathcal{E}^t_{y_r,-}$ will get small values. Computation part is defined as follows:

$$\begin{aligned}\mathcal{E}_{y_r,+} &= \sum_{k\in\mathcal{Y},k\neq y_r}\int_{z\in R_{y_r}} p^t(k)p^t(z|k) \\ &= \sum_{k\in\mathcal{Y},k\neq y_r} p^t(k)\int_{z\in R_{y_r}} p^t(z|k) \\ &= \sum_{k\in\mathcal{Y},k\neq y_r} p^t(k)F_k(R_{y_r})\end{aligned} \tag{6.9}$$

$$\begin{aligned}\mathcal{E}_{y_r,-} &= \sum_{k\in\mathcal{Y},k\neq y_r}\int_{z\in R_k} p^t(y_r)p^t(z|y_r) \\ &= \sum_{k\in\mathcal{Y},k\neq y_r} p^t(y_r)\int_{z\in R_k} p^t(z|y_r) \\ &= \sum_{k\in\mathcal{Y},k\neq y_r} p^t(y_r)F_{y_r}(R_k)\end{aligned} \tag{6.10}$$

where $F_{y_r}(R_k) = \int_{z\in R_k} p^t(z|y_r)$ and $F_k(R_{y_r}) = \int_{z\in R_{y_r}} p^t(z|k)$.

- Compute a weight value w_t by using the following equation:

$$w_t^{\mathbf{MAP}} = 1 - (\mathcal{E}^t_{y_r,+} + \mathcal{E}^t_{y_r,-}) \tag{6.11}$$

A weight vector computation complexity of the MAP approach is $O(mN|\mathcal{Y}|^2)$ for a single time series.

6.2.3 Weighting algorithm with Linear SVM

Finally, we introduce the third algorithm of weight vector computation, where we look at the problem slightly different than before. We do not impose the assumption that the distance values of each class are drawn from Gaussian distribution. The idea is to address the problem of time series weighting as feature selection problem and to learn a weight vector w at once for the whole time series ts_r. Each timestamp is considered as a feature and each time series $ts_i \neq ts_r$ is an observation. Hence, an input of this learning problem is described by m features, each of which takes a value $d^t_{(i,r)}, t \in 1,\ldots,m$. By training a linear classifier, one can obtain a decision hyperplane

described by a set of parameters, where each parameter describes the importance of each feature. Therefore, parameters of a learnt classifier can be taken as timestamps weights. We limit ourselves to linear support vector classifier (SVC). Because of the nature of split, we aim to do at a node of the decision tree.

Let us formalize the learning problem. For each pair of classes $\{y_r, k\} \in \mathcal{Y}$ we train a binary SVC classifier on the training set $\{(\mathbf{d}_{(1,r)}, y_1) \dots (\mathbf{d}_{(l,r)}, y_l)\} \subset \mathbb{R}^m$, $l = N_k + N_{y_r} - 1$. Recall that $\mathbf{d}_{(i,r)}$ is a vector of $\{d_{(i,r)^t}\}_{t=1,\dots,m}$. The aim is to find the decision function:

$$f(\mathbf{d}_{i,r}) = sign(\langle \boldsymbol{\beta}_{(y_r,k)}, \mathbf{d}_{i,r} \rangle + \beta_0) \tag{6.12}$$

that separates classes y_r and k in the best way. We can write this problem as a convex optimization problem:

$$\begin{aligned} \underset{\boldsymbol{\beta}_{(y_r,k)} \in \mathcal{R}^m}{\text{minimize}} \quad & \frac{1}{2}||\boldsymbol{\beta}_{(y_r,k)}||^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{\beta}^T_{(y_r,k)}\mathbf{d}_{i,r} + \beta_0) \geq 1 - \xi_i \quad \xi_i \geq 0, i = 1, \dots, l \end{aligned} \tag{6.13}$$

The coefficients $\boldsymbol{\beta}_{(y_r,k)}$ from a trained classifier are taken as bases for timestamps weights. The high value of a coefficient $\beta^t_{(y_r,k)}$ at a timestamp t indicates its power to discriminate class y_r and k. By training $K-1$ classifier, we obtain $K-1$ coefficient vectors $\beta^t_{(y_r,k)}$. We propose to combine them to obtain the weight vector w of a time series ts_r. A value of weight w_t at timestamp t is defined as the linear combination of squared coefficients of each class k multiplied by its proportion Φ_k.

$$w_t^{\mathbf{SVC}} = \sum_{k \in \mathcal{Y}, k \neq y_r} \Phi_k (\beta^t_{(y_r,k)})^2, \tag{6.14}$$

where Φ_k defined in the same way as in the Equation 6.2.

In addition, we propose the second formula for this approach that takes the square root of the obtained sum. It allows smoothing of highly weighted values. We will check out how it affects results in practice in the section dedicated to the empirical study.

$$w_t^{\mathbf{RSVC}} = \sqrt{\sum_{k \in \mathcal{Y}, k \neq y^w} \Phi_k (\beta^t_{(y_r,k)})^2}, \tag{6.15}$$

Computational complexity of a weight vector of one time series is quadratical $O(m|\mathcal{Y}|N^2)$ with respect to the number of input time series N.

6.3 Application of weights to similarity computation

In the previous section, we formalized how to compute a weight vector for a time series ts_r. In this section, we describe how to obtain sets of $D_t^k, k \in \mathcal{Y}$ and how to use a weight vector w in a pairwise distance computation between two time series. We consider three dissimilarity measures, which are Euclidean, DTW and $(1 - \text{Cort}_\pi)$. Defining this range of basic measures, we are able to take into account value and shape characteristics of time series. Regardless of the distance, the property of symmetry does not hold anymore, which means $d(ts_i^w, ts_j) \neq d(ts_j^w, ts_i)$ since weight vectors $w_i \neq w_j$.

As we mentioned before, weights depend on employed dissimilarity measure to calculate values $d_{(i,r)}^t$. When we compute a weight for static dissimilarity measure, which implies that time series are aligned, only one value $d_{(i,r)}^t$ corresponds to a timestamp t of time series ts_r. However, once we use dynamic measure of dissimilarity, several values of $d_{(i,r)}^t$ can be obtained at a timestamp t, each of which corresponds to a timestamp $t', t \leq t'$ from an alignment path $\boldsymbol{\pi}$ of ts_r with ts_i. Below, we show how to come up with the sets of distances $D_t^k, k \in \mathcal{Y}$ for each dissimilarity measure we use in order to compute a weight vector w.

6.3.1 Euclidean dissimilarity

By using Euclidean distance to compute similarity between two time series, we aim to capture amplitude value differences. From the formula that computes distance values between a split time series ts_r and a time series ts_i from the training set

$$d(ts_r, ts_i) = \Big(\sum_{t=1}^{m} (v_r^t - v_i^t)^2 \Big)^{\frac{1}{2}}$$

We extract $d_{(i,r)}^t = |v_r^t - v_i^t|$ for a timestamp t we aim to weight. Thus, when we speak about classes discriminativeness at a timestamp t we imply that they are separable in terms of time series amplitude difference.

To compute a weighted distance $d(ts_r^w, ts_i)$, we refer to a weight vector w associated with a time series ts_r and formalize it through the following equation:

$$\boldsymbol{l_2}(ts_r^w, ts_i) = \Big(\sum_{t=1}^{m} w_t (v_r^t - v_i^t)^2 \Big)^{\frac{1}{2}} \tag{6.16}$$

6.3.2 DTW dissimilarity

The DTW between two time series ts_r and ts_i is defined by:

$$\mathrm{DTW}(ts_r, ts_i) = \min_{\boldsymbol{\pi}} \Big(\sum_{l=1}^{|\boldsymbol{\pi}|} |v_r - v_i|_{\pi_l} \Big), \tag{6.17}$$

where $\boldsymbol{\pi}$ is a warping path between two time series ts_r and ts_i, defined as a sequence of $|\boldsymbol{\pi}|$ pairs. The lth element of a path $\pi_l = (t, t')_l$ indicates a mapping of a timestamp t to a t' of time series ts_r and ts_i respectively.

Defining set of distances for a weight computation. DTW is able to alleviate distortion along time axis by finding an optimal alignment path $\boldsymbol{\pi}$, leading to the case when a timestamp t of one time series is mapped to multiple timestamps of another time series. Then, we need to define how to take it into account in a representative set of distances $\{(d^t_{(1,r)}, y_1), \ldots, (d^t_{(N,r)}, y_N)\} \in (\mathbb{R} \times \mathcal{Y})^N$ for a weight computation. We distinct two cases of modification of distances, each depends on the weight vector computation algorithm we use.

If a given timestamp t of a time series ts_r is mapped to a set of timestamps $\{t', (t'+1), \ldots, (t'+q)\}, q \in \mathcal{N}, t' \leq q \leq |ts_i|$ of a time series ts_i, then

- in Equation 6.1, 6.3 ($w_t^{\mathbf{BW}}$), or Equation 6.11($w_t^{\mathbf{MAP}}$) the distance scalar $d^t_{(i,r)}$ will be represented by the vector of q distance values

$$\{d^{(t,t')}_{(i,r)}, d^{(t,(t'+1))}_{(i,r)}, \ldots, d^{(t,(t'+q))}_{(i,r)}\}$$

  .

- in Equation 6.14 ($w_t^{\mathbf{SVC}}$) or 6.15 ($w_t^{\mathbf{RSVC}}$) we aggregate multimatched values to one value. In these approaches we consider each timestamp t as a feature, hence we have to end up with the same number for each ts_i. The value $d^t_{(i,r)}$ will be represented by the sum of q distance values

$$d^t_{(i,r)} = \sum_{s=1}^{q} d^{(t,(t'+q))}_{(i,r)}. \tag{6.18}$$

Weighted distance computation. The weighted version of DTW dissimilarity measure between two time series (ts_r, ts_i) is defined as follows:

$$\mathrm{DTW}(ts^w_r, ts_j) = \sum_{l=1}^{|\boldsymbol{\pi}^*|} w_{\pi_l} |v_r - v_i|_{\pi^*_l}, \tag{6.19}$$

where w_{π_l} is a weight value at timestamp t of a time series ts_r corresponding to a mapping $\pi_l = (t, t')_l$. The optimal alignment path $\boldsymbol{\pi}^*$ in the Equation 6.19 is the one taken between non weighted time series, since we used it to compute weight values.

We use the optimal alignment path $\boldsymbol{\pi}^*$ defined between non weighted time series. Note that if a timestamp t is mapped to multiple timestamps, then for all such pairs multiplication is done by the same value of weight w_{π_l}.

6.3.3 $(1 - \mathrm{Cort}_\pi)$ dissimilarity

$(1 - \mathrm{Cort}_\pi)$ dissimilarity measure was developed to capture shape attributes of time series under comparison. Let us recall the version of the formula, proposed in [16] that mitigates the problem of non aligned time series by finding an optimal path $\boldsymbol{\pi}^*$.

$$1 - \mathrm{Cort}_\pi(ts_r, ts_i) = 1 - \min_{\boldsymbol{\pi}} \Big(\frac{\sum_{l=1}^{s} (xy)_{\pi_l}}{\sqrt{\sum_{l=1}^{s} x_{\pi_l}^2} \sqrt{\sum_{l=1}^{s} y_{\pi_l}^2}} \Big), \tag{6.20}$$

where $\pi_l = (t, t')_l$, $x_{\pi_l} = v_r^{(t+1)} - v_r^t$, $y_{\pi_l} = v_i^{(t'+1)} - v_i^{t'}$, $s = |\boldsymbol{\pi}|$.

By using $(1 - \mathrm{Cort}_\pi)$ to compare time series, we measure time dependent correlation of corresponding timestamps taking into account their simultaneous increasing or decreasing behaviour. Hence, when we talk about class separability at a given timestamp t the distances $d_{(i,r)}^t$ is represented by a correlation term $(xy)_{\pi_l}, \pi_l = (t, t')_l \in \boldsymbol{\pi}^*$.

Weighted distance computation The weighted version of $(1 - \mathrm{Cort}_\pi)$ dissimilarity between two time series ts_r and ts_i is formulated as follows:

$$\mathrm{Cort}_\pi(ts_r^w, ts_i) = \frac{\sum_{l=1}^{s} w_{\pi_l} (xy)_{\pi_l}}{\sqrt{\sum_{l=1}^{s} w_{\pi_l} x_t^2} \sqrt{\sum_{l=1}^{s} w_{\pi_l} y_t^2}}, \tag{6.21}$$

In its weighted variant, we use the optimal path π^* between non weighted time series, and weight each local correlation through the path by w_{π_l}. This is done for the consistency, since we computed a weight vector by using distance set derived from the optimal alignment of $1 - \mathrm{Cort}_\pi$.

