Time series classification with Shapelets : Application to predictive maintenance on event logs

Antoine Guillaume

Université d'Orléans

École Doctorale Mathématiques, Informatique, Physique Théorique et Ingénierie des Systèmes

Thèse présentée par :

Antoine Guillaume

soutenue le : 4 janvier 2023

pour obtenir le grade de : Docteur de l'Université d'Orléans

Discipline/ Spécialité : Informatique

Time series classification with Shapelets: Application to predictive maintenance on event logs

Thèse dirigée par :

Pr. VRAIN Christel	Université d'Orléans
Dr. ELLOUMI Wael	Ingénieur R&D, Worldline

RAPPORTEURS :

Pr. BAGNALL Anthony	University of East Anglia
Pr. CORNUÉJOLS Antoine	AgroParisTech

JURY :

Pr. BAGNALL Anthony	University of East Anglia
Pr. CORNUÉJOLS Antoine	AgroParisTech
Dr. ELLOUMI Wael	Ingénieur R&D, Worldline
Pr. FORESTIER Germain	Université de Haute-Alsace
Pr. FROMONT Elisa	**Présidente** - Université de Rennes
Pr. VRAIN Christel	Université d'Orléans

Acknowledgements

I would like to express my deepest gratitude to Pr.Christel Vrain for her continuous support and mentoring during my Ph.D., and for her patience and kindness, even though I was sometime stubborn. I would also like to thank Dr.Wael Elloumi, which accepted to represent Worldline for this thesis, without which this whole endeavour would not even have happened.

My sincere thanks also goes to Pr.Anthony Bagnall and Pr.Antoine Cornuéjols who accepted to take the responsibility of reviewing this manuscript, and to Pr.Germin Forestier and Pr.Elisa Fromont for accepting to participate in the jury.

Last but not the least, I would like to thank my companion Caroline for her unconditional love through the ups and lows of this thesis, and my family and friends for their support, even though I was often caught up in my research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Résumé du chapitre en français
Dans ce chapitre, nous introduisons le contexte de cette thèse ainsi que les deux domaines sur lesquels nous apportons des contributions, à savoir, la classification des séries temporelles et la maintenance prédictive. Dans un premier temps, nous présentons les notions de classification, sur des données tabulaires et sur des séries temporelles, ainsi que la classification précoce. Ensuite, nous introduisons le cas de maintenance prédictive proposé par notre partenaire industriel et les particularités de celui-ci. Enfin, nous définissons informellement les shapelets, qui sont les composants sur lesquels se basent nos contributions pour la classification des séries temporelles. Après un résumé des contributions présentées dans cette thèse, nous définissons les notations utilisées dans ce manuscrit pour les séries temporelles, et concluons ce chapitre sur une présentation des outils que nous utiliserons pour comparer les différents algorithmes de classification que nous présenterons dans nos expérimentations.

Chapter summary
In this chapter, we introduce the context of this thesis as well as the two fields on which we make contributions, namely, time series classification and predictive maintenance. First, we present the notions of classification, time series classification, and early classification. Then, we introduce the industrial use case of predictive maintenance that motivated this thesis and its particularities. Finally, we informally define shapelets, which are the components on which are based our contributions for time series classification. After a summary of our contributions, we define the notations that we will use in the thesis to denote time series, and conclude this chapter with a presentation of the tools that we will use in our experiments to compare the different classification algorithms.

In machine learning, statistics, and more generally, in data analysis, we often distinguish between two tasks, regression and classification. In regression tasks, given a set of input variables, we want to estimate the value of a target variable. An example of regression task is to estimate the housing prices based on the size of the houses and other descriptors, such as the number of bedrooms. Classification tasks, while also considering a set of input variables, aims at grouping samples into predefined groups. For example, classifying species of flowers based on the length of their sepals and petal. In this thesis,

we focus on supervised tasks, where the target variable or group is known for all samples and is used to fit a model. Semi-supervised and unsupervised tasks also exist, where the target variable or the class is only known for some samples or simply unknown.

Those tasks can then be applied on different types of data such as tabular data, images, videos, and time series. In this thesis, we focus on the task of supervised classification for time series data. The goal of supervised time series classification is to build classifiers able to discriminate classes of a given problem, when data depends on time. For example, given a set of time series recorded by meteorological sensors, we want to be able to classify these series into classes representing the weather (e.g. sunny, raining, ...).

In some cases, it might be important to identify the class of a time series as soon as possible, without having to observe it entirely. For example, for earthquake prediction, we want to be able to issue an alert as soon as possible, based on early signals and not on the measurement of the event itself. In such cases, a specific field, called early classification, proposes solutions to adapt time series classification method to this context. More formally, the objective of early classification is twofold, it aims at correctly identifying the class of a time series, while maximizing the notion of earliness of the prediction, defined as the proportion of the time series used to make the prediction, as predictions can be made without having knowledge of the full data (i.e. up to the event we want to predict).

In the time series classification literature, early or not, most works [59, 69, 39, 84, 63] make the assumption that the input time series have the same length. This is due, in part, to the fact that the comparison between time series is less trivial when they are not temporally aligned and/or of the same length. In practice, time series with variable lengths are often converted to the same length via preprocessing operations [76], or by using sliding windows to extract sub-sequences of the same length. The use case that motivated this thesis, namely predictive maintenance on event logs, use time series of variable length as inputs and has particular constraints, close to the ones that motivate the use of early classification methods. One of our objectives is to see if using methods that take into account the entire time series, whatever their length is, has benefits compared to the existing approaches in the predictive maintenance literature which mostly use sliding windows.

In this industrial use case, the aim is to predict the occurrence of soon-to-happen failures on ATMs (Automatic Teller Machines), in order to perform preventive maintenance and to increase machine availability. The particularity is that the data is made up of event log streams, which are primarily designed for software analysis rather than for hardware maintenance. The goal being to raise maintenance alerts, not as soon as possible, as this would lead to unnecessary early maintenance, but within an acceptable time interval before the machine breakdown.

In this thesis, we explore the use of shapelets [86] for the classification of time series, where a shapelet is defined as a short sequence, often extracted directly from the data. In a supervised classification context, the objective of shapelet methods is to find shapelets (seen as patterns), which, by their absence or presence in the time series, make it possible to discriminate the classes of the problems. Shapelets have been used in early classification [39], due to their ability to identify a pattern regardless of its location in the series and because of their interpretability. One problem is that shapelet methods suffer from a lack of performance, both in terms of classification accuracy and speed, compared to the state of the art [63].

Motivated by the need for performance in the predictive maintenance application, we first propose a new shapelet method for the classification of time series. Then, we adapt this new classification algorithm to the classification of multivariate and variable length time series, and we propose a new multi-objective classification model, including business constraints related to the maintenance process.

1.1 Contributions

Given the motivations behind this thesis, we propose novel algorithms supported by large scale experimentations. The contributions of this thesis can be summarized as follows:

- **Random Dilated Shapelet Transform**[31] We present a new classification algorithm based on the notion of shapelets with dilation. It relies on a random generation of shapelets and integrates new descriptors, allowing not only to detect the presence of a pattern but also to measure its frequency. We conduct an experimental study on the sensitivity of the parameters of our method and a comparative study with state-of-the-art algorithms for time series. We also propose modifications of our method by exploring new properties such as phase invariance, and the use of material acceleration to reduce time complexity:
  - Extension of the formulation of shapelets and introduction of new descriptors. We present an extension of shapelets, introducing the notion of dilation, which allows shapelets to identify non-contiguous patterns. We also introduce new descriptors, allowing to measure the number of occurrences of a shapelet in the series.
  - Some invariance properties, such as scale invariance using a normalized distance, have already been established for shapelets. We explore other modifications to the shapelet formulation based on existing work in the time series literature, such as phase invariance, and we measure the impact of these additions on the performance of our method.
  - Many improvements have been proposed to reduce the time complexity of shapelet algorithms, such as the early abandon of distance computation using a lower bound. The performance of such methods has never been compared to hardware acceleration (e.g. use of the "Single Instruction on Multiple Data" architecture [27] of processors).
- **Application on the ATM predictive maintenance use case**
  - A new event logs dataset, extracted from an industrial ATM fleet from Worldline company. Several preprocessing steps have been needed, notably for data slicing and formatting. This dataset could be used to evaluate time series classification algorithms, applied to multivariate and unequal length data.
  - An experimental protocol specifically designed to evaluate predictive maintenance models, using metrics based on the cost of the maintenance process, to estimate the potential gains induced by the model over a baseline maintenance system.

– A time series classification model that uses features extracted by shapelets in a tree based model, with a new splitting criterion using the notion of survival probability to account for different causes of failures (e.g. sudden errors or mechanical wear) based on the current lifetime of a machine. It also includes a trigger system to try to optimize the time when maintenance alerts are raised, regarding the constraints of the maintenance process.

1.2 Thesis Organization

The thesis is organized in six chapters, which contain the introduction and our contributions to the field of time series classification and predictive maintenance, and the conclusion with future works. Chapters 2 and 3 are focused on time series classification. Chapter 2 contains an overview of the field, in which we present the different kinds of algorithms existing in the literature, followed by a more detailed review of shapelet based methods. We then present our contributions to time series classification in Chapter 3, and we present the result of our experiments.

In Chapter 4, we present and formalize the predictive maintenance problem, and present an overview of the different kinds of methods used in the literature to solve it for both classification and regression tasks. We also define an experimental protocol for predictive maintenance models, which allows evaluating the efficiency of such models from a business point of view. To conclude, we introduce the field of early classification and survival analysis, which will be used in our contributions. In Chapter 5, we present the predictive maintenance use case of Worldline, our industrial partner. We introduce the application context, the data collection and preprocessing steps, and conduct experiments using baseline approaches from the literature. We then present our contribution for this use case and perform experiments in order to study how it stands against the baseline results we established.

1.3 Notations

Throughout the manuscript, we use the following notations:

- A univariate time series will be denoted by an uppercase letter $X_i = \{x_1, ..., x_m\}$, with x_j a value and m the length of the time series. This notation extends to the multivariate context with $X_i = \{(x_{1,1}, \ldots, x_{1,k}), \ldots, (x_{m,1}, \ldots, x_{m,k})\}$ with k the number of features. When an ensemble of time series contains time series of variable lengths, the notation changes to $X_i = \{x_1, ..., x_{m_i}\}$ for univariate and to $X_i = \{(x_{1,1}, \ldots, x_{1,k}), \ldots, (x_{m_i,1}, \ldots, x_{m_i,k})\}$ for multivariate time series, with m_i the length of time series X_i.

- An ensemble of time series will be denoted by a calligraphic letter $\mathcal{X} = \{X_1, \ldots, X_n\}$ with X_i a time series and n the number of time series in the ensemble. If needed, the classes of the time series will be denoted as $Y = \{y_1, \ldots y_n\}$, with y_i the class of time series X_i.

The use of upper case and calligraphic letter is applied similarly for any kind of vector or set of elements. Any change or addition to those notations will be introduced at the beginning of each chapter or section.

1.4 Comparison of classifiers

In this section, we introduce tools that we will use multiple times throughout this thesis to analyse experimental results, and we give details about how they are built and how to read them.

1.4.1 Critical difference diagrams

As we will compare multiple classifiers across many datasets, we use critical difference diagrams [24] to produce a robust interpretation of the results, based on statistical testing. To introduce how those diagrams work, let us consider the example given by Figure 1.1.

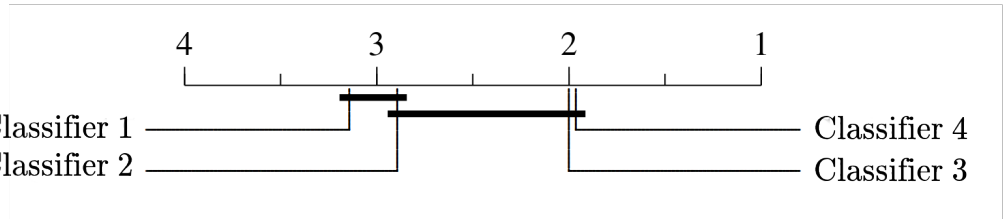

Figure 1.1: Example of critical difference diagram

Each classifier is positioned on the axis by its mean accuracy rank on all the datasets (e.g., classifier 3 has a mean rank of 2 in Figure 1.1), and a bar between a set of classifiers indicates that, despite the differences in accuracy ranks, the difference in accuracy between them is not statistically significant.

More formally, if we have $D = \{d_1, \dots d_r\}$ a set of r datasets and $C = \{c_1, \dots, c_s\}$ a set of s classifiers, we can compute two matrices M and R of size (r, s), where $M_{i,j}$ is the accuracy of the classifier c_j on the dataset d_i, and $R_{i,j}$ is the rank of c_j on the dataset d_i, which is computed using the accuracy of all classifiers on the dataset d_i, contained in the row M_i. The mean rank of c_j is then the mean of the column j of the matrix R, which we will denote as $\bar{r}_j$.

Given M and R, the first step is to test a null hypothesis, stating that the average ranks of classifiers are the same. This hypothesis is then tested by using the Friedman statistic, denoted as F, and approximated with a Chi-squared distribution $\mathcal{X}^2$ with $(r-1)$ degrees of freedom as :

$$\mathcal{X}^2 = \frac{12r}{s(s+1)} \times \sum_{j=1}^{s} \left(\bar{r}_j^2 - \frac{s(s+1)^2}{4} \right) \tag{1.1}$$

Which, as proposed by [24], is then used to approximate F as

$$F = \frac{(r-1)\mathcal{X}^2}{r(s-1) - \mathcal{X}^2} \tag{1.2}$$

If the null hypothesis is rejected, a pairwise Nemenyi tests can then be used to highlight the differences between each pair of classifiers. Given two classifiers c_i and c_j, and the critical value q_α taken from the Studentized range statistic divided by $\sqrt{2}$, with α the p-value (set to 0.05 by default), we can compute the critical difference CD as:

$$CD = q_\alpha \sqrt{\frac{s(s+1)}{6r}} \tag{1.3}$$

The performance of c_i and c_j is significantly different if their average ranks differ by at least CD (i.e. they will not be linked by a black bar in the resulting figure).

1.4.2 Pairwise accuracy plots

Given two classifiers c_1 and c_2 and an ensemble of datasets $D = \{d_1, \dots d_r\}$, a pairwise accuracy plot is a scatter plot which compares the accuracy of c_1 against c_2 for each dataset. A point p_i of this scatter plot is positioned to the coordinates $(a_{c_2,i}, a_{c_1,i})$, with $a_{c_1,i}$ the accuracy of the classifier c_1 on the dataset i. Figure 1.2 shows an example of such plot with 100 datasets.

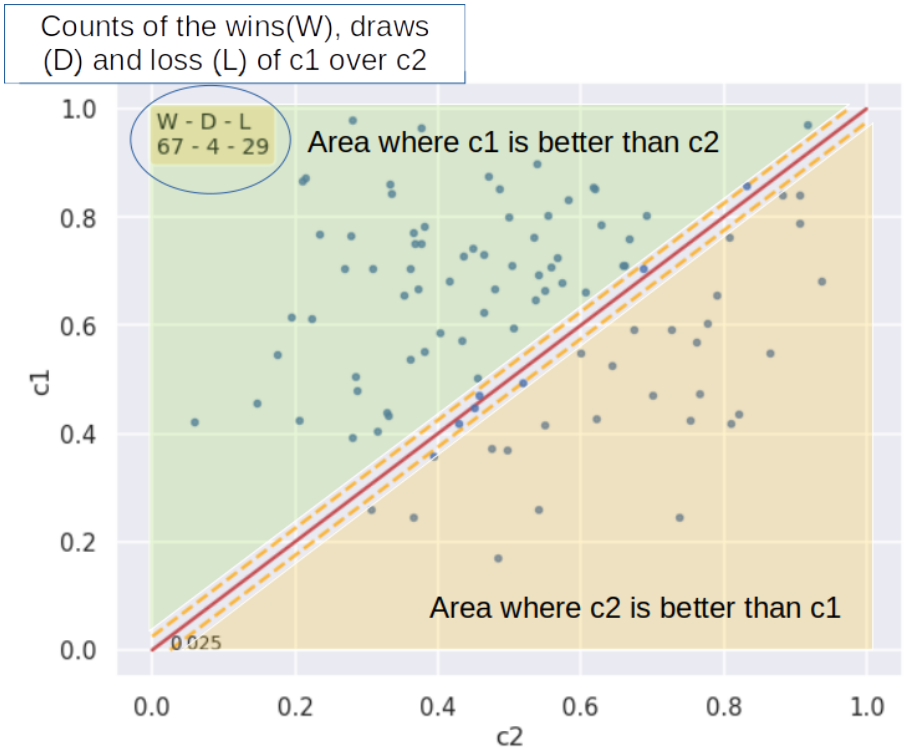

Figure 1.2: Example of pairwise accuracy plot

A line $f(x) = y$ is added to this plot, if a point is below this line, it means that for this dataset, c_2 was better than c_1. If the point is above, c_1 was better, and if it is close to the line $f(x) = y$, it is considered as a draw. The distance threshold for a point to $f(x) = y$ to be considered as a draw is determined by the two dashed oranges lines in Figure 1.2. In this example, they represent draws as differences in accuracy between c_1 and c_2 within +/-2.5%. The total count of wins, draws and losses for c_1 is summarized in the top left corner.

Chapter 2

Time Series Classification

Résumé du chapitre en français
Dans ce chapitre, nous présentons le domaine de la classification des séries temporelles et les approches existantes qui ont été développées pour résoudre ce problème. Dans un premier temps, nous définissons ce qu'est une série temporelle et les différentes propriétés qu'elles peuvent avoir. Nous présentons ensuite les différents types d'approches utilisées pour résoudre le problème de classification : les algorithmes se basant sur les plus K proches voisins, sur des statistiques extraites d'intervalles de temps, la création de dictionnaires contenant le nombre d'occurrences de sous séquences discrétisées dans les séries, les approches utilisant des noyaux de convolution, les méthodes d'ensemble, et enfin, les méthodes d'apprentissage profond. Nous présentons ensuite plus en détails les shapelets, et comment elles sont extraites et utilisées par les algorithmes de la littérature. Pour conclure, nous montrons la performance de ces différentes approches sur un ensemble de 112 jeux de données de classification de séries temporelles.

Chapter summary
In this chapter, we present the field of time series classification and the existing approaches that have been developed to solve this problem. First, we define what are time series and the different properties that they can have. We then present the different types of approaches used to solve the time series classification problem: algorithms based on the K nearest neighbors, on statistics extracted from time intervals, the creation of dictionaries counting the numbers occurrences of discretized subsequences in series, approaches using convolutional kernels, ensemble methods, and finally, deep learning methods. We then define shapelets, and how they are extracted and used by the algorithms of the literature. To conclude, we show the performance of these different approaches on a set of 112 datasets.

2.1 Introduction

Time series classification (TSC) algorithms take as input a set of time series $\mathcal{X} = \{X_1, \ldots, X_n\}$ with $X_i = \{x_1, \ldots, x_m\}$ a (univariate in this example) time series, and $t_{i,j}$ a timestamp associated with the measurement of x_j in X_i. Each time series is also

linked to a class y_i, which represents the group it belongs to. A TSC algorithm aims at learning a model on such data that correctly predicts the class y_j to which a time series $X_j \notin \mathcal{X}$ belongs to.