6.4 Empirical study

In this section we perform experiments where we employ weighted time series in Multi-operator Temporal Decision Trees (MTDT). We selected hyperplane ($\mathcal{F}(H)$) and hypersphere ($\mathcal{F}(S)$) split operators, since for each of them, weighted split candidates

can be generated and assessed. We enlarge the training set $\mathbf{TS}$ by a set of weighted time series $\mathbf{TS}^w$, which were obtained in the root of a tree. Given a node g, the algorithm selects the best split candidate either with uniform weight vector ($f_{(\cdot)} \in \mathbf{TS}$) or with a learned weight vector ($f_{(\cdot)} \in \mathbf{TS}^w$), which identifies the discriminative time series timestamps. The latter points out that a weighted split candidate is capable of improving the data partition.

Since the right decision depends on the class structure of the dataset, which in turn depends on the node position in the tree, the algorithm dynamically choose the best option. It is done by assessment candidates generated from $\mathbf{TS} \cup \mathbf{TS}^w$ and selecting one with the maximum Gini impurity gain.

As an alternative, input set $\mathbf{TS}_g$ of each node g can be enriched by $\mathbf{TS}_g^w$. However, the farther we go down from the root, the smaller size $\mathbf{TS}_g$ is. Therefore, the less and less number of time series will be used to compute weights. As the consequence, uninformative weight vectors can be obtained. Another negative impact is that the risk of yielding overfitted models increases.

In this empirical study we do not employ Local Search Multi-operator Temporal Decision Tree. Due to the fact that application of Local Search algorithm to the weighted version of MTDT is not straightforward. The problem is how to update weights, once we move a weighted time series to its neighbour one.

We perform set of experiments to study impact of weighting approach on classification results. At first, we check out our approach on the synthetic dataset, the goal is to show the distribution of weights along time series and empirically compare different approaches. Then, we apply each weighting approach to obtain $\mathbf{TS}^w$ set of weighted time series for UCR datasets and we build a model by using Weighted Multi-operator Temporal Decision Tree (WMTDT) with extended input set $\mathbf{TS} \cup \mathbf{TS}^w$. We employ only two split operators $\mathcal{F}(H) + \mathcal{F}(S)$ within WMTDT, since pattern-based split operator does not employ weighting. However, in general case all three split operators can be used.

For all experiments, we fix $l_2, \mathrm{DTW}, (1 - \mathrm{Cort}_\pi)$ dissimilarity measures. We employ each of them to compute a weight vector using proposed approaches, that are *BW, BW-RVO, SVC, SVC-loss, RSVC, MAP*. Each learned weight vector is normalized in order to get $\sum_{t=1}^{m} w_t = 1$ and make them comparable. We fix $\lambda = 0.5$ for *BW* and *BW-RVO* to perform experiments. As regularization parameters for *SVC* and *RSVC* we fixed the loss function to be l_2 and C to be in the range $(0.1, 1, 100)$. During weighting process the best parameter C is defined through cross-validation. The

SVC-loss approach is the variant of *SVC* that takes the loss function as the parameter (l_1, l_2) allowing its tuning through cross-validation.

6.4.1 Synthetic dataset

Local discrimination dataset: *Local-Disc*. *Local-Disc* is the synthetic dataset which we use for the purpose to study weights behaviour of each methods described above Chouakria and Amblard [16]. This dataset contains three class labels *Begin, Middle*, and *End*, each reflects the local discriminative patterns. Time series of *Begin* class labels are characterized by a small bell at the beggining of time series; the *End* class label time series has a small bell at the end of time series; and time series of *Middle* class label are similar in global behavior identified by a centered large bell. Classes of this dataset can be more easily separated by capturing local discriminative attributes rather then global. Since the objective of weighting is to highlight such local patterns in a distance computation, their corresponding timestamps should have high weight values. Figure 6.2 presents time series of the dataset, each plot represents one of class labels.

In order to verify how weights contribute to capturing discriminative sub-intervals in time series and then, how they improve classification performance we run $1NN$ classification algorithm with time series weighted by all previously presented methods. In addition, we carried out the experiment by assigning random weights to time series (*RND*). Random weights are sampled from Gaussian distribution ($\mu = 0.8, \sigma = 0.01$) and assigned randomly to 5% of time series timestamps. If even with small proportion of timestamps with randomly assigned weights will negatively affect on classification accuracy, it means that similarity measures are sensitive to weighting. Results are presented in the Table 6.1. Comparing results across distance measures we can notice that the similarity measure $(1 - \mathrm{Cort}_\pi)$ is tailored to this dataset. Surely, looking at original dataset depicted on the Figure 6.2 one can see that, initially, time series are not aligned. Hence, it needs a dissimilarity measure that is capable of dealing with it, i.e., such as DTW or $(1 - \mathrm{Cort}_\pi)$.

Since for this dataset the class discriminativeness is characterized by shape attributes, the $(1 - \mathrm{Cort}_\pi)$ is the most appropriate dissimilarity measure. Without application of weights (marked as *Uniform* in the resulting table), the 97% of classification accuracy is obtained.

One can notice that dissimilarity measures are sensitive to weights assignment. By random weights assignment (*RND* in the resulting table) performance drops significantly for all types of distances. Comparing different approaches, we can remark

that *SVC* and *RSVC* provide more consistent results. It is explainable, since this approach does not depends on the assumption about underlying distribution over class labels. Weights obtained for each presented method and for each employed similarity measure are depicted across time series of each class and shown on Figures 6.3-6.8. Results of *MAP* approach is nearly uniformly distributed as we can see on Figure 6.8. It explains obtained results (Table 6.1). Weight vectors obtained by *SVC* and *RSVC* approaches are more tailored to the data and emphasize underlying class discriminative patterns. And, as we expected, *RSVC* has more smoothed weights across time series. However, it is difficult to say at this stage which of two version is better to use.

In this part of experiment, we saw that learned weights are able to identify local patterns in time series. Together with an appropriate dissimilarity it helps to increase the classification accuracy.

In the second part of these experiments we will build TSC tree model by Multi-operator Temporal Decision Tree, where we include weighted version of time series input set. We will apply each of the proposed approaches to compute time series weight vectors in order to verify which of them has positive impact on interpretability and accuracy of yielded models.

Table 6.1: Classification accuracy results of $1NN$ algorithm on *Local-Disc* dataset.

	l_2	DTW	$(1 - \text{Cort}_\pi)$
Uniform	54.5	77.7	97.0
BW	**58.6**	68.7	79.8
BW-RVO	52.5	76.8	94.9
SVC	50.5	83.8	**98.0**
SVC-loss	50.5	77.7	81.8
RSVC	52.5	**86.9**	**98.0**
MAP	51.5	77.7	94.9
RND	44.4	30.3	81.8

6.4.2 Weighted Multi-operator Temporal Decision Trees.

In the second part of the experiments we applied Multi-operator Temporal Decision Tree on extended training set $\mathbf{TS} \cup \mathbf{TS}^w$. Each time series in input set is represented by its original values and the first order derivative. In this way, the value based distances can capture shape time series attributes as well.

We admit that wide extension of the input set can lead to increase the chances of having split candidates with the same separation ability. Hence, it becomes important

to set up preference on the choice of the best split among splits provided equal quality of data partition. Due to this reason, we prefer to set up preferences for each input parameter such as split operator, distance, weight, and variable. Our reasoning is exclusively based on keeping the highest level of interpretability of yielded trees. We prioritize hyperplane over hypersphere split operator because it provides the split in a node, which is more comprehensible for a user.

Dissimilarity measures are arranged by increasing level of complexity. Though, the algorithm assesses all generated split candidates, the measures with less computational cost are prioritized in order to have classification results faster. Original time series that input set includes are favored over its weighted version or its derivative. If there are two split candidates, one comprises a weighted time series and another contains an original time series, that induce the same node partition, the latter is kept to have the simplest form for visualization. Furthermore, if there is no difference in the partition, a weight vector does not provide interesting information to a user.

We set up the following priorities ($a \succ b$ means that a favoured over b):

- **Split operator:** $\mathcal{F}(H) \succ \mathcal{F}(S)$
- **Distance measures:** $l_1 \succ l_2 \succ \text{DTW} \succ (1 - \text{Cort}_\pi)$
- **Weighting:** $ts \succ ts^w$
- **Variable:** $ts \succ ts^{der}$

Tables 6.2 and 6.3 present obtained averaged results on 46 resampled UCR datasets. Each experiment is carried on 5 bootstrap cross-validation. Averaged results are shown in terms of the number of non-terminal nodes and classification accuracy. Bold numbers indicate the best results. Maximum reduction of tree size is achieved by employing *BW-RVO* and *SVC-loss*. Total averaged number of nodes are 556.6 and 543.9 respectively. Looking at the classification results for each dataset (Table 6.3), we observe that models of *Uniform* MTDT shows the best results only for 9 datasets. *SVC-loss* reaches the best results for 13 datasets, and for 10 out of 46 results are good as results obtained by using other weighting approaches. The *SVC* and RSVC did not prove to be competitive, as we can see in the resulting table. Both approaches employ only l_2 regularization, which, clearly, is not sufficient in general. Including l_1 regularization allows to obtain more sparse weights that can emphasize better discriminative sub-interval. Lack of prior knowledge, which regularizer to use, suggests that defining it as a parameter is the good practice.

Inclusion of weighted time series in the training set positively affects the model readability by trees size reduction. In addition, when weighted time series split the data in a node, a weight vector visual rendering improves the interpretability of the model. However, we did not obtain impact on the model classification accuracy as we expected. Weighted MTDT preserves the same level of accuracy as uniform MTDT. The reason of obtaining such results can be that the diversity of split operators and dissimilarity measures, we feed to the algorithm, is enough to capture class discriminative information in the data. Hence, adding weighted time series do not bring much extra information in order to be able to improve classification results. It lead us to the open question do we have an abundant set of inputs and if the answer is yes, how to select no redundant subset among them. Further, we propose it as one of the future research direction.

Despite the fact, that classification accuracy did not change much, we have got improvement in model readability and, hence, interpretability too. For example, Figure 6.9 shows the decision tree model obtained on uniform version of the training set. We obtained the tree with six internal nodes (including the root) and 7 leaves. Time series of class label $y = 2$ falls into three leaves. And class labels $y = 0$, $y = 1$ can be assigned to a time series by two different ways of tree's traversing. Figure 6.10 reveals the model obtained on enarged training set by weighted time series. Weight vectors are computed by SVC approach. As we can see, the learned model comprises only two internal nodes. Visualization of a weight vector for split time series in a node can navigate a domain expert towards class distinguished subintervals. Therefore, it facilitates understanding of classification process.

6.5 Conclusion

In this section, we introduced three different methods *BW, MAP*, and *SVC* to find a weight vector for time series. Two of them *BW, MAP* are based on the statistical assumption that the class conditional distribution of distance values is Gaussian. *SVC* do not impose any assumption about distribution law and allows to learn the entire vector at once. However, from the computational point of view this approach has higher cost than others.

We made the comparison of weight vectors distribution obtained from different approaches and dissimilarity measures. Weight values are highly dependent on used dissimilarity measure for their computation. As we saw during experimental results, if dissimilarity function is not tailored to the data, weights can be ill-positioned. Oftentime, we do not know a priori which measure will be appropriate for a particular dataset. As the way to tackle with this problem, we suggest using the range of them. By using each of them, we learn weight vectors for time series in the training set. Alternative solution can be to learn a similarity measure before building a tree. This lies beyond this research, and we consider it as one of the axis for future research.

Once weight vectors are learnt, we add them to input set and build Weight Multi-operator Temporal Decision Tree (WMTDT). During experiments we saw that models yielded by WMTDT are more readable, i.e., contains less number of internal nodes. However, we cannot claim the improvement of classification accuracy. By amplifying the training set, the search space of possible split candidates becomes larger too. Hence, the risk to have the set of split candidates in a node with the same separation ability increases. Though, inducing the same data partition, favouring one to another will affect both yielded tree model and classification results. Thus, the question remaining for future investigation is how one can estimate this possible effect and then, how to choose no abundant set of split options.

Empirical evidence that weighting is able to improve model readability and interpretability. The latter one is achieved by providing a user with a model where visualization of a split time series accompanied by its weight vector. If weights are not uniform, rendering them in visualization mode can contribute to a user understanding of classification process and results.

The next chapter contains conclusion on this work and describe possible future directions of research.

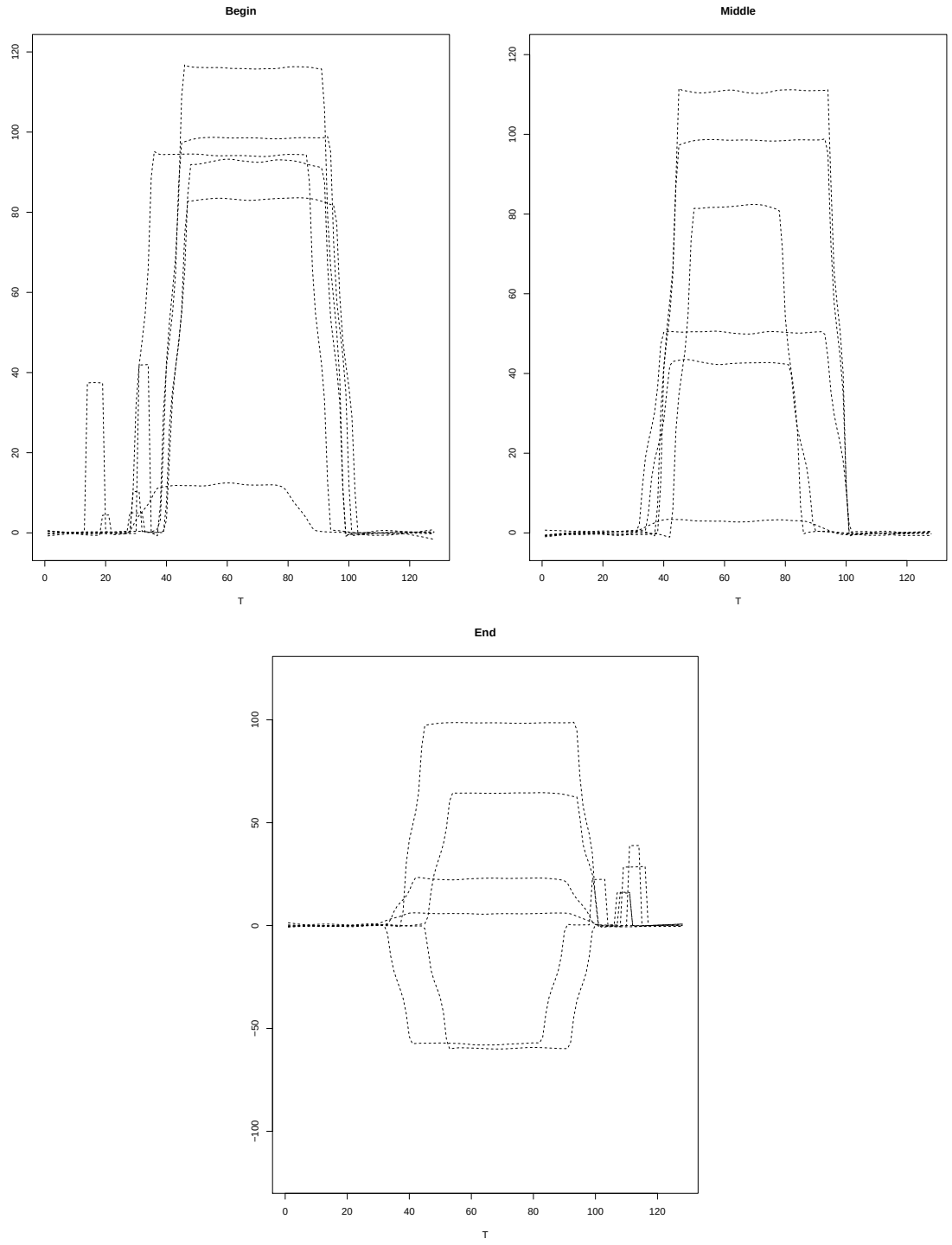

Figure 6.2: Time series of dataset *Local-Disc* divided by class labels.

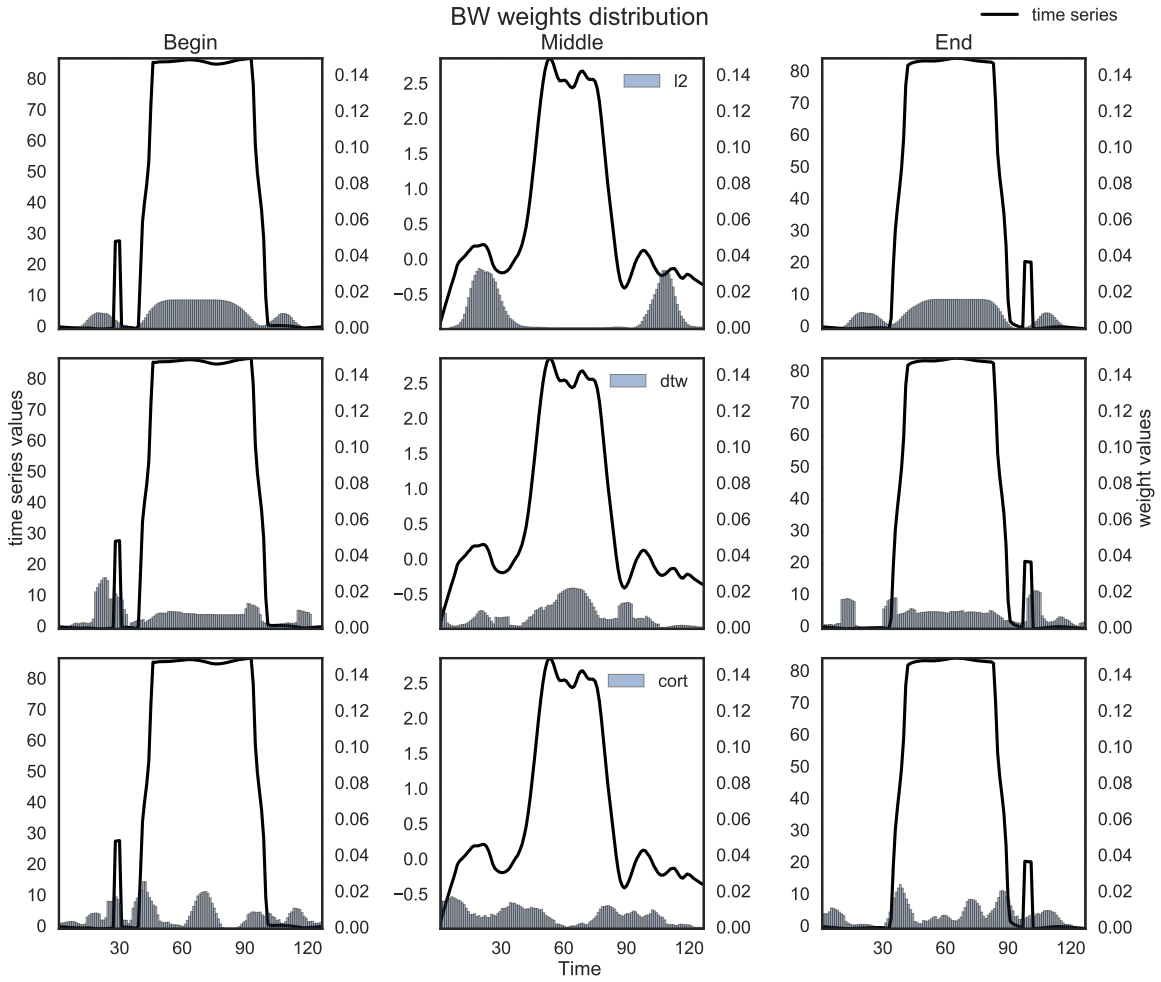

Figure 6.3: Weights vectors computed via *BW* approach for *Local-Disc* dataset. Each line of the figure corresponds to employed dissimilarity measure ($l_2, \mathrm{DTW}, (1 - \mathrm{Cort}_\pi)$) and contains three plots, each of which render a time series and a weight vector of class labels *Begin*, *Middle* and *End* respectively.

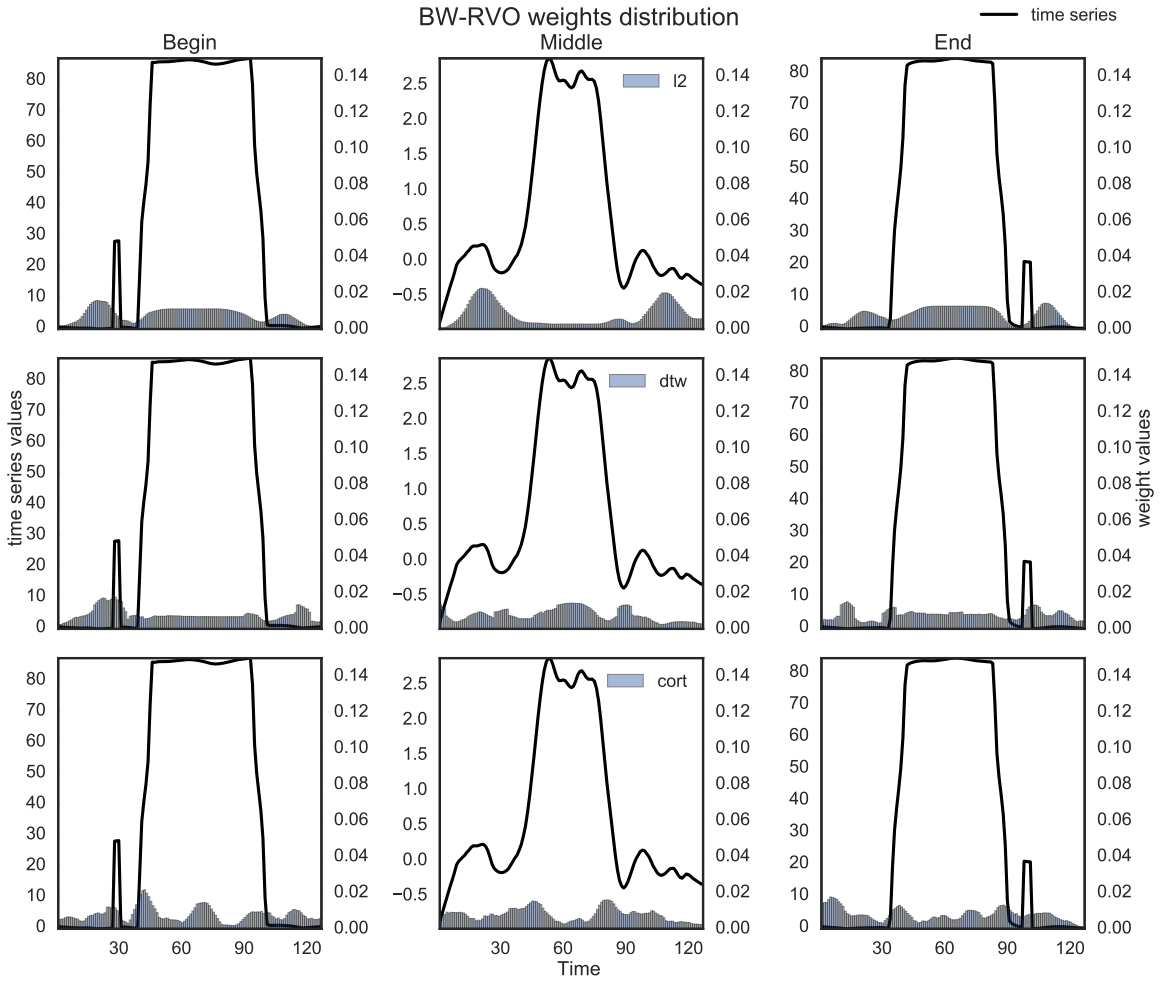

Figure 6.4: Weights vectors computed via *BW-RVO* approach for *Local-Disc* dataset. Each line of the figure corresponds to employed dissimilarity measure ($l_2, \mathrm{DTW}, (1 - \mathrm{Cort}_\pi)$) and contains three plots, each of which render a time series and a weight vector of class labels *Begin*, *Middle* and *End* respectively.