The right types of algorithms to use for a time series classification task can vary based on the characteristics of the input time series. Assessing those characteristics is an important step to know what kind of preprocessing steps and classification models we should consider for a given problem. These characteristics can be listed as follows:

- Are time series univariate, such as $X_i = \{x_1, \ldots, x_m\}$, or multivariate, as $X_i = \{(x_{1,1}, \ldots, x_{1,k}), \ldots, (x_{m,1}, \ldots, x_{m,k})\}$ with k the number of features?
- Are the time series of the same length, such as, given m_i the length of time series X_i, we have $m_i = m_j,\ \forall i, j \in \{1, \ldots, n\}$? Or is the length variable between series?
- Do the series have the same frequency of observations f, such as, given $t_{i,j}$ the time associated with the measurement of $x_j \in X_i$, we have $t_{i,j+1} - t_{i,j} = f,\ \forall i \in [1, n] \wedge\ \forall j \in [1, m]$?
- Do the series contain missing values? Is there a trend or a seasonal component in the data?
- Are the series temporally aligned? In other words, for two series i and a, do we have $t_{i,j} = t_{a,j}$, such as the j^{th} timestamp of both series represent the same time ? If not, are they aligned with respect to a seasonal component (e.g. by weekdays, independently of the week number)?

To simplify a TSC problem, it is desirable that the time series we use share the same properties. Processing operations, such as sampling the series to a common frequency, or removing a trend and seasonal component from the data, allow TSC models to focus on finding information that can discriminate classes without being biased by differences between the series caused by such properties.

Even after preprocessing steps, knowing the properties of the input time series is important, as algorithms often make assumptions on the data. For example, some algorithms use distance or dissimilarity functions to compare time series, such as the Euclidean distance between two series X, Y, which performs a pointwise squared difference, and is defined as :

$$d(X, Y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \tag{2.1}$$

As the Euclidean distance performs a pointwise comparison, it would give misleading results about the distance between X and Y if they are not temporally aligned or do not have the same frequency of observation, as we do not compare the same information. In those cases, a popular alternative is to use Dynamic Time Warping (DTW), which is a dissimilarity function that calculates an optimal match between two series.

Given a set of constraints, such as that every point in X must be matched with at least one point of Y (and vice versa), and a cost function, DTW finds an optimal "path" between the two series that satisfies all constraints with the minimal cost. This cost function is often defined as the sum of absolute differences for each matched pair of points, and the

minimal cost found by DTW is then returned as the result to indicate the dissimilarity between X and Y. As a point of X can be matched with multiple points of Y, a difference in length can be handled, but it is also useful for series of the same length which are not temporally aligned.

In the next section, we present the different types of algorithms that are proposed in the TSC literature and the methodologies they use to learn classification models from time series data.

2.2 Related Work

In this section, we present an overview of the time series classification literature. We focus on building intuition for each method and report the reader to the original papers for further details on each individual method. We present the algorithms of the literature by grouping them into families of methods:

- Distance-based algorithms, which classify time series by their nearest neighbours given a distance or dissimilarity function.
- Interval-based algorithms, which extract summary statistics from the same time interval for all time series and use those statistics to build a classifier.
- Dictionary-based algorithms, which extract words from discretized subsequences of the series and build histograms of words. Series are then classified based on their distance or similarity with the histograms of each class.
- Convolutional kernel algorithms, which transform the series by convolving them with a set of kernels. Features are then extracted from those convolutions and used to build a classifier.
- Ensemble methods, which group together approaches from one or more families and combine their predictions with a voting scheme or a meta estimator.
- Deep Learning methods, which focus on recurrent or convolutional networks to handle time series data.

2.2.1 Distance-based algorithms

Distance-based algorithms rely on the idea that series should be of the same class as their neighbours. They compare two series by using a distance or a dissimilarity function. Most methods use a K-Nearest Neighbors classifier in order to predict the class of time series based on the classes of its neighbourhood. Distance-based algorithms differ by the choice of the distance function and the use of ensemble methods. In the following, we present the most widespread methods.

K-Nearest Neighbors (K-NN)

For time series, similarly to the non-temporal version, a K-NN does not learn a model, but keeps the training set $\mathcal{X} = \{X_1, \ldots, X_n\}$ and uses it to classify new time series.

Given X a time series to be classified, a K-NN first computes $D = \{d(X, X_i) \mid \forall i \in [1, n]\}$, and extracts a subset $\mathcal{K}$ of size K, which contains the series that are the closest to X given d. The predicted class $\hat{y}$, for a weighted K-NN, is then given by Equation 2.2 with v the possible output classes. We use the identity function I, which takes as input a boolean and return an integer, such as $I(True) = 1$ and $I(False) = 0$. Removing $1/d(X, X_i)$ in Equation 2.2 makes it an unweighted majority voting.

$$\hat{y} = \underset{v}{\operatorname{argmax}} \sum_{(X_i, y_i) \in \mathcal{K}} \frac{1}{d(X, X_i)} \times I(v \equiv y_i) \tag{2.2}$$

Any distance or dissimilarity function can be considered, but the most widespread distances in the literature are the Euclidean distance and dynamic time warping (DTW) [64]. Until recently (see [22]), a 1-NN DTW classifier was considered as the standard baseline to compare against new classification algorithms.

If it makes sense for the current dataset and the distance function, a K-NN can be preceded by a data normalization step (e.g. min-max normalization) to remove the sensibility to scale.

Elastic Ensemble (EE)

The Elastic Ensemble method [47] is an ensemble classifier composed of multiple 1-NN classifiers, each using different distance or dissimilarity functions. Amongst those functions, we can find the Euclidean distance, the Longest Common Subsequence distance, as well as variants of the Edit distance and Dynamic Time Warping, which allow for some realignment to compensate for time shifts between series. The voting scheme for this ensemble classifier is a weighted majority voting, with the weight of each classifier determined by its training accuracy in a leave-one-out cross validation.

Variations of this method have been developed, notably to reduce time complexity. For example, FastEE [77] introduces multiple lower bounding techniques to perform early abandon of distance computations, as well as a lazy algorithm to leverage those lower bounds. Other approaches such as TS-QUAD [49] aim at reducing the size of the ensemble, and thus its time complexity, by isolating the most important components of the ensemble. The author shows that this can be achieved without a significative loss of accuracy.

Proximity Forest (PF)

A Proximity Forest [53] is an ensemble method composed of Proximity Trees, which follow the logic of a decision tree, but rather than splitting the data based on the value of a feature, it builds splits based on multiple 1-NN time series classifiers at each node of the tree.

Given a set of distance functions D, such as those used in the Elastic Ensemble, a Proximity Tree recursively builds nodes by sampling "exemplars", which are time series sampled from the input data. Each node of a Proximity Tree contains a distance function, which is chosen at random from D, and as many branches as there are classes in the input data of the node, with each branch containing an exemplar sampled from the class affected to this branch.

Time series in the parent node are then affected to the child node (i.e. branch) from which they are the closest given the distance function of the parent node. Figure 2.1

gives a visual example of this process. To select the set of exemplars used in the branch, multiple candidates splits (i.e. sets of one exemplar per class) are sampled, and the one that maximize the information gain is selected.

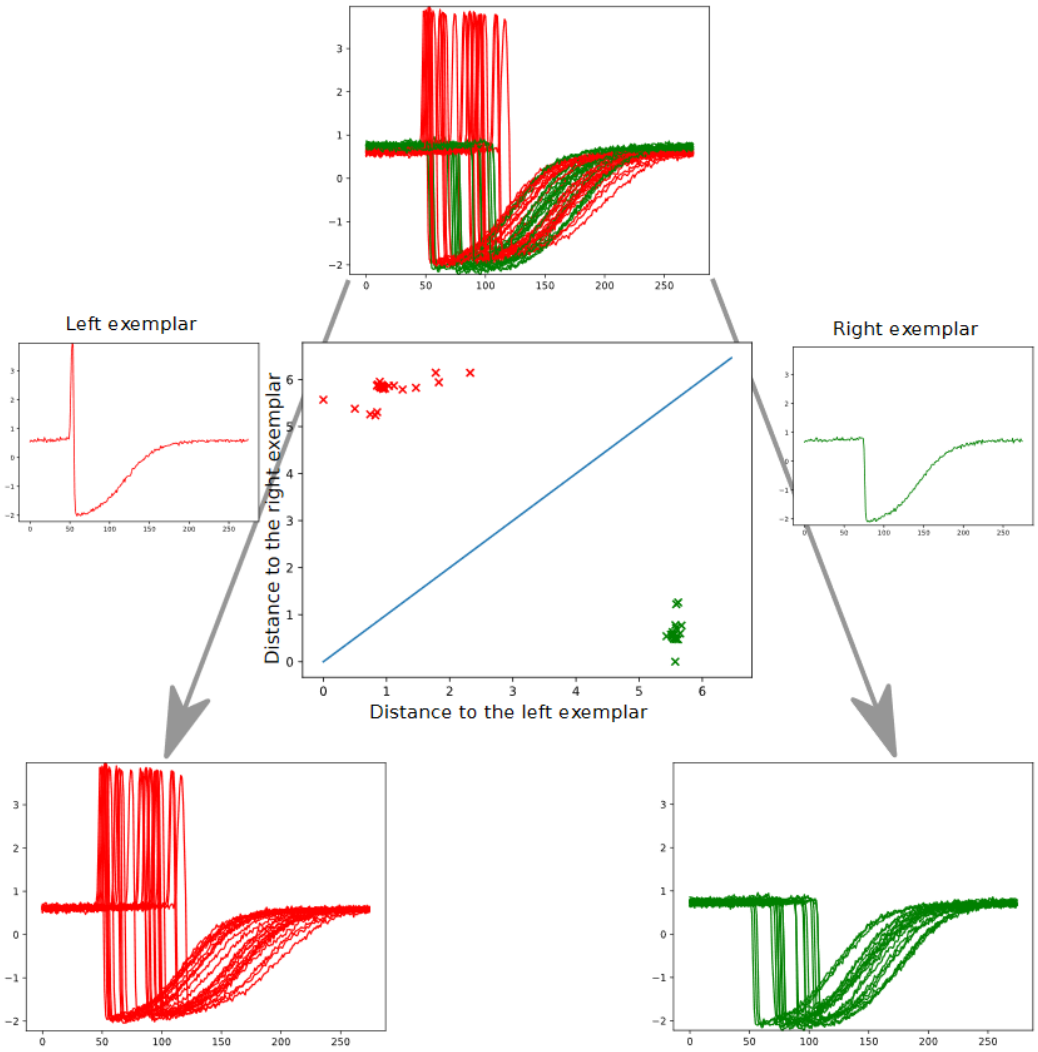

Figure 2.1: An example split made by a proximity tree on two classes of the Trace dataset. Image from the original paper of [53]

2.2.2 Interval-based algorithms

Interval-based algorithms focus on extracting features from time intervals. Their goal is to find discriminative information between time series based on their behaviour (i.e. features) in a specific time window. For example, given $\mathcal{X} = \{X_1, \ldots, X_n\}$ and $X_i = \{x_1, \ldots, x_m\}$, time series could be compared based on the mean value of the interval $[a, b] \subset [1, m]$ for all $X_i \in \mathcal{X}$.

Time Series Forest

A Time Series Forest (TSF) [25] applies the principle of random forests to time series data. A time series tree will randomly sample $\sqrt{p}$ starting positions and $\sqrt{p}$ interval lengths, leading to p intervals on which statistics like mean, standard deviation and slope are computed, resulting in candidates features to be evaluated. The best candidate is the one that maximizes the information gain, and the tree is recursively built until leaves are pure.

Other works, such as Supervised Time series Forest [14], include new input representations to look for discriminative interval-features, for example in the frequency domain, or using the first order difference. Then, they extract interval-features independently for each representation, followed by a supervised feature selection step, independently performed for each representation, before feeding the selected features to a decision tree. Figure 2.2 gives an overview of this process. It was shown empirically that their strategy significantly improves the accuracy and reduces the time complexity compared to Time Series Forest.

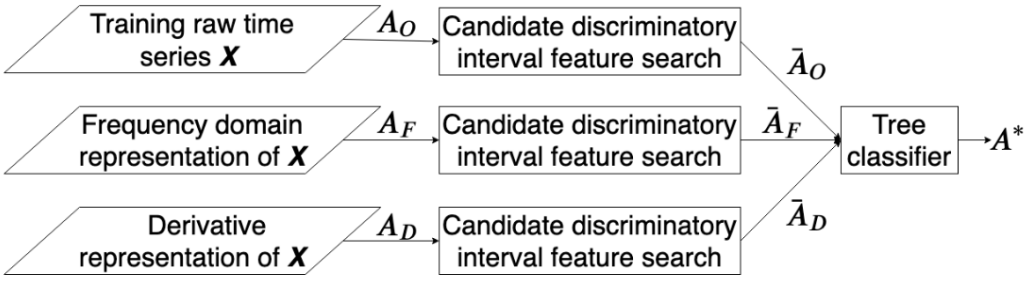

Figure 2.2: Overview of the approach taken by the Supervised Time series Forest. Image from the original paper of [14]. A_O, A_F, A_D are the spaces of possible interval features for each representation. $\bar{A_O}, \bar{A_F}, \bar{A_D}$ are the feature spaces after a feature selection step.

Canonical Interval Forest Classifier (CIF)

In [52], the authors studied the performance of 7658 features from the time series literature, and through multiple filtering steps, including statistical testing based on their individual performance on the datasets of UCR archive [19], extracted a subset of 22 features, which contains as the most discriminant feature subset.

Those 7658 features includes results from most of the existing works on time series, such as statistics of time-series values, linear correlations, measures of stationarity and entropy, results from methods of the time series analysis literature, wavelet coefficients, parameters from linear and nonlinear models such as autoregressive moving average (ARMA), etc...

The Canonical Interval Forest Classifier (CIF) [56] adds these 22 features to the 3 features extracted by Time Series Forest (mean, standard deviation and slope), and randomly samples 8 of those 25 features to be extracted for each tree of the ensemble. Similarly to Time Series Forest, it then samples starting position and intervals lengths, on which these 8 features are computed. An extension to the multivariate context was proposed by [56], allowing tree based models to be applied to the multivariate context.

An extension of this work was proposed by [58], named Diverse Representations Canonical Interval Forest Classifier (DrCIF), where CIF is applied to different input representation, similarly to the Supervised Time Series Forest.

Random Interval Spectral Ensemble (RISE)

Like the Time Series Forest algorithm, the Random Interval Spectral Ensemble (RISE) [50] takes the approach of extracting features from random intervals in the data. The main difference resides in the type of features extracted: while TSF extracts summary statistics like the mean, RISE extracts spectral (i.e. frequency-domain) features such as the power spectrum and the autocorrelation function. Another version of RISE, c-RISE [29] was introduced to drastically reduce the run-time of the algorithm, as well as allowing to be time contracted (i.e. impose a time limit for training), without a significative loss of accuracy. Among other improvements, the random intervals are resized to the nearest power of 2 to allow for faster computation of some spectral features.

2.2.3 Dictionary-based algorithms

Dictionary methods leverage text-mining research and adapt it to the field of time series classification, notably through histograms of words or bigrams extracted from discretized representations of the time series.

Bag of SFA Symbols (BOSS)

The Bag of Symbolic Fourier Approximation Symbols (BOSS) [68] uses a text mining approach to solve a time series classification problem. It takes as inputs a time series X, a word size l, an alphabet a and a window size w, and produces a histogram of Symbolic Fourier Approximation (SFA) words extracted from X.

More precisely, X is divided into moving windows of size w, each window is then approximated as a word of size l ($l \leq w$) using SFA with alphabet a (i.e. a is the set of discrete output values, inducing $a+1$ intervals bounds for discretization). The histogram is then built with all the extracted words, after using a numerosity reduction technique, which prunes successive occurrences of the same SFA word. This transformation process is visually described in Figure 2.3.

To discriminate classes, the authors first proposed an ensemble method based on a 1-NN classification, where each component of the ensemble uses different BOSS parameters. Later, [65] changed the classification model, while still using the BOSS transformation: for each class one histogram is computed by summing the histogram built from the samples of this class, and tf-idf (term frequency inverse document frequency) vector is computed. A new sample is predicted as the class leading to the highest cosine similarity between its tf-idf vector and the tf-idf vectors of the sample to predict.

Word Extraction for Time Series Classification (WEASEL)

Similarly to BOSS, Word Extraction for Time Series Classification (WEASEL) [66] extracts words from moving windows of varying lengths using an approximation technique to discretize the inputs. There are nonetheless major differences between the two approaches,

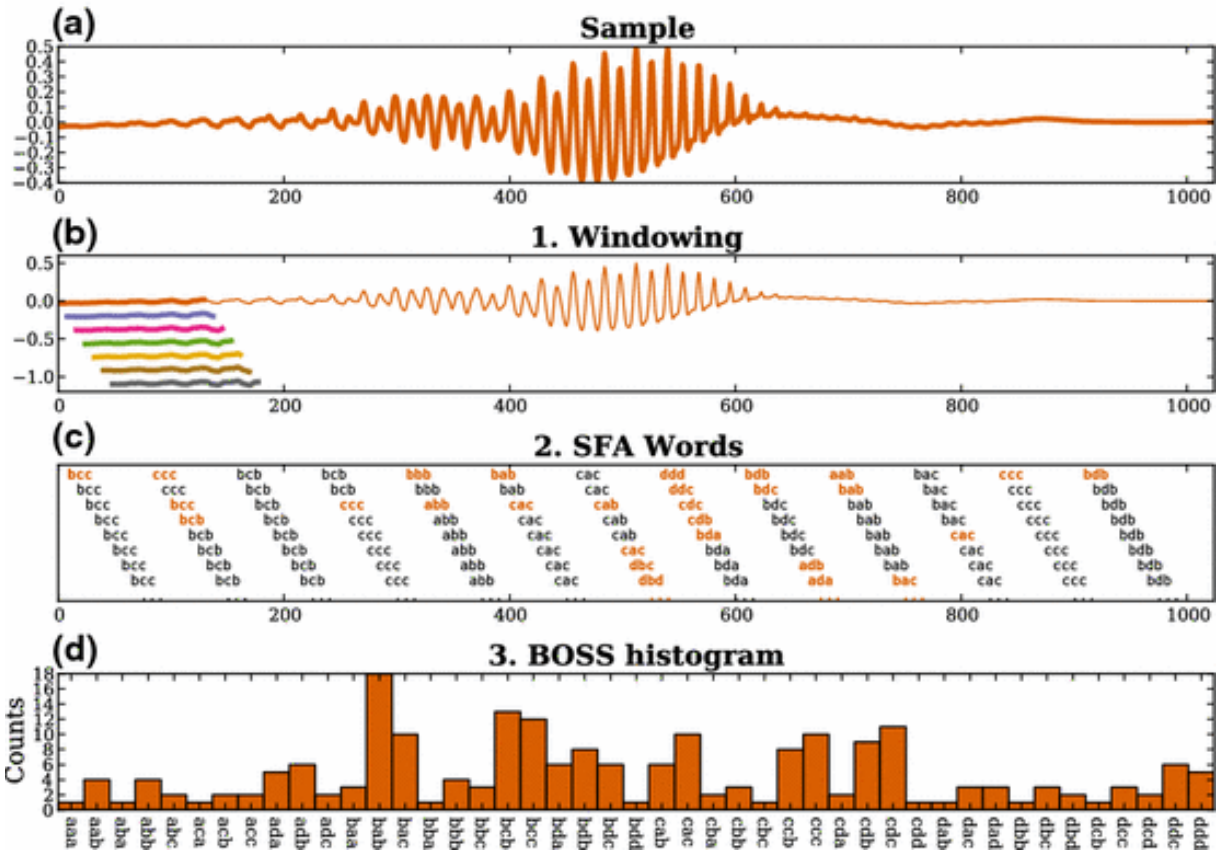

Figure 2.3: Illustration of how BOSS transforms a time series into a histogram of SFA Words. Image from the original paper of [68]. SFA words written in black in (c) are those discarded by the numerosity reduction.

first WEASEL uses z-normalized windows, and rather than using SFA, the Fourier Transform is computed and an ANOVA F-test is performed on the Fourier Coefficients, keeping only those that best separate the classes. Then, again in a supervised manner, the values of these coefficients are discretized into bins, with the boundaries chosen to best separate the classes.