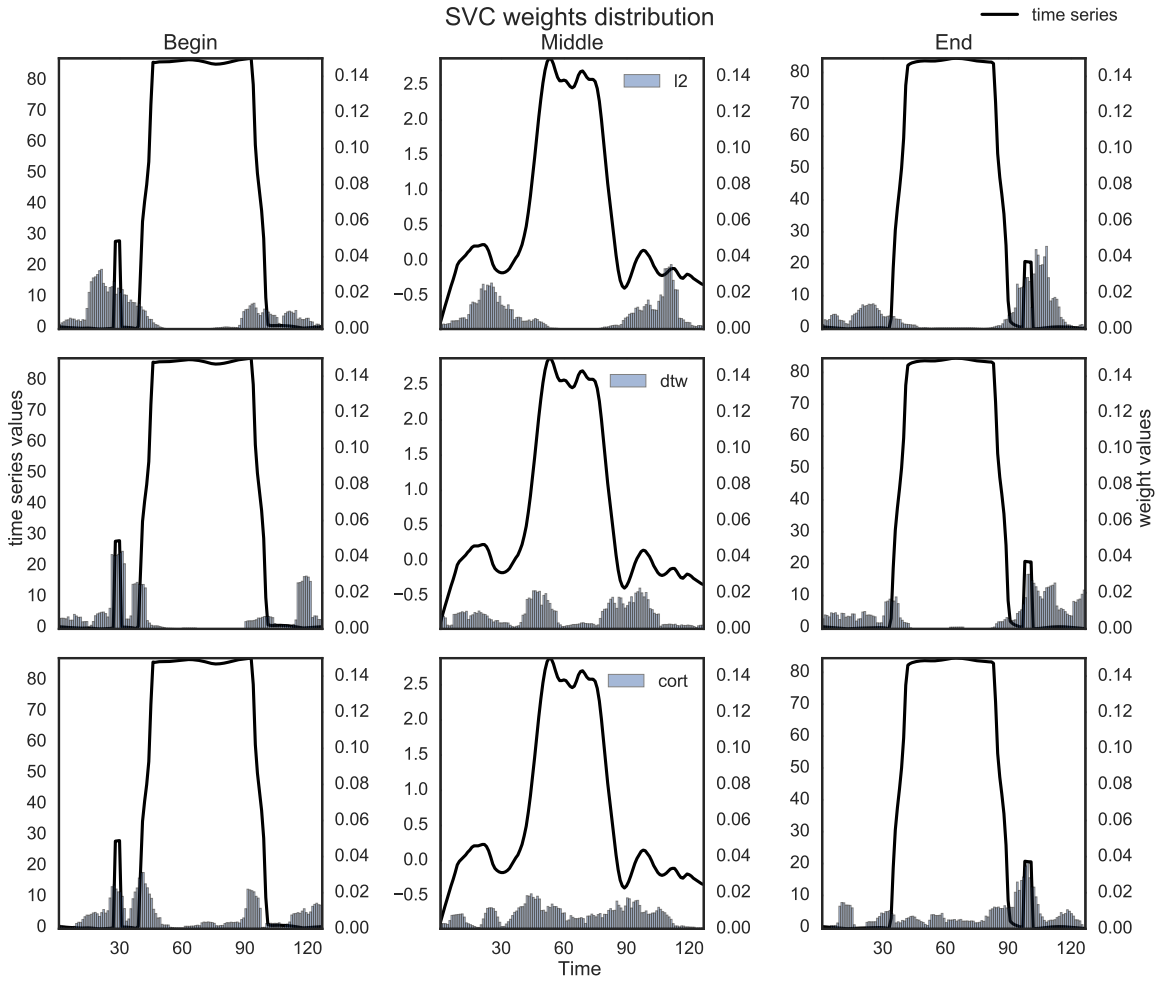

Figure 6.5: Weights vectors computed via *SVC* approach for *Local-Disc* dataset. Each line of the figure corresponds to employed dissimilarity measure ($l_2, \mathrm{DTW}, (1 - \mathrm{Cort}_\pi)$) and contains three plots, each of which render a time series and a weight vector of class labels *Begin*, *Middle* and *End* respectively.

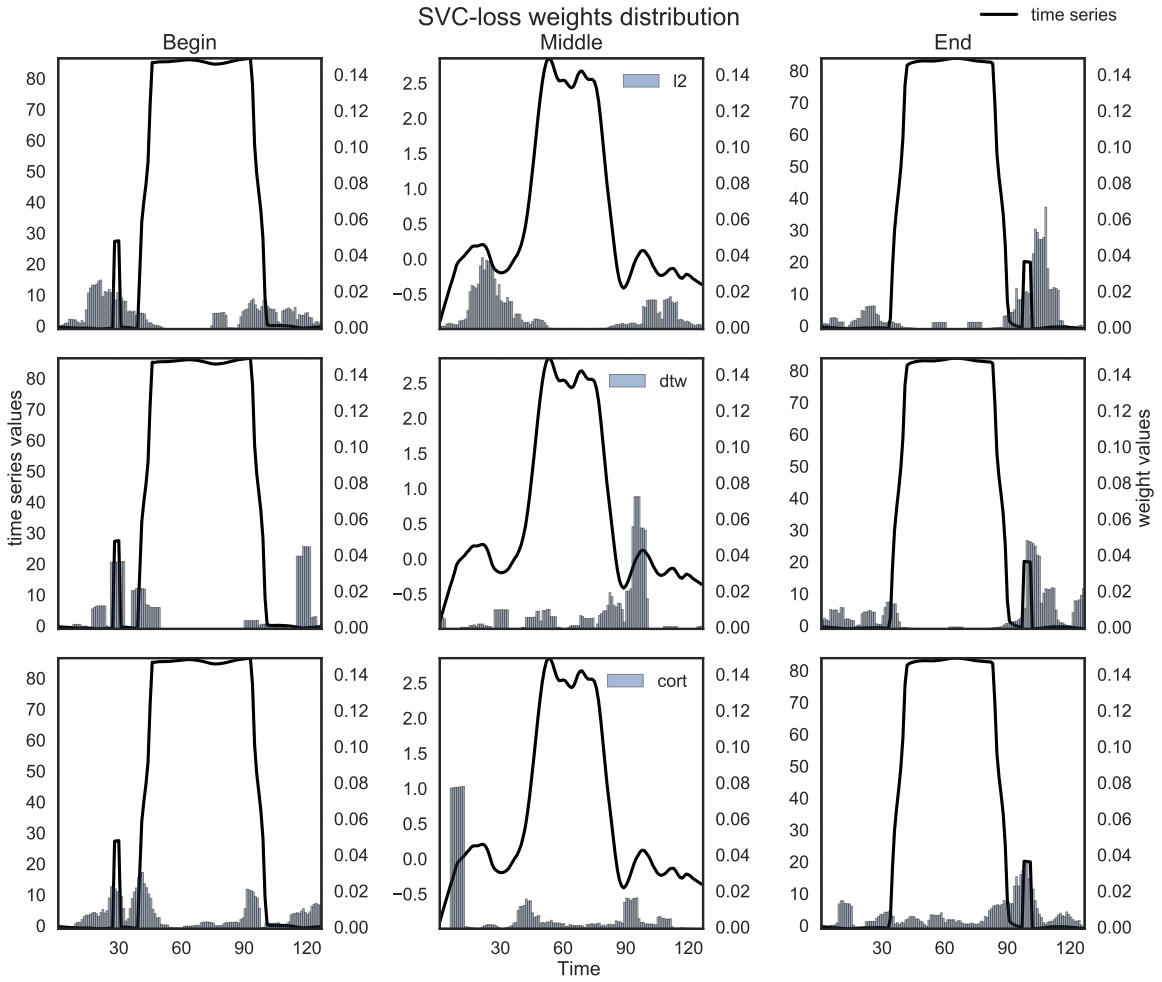

Figure 6.6: Weights vectors computed via *SVC-loss* approach for *Local-Disc* dataset. Each line of the figure corresponds to employed dissimilarity measure ($l_2, \mathrm{DTW}, (1 - \mathrm{Cort}_\pi)$) and contains three plots, each of which render a time series and a weight vector of class labels *Begin*, *Middle* and *End* respectively.

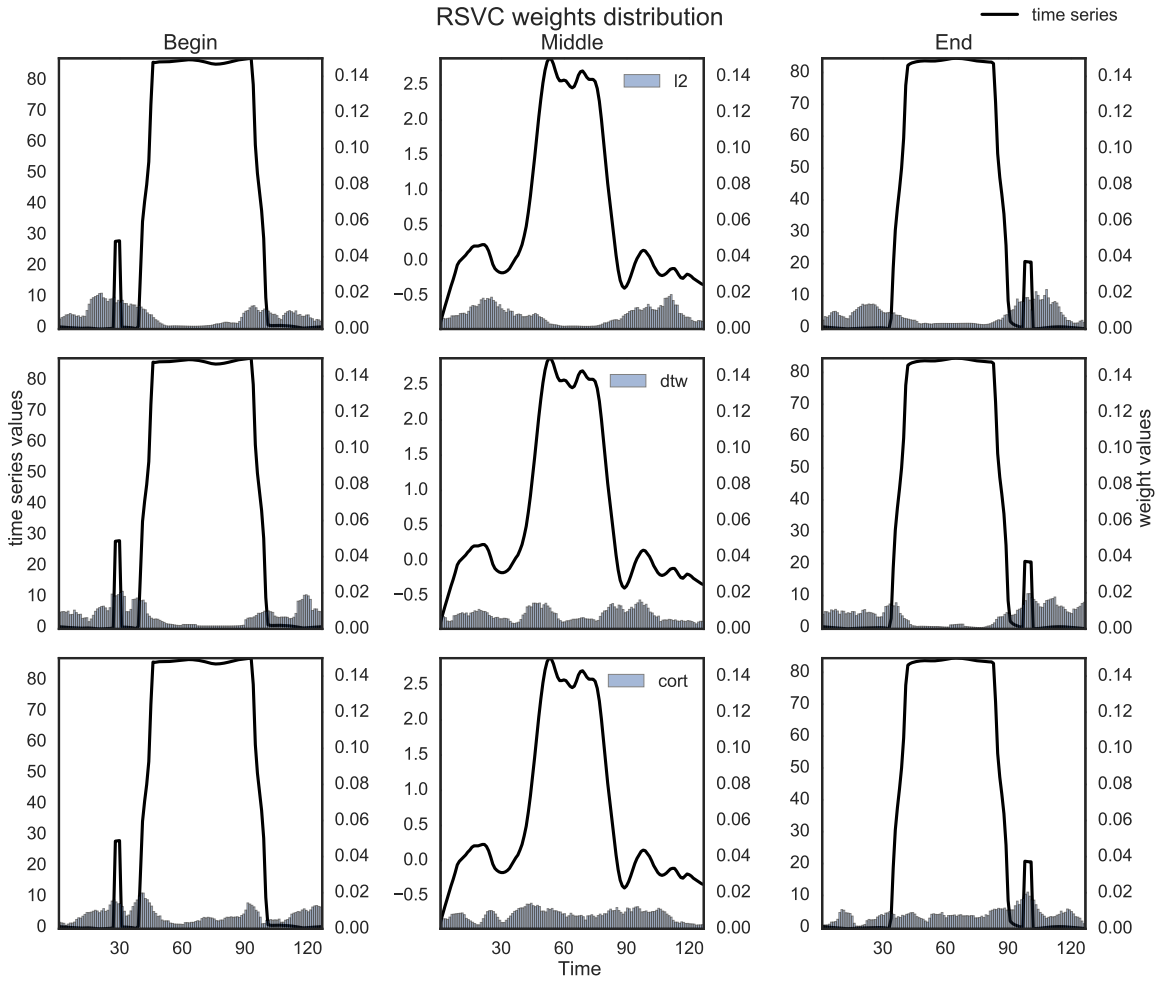

Figure 6.7: Weights vectors computed via *RSVC* approach for *Local-Disc* dataset. Each line of the figure corresponds to employed dissimilarity measure ($l_2, \mathrm{DTW}, (1 - \mathrm{Cort}_\pi)$) and contains three plots, each of which render a time series and a weight vector of class labels *Begin*, *Middle* and *End* respectively.

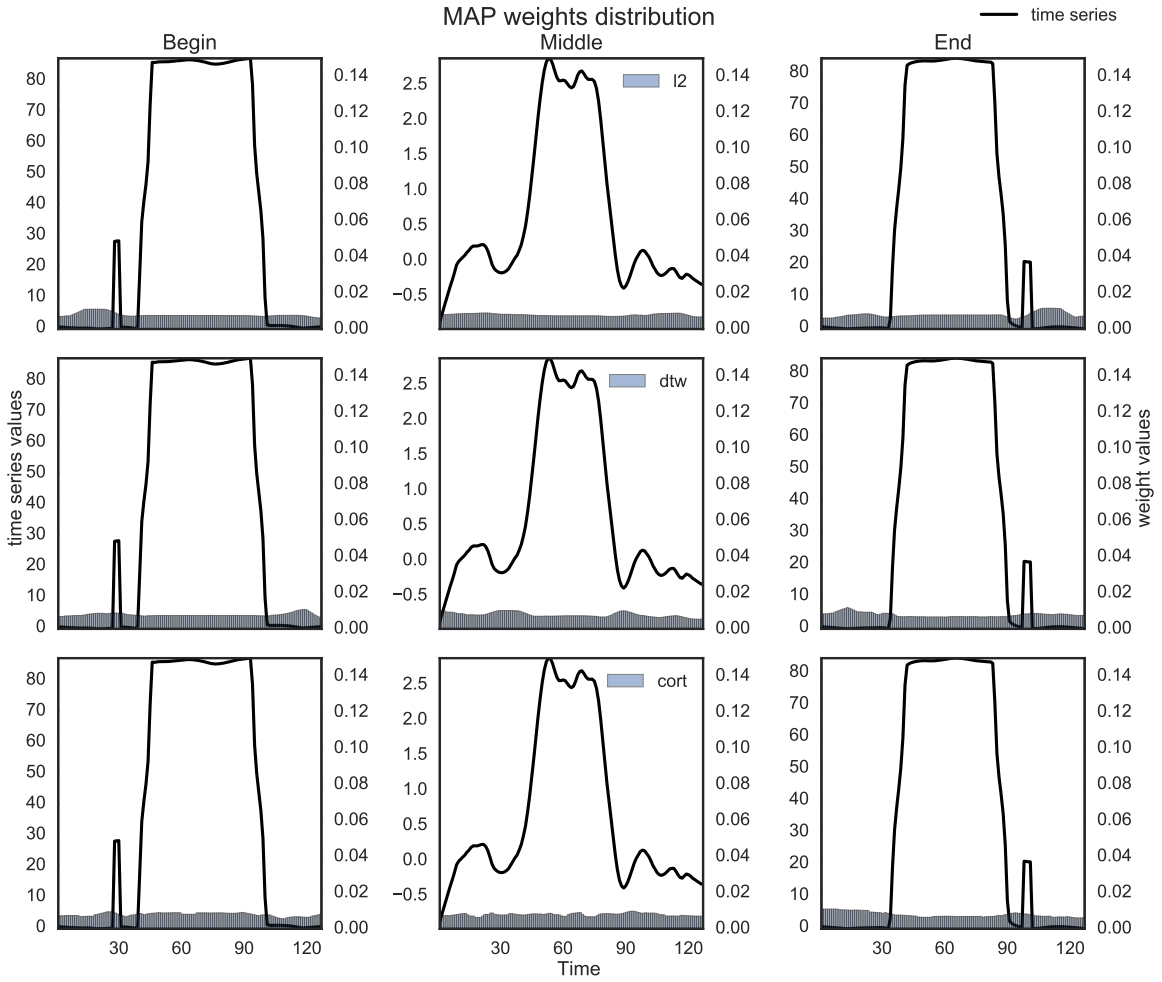

Figure 6.8: Weights vectors computed via *MAP* approach for *Local-Disc* dataset. Each line of the figure corresponds to employed dissimilarity measure (l_2, DTW, $(1 - \mathrm{Cort}_\pi)$) and contains three plots, each of which render a time series and a weight vector of class labels *Begin*, *Middle* and *End* respectively.

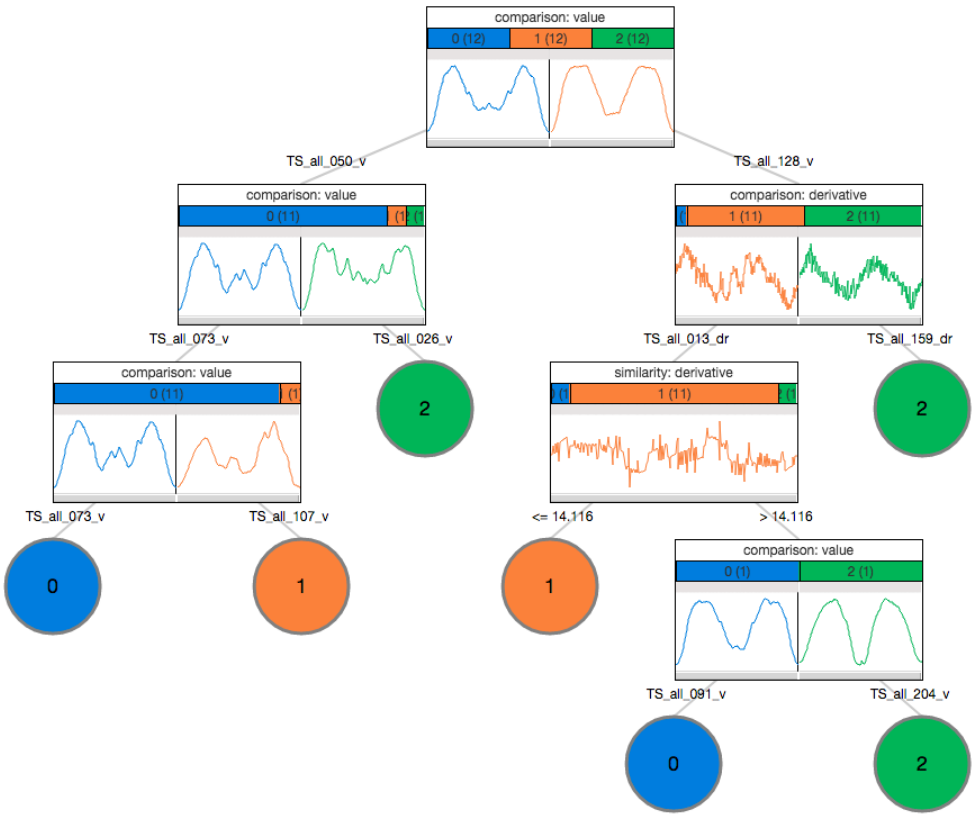

Figure 6.9: Classification decision tree for *ArrowHead* UCR dataset obtained by MTDT only with **TS** training set of non weighted time series set.

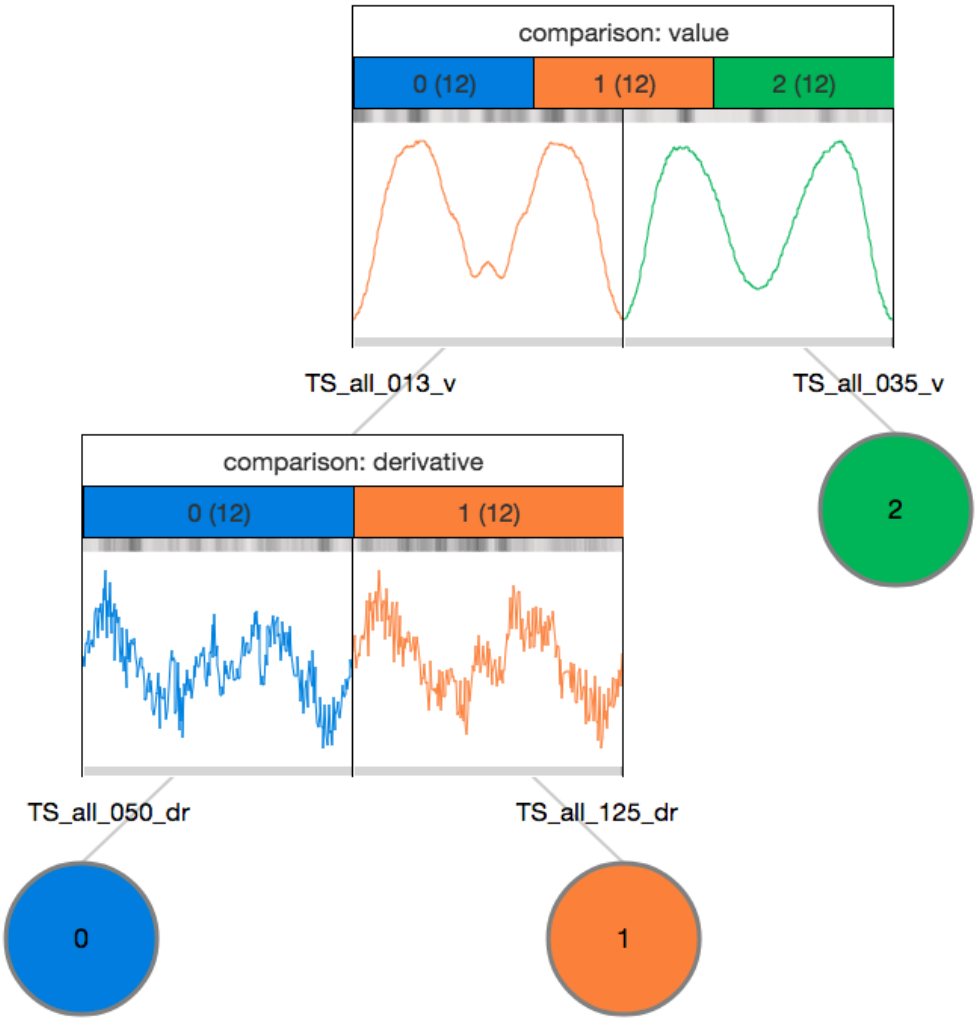

Figure 6.10: Classification decision tree obtained by WMTDT for *ArrowHead* UCR dataset. The *SVC* weighting approach was used.

Table 6.2: The number of non-terminal nodes of trees built on 46 UCR dataset with Weighted Multioperator Temporal Decision Tree (WTDT).

		U	BW	BW − RVO	SVC	SVC − loss	RSVC	MAP
Total		611.2	564.0	556.6	562.6	543.9	577.5	598.3
Average		13.29	12.26	12.10	12.23	11.82	12.55	13.01
1	Adiac	73.6± 5.8	72.6± 3.6	67.8± 2.3	**64.0**± 4.1	68.0± 3.7	70.0± 2.9	71.3± 5.3
2	ArrowHead	4.6± 0.9	4.2± 0.8	3.2± 0.4	**2.8**± 0.8	**2.8**± 0.8	4.2± 0.4	4.0± 0.7
3	Beef	7.2± 1.5	6.4± 1.1	6.6± 0.9	6.0± 1.2	**5.6**± 0.9	6.2± 0.8	6.6± 0.9
4	BeetleFly	1.6± 0.5	**1.0**± 0.0	1.6± 0.5	1.4± 0.5	1.2± 0.4	1.8± 0.4	1.8± 0.4
5	BirdChicken	1.2± 0.4	1.0± 0.0	1.2± 0.4	**1.0**± 0.0	**1.0**± 0.0	1.2± 0.4	1.4± 0.5
6	Car	7.0± 0.7	5.8± 1.1	5.0± 0.7	**4.8**± 0.8	6.6± 0.9	6.6± 0.5	6.4± 1.1
7	CBF	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0
8	CincECGtor	4.6± 0.5	**3.0**± 0.0	3.4± 0.5	4.2± 0.8	3.2± 0.4	4.8± 0.8	4.5± 0.7
9	DistPhOutAge	13.2± 2.4	**11.0**± 2.5	**11.0**± 2.2	11.4± 2.3	11.8± 1.6	11.8± 2.8	13.4± 1.1
10	DistPhOutCor	20.0± 3.2	**18.2**± 1.6	19.2± 2.9	19.0± 2.3	19.0± 2.1	19.0± 1.9	21.0± 2.7
11	DistPhTW	20.8± 1.9	19.6± 3.2	**17.4**± 1.8	20.4± 1.8	19.4± 3.0	20.4± 2.1	20.4± 1.9
12	Earthquakes	13.4± 1.8	**10.8**± 1.3	12.2± 1.1	11.4± 1.1	12.0± 1.2	11.8± 0.8	12.7± 1.6
13	ECG200	6.6± 1.3	6.6± 0.9	7.0± 1.6	6.0± 1.4	**5.4**± 0.9	6.2± 0.8	6.0± 1.6
14	ECG5000	16.2± 2.2	16.0± 1.9	15.4± 1.1	13.8± 1.6	**11.8**± 1.9	15.0± 1.9	15.8± 1.8
15	ECGFiveDs	2.2± 0.4	**1.4**± 0.5	2.0± 0.7	2.0± 0.7	2.2± 0.4	2.0± 0.0	2.2± 1.1
16	FaceFour	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0
17	FacesUCR	25.2± 2.2	23.6± 1.9	**22.2**± 1.3	25.4± 3.2	23.2± 1.1	25.0± 2.4	24.7± 2.3
18	GunPoint	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0
19	Ham	10.4± 1.5	9.2± 1.1	11.0± 0.7	9.8± 0.4	**9.0**± 0.7	10.4± 1.3	10.0± 1.4
20	Herring	7.6± 1.1	6.6± 0.5	6.8± 1.3	7.8± 1.5	**6.0**± 0.7	7.2± 1.1	7.4± 1.3
21	InsectWngS	45.4± 4.7	40.4± 1.8	39.4± 1.1	40.8± 3.3	**39.0**± 3.2	41.8± 3.8	47.4± 2.8
22	ItalyPowDem	1.2± 0.4	1.4± 0.5	1.6± 0.5	1.6± 0.5	**1.2**± 0.4	1.6± 0.5	2.0± 0.7
23	Lighting2	5.2± 0.8	**4.6**± 0.5	5.8± 0.8	**4.6**± 0.5	5.4± 0.5	5.4± 0.9	5.0± 0.7
24	Lighting7	10.6± 0.9	10.2± 1.3	9.0± 1.6	10.2± 1.5	10.2± 0.8	10.0± 1.2	**8.8**± 0.8
25	MALLAT	**7.0**± 0.0	**7.0**± 0.0	**7.0**± 0.0	7.4± 0.5	7.2± 0.4	7.2± 0.4	7.2± 0.4
26	Meat	2.2± 0.4	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0	**2.0**± 0.0
27	MedicalImag	57.8± 5.4	54.6± 2.5	**51.2**± 1.9	55.0± 3.7	51.6± 2.3	53.2± 4.8	53.6± 3.5
28	MidPhAge	21.4± 5.2	20.2± 3.1	19.2± 5.8	20.0± 2.8	**18.2**± 1.9	21.0± 2.4	22.2± 4.1
29	MidPhCor	22.0± 1.4	**20.0**± 2.2	21.0± 1.6	23.6± 2.7	20.8± 2.9	22.2± 4.0	23.4± 2.1
30	MiddPhTW	30.6± 3.4	29.2± 1.5	29.2± 1.9	31.2± 1.3	**28.4**± 1.5	29.4± 2.3	33.2± 1.3
31	MoteStrain	1.4± 0.5	**1.0**± 0.0	1.4± 0.5	1.2± 0.4	1.2± 0.4	**1.0**± 0.0	1.4± 0.5
32	OliveOil	3.8± 0.8	**3.0**± 0.0	**3.0**± 0.0	3.6± 0.9	3.4± 0.5	3.4± 0.5	3.2± 0.4
33	ProxPhAge	26.0± 3.5	25.2± 2.2	27.2± 4.3	25.2± 0.4	**24.2**± 1.9	25.6± 1.5	25.5± 2.3
34	ProxPhCor	36.6± 1.8	33.6± 1.1	33.8± 2.5	**29.0**± 1.9	29.2± 1.6	30.6± 2.9	33.6± 3.2
35	ProxPhTW	20.8± 1.9	18.6± 1.5	**17.4**± 1.1	19.6± 1.5	18.8± 1.9	19.0± 2.2	18.6± 0.9
36	ShapeletSim	1.6± 0.5	1.6± 0.5	1.6± 0.5	1.6± 0.5	**1.4**± 0.5	**1.4**± 0.5	1.6± 0.5
37	SonyAIBO1	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0
38	SwedishLeaf	30.4± 4.3	**27.8**± 1.5	**27.8**± 1.3	28.0± 1.9	29.4± 2.7	31.0± 2.3	30.0± 2.4
39	SynControl	5.2± 0.4	5.6± 0.5	5.6± 0.5	**5.2**± 0.4	**5.2**± 0.4	**5.2**± 0.4	**5.2**± 0.4
40	ToeSegm1	4.0± 0.7	2.8± 0.8	2.6± 0.9	3.0± 1.2	**2.3**± 1.5	3.3± 1.2	3.4± 1.5
41	ToeSegm2	3.2± 0.8	2.0± 0.0	2.0± 0.0	2.2± 0.4	**1.8**± 0.4	2.4± 0.5	2.2± 0.4
42	Trace	3.4± 0.5	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0	**3.0**± 0.0
43	TwoLdECG	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0	**1.0**± 0.0
44	Wine	6.2± 1.3	**5.0**± 1.4	6.2± 1.9	5.6± 1.5	5.4± 0.9	5.2± 0.4	6.0± 1.2
45	Worms	14.2± 2.8	12.8± 2.6	12.6± 2.3	14.0± 1.9	**11.4**± 0.5	13.6± 1.7	13.8± 2.6
46	Worms2Class	7.6± 0.9	7.4± 1.5	**6.0**± 0.7	6.2± 1.3	6.8± 1.0	7.4± 0.9	8.4± 0.5
	w/d/l	**0/5/41**	**6/11/29**	**5/10/31**	**3/10/33**	**13/10/23**	**0/7/39**	**1/6/39**