After performing this transformation on multiple window lengths, all words (including bigrams of multiple window lengths) are grouped into a single feature vector, where, once again, a supervised test (Chi-squared) is performed to remove irrelevant features. This feature vector, representing the frequencies of occurrences of each word for each time series, is then given to a logistic regression classifier. This whole process is summarized in Figure 2.4.

Temporal Dictionary Ensemble (TDE)

The Temporal Dictionary Ensemble (TDE) [57] is introduced as an ensemble of dictionary-based classifiers such as BOSS. In addition to the transformation of a time series into a histogram of SFA words, TDE also counts the frequency of bigrams from non-overlapping windows. This process is done for the whole series, but also for increasingly smaller subsequences, with each subsequence size producing a histogram by concatenating the

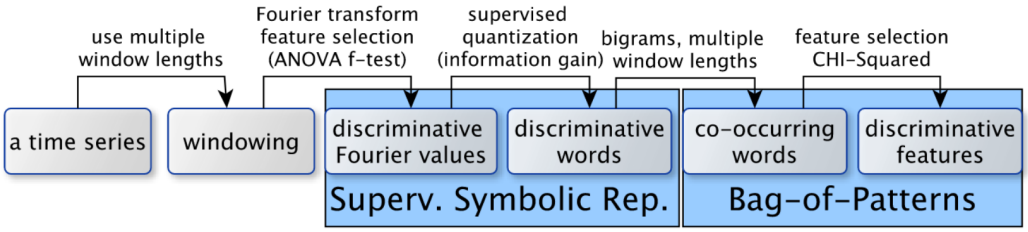

Figure 2.4: The WEASEL data transformation pipeline, from raw time series to the feature vector given to the classifier. Image from the original paper of [66].

histograms of all subsequences of this size.

Each component of the ensemble computes the SFA bounds either in an unsupervised way (i.e. as in BOSS using Multiple Coefficient Binning) or in a supervised way (i.e. as in WEASEL using information gain). The choice to use normalized windows also differs between components. To build the ensemble, a set of candidates component is produced, and then filtered using leave one out cross-validation, with each candidate using 70% of the data with sampling.

2.2.4 Convolutional Kernel algorithms

This family of methods uses a set of convolutional kernels to extract features from the data. Given a kernel and a time series, they look at the resulting convolution of both inputs, and extract features from it, similarly to pooling operations in neural networks. These methods differ from neural network approaches as they do not perform any optimization on any parameters of the kernels, and are rather used as a transformation step before a classifier.

Random Convolutional Kernel Transform (ROCKET)

The Random Convolutional Kernel Transform (ROCKET) [22] randomly initializes a huge number (10.000 by default) of convolutional kernels K, $K = \{l, \{w_1, ..., w_l\}, b, d, p\}$, with l the length of the kernel, w its weights, b the bias, d the dilation and p the padding. If N kernels are generated, ROCKET will output a 1-dimensional array containing $2N$ features, since 2 features are extracted for each convolution of the input by a kernel. More precisely, for a time series $X = \{x_1, ..., x_m\}$ and a kernel $K = \{l, \{w_1, ..., w_l\}, b, d, p\}$, the convolution will output a vector of size $m - ((l-1) * d) + (2 * p)$, with value at position i defined by:

$$c_i = b + \sum_{j=1}^{l} w_j \times X_{i+(j-1)\times d} \tag{2.3}$$

When padding is used, zeros are added at the beginning and at the end of X so that the middle of the kernel is centred on every point. The two features extracted from the convolution between a kernel and a time series are the maximum value and the proportion of positive values (PPV) generated by the convolution. This operation is repeated for each

kernel that was randomly initialized. Figure 2.5 gives a visual representation of the feature extraction process.

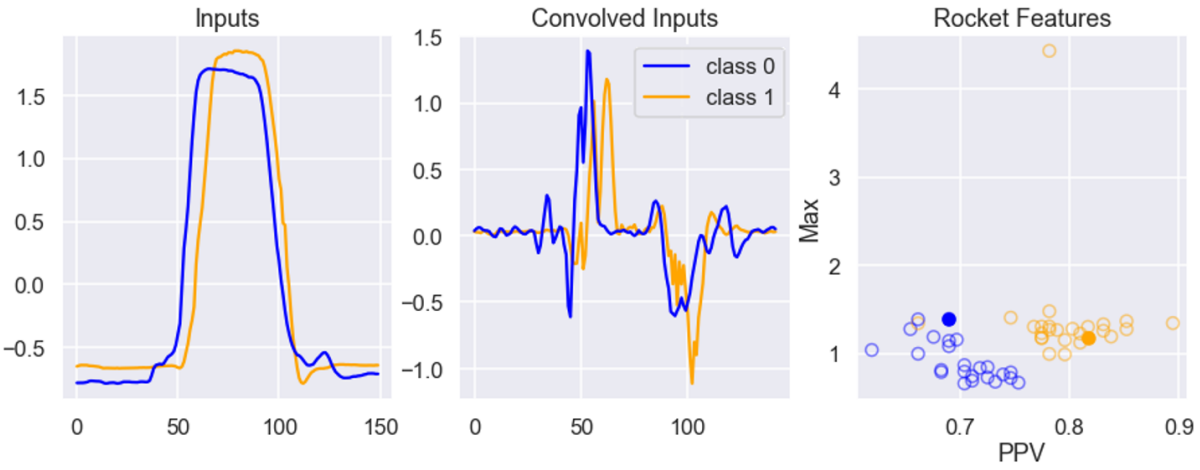

Figure 2.5: The different steps of the ROCKET method: first, transformation of the inputs using convolutional kernels, and then feature extraction from those convolved inputs

Extensions of this method were proposed, such as Mini-ROCKET [23], which, by using only two discrete values for kernel weights and with computational optimizations, reduces the time complexity without a significative loss of accuracy. ARSENAL [58] introduces an ensemble scheme with ROCKET as base components, improving the accuracy of the method. Finally, Multi-ROCKET [75] extracts more features from the distance vector, and uses of the first order difference as an alternative representation of the inputs. It then feeds both the original and the differentiated time series to the model, with each representation having its own set of kernels.

2.2.5 Ensemble methods

Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE)

Following the success of ensemble methods for time series classification, [50] introduced HIVE-COTE, a meta-ensemble method that uses the most successful time series classification algorithms as components of the ensemble. Specifically, it uses the Elastic Ensemble (EE) [47], the Binary Shapelet Transform [11], BOSS [65], Time Series Forest [25] and Random Interval Features [50].

Those classifiers are combined under a hierarchical voting structure, where the class probabilities obtained as output of each classifier are weighted by its performance on the training data (i.e. by the training accuracy). The output of HIVE-COTE is then the class which, when summing the weighted class probabilities of each component, has the highest probability. This process is described in Figure 2.6.

An update to HIVE-COTE, HIVE-COTE 2 [58] has been proposed, where the base components have been updated to the newest best performing classifier for each type of algorithms (i.e. interval-based, dictionary-based, etc.). The voting scheme has also been slightly modified to include a parameter influencing the importance of weights given to

each component, based on their training accuracy obtained from a leave-one-out validation.

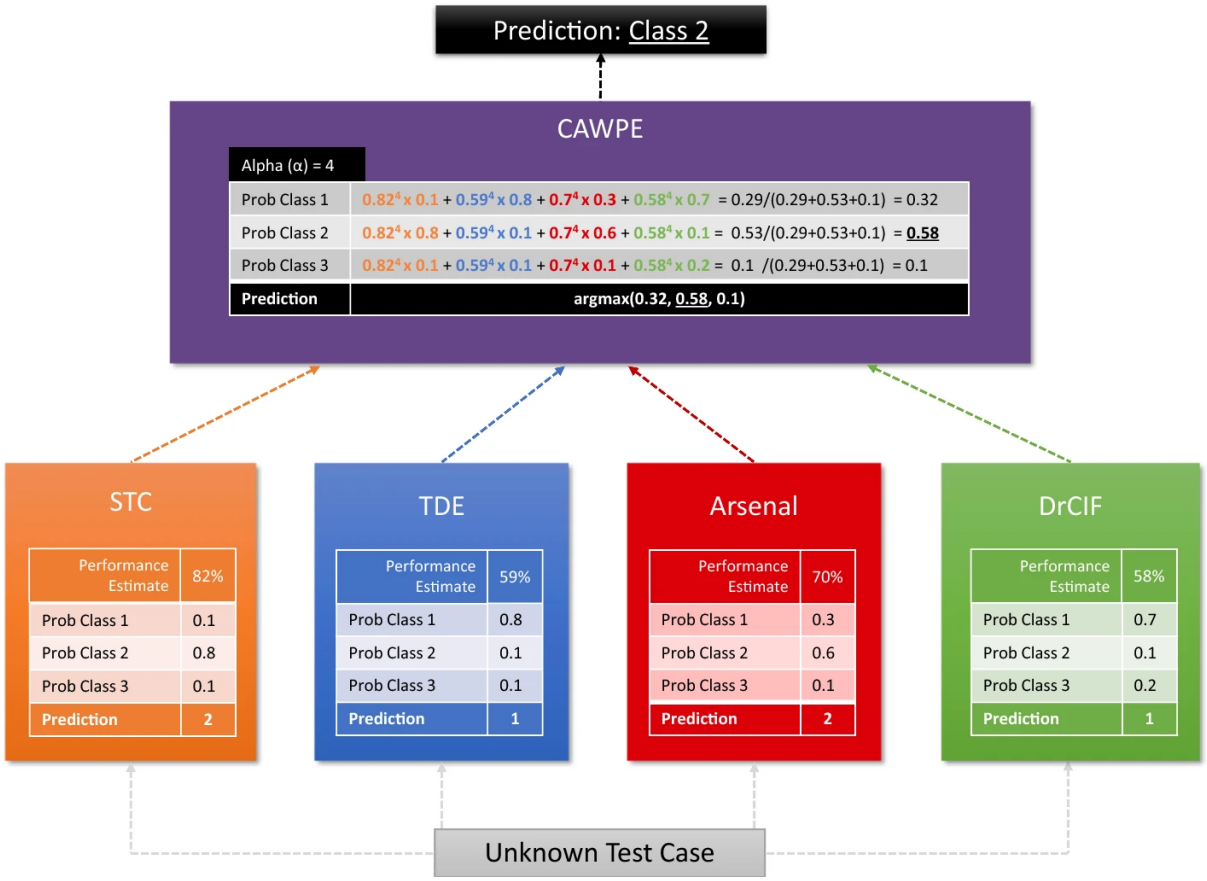

Figure 2.6: How the HIVE-COTE 2 ensemble predict the class of a time series. Image from the original paper of [58]

Time Series Combination of Heterogeneous and Integrated Embedding Forest (TS-CHIEF)

TS-CHIEF [72] is another ensemble, tree based algorithm. The authors define multiple splitting methods at node level, based on popular algorithms used in TSC. To split a node, those different methods are randomly initialized multiple times, and the one that maximizes a splitting criterion is chosen as the splitting method for this node. The algorithms that are re-defined as splitting methods in TS-CHIEF are: Proximity Forest, BOSS and RISE. The methodology is close to the one used in HIVE-COTE, but with fewer splitting methods and random initialization, which presents a huge gain in time complexity without a significative loss in performance.

2.2.6 Deep Learning methods

While any recurrent (e.g. RNN, LSTM, etc.) or convolutional network can be applied to time series, InceptionTime [36] is regarded as the most accurate for time series classification. InceptionTime is an ensemble method that combines five residual networks using

inception modules [74] as base components. Each network is made up of two blocks of three Inception modules, described in Figure 2.7, which maintain residual connections, followed by a global average pooling and softmax layers. Each network in the ensemble is initialized with random weights.

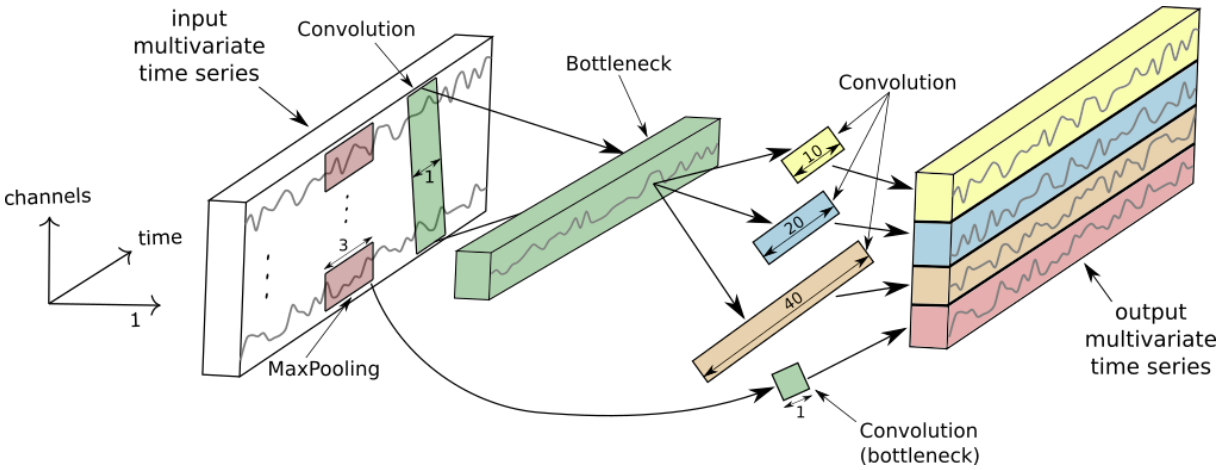

Figure 2.7: Visual description of an Inception Module (with bottleneck layer of size $m = 1$). Image from the original paper of [36]

2.3 Shapelets

In the following, given a time series $X = \{x_1, \ldots, x_m\}$, we use the notation $X^{i,l} = \{x_i, x_{i+1}, \ldots, x_{i+(l-1)}\}$ to denote the subsequence of size l starting at index i in X. We also use $X_j^{i,l}$ to denote a subsequence of size l starting at index i in the time series $X_j \in \mathcal{X} = \{X_1, \ldots, X_n\}$.

2.3.1 Introduction

Shapelets [86] were originally defined as time series subsequences representative of class membership, and they can be represented as a time series $S = \{s_1, ..., s_l\}$ with l the length parameter of the shapelet, and $l << m$. Given a shapelet S and a time series X, shapelet algorithms compute a distance vector $V_{S,X} = \{v_1, ..., v_{m-(l-1)}\}$ using a sliding window approach, where each subsequence of size l in X is compared to S:

$$v_i = dist(S, X^{i,l}) \tag{2.4}$$

with d a distance function. The original formulation uses the Euclidean distance as a distance function, giving the following formula for any v_i:

$$v_i = dist(S, X^{i,l}) = \sqrt{\sum_{j=1}^{l} (x_{i+(j-1)} - s_j)^2} \tag{2.5}$$

The minimum value of $V_{S,X}$ is then extracted as a feature, which can be interpreted as an indicator of the presence of the pattern represented by S in X. The closer $\min V_{S,X}$ is to 0, the higher the similarity between S and a subsequence of X. This formulation allows shapelets to be translation invariant, as $\min V_{S,X}$ does not depend on the localization of the pattern matched by S in X.

Another interesting property that can be added to this formulation is scale invariance. The goal of scale invariance is to ignore the scale of both S and $X^{i,l}$ when computing the distance, which is equivalent to satisfying the following property :

$$dist(S, X^{i,l}) = dist(\alpha S, \beta X^{i,l}) \mid \forall\ (\alpha, \beta) \in \mathbb{R}^* \tag{2.6}$$

To achieve this property, the z-normalized Euclidean distance is used as the distance function. Given μ_S and σ_S, the mean and the standard deviation of S (respectively for $X^{i,l}$), the z-normalized Euclidean distance is computed as :

$$v_i = dist(\hat{S}, \hat{X}^{i,l}) = \sqrt{\sum_{j=1}^{l} (\hat{x}_j^{i,l} - \hat{s}_j)^2} \tag{2.7}$$

$$\hat{S} = \frac{S - \mu_S}{\sigma_S},\ \hat{X}^{i,l} = \frac{X^{i,l} - \mu_{X^{i,l}}}{\sigma_{X^{i,l}}} \tag{2.8}$$

In the original method of [86], shapelets were used to build a shapelet tree. In the brute force approach, given $\mathcal{X}$ the samples at the root of a decision tree and L a set of possible lengths for the shapelets, the set of all possible subsequences $\{X_j^{i,l} \mid \forall j \in \mathcal{X},\ \forall\ i \in [1, (m - (l-1))],\ \forall\ l \in L\}$ is extracted as the shapelet candidates pool.

At the root node, all candidates $X_j^{i,l}$ are evaluated. Using all the series in $\mathcal{X}$, the minimum values are extracted from each distance vector and, using a dichotomy, each possible threshold value is tested to find the one that maximize the information gain. The pair of shapelet and threshold that maximizes the information gain is selected as a splitting function for the node. This process is repeated using the input data of each leaf instead of $\mathcal{X}$, until all leaves are pure. Some speed-up techniques, which we detail in Section 2.3.6, were also introduced to reduce the time complexity by multiple orders of magnitude, such as entropy pruning, early distance abandon, or input sampling.

2.3.2 Shapelet Transform

This use of shapelets was replaced a few years later by the Shapelet Transform [48], which, until now, remains the standard framework for shapelet based algorithms. The authors of [48] demonstrated that shapelets can be used as a transformation step before using non-temporal classifiers (e.g. decision trees) without loss of accuracy, instead of embedding them in a classifier as [86].

Given a set of time series $\mathcal{X} = \{X_1, \ldots, X_n\}$ and a set of shapelets $\mathcal{S} = \{S_1, \ldots, S_k\}$, the transformation outputs a matrix M of distance vector of size (n, k) such as:

$$M = \begin{bmatrix} V_{S_1,X_1} & \ldots & V_{S_k,X_1} \\ \vdots & \ddots & \vdots \\ V_{S_1,X_n} & \ldots & V_{S_k,X_n} \end{bmatrix} \tag{2.9}$$

Then, feature are extracted from each distance vector V_{S_i,X_j} (only the minimum in case of [48]), giving a feature matrix useable by any non-temporal classifier. The enumeration of all possible candidates, along with existing and new speed-up techniques such as shapelet self similarity (do not sample shapelets with overlapping indexes in the input) or time contracting, is used to generate shapelets. We detail those speed-up techniques in Section 2.3.6.

It is worth mentioning that applications to the unsupervised context, for example in clustering tasks, have also been explored in the literature, starting with u-shapelets [87], but are outside the scope of this manuscript.

2.3.3 Distance function

We can distinguish between two families of distance functions: the non-elastic distances such as the Euclidean distance, and the elastic distances such as dynamic time warping (DTW) [64], which allow for some realignment to compensate for time shifts between series. Few papers in the literature explore alternatives to the Euclidean and z-normalized Euclidean distances.

To the best of our knowledge, there has been no experiments on non-elastic distances other than the Euclidean distance, with the use of normalization to induce scale invariance. A few methods use DTW in place of the Euclidean distance such as [71], which follows the methodology of Learning Shapelets [30], but replace the Euclidean distance with DTW. This induced minor gains on a few datasets, but it is hard to say from their experiments if this change is worth the additional complexity.

We will nevertheless experiment three different non-elastic distance functions, we do not expect significative differences in accuracy between them, but are more interested in their impact on the run-time of the shapelet transform algorithm. These distances are :

- The Euclidean distance, as $d(S, X^{i,l}) = \sqrt{\sum_{j=1}^{l}(x_{i+j-1} - s_j)^2}$. This is the default option for shapelet methods.
- The squared Euclidean distance, as $d(S, X^{i,l}) = \sum_{j=1}^{l}(x_{i+j-1} - s_j)^2$, which will further increase high pointwise differences (> 1) and reduce small pointwise differences (< 1). This can have an influence depending on the type of classifier used.
- The Manhattan distance, or L1 norm, as $d(S, X^{i,l}) = \sum_{j=1}^{l}|x_{i+j-1} - s_j|$, which requires fewer operations and should thus be faster than the Euclidean distance.

2.3.4 Shapelet features

Since the publication of Localized Random Shapelet (LRS) [32], which showed the benefit of extracting argmin $V_{S,X}$ to discriminate time series based on the location of the minimum between S and X, it has been included in most recent approaches.

Outside the extraction of min $V_{S,X}$ and argmin $V_{S,X}$, to the best of our knowledge, there have been no study of alternative feature sets, similarly to the work on the canonical time-series characteristics (Catch22) [52]. One of the reasons behind this may be that most heuristics or speed-up techniques proposed in the literature implicitly use has assumption that only the minimum value of the distance vector is needed.

2.3.5 Shapelet generation

One of the most studied part of shapelet based algorithms is the generation of the shapelets. Except for a few papers, most of the literature share the common objective of building heuristics to be close to the classification accuracy of the brute force method, which enumerates and assesses the quality of all possible candidates. Different kinds of heuristics have been developed to extract a set of shapelets for a given dataset, we regroup those approaches into three families that we detail below.

Random extraction

One of the simplest shapelet extraction scheme is the random extraction. Given a time series dataset $\mathcal{X} = \{X_1, \ldots, X_n\}$ with $X_i = \{x_1, ..., x_m\}$ a time series, and l the length of the shapelet, a total of $n \times (m - (l - 1))$ shapelets can be sampled from $\mathcal{X}$.

Assuming a uniform probability distribution, the quality of shapelets extracted from this space cannot be guaranteed, and hence a lot of shapelets have to be extracted to compensate. This was the approach taken by Ultra-Fast Shapelets [82], which randomly samples a percentage of all possible candidates. The Localized Random Shapelet [32] also uses this approach to sample candidates.

Shapelet optimization

This type of approach allows generating shapelets that do not exist in the input space, as their values are iteratively modified based on an optimization scheme.

Learning Time Series Shapelets (LTS) [30] was the first method to approach the problem of finding the best shapelets as an optimization problem. The initialization phase of LTS uses a clustering method to sample the initial shapelet candidates. Given the input time series and a shapelet length l, a K-means clustering is performed on all subsequences of size l. Then, the centroids of each cluster are extracted as shapelet candidates, and affected a random weight.

The first step of the method is to extract the soft minimum distance between the shapelets and the input time series. These distances are then multiplied by the weight associated with the shapelets they come from. A logistic regression classifier is then used to predict the class of each time series, and regularized logistic loss is used to estimate the error. Then, using a gradient descent optimization, it will change the values of the shapelets as well as their weight. The method is then repeated until convergence.

Similarly, GENDIS [79] uses an optimization scheme based on a genetic algorithm. The mutation and crossover operators are defined to either modify parts of the shapelets, or to modify the set of candidates, notably by randomly sampling new ones from the dataset. The initial population is created with a clustering step on random subsets of the possible candidates in the input space, with the centroids selected as the initial candidates.

The issue with such methods is that, as they produce out-of-sample shapelets, the interpretability is often loss as the resulting shapelets do not resemble the input data at all. A correction for this issue was proposed by [81], by using an adversarial approach to regularize shapelets to be similar to subsequences found in the data.

Heuristics for candidates enumeration

Outside the random and the optimization schemes, multiple heuristics were proposed to speed up the enumeration and evaluation of the best shapelets.

A first family of heuristics uses approximation techniques such as SAX [46] to reduce the search space by producing an approximation of the inputs (i.e. reducing the length of the time series and/or discretizing the values). This is the case of Fast Shapelets [60], which then establishes a quality measure for candidates based on their similarity with each other in the approximated space. If a candidate often appears in a class but not in the others, it will be considered for evaluation in the input space using information gain. The ELIS++ [88] method also uses a similar approach, but rely on the TF-IDF score to formalize the quality measure between discretized candidates. They also introduce multiple pruning techniques along with data augmentation and a Bayesian optimization of the parameters of their method to try to find the best set of candidates in approximated inputs.

Another family of heuristics takes an iterative approach to the problem, and reduces the search space given the shapelets selected in previous iterations. An example of such heuristics, is the Tabu search described in [10]. The Tabu search exploits the fact that candidates, when sampled in the neighbourhood of shapelets found in previous iterations, are likely to be redundant (e.g a shapelet of size l sampled at x_{i+1} will be very similar to the shapelet sampled at x_i). Given this observation, the method constraints a random candidates search by not considering the neighbourhood of points used to sample a shapelet in previous iterations.

It is also possible to "contract" a method by limiting its run-time. Such contracted methods will search for the best shapelets until time is out, and return the best-so-far found shapelets. An example of such a classifier is given in [12], using a random search. A skipping search for a time contracted method was also defined in [10], where given the size of the input data and the time constraint, an amount of neighbouring candidates will be skipped. The goal is to maximize the diversity of shapelets evaluated in the given time contract, based on the assumption of similarity between neighbouring candidates.

Lastly, subsampling the dataset, by considering only $r\%$ of the samples in the training data, is also another way of reducing the size of the search space to $(\frac{r}{100} \times n) \times (m-(l-1))$. This assumes that sufficient discriminative information is still present in the fraction of the dataset considered.

2.3.6 Speed up techniques

A number of techniques designed to speed up the distance computations or the shapelet selection have also been proposed in the original approach of [86]. Most are targeted at finding a lower-bound on either the distance to the shapelet or its estimated quality, to be able to perform early abandon or pruning. The general idea is that, if a measure goes above this lower bound, it will not be helpful for the problem, hence its evaluation can be stopped even if it is not finished.

An example of this approach is the distance early abandon, displayed on Figure 2.8. Given a shapelet $S = \{s_1, \ldots, s_l\}$ and a time series $X = \{x_1, \ldots, x_m\}$, S is slided across X,

computing the Euclidean distance between S and all $m-(l-1)$ subsequences of size l in X. Let min_i the best-so-far minimum value found before step i (i.e $\min_{1 \leq k < i} dist(S, X^{k,l})$), if at any step of the computation of $\sum_{j=1}^{l}(x_{i+(j-1)} - s_j)^2$ the current squared sum is superior or equal to min_i, we can ignore the remaining part of the sum, as it will not be inferior to min_i. The square root can then be applied at the end of the computations if we want to use the Euclidean distance rather than the squared Euclidean distance.

A change to the evaluation order of the shapelet was proposed by [10] to further increase the number of ignored operations with early abandon. The idea was that instead of sliding a shapelet S on X from 1 to $m-(l-1)$, we could start at the point i where s was sampled, and evaluate $i+1$, then $i-1$, then $i+2$, etc. They show that this increases the average number of operations ignored, as the global minimum has a better chance of being found earlier by evaluating candidates that are increasingly further from each other. It is also possible to use a random order.

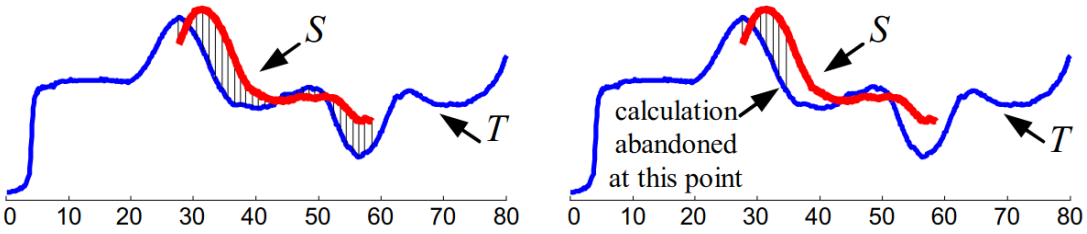

Figure 2.8: Comparison of full distance computation (left) against of early distance abandon (right), with S a shapelet and T a time series. Image from the original paper of [86]

One issue with those lower bound techniques, is that they do not consider their impact on material acceleration techniques, notably on vectorization (i.e. Single Instruction Multiple Data). For example, the cost of checking the value of the current sum against the best-so-far minimum might not be worth, compared to a version using vectorization that would compute the full distance. We evaluate this assumption during our experiments in Section 3.9.7.

2.3.7 Invariance properties and robustness

Shapelets are known to be translation invariant, due to the sliding of the shapelet on the series when computing the distance vector and the extraction of the minimum as a feature. This means that $\min V_{S,X}$, which represents the best match between a series X and a shapelet S, is not influenced by the position of this best match in $\min V_{S,X}$. Shapelets can also be scale invariant, when a z-normalized distance function is used. This makes the scale of the shapelet and of the subsequences being compared irrelevant when computing the distance vector, as described by Equation 2.6. Hence, the minimum value extracted from the distance vector is also scale invariant.

An invariance is a property that guarantee that a mathematical object is not modified after some type operations or transformations are applied to it. In the context of shapelets, such property target either the whole distance vector, such as scale invariance, or the features extracted from it, such as the translation invariance for $\min V_{S,X}$. We also consider additions which induce robustness to a phenomenon (e.g. noise, time warping), which,

while not guaranteeing an invariance, allow negating the effect of this phenomenon on the distance vector or the features extracted by the shapelets.

In the literature, other properties have been defined, mostly between two time series. We introduce those properties for the case of two time series X_1 and X_2, and show in our contributions how we can adapt those properties to the case of shapelets.

Phase invariance: When recording a cyclic phenomenon (e.g. heartbeats, a sinusoidal function, etc.), the moment when the recording starts, known as the initial phase of the signal, can cause massive point-wise difference between time series. For example, if we were to compare two similar signals with a different initial phase, the Euclidean distance would give misleading results. To the best of our knowledge, the only way to guarantee phase invariance between two series is to test all possible alignments of X_1 while maintaining X_2 fixed, and select the smallest resulting distance.

Robustness to time warping: In some cases, the data can be misaligned due to natural shifts in the recorded phenomenon. Consider the case where we record the hand movement of a person writing the same word multiple times. Even if the resulting time series represent the same word, the hand movement will differ slightly from each recording, as well as the time spent on each part of the word. In this case, the Euclidean distance can give misleading results as the data is not aligned. In such cases, it is common to use DTW to compare time-warped series.

In the case of Shapelets, DTW has been used as an alternative to the Euclidean distance [71]. One downside of their formulation is that they do not change the sliding window scheme used to compute the distance vectors. This does not allow DTW to find the path between the shapelet S of size l and a theoretical "optimal" subsequence $X^{i,l'}$ if $l' > l$, as we only use $DTW(S, X^{i,l})$. As the experiments of methods using DTW for shapelets do not seem to justify the additional complexity by significant improvements of the results, we do not pursue warping invariance for shapelets in this thesis. We think a good first step towards this goal would be to modify the sliding window scheme, to allow larger patterns to be matched as well.

Robustness to noise: When the time series are subject to noise, a simple Euclidean distance might not be able to find similarities between the underlying signals. While noise can affect the whole time series due to the recording process (e.g. background sounds from an audio recording), its amplitude can vary in the signal itself. While not strictly producing invariance, techniques based on the Fourier Transform can be used to try to filter-out the noise, or to design methods that are robust to noisy data.

Robustness to complexity: The intuition behind the idea of a complex time series can first be explained as a time series with a lot of peaks and/or higher variability between successive data points. This variability can cause errors when trying to classify (e.g. with a 1-NN with Euclidean distance) time series where one class is "complex" and the other "simple", as a complex time series might be closer to a simple one under Euclidean distance, as illustrated by Figure 2.9.

A correction factor for any distance function was proposed by [8], which takes into account the complexity of the time series being compared. Given d a distance function,

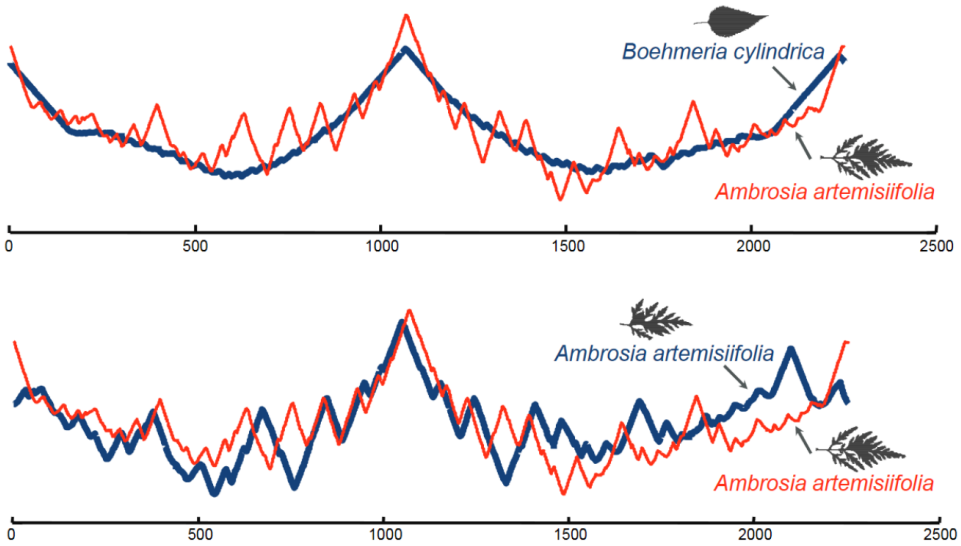

Figure 2.9: An example case where, at the top, a simple-shaped (blue) and a complex-shaped (red) time series result in a smaller Euclidean distance than, at the bottom, two complex time series (of the same class). Image from the original paper of [8]

the complexity invariant distance (CID) can be computed as :

$$CID(X_1, X_2) = dist(X_1, X_2) \times CF(X_1, X_2) \tag{2.10}$$

with CF the complexity correction factor computed as :

$$CF(X_1, X_2) = \frac{max(CE(X_1), CE(X_2))}{min(CE(X_1), CE(X_2))}, \; CE(X) = \sqrt{\sum_{i=1}^{n-1}(x_i - x_{i+1})^2} \tag{2.11}$$

The authors of [8] then empirically demonstrate that CID improves the accuracy of a 1-NN classifier, with both Euclidean distance and DTW, while obeying the p-relaxed triangular inequality and still being subject to lower-bounding techniques for early distance abandon.

2.4 State-of-the-art evaluation

Given the algorithms presented in Section 2.2, we consider the Shapelet Transform Classifier (STC) [12] as the state of the art for shapelet algorithm. It uses a time contract to find the best set of candidate shapelets given their information gain and build a Rotation Forest [4] classifier, using the minimum distance between the shapelets and the series as features.

Figure 2.10 summarize the latest published accuracy results from [5] on the UCR archive [19] with a critical difference diagram. This archive contains 112 datasets for univariate time series classification, from a variety of domains and applications. All datasets are defined with a base training and testing set, which may reflect some particularity of the application (e.g. a small training set and a bigger testing set). The standard evaluation protocol of the archive is to use resamples, in order to have the same class repartition than in original training and testing sets. The accuracy results are obtained with 30

resamples. We see that shapelet methods have fallen behind in terms of classification accuracy compared to the rest of the state of the art. In the next section, we present our contributions, and show in our experimental section how they improve on the state of the art for shapelet algorithms.

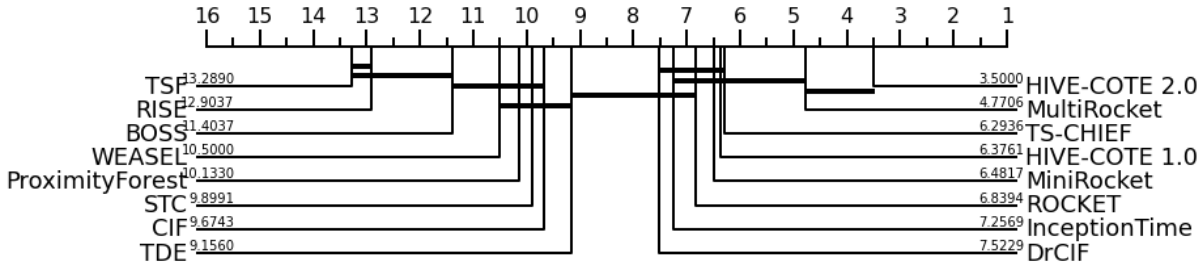

Figure 2.10: Mean accuracy rank (numbers next to the name of the algorithms) of state-of-the-art classifiers for univariate time series classification, evaluated on 30 resamples on the univariate datasets of the UCR archive

Chapter 3

Contributions to Time Series Classification

Résumé du chapitre en français
Dans ce chapitre, nous présentons nos contributions au domaine de la classification des séries temporelles. Dans un premier temps, nous proposons une nouvelle définition des shapelets, intégrant la notion de dilatation et un nouveau descripteur, et présentons un algorithme aléatoire de transformation par shapelets utilisant ces nouveaux composants. Nous présentons ensuite des améliorations pouvant être ajoutées à cet algorithme, comme la réduction de l'espace de recherche, et des modifications sur la méthode de calcul de la distance entre une shapelet et une série. Nous définissons ensuite une version de cet algorithme adaptée aux séries multivariées et de longueur différente, ainsi que les optimisations apportées à l'algorithme, notamment pour permettre de paralléliser l'initialisation des shapelets et les calculs de distance. Nous menons ensuite plusieurs expériences pour étudier l'impact de chacune de nos contributions sur notre algorithme, à la fois en termes de précision et de complexité temporelle. Pour conclure, nous comparons notre algorithme aux autres méthodes de la littérature sur deux archives de données couramment utilisées pour comparer les méthodes de classification des séries temporelles univariées et multivariées.