Table 6.3: Accuracy of trees built on 46 UCR dataset with Weighted Multioperator Temporal Decision Tree (WTDT).

		U	BW	BW − RVO	SVC	SVC − loss	RSVC	MAP
Average		0.759	0.753	0.757	0.755	0.756	0.754	0.756
1	Adiac	0.600± 0.02	0.615± 0.03	**0.627**± 0.04	**0.627**± 0.04	0.611± 0.03	0.612± 0.01	0.606± 0.02
2	ArrowHead	0.728± 0.06	0.723± 0.08	0.693± 0.08	**0.752**± 0.06	0.714± 0.05	0.690± 0.05	0.703± 0.04
3	Beef	0.560± 0.12	0.640± 0.09	0.547± 0.07	0.573± 0.10	0.613± 0.13	**0.647**± 0.07	0.620± 0.04
4	BeetleFly	0.730± 0.07	0.730± 0.04	**0.780**± 0.04	**0.780**± 0.08	0.720± 0.11	0.660± 0.16	0.680± 0.14
5	BirdChicken	0.900± 0.08	0.860± 0.10	0.840± 0.24	0.920± 0.07	**0.970**± 0.04	0.860± 0.11	0.910± 0.12
6	Car	0.693± 0.03	0.650± 0.03	**0.797**± 0.03	0.750± 0.02	0.767± 0.03	0.743± 0.05	0.693± 0.06
7	CBF	**0.945**± 0.01	0.883± 0.09	0.900± 0.06	0.912± 0.04	0.908± 0.04	0.900± 0.07	0.918± 0.04
8	CincECGtor	0.758± 0.07	**0.837**± 0.04	0.806± 0.06	0.744± 0.10	0.805± 0.06	0.695± 0.05	0.717± 0.08
9	DistPhOutAge	**0.795**± 0.02	0.771± 0.02	0.781± 0.02	0.784± 0.02	0.771± 0.03	0.785± 0.02	0.788± 0.03
10	DistPhOutCor	0.766± 0.03	0.777± 0.02	**0.787**± 0.02	0.759± 0.02	0.770± 0.01	0.766± 0.02	0.764± 0.02
11	DistPhTW	0.750± 0.02	0.735± 0.02	0.715± 0.02	0.705± 0.03	0.715± 0.02	0.717± 0.01	0.728± 0.01
12	Earthquakes	0.698± 0.03	0.709± 0.05	0.670± 0.02	0.706± 0.03	0.698± 0.03	0.686± 0.04	**0.712**± 0.02
13	ECG200	0.780± 0.04	0.802± 0.03	0.764± 0.02	**0.812**± 0.05	0.782± 0.04	0.808± 0.03	0.808± 0.02
14	ECG5000	**0.923**± 0.00	0.918± 0.01	0.918± 0.00	0.919± 0.01	0.918± 0.01	0.918± 0.01	0.920± 0.01
15	ECGFiveDs	0.826± 0.06	0.774± 0.03	0.762± 0.10	0.800± 0.05	0.792± 0.08	0.803± 0.05	**0.853**± 0.07
16	FaceFour	0.732± 0.08	0.757± 0.06	0.800± 0.05	0.773± 0.12	0.773± 0.04	0.775± 0.09	**0.823**± 0.07
17	FacesUCR	**0.757**± 0.03	0.747± 0.01	0.745± 0.02	0.738± 0.01	0.742± 0.03	0.747± 0.02	0.747± 0.02
18	GunPoint	0.977± 0.01	**0.981**± 0.01	0.965± 0.02	0.956± 0.07	0.965± 0.03	0.977± 0.01	0.972± 0.01
19	Ham	0.699± 0.03	0.714± 0.04	**0.716**± 0.07	0.644± 0.08	0.678± 0.05	0.669± 0.06	0.697± 0.05
20	Herring	0.622± 0.06	0.591± 0.06	0.600± 0.07	0.581± 0.09	**0.641**± 0.06	0.600± 0.04	0.544± 0.11
21	InsectWngS	0.542± 0.01	0.540± 0.01	0.541± 0.01	0.522± 0.02	0.524± 0.03	**0.550**± 0.02	0.523± 0.02
22	ItalyPowDem	0.957± 0.01	0.954± 0.02	0.952± 0.02	0.953± 0.01	**0.961**± 0.00	0.954± 0.01	0.957± 0.02
23	Lighting2	0.715± 0.06	**0.754**± 0.04	0.718± 0.07	0.738± 0.06	0.744± 0.05	0.721± 0.07	0.711± 0.05
24	Lighting7	**0.641**± 0.06	0.625± 0.05	0.600± 0.05	0.614± 0.07	0.622± 0.05	0.616± 0.05	0.611± 0.08
25	MALLAT	0.922± 0.01	0.910± 0.03	0.923± 0.02	**0.925**± 0.04	0.901± 0.04	0.916± 0.02	0.908± 0.04
26	Meat	0.957± 0.03	0.943± 0.03	0.963± 0.03	0.960± 0.04	0.963± 0.03	0.947± 0.04	**0.967**± 0.03
27	MedicalImag	0.658± 0.04	0.670± 0.02	0.663± 0.01	0.657± 0.03	0.677± 0.04	**0.680**± 0.01	0.657± 0.02
28	MidPhAge	0.678± 0.05	0.690± 0.04	0.700± 0.03	0.683± 0.03	**0.717**± 0.04	0.672± 0.03	0.685± 0.03
29	MidPhCor	0.762± 0.03	0.759± 0.02	**0.768**± 0.01	0.723± 0.06	0.747± 0.03	0.750± 0.01	0.740± 0.02
30	MiddPhTW	0.556± 0.02	0.547± 0.02	**0.566**± 0.03	0.555± 0.01	0.552± 0.02	0.545± 0.02	0.548± 0.04
31	MoteStrain	0.842± 0.02	0.844± 0.04	**0.864**± 0.01	0.849± 0.05	0.840± 0.06	0.849± 0.05	0.846± 0.04
32	OliveOil	0.760± 0.06	0.767± 0.08	0.820± 0.06	0.807± 0.06	0.760± 0.07	0.800± 0.07	**0.840**± 0.07
33	ProxPhAge	0.804± 0.02	0.809± 0.03	0.799± 0.01	0.779± 0.01	**0.810**± 0.03	0.783± 0.02	0.797± 0.02
34	ProxPhCor	**0.840**± 0.02	0.826± 0.04	0.837± 0.01	0.830± 0.02	0.833± 0.03	0.839± 0.02	0.838± 0.02
35	ProxPhTW	0.741± 0.03	**0.774**± 0.01	0.758± 0.01	0.751± 0.02	0.735± 0.03	0.744± 0.03	0.744± 0.04
36	ShapeletSim	0.704± 0.10	0.639± 0.10	0.730± 0.07	0.730± 0.06	0.680± 0.12	**0.750**± 0.05	0.733± 0.06
37	SonyAIBO1	0.817± 0.03	0.780± 0.12	0.806± 0.05	0.808± 0.07	0.735± 0.10	0.786± 0.10	**0.827**± 0.04
38	SwedishLeaf	**0.855**± 0.01	0.848± 0.02	0.852± 0.01	**0.855**± 0.02	0.853± 0.01	**0.855**± 0.01	0.850± 0.01
39	SynControl	0.965± 0.01	0.945± 0.03	0.963± 0.00	0.965± 0.01	0.961± 0.02	0.958± 0.01	**0.970**± 0.01
40	ToeSegm1	0.779± 0.08	0.760± 0.04	0.775± 0.06	0.739± 0.06	**0.789**± 0.04	0.784± 0.03	0.697± 0.06
41	ToeSegm2	0.798± 0.07	0.774± 0.08	0.725± 0.09	0.789± 0.04	0.797± 0.04	**0.840**± 0.01	0.763± 0.12
42	Trace	0.980± 0.01	0.978± 0.02	0.986± 0.01	**0.988**± 0.01	0.972± 0.01	0.974± 0.03	0.986± 0.01
43	TwoLdECG	**0.804**± 0.05	0.748± 0.07	0.763± 0.12	0.796± 0.07	0.767± 0.12	0.785± 0.04	0.789± 0.06
44	Wine	**0.804**± 0.05	0.748± 0.07	0.763± 0.12	0.796± 0.07	0.767± 0.12	0.785± 0.04	0.789± 0.06
45	Worms	0.555± 0.06	**0.556**± 0.05	0.537± 0.04	0.529± 0.04	0.511± 0.04	0.539± 0.05	0.524± 0.04
46	Worms2Class	0.673± 0.03	0.608± 0.08	0.678± 0.03	**0.705**± 0.05	0.662± 0.05	0.687± 0.03	0.669± 0.03
	w/d/l	**8/1/37**	**5/0/41**	**6/2/38**	**5/3/38**	**6/0/40**	**5/1/40**	**7/0/39**

Chapter 7

Publications

1. Vera Shalaeva, Sami Alkhoury, Julien Marinescu, Cécile Amblard, Gilles Bisson. Multi-operator Decision Trees for Explainable Time-Series Classification. IPMU, 2018

2. Vera Shalaeva, Sami Alkhoury, Julien Marinescu, Cécile Amblard, Gilles Bisson. Arbres de décision multi-opérateurs pour la classification efficace et intelligible de séries temporelles. CAp, 2018

3. Demo of IKATS visualization tool, 2018

Chapter 8

Conclusions and Future Directions

8.1 Conclusions

Time series data is represented by ordered sequences of real values. Such data appears in many domains, where an expert may need to analyze this data. In this research, we focus on the problem of time series data classification (TSC). There are lots of Machine Learning (ML) algorithms that are capable of working on time series data. However, there are only a few of them which a domain practitioner without ML knowledge can employ. Moreover, results of many methods designed for TSC are very difficult to interpret, even though they provide high classification performance. Therefore, the ML research community have been motivated to develop algorithms which are capable of providing both accurate and interpretable results.