Chapter summary
In this chapter, we present our contributions to the field of time series classification. First, we define new components to add to the existing definition of shapelets, and present a new shapelet transform algorithm using these new components. We then present other modifications that can be added to this algorithm, such as reducing the number of candidate shapelets and changes to the computation of the distance between a shapelet and a time series. We then define a version of this algorithm adapted to multivariate and variable length time series, as well as optimizations made to the algorithm, notably to allow to parallelize the initialization of the shapelets and the computation of distances. We then conduct several experiments to study the impact of each contribution on our algorithm, both in terms of accuracy and time complexity. To conclude, we compare our algorithm to the other methods of the literature on two data archives, commonly used to evaluate univariate and multivariate time series classification methods.

3.1 Introduction

Before presenting our contributions to time series classification, let us remind the context on which we build upon, which is shapelet-based classification. A shapelet S is defined as a vector $S = \{s_1, \ldots, s_l\}$ with l its length. Given a time series $X = \{x_1, \ldots, x_m\}$, we slide S on X and compute the distance between S and a subsequence $X^{i,l} = \{x_i, \ldots, x_{i+(l-1)}\} \in X$ of size l, for all steps i of the sliding window. This operation generates a distance vector $V_{S,X}$ from which two features are extracted, $\min V_{S,X}$ and $\operatorname{argmin} V_{S,X}$, giving respectively how close the best match between S and X is, and where this best match is located.

More formally, we have $V_{S,X} = \{v_1, \ldots, v_{m-(l-1)}\}$ the distance vector resulting from the computation of the sliding distance between S and X. A point of this distance vector is computed with the Euclidean distance as:

$$v_i = d(S, X^{i,l}) = \sqrt{\sum_{j=1}^{l} (x_{i+(j-1)} - s_j)^2} \tag{3.1}$$

To build a time classification model with shapelets, we first have to generate a set of shapelets $\mathcal{S}$, using one of the methods we described in Section 2.3.5. This set of shapelets is then applied to the set of time series $\mathcal{X}$ on which we want to learn a classifier. If we extract $\min V_{S_i,X_j}$ and $\operatorname{argmin} V_{S_i,X_j}$, $\forall\ X_j \in \mathcal{X}, \forall\ S_i \in \mathcal{S}$, we obtain a feature matrix M of size $(||\mathcal{X}||, 2 \times ||\mathcal{S}||)$.

This feature matrix M can then be used along with the classes of the time series to learn a tabular classifier such as a decision tree. To predict the class of new samples, $\mathcal{S}$ is applied again on the unseen series to build another feature matrix, which is fed to the classifier to predict the classes of the time series.

3.2 Dilated Shapelets

Dilation is often used in the context of kernel operations, and most famously in convolutional kernels. Let us first recall that, given a kernel $K = \{k_1, \ldots, k_l\}$ and a time series $X = \{x_1, \ldots, x_m\}$, the resulting convolution $C = \{c_1, \ldots, c_{m+l-1}\}$ is computed as described by Figure 3.1.

The time series is usually padded with zeros at the beginning and the end of the convolution, but it is also possible to start sliding operation with k_1 aligned on x_1. When a dilation d is used, the kernel is "stretched", such as the point x_b to be multiplied by the coefficient k_{i+1} is spaced by a value of d compared to the point x_a (i.e. $b = a + d$) used for k_i. Figure 3.2 describes this process with $d = 2$. Our first contribution is to apply this idea of dilation to the distance vector computation of shapelets.

3.2.1 Definition

We define a dilated shapelet S as $S = \{\{s_1, \ldots, s_l\}, d\}$ with l the length parameter and d the dilation parameter. The dilation parameter is used in the distance function, with the shapelet being compared to a dilated subsequence of the input time series. More formally, consider a time series $X = \{x_1, \ldots, x_m\}$ and a dilated shapelet S, we now define

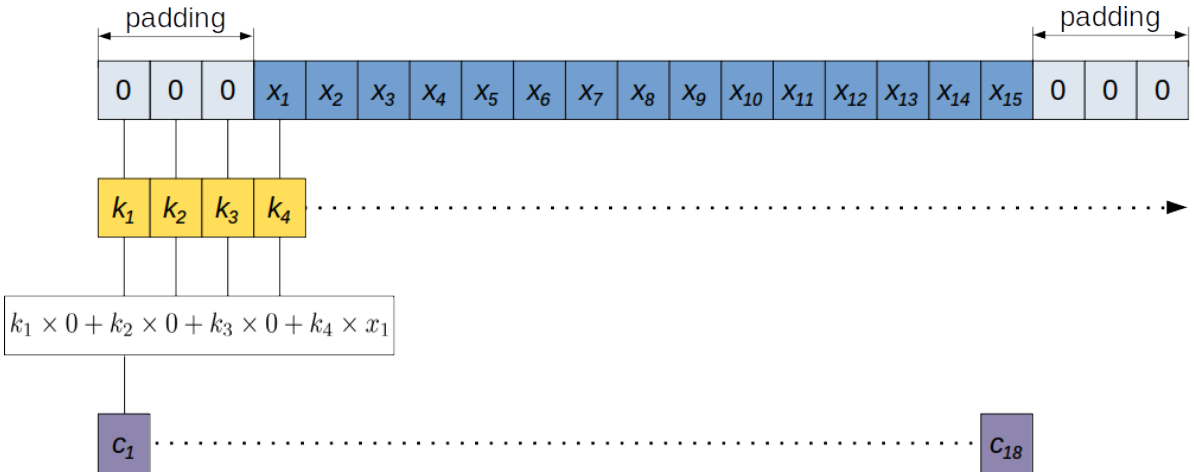

Figure 3.1: Illustration of the convolution between a time series (in blue) and a kernel (in yellow)

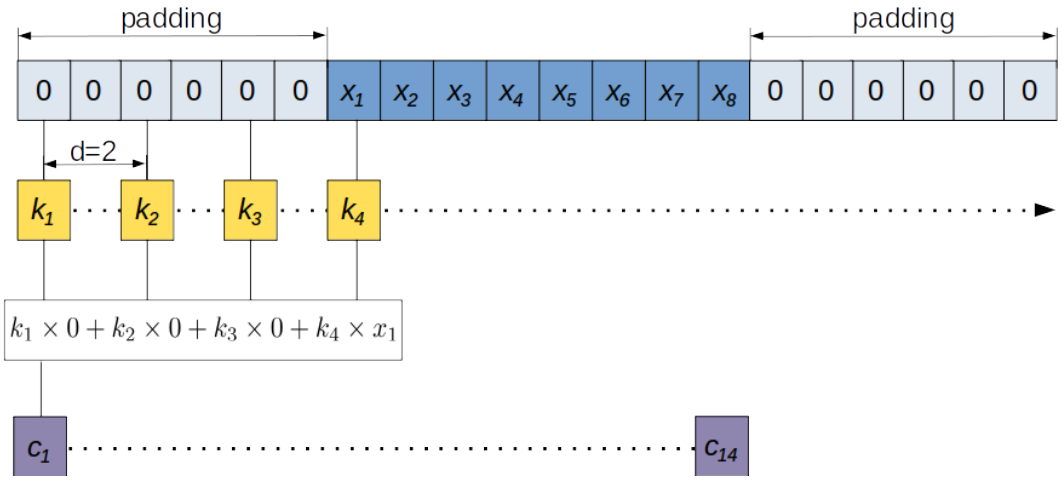

Figure 3.2: Illustration of a dilated convolution between a time series (in blue) and a kernel (in yellow), with $d = 2$

the distance vector between S and X as $V_{S,X} = \{v_1, \ldots, v_{m-(l-1)\times d}\}$, where v_i is equal to:

$$v_i = \sqrt{\sum_{j=1}^{l} (x_{i+(j-1)\times d} - s_j)^2} \tag{3.2}$$

Note that the formulation with dilation is equivalent to the original shapelet formulation when $d = 1$. From now on we refer to a subsequence of size l starting at index i with a dilation d as $X^{i,l,d} = \{x_i, x_{i+d}, \ldots, x_{i+(l-1)\times d}\}$. We can also use a z-normalized distance with dilated shapelets, similarly to the definition given in Equation 2.6, as :

$$v_i = dist(\hat{S}, \hat{X}^{i,l,d}) = \sqrt{\sum_{j=1}^{l} (\hat{x}_j^{i,l,d} - \hat{s}_j)^2} \tag{3.3}$$

$$\hat{S} = \frac{S - \mu_S}{\sigma_S}, \quad \hat{X}^{i,l,d} = \frac{X^{i,l,d} - \mu_{X^{i,l,d}}}{\sigma_{X^{i,l,d}}} \tag{3.4}$$

3.2.2 Why use dilated shapelets?

The interest of using dilation in shapelets is to make them non-contiguous subsequences. It allows a shapelet with $d > 1$ to match non-contiguous discriminative patterns. It is also possible to partially match contiguous discriminative patterns, by only considering key points of the patterns. Focusing only on key points can make the pattern matching less sensitive to noise during the distance vector computation. Contiguous patterns can still be matched when $d = 1$.

Let us illustrate the influence of dilation with an example. Consider two samples from two different classes in the PigCVP dataset, which records central venous pressure (CVP) measurements of pigs under different conditions, and a random shapelet extracted from the blue time series, shown by Figure 3.3.

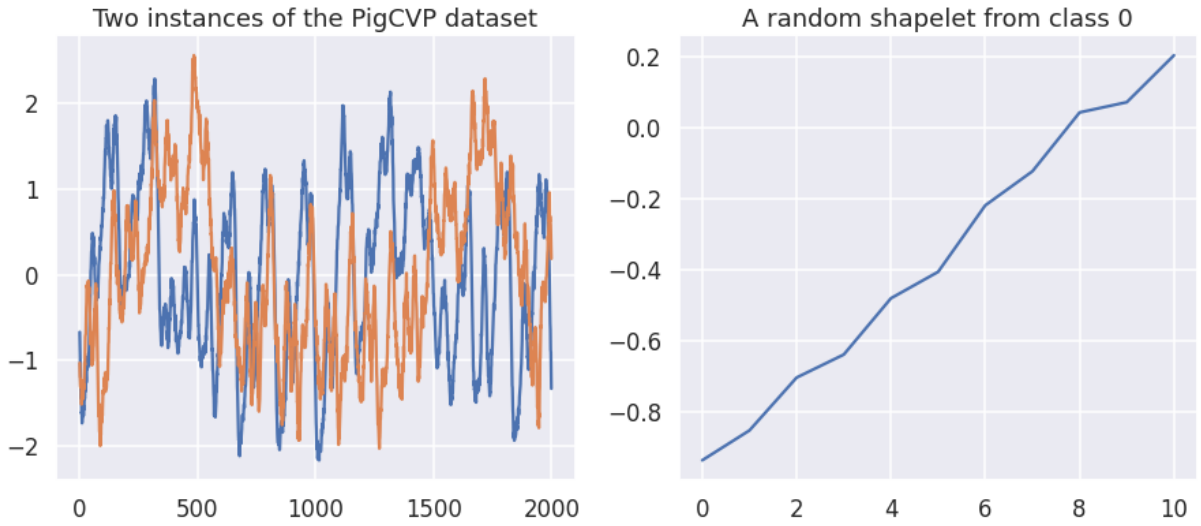

Figure 3.3: (a) Two time series of different classes from the PigCVP dataset, and (b) a random shapelet of length 11 extracted from the blue time series. The x-axis is the indexes, and the y-axis the values of the time series and the shapelet.

The resulting distance vectors, using a normalized Euclidean distance between the two samples and the shapelet, and computed with different values of dilation, are shown by Figure 3.4. As the measure of CVP is directly influenced by the heart rate, it is possible that some values of dilations allow a shapelet to "synchronize" with the heart frequency, such that x_i and x_{i+d} are spaced by one period (i.e. the time between two heart contractions). Given the shapelet of Figure 3.3, if such a dilation was applied, we would have a low distance if measurements spaced by the period between two heart beats successively increase for a number of heartbeats equal to the shapelet length.

Considering that the PigCVP dataset is collected at a frequency of 250Hz, giving one measurement every $\frac{1}{250}$ second, if we use the expected heart rate for young pigs (the age of pigs is not specified in the original study), which is of 90 contractions per minutes, or $\frac{90}{60}$ contractions per second, this "synchronization" would happen around $d = (90/60)/(1/250) = 166.66$, rounded to 167, as displayed in the last plot of Figure 3.4. If we consider the minimum extracted from these distance vectors, we see that using $d > 1$ result in more discriminative feature than with $d = 1$.

Figure 3.4: Resulting distance vector between the samples and the shapelet of Figure 3.3 for different values of dilation.

Existing state-of-the-art approaches for time series classification, such as HIVE-COTE, have shown the importance of diversifying the representation and methods to extract features. The use of dilation for shapelets could then be considered, in addition to the ability to match non-contiguous patterns, as a parameter which favour the diversity of input representations.

3.3 Shapelet Occurrence feature

In the initial formulation of [86], given a shapelet S and a time series X, after computing the distance vector $V_{S,X}$, only $\min V_{S,X}$ was extracted as a feature. Later [32] added $\operatorname{argmin} V_{S,X}$ as feature during the extraction, notably to be able to discriminate cases where only the location of the best match of a pattern is discriminative between classes.

Using only these two features still leave some type of possibly discriminative information unused, such as the number of occurrences of a pattern, which cannot be captured by statistics that give information about a single point of $V_{S,X}$ as $\min V_{S,X}$ and $\operatorname{argmin} V_{S,X}$ do. We propose a new feature computed from the distance vectors generated by shapelets. This could also be compared to the use of a global pooling operator in neural networks.

3.3.1 Definition

The definition of the Shapelet Occurrence (SO) feature requires the introduction of a second parameter in the definition of a shapelet S as $S = \{\{s_1, \ldots, s_l\}, d, \lambda\}$, with λ a threshold allowing us to compute SO, using the identity function I with $I(True) = 1$ and $I(False) = 0$, as follows:

$$SO(S, X) = \sum_{i}^{m-(l-1)\times d} I\left(dist(S, X^{i,l,d}) \leq \lambda\right) \tag{3.5}$$

Given the threshold λ, the Shapelet Occurrence feature counts how many times the shapelet was at least λ-close to the time series. Algorithm 1 gives the pseudocode corresponding to the distance vector computation with dilated shapelets and the feature extraction. It takes as input a time series X and a shapelet S with its parameters, l, d, λ along with a Boolean to indicate whether the distance computation should be z-normalized.

Algorithm 1 extract_features(X, S, l, d, normalize, λ)

$min = \infty$, $argmin = \infty$, $SO = 0$
for i in $[1, m - (l-1) \times d]$ **do**
  $X^{i,l,d} \leftarrow \{x_i, x_{i+d}, \ldots, x_{i+(l-1)\times d}\}$
  if normalize **then**
    $X^{i,l,d} \leftarrow$ z-normalize($X^{i,l,d}$) ▷ S is z-normalized at extraction in this case
  end if
  $v_i \leftarrow \sqrt{\sum_{j=1}^{l}(x_j^{i,l,d} - s_j)^2}$ ▷ $x_j \in X^{i,l,d}$
  if $v_i < min$ **then**
    $min \leftarrow v_i$, $argmin \leftarrow i$
  end if
  if $v_i < \lambda$ **then**
    $SO \leftarrow SO + 1$
  end if
end for
return $min, argmin, SO$

3.3.2 Why use Shapelet Occurrence?

If we consider a shapelet S and two time series X_1 and X_2, we can imagine multiple ways of discriminating X_1 and X_2 using S.

1. S can be present in both series, but not at the same scale. In this case, a z-normalized distance would not be able to discriminate the series. Deciding whether scaling is important or not is highly dependent on the application, but without prior knowledge, one cannot know which to choose.

2. S can be present in X_1 but not in X_2. This is captured by $min\ V_{S,X_i}$, with smaller distances indicating better matches between the shapelet and the series.

3. S can be present in both series, but not at the same place. This is captured by the *argmin* feature introduced in the Localized Random Shapelet [32] algorithm.

4. S can be present in both series, but occurs a different number of times in X_1 compared to X_2. This is captured by our feature, Shapelet Occurrence (SO).

Those points are illustrated in Figure 3.5 with synthetic examples and their distance vector computed using the shapelet S, using a z-normalized Euclidean distance. Each column of Figure 3.5 illustrates the corresponding points of the above list.

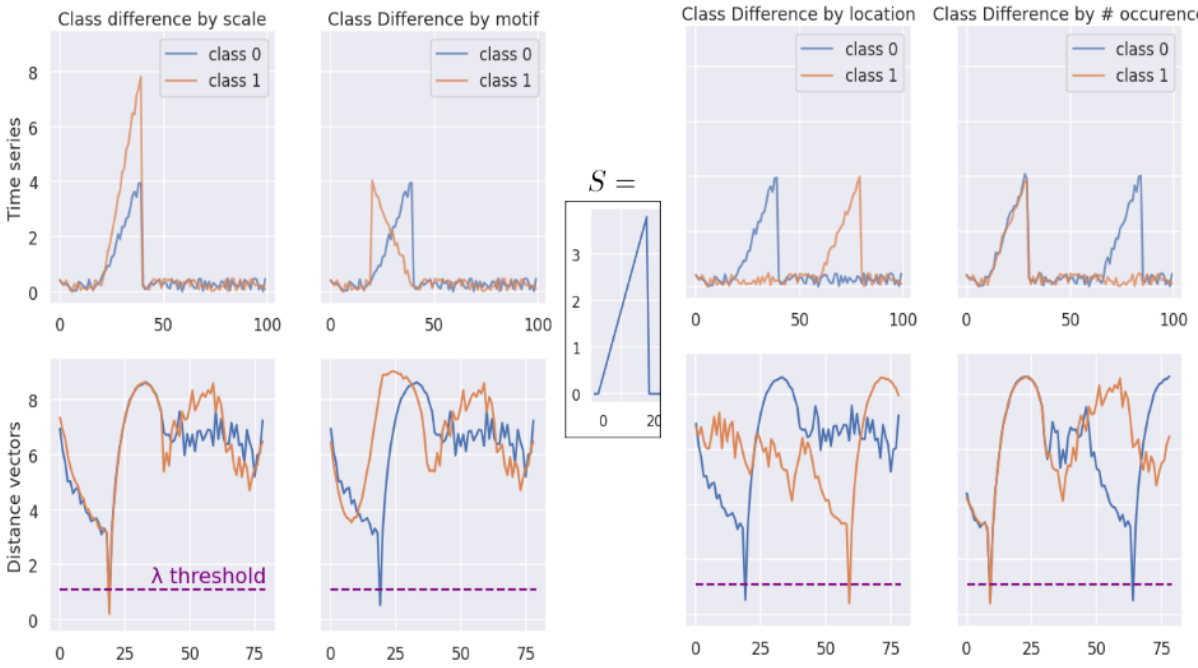

Figure 3.5: Synthetic examples with their distance vector computed using a shapelet S and a z-normalized Euclidean distance. The first row give the input time series and the second give the resulting distance vectors. Each column illustrates a different way for the same shapelet to discriminate the classes.

3.4 Random Dilated Shapelet Transform (RDST)

Considering the notion of dilated shapelet and the Shapelet Occurrence feature that we introduced, we propose a new shapelet algorithm. We choose to use a random approach for the shapelet generation for multiple reasons:

- We wanted to produce a scalable algorithm, which could compete with methods like ROCKET, which have shown that random methods can achieve state-of-the-art accuracy.

- Adding the dilation parameter and the λ threshold would make the search for optimal shapelets even slower, as the possible range of those two parameters makes the search space much larger.

- The recent tendency of the machine learning community to consider accuracy (i.e. improving state-of-the-art on a benchmark) as the most important factor of success has led to a loss of interest in presenting, or even considering, the optimization of the implementation of algorithms, as shown by [67]. In the following, we thus give a particular focus on increasing efficiency, rather than solely focusing on accuracy.

3.4.1 Definition

For simplicity, we define our approach in the context of univariate and same length time series, with $\mathcal{X} = \{X_1, \ldots, X_n\}$ a set of time series with $X_i = \{x_1, \ldots, x_m\}$, and $Y = \{y_1, \ldots, y_n\}$ their respective classes. We show in Section 3.7 how it extends to other contexts. Our method takes as input four parameters that are:

- n_{shp} : the number of shapelets to generate,
- L : a set of possible lengths for the shapelets,
- P_{norm} : the proportion of shapelets that will be used with z-normalization,
- (p_{min}, p_{max}) : a pair used as percentile bounds for the sampling of the λ threshold.

As stated in Section 3.3.2, the choice of whether to use a normalized distance is mostly based on prior knowledge of the data. Using the parameter P_{norm} allows to either set values that are known to be acceptable for a use case, or to search for it by trying multiple values of the parameter.

Given the definition of a dilated shapelet with λ threshold $S = \{\{s_1, \ldots, s_l\}, d, \lambda\}$, we initialize each parameter as follows:

- the length l is uniformly drawn from L,
- the dilation d, in the same way as ROCKET [22], is set to $d = \lfloor 2^x \rfloor$ with x uniformly drawn in $[0, log_2 \frac{m}{l}]$. In addition to limiting the maximum dilation to reasonable values to avoid generating tiny distance vectors, sampling possible dilation uniformly from $[0, log_2 \frac{m}{l}]$ generates a distribution of dilations favouring smaller values of dilation due to log_2. An example of the distribution of dilation obtained with this method is shown by Figure 3.6, for 10000 generated shapelets.

  The sensitivity analysis conducted by [22] on 40 datasets show that this is significantly more accurate on average than using a uniform distribution of dilation. This distribution of dilation also makes sense given how we sample shapelets, as the larger the dilation, the smaller the number of candidate shapelets, as we have $n \times (m - (l-1) \times d)$ admissible sampling points for candidate shapelets with a dilation of d.

- choose if the shapelet will use a z-normalized distance with probability P_{norm},
- generate the values $\{s_1, \ldots, s_l\}$ of the shapelet, a sample X is uniformly drawn from $\mathcal{X}$, and an admissible start point $i \in [1, m - (l-1) \times d]$ is randomly selected with uniform probability. Then, the values are set as $S = X^{i,l,d}$.

- finally, given a shapelet S, to fix the value of λ, we randomly choose a sample X from the same class as the one used for extracting the shapelet value, and uniformly draw a value between the two percentiles (p_{min}, p_{max}) of $V_{S,X}$.

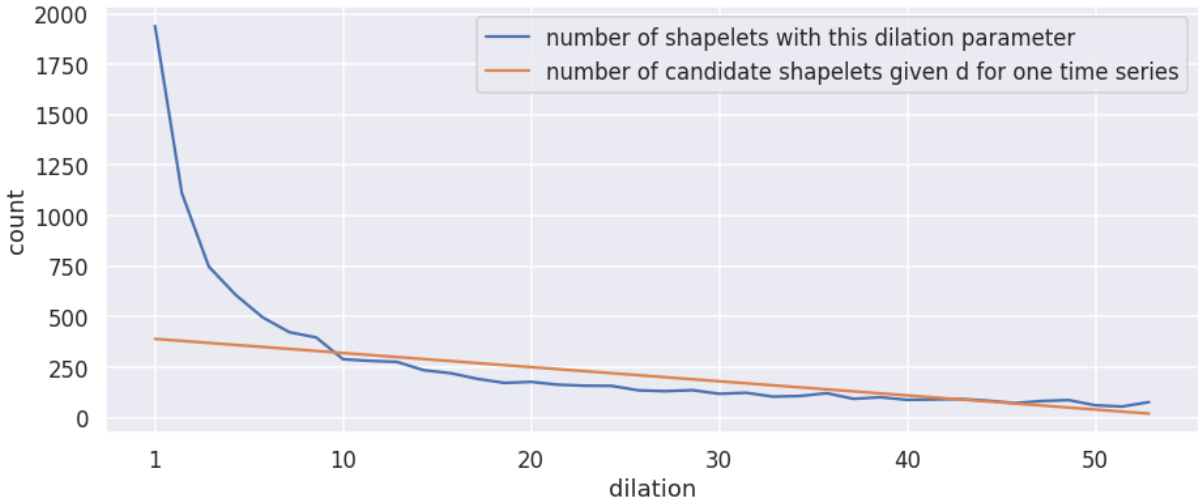

Figure 3.6: Count of the values of dilation obtained by using $d = \lfloor 2^x \rfloor$ with x uniformly drawn in $[0, log_2 \frac{m}{l}]$ with $l = 11$ and $m = 400$ for 10000 draw. The number of admissible candidate shapelets for one time series (in orange) is given by $m - (l-1) \times d$

Algorithm 2 gives the pseudocode for the initialization of the shapelets. After computing the distance vector between all pairs of time series and shapelets, the output of our method is a feature matrix of size $(|\mathcal{X}|, 3 \times n_{shp})$, with the three features extracted from the distance vector $V_{S,X}$ being the $min, argmin, SO(S, X)$. We choose to use a Ridge Classifier after the transformation of $\mathcal{X}$, as the L2 regularization used in Ridge is of critical importance due to the high number of features that are generated, while also being interpretable and scalable even when the number of features is high. The pseudocode for the whole RDST method is given in Algorithm 3.

3.4.2 Alternative strategies for the λ threshold

Multiple strategies can be employed to fix the λ threshold, which are subject to the trade-off between time and accuracy. Consider a shapelet $S = \{\{s_1, \ldots, s_l\}, d\}$ and a set of time series $\mathcal{X} = \{X_1, \ldots, X_n\}$ with $X_i = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$ their respective classes. We can then compute a set of distance vectors $\mathcal{V} = \{V_1, \ldots, V_n\}$ with $V_i = \{v_1, \ldots, v_{m-(l-1)\times d}\}$.

The first alternative is to use a random approach, where we set the λ threshold to a random value in $[min(\mathcal{V}), max(\mathcal{V})]$. Another option is to find the optimal value for the λ threshold by computing the information gain of the Shapelet Occurrence feature for every possible value of $\lambda \in [min(\mathcal{V}), max(\mathcal{V})]$. At the expense of optimality, multiple speed-ups techniques could be proposed, like subsampling or reducing the range of value for λ, either by imposing a higher bound or by discretizing the interval $[min(\mathcal{V}), max(\mathcal{V})]$.

Algorithm 2 initialize_shapelets($\mathcal{X}, Y, n_{shp}, L, P_{norm}, p_{min}, p_{max}$)

$n, m \leftarrow$ shape($\mathcal{X}$)
$\mathcal{S} \leftarrow \varnothing$
for $i_{shp} \in [1, n_{shp}]$ **do**
  $l \leftarrow$random_uniform(L)
  power $\leftarrow$random_uniform($[0, log_2(m/l)]$)
  $d \leftarrow \lfloor 2^{power} \rfloor$
  dist_len $\leftarrow m - (l-1) \times d$ ▷ Number of admissible sampling points for a series
  normalize $\leftarrow$random_uniform($[0,1]$) $\leq P_{norm}$
  location $\leftarrow$ random_uniform($[1, n \times$dist_len$]$)
  $i_x, i_t \leftarrow \lfloor$location/dist_len$\rfloor$, location % dist_len
  $S \leftarrow X_{i_x}^{i_t,l,d}$ ▷ The subsequence of size l starting at index i_t in sample i_x with d
  if normalize **then**
    $S \leftarrow$z_normalize(S)
  end if
  $i_\lambda \leftarrow$random_uniform(where($Y = y_{i_x}$)) ▷ Another sample of the same class as i_x
  dist $\leftarrow$ distance_vector($X_{i_\lambda}, S,\ l,\ d$, normalize)
  $\lambda_S \leftarrow$random_uniform($[P(dist, p_{min}), P(dist, p_{max})]$) ▷ P as percentile function
  $\mathcal{S}.add(\{S, l, d, \lambda_S,$ normalize$\})$
end for
return $\mathcal{S}$

Algorithm 3 RDST($\mathcal{X}, Y, n_{shp}, L, P_{norm}, p_{min}, p_{max}$)

$n, m \leftarrow$ shape($\mathcal{X}$)
$\mathcal{S} \leftarrow$ initialize_shapelets ($\mathcal{X}, Y, n_{shp}, L, P_{norm}, p_{min}, p_{max}$)
$M \leftarrow$ empty_array(size=$(n, 3 \times n_{shp})$)
for $i_{shp} \in [1, n_{shp}]$ **do**
  for $i_x \in [1, n]$ **do**
    $min, argmin, SO \leftarrow$ extract_features($\mathcal{X}[i\_x]$, $\mathcal{S}[i\_shp]$)
    $M[i_x, (i_shp \times 3)] \leftarrow min$
    $M[i_x, (i_shp \times 3) + 1] \leftarrow argmin$
    $M[i_x, (i_shp \times 3) + 2] \leftarrow SO$
  end for
end for
RidgeClf $\leftarrow$ RidgeClassifier().fit(M, Y) ▷ Standardize each feature before fitting
return RidgeClf, $\mathcal{S}$

In RDST, we chose to use a semi-random approach, in which, given the sample X_i from which S was extracted, we select a sample X_j, of the same class as y_i. Then, given its associated distance vector V_j and two bounds p_{min}, p_{max}, we compute the possible range for the values of λ, based on the p_{min} and p_{max} percentiles of V_j, and select a random value in this range. The rationale behind this idea is that, given S a shapelet extracted from class y_i, we can distinguish two cases: S also "appears", one or more times, in other samples of the same class, or it is unique to X_i. If S only appears in X_i, it is irrelevant to recognize members of class y_i (e.g. noise). Hence, we can choose p_{min} and p_{max} such

as the λ threshold value characterizes an expected number of λ-close occurrences of S in another sample of class y_i, based on the observed values of the distance vectors of this sample. We show in our experimental section that this strategy is competitive with the optimal solution.

3.5 Reducing the number of candidate shapelets

In this section, we present two techniques that we use to reduce the number of candidate shapelets that our algorithm can select from. The goal of such methods is to try to remove redundant shapelets, so that we have more chance of sampling useful shapelets for the classification task.

3.5.1 Shapelet similarity

In order to reduce the search space and to improve the time complexity of the shapelet generation, [48] defines the notion of shapelet self-similarity, which they define as follows: "We define two shapelets as being self-similar if they are taken from the same series and have any overlapping indices". This means that, given a shapelet sampled from a point i, no shapelet can be sampled in the range $[i - (l - 1), i + (l - 1)]$. While this allows to greatly reduce the number of possible candidates, one could have the intuition that it could possibly discard the "suffix" or "prefix" of discriminative candidates. We can nevertheless adapt this idea to the case of dilation, and compare it against other baselines in our experiments.

To adapt this notion of self-similarity to our new formulation, we need to take into account the dilation and the normalization parameters of the shapelet. As each combination of those two parameters produces a different view of the input data, we propose to apply self-similarity independently on each combination of dilation and normalization. Furthermore, to test whether this can impact the performance of our method in our experiments, we propose to relax this definition to α-similarity ($\alpha \in [0, 1]$).

Given L the possible shapelets lengths, a candidate shapelet S will be pruned by α-similarity, if it has $\max(1, \lfloor(1 - \alpha) \min(L)\rfloor)$ or more shared indexes with shapelets previously sampled in the same time series. Figure 3.7 gives a visual comparison of α-similarity with $\alpha = 0.5$ and $\alpha = 1.0$, which is equivalent to self-similarity, as all the indexes of a shapelet must have never been used before.

In practice, for each possible values of dilation and normalization, we maintain a distinct Boolean mask during the generation of the shapelets, which indicate admissible sampling points. This mask is shared by all shapelets with the same dilation and normalization parameters, **even if they have a different length parameter**. The mask is updated after the sampling of each shapelet, and takes into account the effect of dilation, as neighbouring sampling points are not to be pruned when $d > 1$.

For example, given $\alpha = 1$, $d = 2$ and $l = 3$ and a sample X, if the point i is used to sample $S = \{x_i, x_{i+2}, x_{i+4}\}$, $\{i - 4, i - 2, i, i + 2, i + 4\}$ cannot be considered as sampling points anymore, while $\{i - 3, i - 1, i + 1, i + 3\}$ are still valid, as they do not share any input indexes with $X^{i,l,d}$. We thus need to set to false indexes i (+/-) $j \times d \mid \forall j \in \{0, 1, \ldots, l - \max(1, \lfloor(1 - \alpha) \min(L)\rfloor)\}$, if the corresponding indexes exist

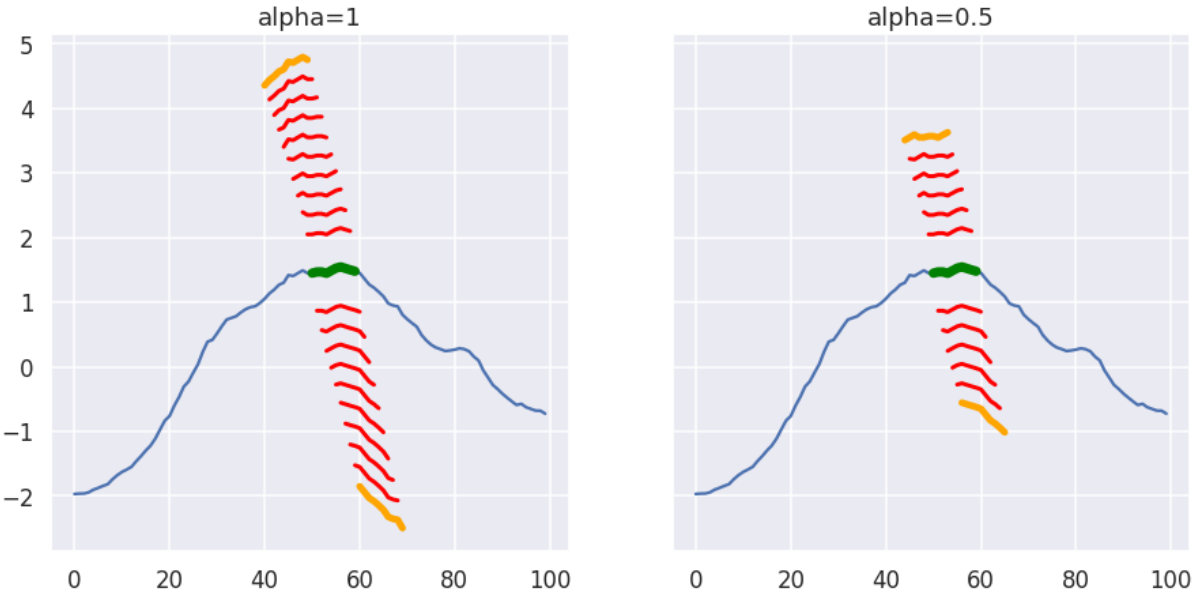

Figure 3.7: Illustration of α-similarity with $\alpha = 1$ (left), equivalent to self-similarity, and $\alpha = 0.5$ (right). The green part represents a sampled shapelet, the red parts are candidates excluded by α-similarity (includes the green part), and in orange are the closest admissible candidate to the shapelet.

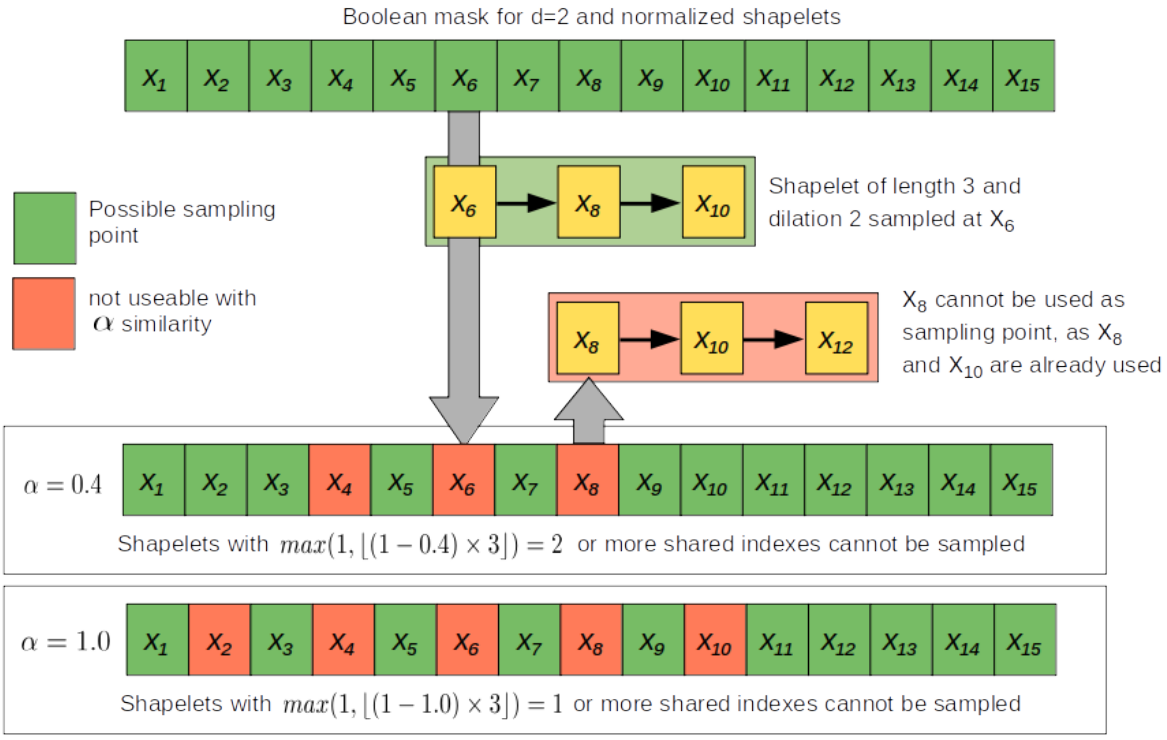

Figure 3.8: Illustration of the Boolean mask used to know which points of a time series are admissible sampling points for candidate shapelets, with an update to the mask made for $\alpha = 0.6$ and $\alpha = 1.0$

in the mask. Figure 3.8 gives a visual example of this process for a shapelet with $l = 3$ and $d = 2$.