In this research, we as well explore the problem of interpretable time series classification through the Temporal Decision Trees (TDT). We first investigated the state-of-the-art studies on TSC models where one can see that classification trees are generally recommended when it is important to yield an interptetable model. In addition, the decision tree methods show good predictive performance. For an expert from any domain, a classification model with such characteristics facilitates understanding of the obtained results. At the same time, interpreting the output of these techniques does not require prerequisites of ML knowledge. We aimed to reinforce interpretable properties of the yielded tree models by employing several comprehensible split operators. Readability of a model is another important characteristic that we targeted to improve. In addition, we attemted to keep the same level of classification accuracy.

To boost the capacity of interpretability and readability of the Temporal Decision Trees, we proposed to employ different split operators in order to build a more interpretable classification tree. Assessment of split candidates generated from split

operators of different nature allows capturing different underlying structures in the data. To do that, we presented Multi-operator Temporal Decision Trees which build models of significantly smaller size than Mono-operator Temporal Decision Trees. The performance criteria we measured are the size of generated trees in terms of the number of internal nodes and classification accuracy. We obtained readable classification models which are in average two times smaller than those obtained by Mono-operator TDT. As a result, these models can be easily visualized and a user can efficiently analyze them.

The flip side of the proposed algorithm is the computational complexity to train the model. The partition induced by each generated split candidate has to be estimated in order to get the best possible split at each node of the tree. Hence, by enriching the set of split candidates, the running time to learn the classification model is growing as well. The total complexity of Multi-operator Temporal Decision Tree increases with the respect to the number of split candidates to examine. The complexity with respect to the distance computations does not change, however, it depends on the number of used dissimilarity measures during the learning process. In this work, we chose to employ three dissimilarity measures $(l_2, DTW, (1 - \mathrm{Cort}_\pi))$, which are capable of capturing value and shape time series attributes.

To deal with the training time complexity, we proposed an approximation algorithm, called Local Search Temporal Decision Trees ($LSTDT$), which iteratively explores split candidates starting with a random one and searching in its vicinity for the better split. Using the triangle inequality and bounds on real pairwise distance values allows skipping unnecessary distance computations. Exploiting these properties, the proposed algorithm is capable of yielding similar results in terms of model size and classification accuracy. The volume of search space is defined by a user via input parameters, which control the number of distance computations the algorithm can compute on the training step. With higher values of input parameters, the $LSTDT$ explores more and more split candidates, finally going towards to the complete search space of split candidates. During experimental study, we observed that beyond a certain level of search space the algorithm becomes computationally inneficient and can be simply be switched to the original one.

Last part of the current work investigates the problem of finding local class discriminative subsequences of time series. We suggest resolving this task by finding a weight vector for each time series from the training set. Several techniques were proposed to learn weight vectors, among which two, namely BW and SVC, show more interesting properties. The former has an advantage of fast computation, but assumes

that distance values of each class are drawn from the Gaussian distribution. The latter do not make any statistical assumption on the data; however, is computationally more expensive. We observed positive results on the model readability, by obtaining trees of compact size, keeping the same classification accuracy. Despite this fact, we can conclude that employing weighted time series into the training set is able to improve model's interpretability and contribute to its explainability. On one hand, if a node of yielded tree contains split annotated by a weight vector, it points out existence of local class distinguishing sub-intervals. In this case, providing a user with the visualization of a weight vector will facilitate the data analysis and can help to explain obtained classification results. On the other hand, if a node split comprises time series with uniform weight vector, a practitioner understands that discriminative information is contained into the entire time series. In the next section, we describe some future directions of research that we found to be interesting and useful to explore.

8.2 Future Directions

- Multi-operator Temporal Decision Trees examines a wide range of split candidates generated from split operators of different nature. Two questions can be asked here:

    1. how many split candidates are sufficient to feed to such algorithms to learn class discriminative concepts of the data?
    2. how to select the best split candidate among a set of candidates, where each of them induces the same data partition?

    One possibillity to address the first question could be done by employing risk bounds during the training process. The second one requires consideration some additional criteria to evaluate split partition, the classification risk bounds can be used as well.

- Hyperplane and hypersphere split operators can employ different dissimilarity measures in order to capture value or shape attributes of time series. Since a domain expert rarely has a prior knowledge about which dissimilarity function to use, they either employ many of them (which leads to high computational complexity) or risk to pick the suboptimal one. Hence, embedding metric learning within building a tree node can be an interesting direction of the algorithm improvement.

- We proposed approximated Local Search Temporal Decision Tree algorithm which explores subset of split candidates at each internal node depending on limits, provided by a user, on the number of distance computations. While it is able to yield a model by taking the non-weighted time series data as input, it cannot be applied when time series have weight vectors. The problem appears on the step of exploration neighbour split candidates, when bounds on the real distances have to be updated. Then, the point is how one can update weight values for a new split candidate, which lies in vicinity of the initial one.

- The Multi-operator Temporal Decision Tree algorithm is computationally demanding algorithm. As a consequence, it does not scale when the datasets grow in size both in terms of number of time series and their length. With rapid increase of the former, the number of split candidates to evaluate increases too. When the latter term grows, computation of dissimilarity measure becomes costly. As an axis of future work, parallelization of the training process can be considered.

- Trees models suffer from the high variance. In practice, Random Forest classifier is commonly used to overcome this problem. Tree ensembles also have higher classification accuracy than a single tree. However, the intepretability of such models is very limited. When the number of built models grows, it becomes difficult to visualize them. Also, for a domain expert, it is challenging to analyze and efficiently aggregate classification results obtained from multiple models. Hence, the interest of future research would be decreasing variance of a tree model, and at the same time keeping its classification accuracy.

- In this research, we construct binary classification trees. It is assumed that the data can be efficiently bi-partitioned, which, clearly, is not always true, especially in multi-class classification problem. The data structure might change from a node to node, hence producing a binary split can lead to yielding bigger, less accurate and interpretable trees. It leads to the open question, how to built trees with a multiway split in a node. Thus, we suggest exploration of this problem as the one of prominent future directions in order to improve interpretability of tree models.

Bibliography

[1] . Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, ICDE '99, pages 126–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0071-4. URL `http://dl.acm.org/citation.cfm?id=846218.847201`.

[2] Innovative tool-kit for analysig time series. https://ikats.org, 2019.

[3] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001. ISBN 3-540-41456-8. doi: 10.1007/3-540-44503-X_27. URL `https://doi.org/10.1007/3-540-44503-X_27`.

[4] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, FODO '93, pages 69–84, Berlin, Heidelberg, 1993. Springer-Verlag. ISBN 3-540-57301-1. URL `http://dl.acm.org/citation.cfm?id=645415.652239`.

[5] Anthony Bagnall, Luke M. Davis, Jon Hills, and Jason Lines. Transformation based ensembles for time series classification. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012.*, pages 307–318. SIAM / Omnipress, 2012. ISBN 978-1-61197-232-0. doi: 10.1137/1.9781611972825.27. URL `https://doi.org/10.1137/1.9781611972825.27`.

[6] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with COTE: the collective of transformation-based ensembles. In

32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016, 2016.

[7] Suhrid Balakrishnan and David Madigan. Decision trees for functional variables. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 798–802. IEEE Computer Society, 2006. ISBN 0-7695-2701-9. doi: 10.1109/ICDM.2006.49. URL https://doi.org/10.1109/ICDM.2006.49.

[8] Gustavo E. A. P. A. Batista, Eamonn J. Keogh, Oben Moses Tataw, and Vinícius M. A. de Souza. CID: an efficient complexity-invariant distance for time series. *Data Min. Knowl. Discov.*, 28(3):634–669, 2014. doi: 10.1007/s10618-013-0312-3. URL https://doi.org/10.1007/s10618-013-0312-3.

[9] Mustafa Gokce Baydogan, George C. Runger, and Eugene Tuv. A bag-of-features framework to classify time series. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(11):2796–2802, 2013. doi: 10.1109/TPAMI.2013.72. URL https://doi.org/10.1109/TPAMI.2013.72.

[10] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kavé Salamatian. Traffic classification on the fly. *Computer Communication Review*, 36(2):23–26, 2006. doi: 10.1145/1129582.1129589. URL http://doi.acm.org/10.1145/1129582.1129589.

[11] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, July 1994. Technical Report WS-94-03*, 1994.

[12] Aaron Bostrom and Anthony Bagnall. Binary shapelet transform for multiclass time series classification. *T. Large-Scale Data- and Knowledge-Centered Systems*, 32:24–46, 2017. doi: 10.1007/978-3-662-55608-5_2. URL https://doi.org/10.1007/978-3-662-55608-5_2.

[13] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[14] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984. ISBN 0-534-98053-8.

[15] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, July 2015. www.cs.ucr.edu/~eamonn/time_series_data/.

[16] Ahlame Douzal Chouakria and Cécile Amblard. Classification trees for time series. *Pattern Recognition*, 2012.

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018.

[18] Stuart Daw, Charles Finney, and Eugene Tracy. A review of symbolic analysis of experimental data. 74:915–930, 02 2003.

[19] Houtao Deng, George C. Runger, Eugene Tuv, and Vladimir Martyanov. A time series forest for classification and feature extraction. *Inf. Sci.*, 239:142–153, 2013. doi: 10.1016/j.ins.2013.02.030. URL https://doi.org/10.1016/j.ins.2013.02.030.

[20] Mukund Deshpande and George Karypis. Evaluation of techniques for classifying biological sequences. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD '02, pages 417–431, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43704-5. URL http://dl.acm.org/citation.cfm?id=646420.693671.

[21] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008. URL http://www.vldb.org/pvldb/1/1454226.pdf.

[22] Cao-Tri Do, Ahlame Douzal-Chouakria, Sylvain Schneider Marié, Michèle Rombaut, and Saeed Varasteh. Multi-modal and multi-scale temporal metric learning for a robust time series nearest neighbors classification. *Information Sciences*, pages 418–419, 2017.

[23] Omer Duskin and Dror G. Feitelson. Distinguishing humans from robots in web search logs: preliminary results using query rates and intervals. In *WSCD@WSDM*, 2009.

[24] Charles Elkan. Using the triangle inequality to accelerate k-means. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 147–153. AAAI Press, 2003. ISBN 1-57735-189-4. URL http://www.aaai.org/Library/ICML/2003/icml03-022.php.

[25] Philippe Esling and Carlos Agón. Time-series data mining. *ACM Comput. Surv.*, 2012.

[26] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994.*, pages 419–429. ACM Press, 1994. doi: 10.1145/191839.191925. URL http://doi.acm.org/10.1145/191839.191925.

[27] Cedric Frambourg, Ahlame Douzal-Chouakria, and Eric Gaussier. Learning multiple temporal matching for time series classification. In *International Symposium on Intelligent Data Analysis*, pages 198–209. Springer, 2013.

[28] Cédric Frambourg. *Apprentissage d'appariements pour la discrimination de séries temporelles*. PhD thesis, 2013. URL http://www.theses.fr/2013GRENS025. Thèse de doctorat dirigée par Demongeot, Jacques et Douzal-Chouakria, Ahlame Modèles, méthodes et algorithmes en biologie, santé et environnement Grenoble 2013.

[29] Tak-chung Fu. A review on time series data mining. *Eng. Appl. of AI*, 24(1): 164–181, 2011. doi: 10.1016/j.engappai.2010.09.007. URL https://doi.org/10.1016/j.engappai.2010.09.007.

[30] Ben D. Fulcher. Feature-based time-series analysis. *CoRR*, abs/1709.08055, 2017. URL http://arxiv.org/abs/1709.08055.