More formally, consider $\mathcal{S} = \{S_1, \ldots, S_p\}$ a set of shapelets already sampled from $\mathcal{X} = \{X_1, \ldots X_n\}$, a set of time series of size m, and L the set of possible length for the shapelets. To know which points are valid sampling points for new shapelets, we use a boolean mask $\mathcal{B}^{d,norm} = \{B_1, \ldots, B_n\}$ for each pair of dilation d and normalization $norm$, with $B_i = \{b_1, \ldots, b_{m-(\min(L)-1)\times d}\}$ the mask for possible sampling points in sample X_i. Note that we use $m - (\min(L) - 1) \times d$ as the size of the boolean mask, as the smaller the length of the shapelet, the higher the number of possible candidates (i.e. the closer to m the sampling point can be). Shapelets with length $l > \min(L)$ cannot be sampled after point $m - (l - 1) \times d$, but can still influence the sampling of the shapelets of length $l' < l$, based on the value of α.

Given $i \in [1, n]$ the sample index, and $j \in [1, m - (l-1) \times d]$ the index of the sampling point, we define this $X_i^{j,l}$ as a valid sampling point if $b_j \in B_i$ is True. Then, to update the mask, we apply Equation 3.6, which marks as False all points in B_i which would result in shapelets pruned by α-similarity.

$$b_{j+q\times d} = False,\ \forall q \in \{0, \ldots, l - \max(1, \lfloor(1 - \alpha)\min(L)\rfloor)\} \mid B_i \in \mathcal{B}^{d,norm} \tag{3.6}$$

3.5.2 Input subsampling

Another method, which was already used in previous works, is to use subsampling to reduce the search space. Given a set of time series $\mathcal{X} = \{X_1, \ldots, X_n\}$ used to sample shapelets and then train a model, subsampling removes a proportion $r \in [0, 1]$ of samples from $\mathcal{X}$ before it is used to sample shapelets. It is often done in a stratified manner, so that the proportion of each class in the subsampled set is the same as in $\mathcal{X}$.

The rationale for subsampling is that, theoretically, a good candidate shapelet, one that is unique to a class, can be found in any sample of the said class, and thus, using fewer samples could help the algorithm by reducing the search space to find such shapelets. While this strategy may work for a well-defined and clean dataset, the presence of outliers or noisy samples could make outliers more frequent in the training data and thus reduce the performance of the classifier. Another problematic case with subsampling arises when, within the same class, we can distinguish subclasses. For example, in the Gunpoint dataset of the UCR archive [19] two actors perform a hand movement with and without a gun, the goal being to identify which time series are linked to a movement performed with, or without a gun, independently of the actor. In the case where subsampling completely removes one actor from the training data, the performance on the test set, where the two actors are present, could be reduced.

3.6 Invariance properties and robustness for shapelets

In this section, we propose an adaptation of the properties presented in Section 2.3.7 to the context of shapelets, except the robustness to time warping, as discussed in Section 2.3.7. We study the impact on accuracy and time complexity of these potential additions to RDST in our experimental section.

3.6.1 Phase invariance

Given a time series $X = \{x_1, \dots x_m\}$ and a shapelet S with parameters l, d, we compute a distance vector $V_{S,X}$ of size $m - (l-1) \times d$. When extracting $\min V_{S,X}$, the position of the best match in the series do not influence the value, thus $\min V_{S,X}$ is translation invariant. This is also true for the Shapelet Occurrence feature.

In cases where the data represented by X is cyclic, the initial phase of the underlying signal could cause the pattern matched by S to be split between the end and the start of X, meaning that the starting index of the pattern would be in the interval $[1 + m - (l - 1) \times d, m]$, as shown in Figure 3.9

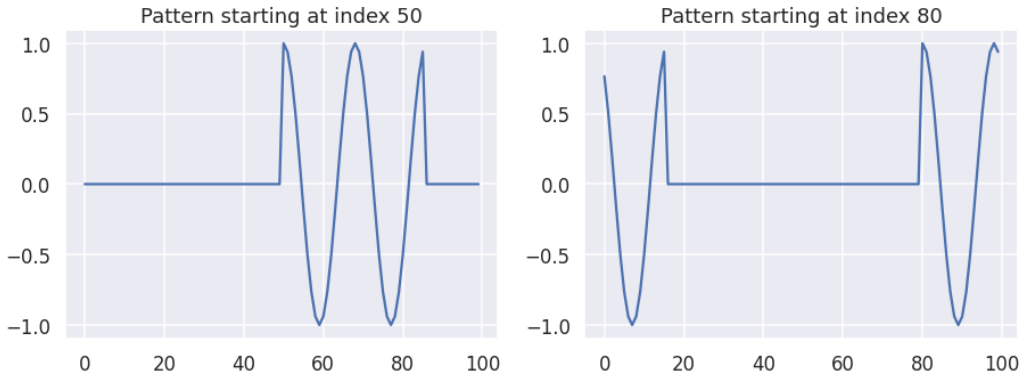

Figure 3.9: Example of a cyclic signal with a different initial phase. The pattern of interest can be split between the end and the start of the time series.

We can add phase invariance for shapelets by modifying the distance computation. The distance vector resulting of the operation between a shapelet S and time series $X = \{x_1, \dots x_m\}$ is now defined as $V_{S,X} = \{v_1, \dots v_m\}$, with v_i computed as :

$$v_i = \sqrt{\sum_{j=1}^{l} (x_{(i+(j-1)\times d)\%m} - s_j)^2} \tag{3.7}$$

To give a visual intuition, we can see this as wrapping the time series around a circle, as represented in Figure 3.10, and sliding the shapelet (projected into this 2D space) around the circle until it has used all points (i.e. from 1 to m) as starting points.

Another advantage of this operation is that all points of X will be used in the distance computations exactly l times, whereas without phase invariance, indexes 1 from $(l-1)\times d$ (respectively for $m-(l-1)\times d$ to m) are seen less often. The sampling process for shapelets can also be adapted for phase invariance, by considering candidates starting at indexes $[1, m]$ rather than only in indexes $[1, m - (l - 1) \times d]$. This also apply to α similarity, by extending the size of the Boolean mask of each sample from $m - (l - 1) \times d$ to m.

While, due to lack of time, we do not explore this possibility in this thesis, interesting optimization could be made with this property. When using phase invariance, the length of the distance vector is a constant equal to the number of timestamp m, independently of the length and dilation parameter of the shapelet. This could allow for significative speed-up, by transitioning to large matrix operation computed on a GPU, if the cost of transferring the data is worth it (i.e. for big datasets).

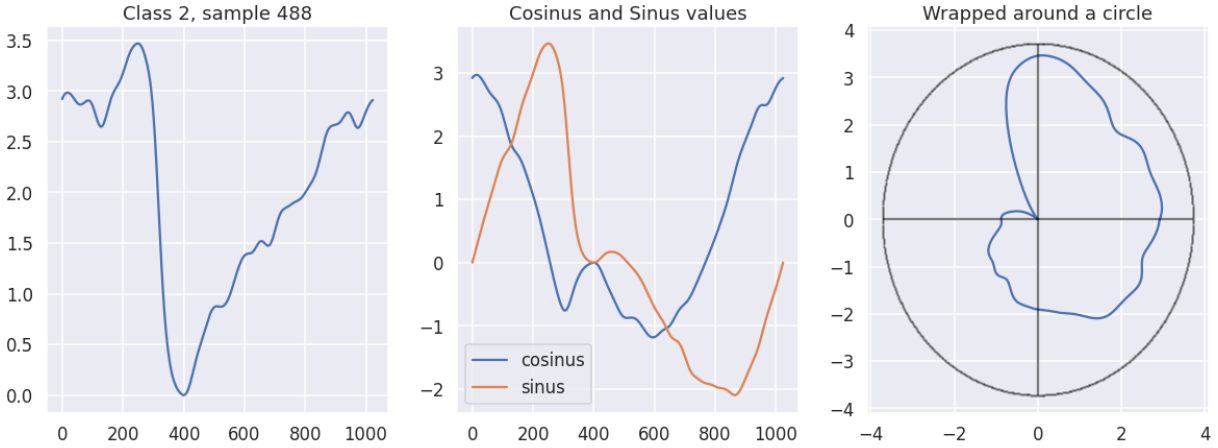

Figure 3.10: An example from the StarLightCurves dataset, wrapped around a circle.

3.6.2 Robustness to complexity

We can also apply the complexity invariance correction, simply by modifying the distance function used to compute the value of each point of the distance vector between a shapelet and a time series. As described by [8], this distance function becomes:

$$v_i = CF(S, X^{i,l,d}) \times \sqrt{\sum_{j=1}^{l}(x_{(i+(j-1)\times d)} - s_j)^2} \tag{3.8}$$

with CF the complexity correction factor computed as described in Section 2.3.7. The impact of this addition is presented in the experimental section.

3.6.3 Robustness to noise

To the best of our knowledge, dealing with noise on a local level (i.e. at each step of the sliding window used by the shapelets) has not been studied. It is more common to apply a filter on the whole time series as a preprocessing step, to try to filter out a noise process (e.g., Gaussian noise) that has been applied to the data. The best kind of filter to be used is entirely dependent on the data, and thus, on expert or domain knowledge.

A possible solution without prior knowledge, at the expense of time complexity, could be to apply multiple filters on the training data, and select the one that maximize training accuracy in a leave-one-out validation, or to perform a Discrete Fourier Transform and select discriminative Fourier coefficients, for example with a one-way ANOVA test.

We can also use an ensemble of classifiers using different input representations, with a weighted majority voting, based on classification accuracy on training data, using for example a leave-one-out cross validation, in order to favour discriminative representations. We explore the use of an ensemble classifier in our experimental section, using diverse representations of the inputs, notably in the frequency domain, in an attempt to gain robustness toward noise.

3.7 Generalization of RDST

Multiple extensions to the multivariate context have already been proposed by [12]. Due to the modifications of the shapelet formulation and the new feature we extract, and based on the previous results of [12], we focus on the time-dependent method, where a multivariate shapelet is slided across the series, rather than a time-independent method, where each feature is computed independently (i.e. the best match is found independently for each feature rather than globally on all features). Both approaches are illustrated in Figure 3.11

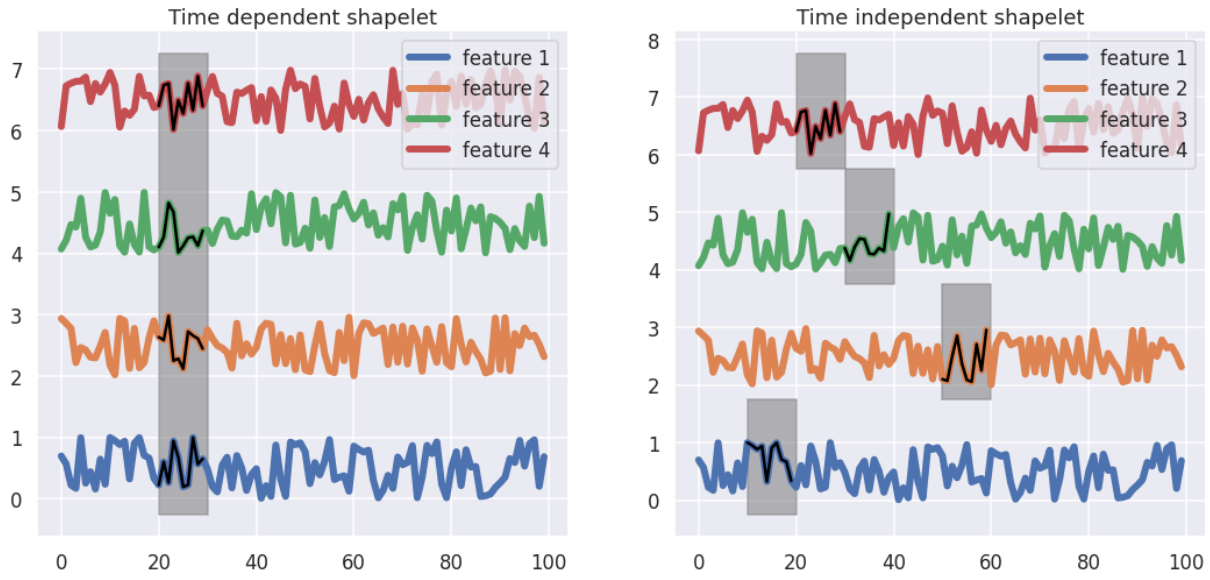

Figure 3.11: Example of (a) time dependent shapelet and (b) time independent shapelet in black, with the moving window positioned on the best match between the shapelet and the time series illustrated in grey

In the multivariate context, a time series with k features is defined as $X = \{(x_{1,1}, \ldots, x_{1,k}), \ldots, (x_{m,1}, \ldots, x_{m,k})\}$, with $x_{i,j}$ the value at index i for feature j. Similarly, a multivariate shapelet of length l with k features is defined as $S = \{(s_{1,1}, \ldots, s_{1,k}), \ldots, (s_{l,1}, \ldots, s_{l,k})\}$.

To adapt RDST to the multivariate context, when creating a shapelet, we randomly sample a subset of the input features, with the size of the subset randomly selected from $[1, k]$ (i.e., univariate up to all the features of the input series), we denote this subset as $F = \{f_1, \ldots\}$ and the associated shapelet as $S = \{(s_{1,1}, \ldots, s_{1,||F||}), \ldots, (s_{l,1}, \ldots, s_{l,||F||})\}$. Figure 3.12 shows an example of such shapelets. For the multivariate context, the distance vector $V_{S,X} = \{v_1, \ldots, v_{m-(l-1)\times d}\}$, with f_q the q^{th} selected feature, is then computed as:

$$v_i = \sum_{q=1}^{||F||} \sqrt{\sum_{j=1}^{l} (x_{i+(j-1)\times d, f_q} - s_{j,q})^2} \tag{3.9}$$

If S uses z-normalization, all subsequences of X are z-normalized independently of each other and for each feature prior to the distance computation.

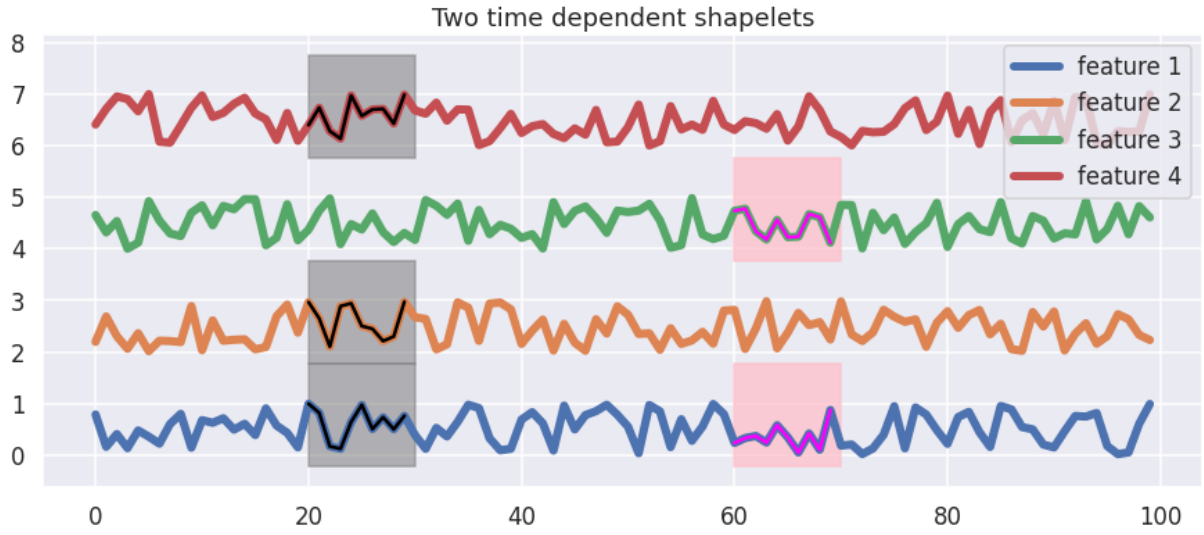

Figure 3.12: Example of two multivariate shapelets (black and fuchsia) created by RDST, with each shapelet affected to a specific subset of features

3.7.1 α-similarity for multivariate shapelets

We also need to adapt the notion of α-similarity presented in Section 3.5 to the case of multivariate data.

Consider $\mathcal{S} = \{S_1, \ldots, S_p\}$ a set of multivariate shapelets, with L the set of possible lengths. We consider that $\mathcal{S}$ was already sampled from $\mathcal{X} = \{X_1, \ldots X_n\}$ with $X_i = \{(x_{1,1}, \ldots, x_{1,k}), \ldots, (x_{m,1}, \ldots, x_{m,k})\}$, a set of multivariate time series of size m with k features.

The Boolean mask $\mathcal{B}^{d,norm} = \{B_1, \ldots, B_n\}$ defined for each pair of dilation d and normalization $norm$, is extended to the multivariate context as $B_i = \{(b_{1,1}, \ldots, b_{1,k}), \ldots, (b_{m-(min(L)-1)\times d,1}, \ldots, b_{m-(min(L)-1)\times d,k})\}$. Similarly to the univariate version, it is shared by all shapelets with parameters d and $norm$, independently of their length or the subset of feature they use.

Given $i \in [1, n]$ the sample index, $j \in [1, m - (l-1) \times d]$ the timestamp index and F the subset of input features used, the multidimensional sampling point is considered as valid if the condition described by Equation 3.10 is true, with I the identity function ($I(True) = 1$ and $I(False) = 0$).

$$\max(1, ||F||\alpha) \geq \sum_{q=1}^{||F||} I(b_{j,f_q}) \mid B_i \in \mathcal{B}^{d,norm} \tag{3.10}$$

In other words, a proportion of at least α (e.g. 0.5) of the $||F||$ selected points must be True for this multidimensional sampling point to be valid. The mask is then updated similarly to the univariate version for each $f_q \in F$.

3.7.2 Extension to variable length time series

To adapt to variable length time series, given m_i the length of the series X_i, we do not have to modify the computation of the distance vector V_{S,X_i}. A shapelet can still be applied

by sliding it across X_i, independently of the length of the other series in the dataset. In terms of adaptation for RDST, we need to perform some minor modifications:

- Take as input a parameter min_len, giving the minimum possible length of a time series, to cover the case where such series of minimal length are not included in the training data. If a series is smaller than min_len, it will be rejected, and we should have $l \leq min\_len,\ \forall\ l \in L$.