[31] Pierre Geurts. Pattern extraction for time series classification. In Luc De Raedt and Arno Siebes, editors, *Principles of Data Mining and Knowledge Discovery, 5th European Conference, PKDD 2001, Freiburg, Germany, September 3-5, 2001, Proceedings*, volume 2168 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2001. ISBN 3-540-42534-9. doi: 10.1007/3-540-44794-6_10. URL https://doi.org/10.1007/3-540-44794-6_10.

[32] Tomasz Górecki and Maciej Luczak. Using derivatives in time series classification. *Data Min. Knowl. Discov.*, 26(2):310–331, 2013. doi: 10.1007/s10618-012-0251-4. URL https://doi.org/10.1007/s10618-012-0251-4.

[33] Tomasz Górecki and Maciej Luczak. First and second derivatives in time series classification using DTW. *Communications in Statistics - Simulation and Computation*, 43(9):2081–2092, 2014. doi: 10.1080/03610918.2013.775296. URL https://doi.org/10.1080/03610918.2013.775296.

[34] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, 2014.

[35] Steinn Gudmundsson, Thomas Philip Runarsson, and Sven Sigurdsson. Support vector machines and dynamic time warping for time series. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*, pages 2772–2776. IEEE, 2008. ISBN 978-1-4244-1820-6. doi: 10.1109/IJCNN.2008.4634188. URL https://doi.org/10.1109/IJCNN.2008.4634188.

[36] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Min. Knowl. Discov.*, 28(4):851–881, 2014. doi: 10.1007/s10618-013-0322-1. URL https://doi.org/10.1007/s10618-013-0322-1.

[37] Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675, 1977. doi: 10.1145/322033.322044. URL http://doi.acm.org/10.1145/322033.322044.

[38] Gísli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003. doi: 10.1145/958942.958948. URL http://doi.acm.org/10.1145/958942.958948.

[39] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-32:67–72, February 1975.

[40] Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9): 2231 – 2240, 2011. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2010.09.022. URL http://www.sciencedirect.com/science/article/pii/S003132031000484X. Computer Analysis of Images and Patterns.

[41] Rohit J. Kate. Using dynamic time warping distances as features for improved time series classification. *Data Min. Knowl. Discov.*, 2016.

[42] Eamonn J. Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Min. Knowl. Discov.*, 7(4):349–371, 2003. doi: 10.1023/A:1024988512476. URL https://doi.org/10.1023/A:1024988512476.

[43] Eamonn J. Keogh and Michael J. Pazzani. Scaling up dynamic time warping for datamining applications. In Raghu Ramakrishnan, Salvatore J. Stolfo, Roberto J. Bayardo, and Ismail Parsa, editors, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 285–289. ACM, 2000. ISBN 1-58113-233-6. doi: 10.1145/347090.347153. URL http://doi.acm.org/10.1145/347090.347153.

[44] Eamonn J. Keogh and Michael J. Pazzani. Derivative dynamic time warping. In Vipin Kumar and Robert L. Grossman, editors, *Proceedings of the First SIAM International Conference on Data Mining, SDM 2001, Chicago, IL, USA, April 5-7, 2001*, pages 1–11. SIAM, 2001. ISBN 978-0-89871-495-1. doi: 10.1137/1.9781611972719.1. URL https://doi.org/10.1137/1.9781611972719.1.

[45] Eamonn J. Keogh and Thanawin Rakthanmanon. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *Proceedings of the 13th SIAM International Conference on Data Mining, May 2-4, 2013. Austin, Texas, USA.*, pages 668–676. SIAM, 2013. ISBN 978-1-61197-262-7. doi: 10.1137/1.9781611972832.74. URL https://doi.org/10.1137/1.9781611972832.74.

[46] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa*

Barbara, CA, USA, May 21-24, 2001, pages 151–162. ACM, 2001. ISBN 1-58113-332-4. doi: 10.1145/375663.375680. URL http://doi.acm.org/10.1145/375663.375680.

[47] Flip Korn, H. V. Jagadish, and Christos Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. *SIGMOD Rec.*, 26(2):289–300, June 1997. ISSN 0163-5808. doi: 10.1145/253262.253332. URL http://doi.acm.org/10.1145/253262.253332.

[48] Ludmila I. Kuncheva. Diversity in multiple classifier systems. *Information Fusion*, 6(1):3–4, 2005. doi: 10.1016/j.inffus.2004.04.009. URL https://doi.org/10.1016/j.inffus.2004.04.009.

[49] Terran Lane and Carla E. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, pages 150–158, New York, NY, USA, 1998. ACM. ISBN 1-58113-007-4. doi: 10.1145/288090.288122. URL http://doi.acm.org/10.1145/288090.288122.

[50] Daniel Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *CoRR*, abs/0811.3301, 2008. URL http://arxiv.org/abs/0811.3301.

[51] Christina S. Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the 7th Pacific Symposium on Biocomputing, PSB 2002, Lihue, Hawaii, USA, January 3-7, 2002*, pages 566–575, 2002. URL http://psb.stanford.edu/psb-online/proceedings/psb02/leslie.pdf.

[52] Christina S. Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004. doi: 10.1093/bioinformatics/btg431. URL https://doi.org/10.1093/bioinformatics/btg431.

[53] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 2007.

[54] Jessica Lin, Rohan Khade, and Yuan Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *J. Intell. Inf. Syst.*, 39(2):287–315, October 2012. ISSN 0925-9902. doi: 10.1007/s10844-012-0196-5. URL http://dx.doi.org/10.1007/s10844-012-0196-5.

[55] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Min. Knowl. Discov.*, 29(3):565–592, 2015. doi: 10.1007/s10618-014-0361-2. URL https://doi.org/10.1007/s10618-014-0361-2.

[56] Jason Lines, Luke M. Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 289–297. ACM, 2012. ISBN 978-1-4503-1462-6. doi: 10.1145/2339530.2339579. URL http://doi.acm.org/10.1145/2339530.2339579.

[57] Zachary Chase Lipton. The mythos of model interpretability. *CoRR*, 2016.

[58] Fang-Jun Luan, Kai Li, and Si-Liang Ma. The algorithm of online handwritten signature verification based on improved dtw. 10:81 – 86, 07 2010.

[59] Pierre-Francois Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(2):306–318, 2009. doi: 10.1109/TPAMI.2008.76. URL https://doi.org/10.1109/TPAMI.2008.76.

[60] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *CoRR*, abs/1706.07269, 2017. URL http://arxiv.org/abs/1706.07269.

[61] Theophano Mitsa. *Temporal Data Mining*. Chapman & Hall/CRC, 1st edition, 2010. ISBN 1420089765, 9781420089769.

[62] Norrima Mokhtar, H Arof, and Masahiro Iwahashi. One dimensional image processing for eye tracking using derivative dynamic time warping. 5:2947–2952, 01 2010.

[63] Abdullah Mueen, Eamonn J. Keogh, and Neal E. Young. Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, 2011.

[64] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. Information processing and technology. chapter Feature-based Classification of Time-series Data, pages 49–61. Nova Science Publishers, Inc., Commack, NY, USA, 2001. ISBN 1-59033-116-8. URL http://dl.acm.org/citation.cfm?id=766914.766918.

[65] Chang-Shing Perng, Haixun Wang, Sylvia R. Zhang, and Douglas Stott Parker Jr. Landmarks: a new model for similarity-based pattern querying in time series databases. In *ICDE*, pages 33–42. IEEE Computer Society, 2000.

[66] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi: 10.1023/A:1022643204877. URL https://doi.org/10.1023/A:1022643204877.

[67] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. ISBN 1-55860-238-0.

[68] Laura Elena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, May 2004. ISSN 1573-7470. doi: 10.1023/B:AMAI.0000018580.96245.c6. URL https://doi.org/10.1023/B:AMAI.0000018580.96245.c6.

[69] Chotirat (Ann) Ratanamahatana and Eamonn J. Keogh. Making time-series classification more accurate using learned constraints. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*, pages 11–22, 2004. doi: 10.1137/1.9781611972740.2. URL https://doi.org/10.1137/1.9781611972740.2.

[70] Xavier Renard, Maria Rifqi, Walid Erray, and Marcin Detyniecki. Random-shapelet: an algorithm for fast shapelet discovery. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.

[71] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM, 2016. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778. URL http://doi.acm.org/10.1145/2939672.2939778.

[72] John F. Roddick, Kathleen Hornsby, and Myra Spiliopoulou. An updated bibliography of temporal, spatial, and spatio-temporal data mining research. In John F. Roddick and Kathleen Hornsby, editors, *Temporal, Spatial, and Spatio-Temporal Data Mining, First International Workshop TSDM 2000 Lyon, France, September*

12, 2000, Revised Papers, volume 2007 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2000. ISBN 3-540-41773-7. doi: 10.1007/3-540-45244-3_12. URL https://doi.org/10.1007/3-540-45244-3_12.

[73] Juan José Rodríguez, Ludmila I. Kuncheva, and Carlos J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1619–1630, 2006. doi: 10.1109/TPAMI.2006.211. URL https://doi.org/10.1109/TPAMI.2006.211.

[74] H Sakoe and S Chiba. Dynamic programming algorithm optimization for spoken word recognition. *EEE Transaction on Acoustic Speech Signal Processing*, 26(1): 43–49, 1978.

[75] Gerard Salton, A. Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975. doi: 10.1145/361219.361220. URL http://doi.acm.org/10.1145/361219.361220.

[76] Steven Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Min. Knowl. Discov.*, 1(3):317–328, 1997. doi: 10.1023/A:1009752403260. URL https://doi.org/10.1023/A:1009752403260.

[77] Patrick Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Min. Knowl. Discov.*, 29(6):1505–1530, 2015.

[78] Patrick Schäfer. Scalable time series classification. *Data Min. Knowl. Discov.*, 30 (5):1273–1298, 2016. doi: 10.1007/s10618-015-0441-y. URL https://doi.org/10.1007/s10618-015-0441-y.

[79] Pavel Senin and Sergey Malinchik. SAX-VSM: interpretable time series classification using SAX and vector space model. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, 2013.

[80] Hiroshi Shimodaira, Ken-ichi Noma, Mitsuru Nakai, and Shigeki Sagayama. Dynamic time-alignment kernel in support vector machine. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 921–928. MIT Press, 2001. URL http://papers.nips.cc/paper/2131-dynamic-time-alignment-kernel-in-support-vector-machine.

[81] Saeid Soheily-Khah, Ahlame Douzal-Chouakria, and Eric Gaussier. Generalized k-means-based clustering for temporal data under weighted and kernel time warp. *Pattern Recognition Letters*, 75:63 – 69, 2016. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2016.03.007. URL http://www.sciencedirect.com/science/article/pii/S0167865516000763.

[82] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. The move-split-merge metric for time series. *IEEE Trans. Knowl. Data Eng.*, 25(6):1425–1438, 2013. doi: 10.1109/TKDE.2012.88. URL https://doi.org/10.1109/TKDE.2012.88.

[83] Xiaoyue Wang, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *CoRR*, abs/1012.2789, 2010. URL http://arxiv.org/abs/1012.2789.

[84] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. *CoRR*, abs/1611.06455, 2016.

[85] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2006.

[86] Martin Wistuba, Josif Grabocka, and Lars Schmidt-Thieme. Ultra-fast shapelets for time series classification. *CoRR*, abs/1503.05018, 2015.

[87] Xiaopeng Xi, Eamonn J. Keogh, Christian R. Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In William W. Cohen and Andrew Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 1033–1040. ACM, 2006. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143974. URL http://doi.acm.org/10.1145/1143844.1143974.

[88] Zhengzheng Xing, Jian Pei, and Eamonn J. Keogh. A brief survey on sequence classification. *SIGKDD Explorations*, 12(1):40–48, 2010. doi: 10.1145/1882471.1882478. URL http://doi.acm.org/10.1145/1882471.1882478.

[89] Yuu Yamada, Einoshin Suzuki, Hideto Yokoi, and Katsuhiko Takabayashi. Decision-tree induction from time-series data based on a standard-example split test. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, 2003.

[90] Kiyoung Yang and Cyrus Shahabi. An efficient k nearest neighbor search for multivariate time series. *Information and Computation*, 205(1):65–98, 2007.

[91] Lexiang Ye and Eamonn J. Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD*, 2009.

[92] Jidong Yuan, Zhihai Wang, and Meng Han. A discriminative shapelets transformation for time series classification. *IJPRAI*, 28(6), 2014. doi: 10.1142/S0218001414500141. URL `https://doi.org/10.1142/S0218001414500141`.

[93] Jidong Yuan, Ahlame Douzal-Chouakria, Saeed Varasteh Yazdi, and Zhihai Wang. A large margin time series nearest neighbour classification under locally weighted time warps. *Knowledge and Information Systems*, Mar 2018. ISSN 0219-3116. doi: 10.1007/s10115-018-1184-z. URL `https://doi.org/10.1007/s10115-018-1184-z`.