- When sampling the dilation parameter of a shapelet, we need to consider the new parameter min_len as $d = \lfloor 2^x \rfloor$ with x uniformly drawn in $[0, log_2 \frac{min\_len}{l}]$. Sampling dilation with the length of a series m_i might lead to shapelets longer than other series where $m_i >> m_j$, making the Euclidean distance impossible to apply.

- The length of the Boolean mask used for α-similarity is now variable between each sample.

Additionally, considering a specific use case, if biases can be introduced based on the length of the series (i.e. if the distribution of the length of the series is different between classes but should not be considered as discriminative), we have to normalize $\text{argmin}\, V_{S,X_i}$ and the Shapelet Occurrence feature by dividing them by m_i, in order to remove the influence of the length of the series in the features generated by the shapelets.

3.8 Parallelization and optimization of RDST

In this section, we detail the implementation of RDST and how it has been optimized. To improve the overall efficiency of our method compared to a pure Python implementation, we use Numba [43], a Python compiler, capable of generating optimized machine code from Python-like code. Further improvements are made possible with the use of Numba, notably, parallelism and vectorization with the Single Instruction Multiple Data (SIMD) protocol.

The goal of parallel computing is to perform operations simultaneously on different computing units, and to regroup the results once all operations are done. The idea behind the use of parallelism is that large problems can often be divided into smaller, and ideally independent ones, which can be solved on different computing units.

One constraint when implementing a parallel algorithm is to choose whether we need to use processes or threads. The main difference between the two is that all threads use the same memory space, while each process makes a copy of the data needed to perform the task. Processes are less prone to errors compared to threads as they have their own memory space, and thus are unlikely to run into race conditions when modifying the memory space, but they consume more resources due to the data copy. When using Numba, threads are used as the default paradigm for parallelization. This is not a problem as RDST is easy to parallelize, as in some "for" loops, all iterations are completely independent of each others.

Given n the number of samples, m the number of timestamps, $\bar{L}$ the mean of the possible shapelet lengths, and n_{shp} the number of shapelets, we estimate the time complexity using the O notation. For the shapelet initialization, described by Algorithm 2, as $O(\bar{L} n_{shp} m)$, and for the shapelet transform, described by Algorithm 3, as $O(\bar{L} n_{shp} m n)$.

While the changes we present in the following sections do not change the O notation, they influence the hidden factors affected to each variable, and parallelization divide it by the number of cores available on the machine.

3.8.1 Shapelet initialization

In Algorithm 2, which describes how RDST initialize the shapelets, each shapelet initialization is independent of the others, and only read operations are performed on the data shared by the iterations of the loop (i.e. $\mathcal{X}$ and Y). This makes the loop iterating on the number of shapelets an ideal candidate for parallelization, as it does not have any synchronization issue or race conditions between threads.

A synchronization problem is although introduced when using α similarity, as shapelets with similar parameters share the same Boolean mask, and the sampling location of one shapelet depends on the locations of the previous ones. To bypass this problem, we implement the parallelism on a loop iterating on the shapelet parameters, as described by Algorithm 4, so that each thread manages one set of dilation and normalization parameter and its associated Boolean mask.

Algorithm 4 initialize_shapelets parallel $(\mathcal{X}, Y, n_{shp}, L, P_{norm}, p_{min}, p_{max}, \alpha)$

$n, m \leftarrow \text{shape}(\mathcal{X})$
$Lengths \leftarrow \text{random\_uniform}(L, size \leftarrow n_{shp})$
$powers \leftarrow \text{random\_uniform}([0, log_2(n/Lengths)])$
$Dilations \leftarrow \lfloor 2^{powers} \rfloor$
$Normalize \leftarrow \text{random\_uniform}([0, 1], size \leftarrow n_{shp}) \leq P_{norm}$
$Thresholds \leftarrow \text{empty\_array}(size \leftarrow n_{shp})$
$Values \leftarrow \text{empty\_array}(size \leftarrow (n_{shp}, \max(L)))$

for $(d, norm) \in \text{Combinations}(Dilations, Normalize)$ **do** ▷ Do this loop in parallel
  $mask \leftarrow \text{array}(True, size \leftarrow (n, m - (\min(L) - 1) \times d))$
  $I_{shps} \leftarrow \text{where}(Dilations = d \ \wedge \ Normalize = norm)$
  for $i_{shp} \in I_{shps}$ **do** ▷ If mask all false, break the loop
    $i_x, i_t \leftarrow \text{random\_uniform}(\text{where}(mask = True))$
    $l \leftarrow Lengths[i_{shp}]$
    $mask \leftarrow \text{update\_mask}(mask, i_x, i_t, l, d, \alpha)$
    $val \leftarrow X_{i_x}^{i_t, l, d}$
    if $norm$ **then**
      $val \leftarrow \text{z\_normalize}(val)$
    end if
    $i_\lambda \leftarrow \text{random\_uniform}(\text{where}(Y = y_{i_x}))$
    $\lambda \leftarrow \text{get\_threshold}(X_{i_\lambda}, val, l, d, norm, p_{min}, p_{max})$
    $Values[i_{shp}, : l] \leftarrow val$
    $Thresholds[i_{shp}] \leftarrow \lambda$
  end for
end for
$I \leftarrow \text{where}(Thresholds \neq \emptyset)$ ▷ Index of shapelets not skipped due mask all False
return $Lengths[I], Dilations[I], Normalize[I], Thresholds[I], Values[I]$

3.8.2 Shapelet Transformation

Similarly to the shapelet initialization, the computation of the distance vectors $V_{S,X}$ and the extraction of the three features can be made in parallel. Additionally, we should be careful of not computing the same operation multiple times. For example, if we use an outer loop iterating through each shapelet and an inner loop iterating through the time series, if a shapelet S_i use normalization, we will normalize each time series in the inner loop. If another shapelet S_j, with the same length and dilation parameter, also use normalization, we will need to compute the normalization again. We could of course cache the normalized dataset for each combination of parameters l, d, but this would massively increase the memory usage.

To address both of these points, we implement parallelization on an outer loop iterating through each sample, then an inner loop, which can also be parallelized, iterates through the combinations of length l and dilation d in the set of shapelets. After extracting the subsequences of size l, d from a sample X, we extract features for non-normalized shapelets, and if some shapelets with (l, d) use normalization, we z-normalize each subsequence. Algorithm 5 describes our parallel implementation of the shapelet transformation.

3.8.3 Distance computation and normalization

In RDST, excluding operations on indexes and the random choices, we can distinguish two kinds of mathematical operations that are repeated often, and thus are important to optimize:

- the Euclidean distance between two vectors, which is used to compute the individual points of each distance vector $V_{S,X}$
- the z-normalization, which is used before the Euclidean distance for normalized shapelets.

We can compare the efficiency of Numba against Numpy, a Python package dedicated to efficient array operations and written in CPython, as well as against a pure Python implementation. Numba and Numpy both make use of SIMD operations, and, for computing a Euclidean distance, both theoretically have a similar time complexity. Figure 3.13 shows the implementation of the Euclidean distance for the three methods. The python implementation uses the zip method, which takes a set of list as input, and give an iterator. The i^{th} element of this iterator is a set containing the i^{th} elements of the lists given as inputs.

In practice, the Numba function is faster compared to the Numpy function as it does not have all the input checks and storage of some intermediate results that Numpy have, hidden in the implementation of the functions it provides. This difference becomes less noticeable the bigger the input arrays are (>10000).

Numba can push performance further by using the LLVM compiler [44] "fastmath" option, which relaxes the IEEE standard for floating-point Arithmetic (IEEE 754) to perform optimizations on sequences of arithmetic instructions, which under IEEE 754, would lead to violations of the standard. The cost of this optimization is a small loss of floating point precision in results, but the reduction in run time is substantial. For example, with the Euclidean distance, the mean difference from the true result was of

Algorithm 5 Transform parallel($Lengths, Dilations, Normalize, Thresholds, Values$)

$n, m \leftarrow$ shape($\mathcal{X}$)
$n_{shp} \leftarrow$ shape($Lengths$)
$X_{new} \leftarrow$ empty_array($size \leftarrow (n, 3 \times n_{shp})$)
$param\_combs \leftarrow$ Combinations($Lengths, Dilations$) ▷ Only existing combinations
$n_{comb} \leftarrow$ shape($param\_combs$)[0]
$param\_indexes \leftarrow$ empty_array($size \leftarrow 2n_{comb}$)
for $i \in [1, n_{comb}]$ **do**
  $param\_indexes[2i] \leftarrow$ where $(Dilations = d \wedge Lengths = l \wedge \neg Normalize)$
  $param\_indexes[2i+1] \leftarrow$ where$(Dilations = d \wedge Lengths = l \wedge Normalize)$
end for

for $i_x \in [1, n]$ **do** ▷ Do this in parallel
  for $i \in [1, n_{comb}]$ **do** ▷ Do this in parallel if more threads than samples
    $Subs \leftarrow$ generate_subsequences(X_{i_x}, l, d) ▷ 2D array of size $(m - (l-1) \times d, l)$
    for $i_{shp} \in param\_indexes[2i]$ **do**
      $min, argmin, SO \leftarrow$ get_features$(Subs, Values[i_{shp}, : l], Threshold[i_{shp}])$
      $i_{out} \leftarrow 3 \times i_{shp}$
      $X_{new}[i_x, i_{out} : (i_{out} + 3)] \leftarrow min, argmin, SO$
    end for
    $I_{shps} \leftarrow param\_indexes[1 + 2i]$
    if $I_{shps} \neq \emptyset$ **then**
      $Subs \leftarrow$z_normalize_2D$(Subs)$ ▷ Each normalized independently
      for $i_{shp} \in I_{shps}$ **do**
        $min, argmin, SO \leftarrow$ get_features$(Subs, Values[i_{shp}, : l], Threshold[i_{shp}])$
        $i_{out} \leftarrow 3 \times i_{shp}$
        $X_{new}[i_x, i_{out} : (i_{out} + 3)] \leftarrow min, argmin, SO$
      end for
    end if
  end for
end for
return X_{new}

$1.06e-13$ with arrays of size 10 to 10000, but was up to 5.3 times faster for larger arrays. Table 3.1 gives a comparison of the mean performance of each approach with various array size, for 100000 repetitions.

Table 3.1: Comparison of the mean performance of the Euclidean distance computation using pure Python, Numpy and Numba with the fastmath option, for 100000 repetitions.

Array size	Python	Numpy	Numba
10	$4.66\mu s$(+/- $148ns$)	$5.22\mu s$(+/- $427ns$)	$303ns$(+/- $10.2ns$)
100	$36.8\mu s$(+/- $3.02\mu s$)	$5.06\mu s$(+/- $186ns$)	$339ns$(+/- $18.8ns$)
1000	$334.\mu s$(+/- $1.54\mu s$)	$7.04\mu s$(+/- $100ns$)	$408ns$(+/- $10.6ns$)
10000	$3.56ms$(+/- $238\mu s$)	$16.2\mu s$(+/- $508ns$)	$2.08\mu s$(+/- $36.3ns$)

```python
def euclidean_python(x, y):
    dist = [(a - b)**2 for a, b in zip(x, y)]
    dist = sqrt(sum(dist))
    return dist

def euclidean_numpy(x,y):
    return sqrt(numpy.sum((x - y)**2))

@njit(
  "float64(float64[:], float64[:])",
  fastmath=True, cache=True
)
def euclidean_numba(x,y):
    s = 0
    for i in prange(x.shape[0]):
        s += (x[i]-y[i])**2
    return s**0.5
```

Figure 3.13: Implementation of the Euclidean distance using pure Python, Numpy and Numba.

Concerning z-normalization, some gains can be obtained by reducing the number of time the array is iterated through. Consider the case where, given a vector X, we want to extract $X^{i,l,d}$ and z-normalize it. In a naïve implementation, we would extract the subsequence, and then, call two functions on the subsequence to obtain the mean and the standard deviation, which would lead to a total of 3 iterations over the array. We can extract all those information in one iteration by storing the sum and the squared sum of each point from the subsequence we extract, as the standard deviation σ can be equivalently computed as :

$$\sigma = \sqrt{\frac{1}{n}\sum_{i}^{n}(x_i - \mu)^2} = \sqrt{\frac{\sum_i^n x_i^2}{n} - \mu^2} \tag{3.11}$$

Figure 3.14 gives the implementation of the two cases and Table 3.2 shows the difference in performance for a loop of varying size.

Table 3.2: Comparison of the mean performance of the normalization of a subsequence of a vector using in loop sums and functions called after the loop, for 100000 repetitions.

Array size	with sums in loop	with functions after loop
10	$865ns(+/- 25ns)$	$904ns(+/- 62ns)$
100	$981ns(+/- 27ns)$	$1.01\mu s(+/- 12ns)$
1000	$2.18\mu s(+/- 35ns)$	$2.41\mu s(+/- 49ns)$
10000	$14.1\mu s(+/- 97ns)$	$17.6\mu s(+/- 99ns)$

```python
@njit(fastmath=True)
def get_subsequence(X, i, l, d):
    n_ts = X.shape[0]
    sub = np.empty(l)
    _sum = 0
    _sum2 = 0

    for j in prange(l):
        v = X[i+j*d]
        sub[j] = v
        _sum += v
        _sum2 += v**2

    mean = _sum/l
    std = (_sum2/l - mean**2)**0.5
    return (sub - mean)/std

@njit(fastmath=True)
def get_subsequence_naive(X, i, l, d):
    n_ts = X.shape[0]
    sub = np.empty(l)
    for j in prange(l):
        sub[j] = X[i+j*d]
    return (sub - sub.mean())/sub.std()
```

Figure 3.14: Implementation of a function extracting a subsequence from a vector and normalizing it, one storing the sums to compute the normalization, the other directly calling two functions to obtain the mean and standard deviation

3.9 Experiments

In this section, we perform an empirical validation of our contributions and provide answers to the following questions:

- What is the sensitivity of the parameters of RDST, and how do they influence its accuracy? (see Section 3.9.1)
- Is there any advantage to use a squared Euclidean distance or a Manhattan distance compared to the Euclidean distance? (see Section 3.9.2)
- What is the individual impact of each contribution on the performance of RDST? Which are worth keeping, and which one should we discard? (see Section 3.9.3)
- What is the time complexity of our method, and how the optimization we propose perform against the speed-up techniques proposed in the literature? (see Section 3.9.7)
- How does our method perform in terms of accuracy compared to the existing state of the art? Can we further improve the performance of our method by using an ensemble scheme ? (see Section 3.9.8)

All experiments were run on a DELL PowerEdge R730 on Debian 9 with 2 XEON E5-2630 Corei7 (92 cores) and 64 GB of RAM. To benchmark our results against other state-of-the-art methods, we use the 112 univariate datasets of the UCR archive [19] and the 26 multivariate datasets from the UEA archive [3]. Those datasets are defined by an original training and testing set for each problem, sometime introducing additional difficulties (e.g. a small training set and huge testing set). Hence, the standard validation protocol for this archive is to make n **resamples** rather than n **folds**, in order to preserve the distribution of the classes and the sizes of the original training and testing sets.

3.9.1 Sensitivity analysis

We conduct a sensitivity analysis on the four input parameters of our algorithm and their effect on classification accuracy on 40 datasets selected randomly out of the 112 univariate datasets. For each parameter analysis, all other parameters remain fixed at the following arbitrary default values: $n_{shp} = 10000$, $P_{norm} = 0.9$, $L = \{7, 9, 11\}$, $p_{min} = 5$ and $p_{max} = 15$. Figure 3.15 and Figure 3.16 give the mean accuracy ranks of each set of parameters over the 40 datasets averaged over 10 resamples.

Given the tested set of values, the most impactful parameter is the number of shapelets, with a noticeable increase in performance above 10000 shapelets. All other parameters, for the tested ranges, only show minor gains and thus seem to be stable on those 40 datasets. Based on those results, for all further experiments we set as default parameters $n_{shp} = 10000$, $P_{norm} = 0.8$, $L = \{11\}$, $p_{min} = 5$ and $p_{max} = 10$.

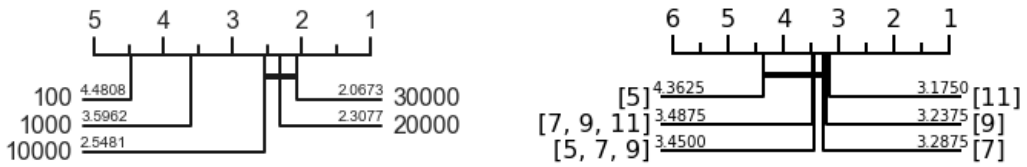

Figure 3.15: Mean accuracy ranks for (a) different number of shapelets, and (b) different shapelet lengths

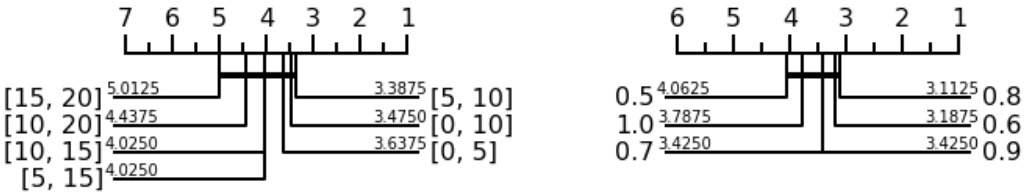

Figure 3.16: Mean accuracy ranks for (a) different percentiles bounds, and (b) proportion of z-normalized shapelets

3.9.2 Distance functions

In this section, we study the impact of using different distance functions for RDST. We performed a 10-resample validation for the 112 univariate datasets with the following distances :

- The Euclidean distance, as $d(S, X^{i,l}) = \sqrt{\sum_{j=1}^{l}(x_{i+j-1} - s_j)^2}$. This is the default option for shapelet methods.
- The squared Euclidean distance, as $d(S, X^{i,l}) = \sum_{j=1}^{l}(x_{i+j-1} - s_j)^2$, which will further increase high pointwise differences (> 1) and reduce small pointwise differences (< 1). This will influence the distance vector computation and the value of the minimum.
- The Manhattan distance, or L1 norm, as $d(S, X^{i,l}) = \sum_{j=1}^{l}|x_{i+j-1} - s_j|$, which requires fewer operations and should thus be faster than the Euclidean distance.

Figure 3.17 gives the critical difference diagram for those three variations, and 3.18 the pairwise accuracy plot. While there is no extreme difference in accuracy in any dataset, a difference in run-time is noticeable. The Manhattan distance took on average 3081.43 seconds to evaluate a single resample of each of the 112 datasets, while the Euclidean distance took 3604.77 seconds and the squared Euclidean distance 4036.11. While the distance computation squared Euclidean distance by itself is faster than the Euclidean distance, it makes the algorithm manage distance vectors with bigger numbers, which slow down the following operations. This may be due to some optimization performed by the LLVM compiler on small numbers (e.g. storing them with 8 bits rather than 64 bits).

On average for all datasets, the Manhattan distance was 38% faster than the squared Euclidean distance, and 16% faster than the Euclidean distance. Considering those results, we choose to use the Manhattan distance as the distance function in the following experiments.

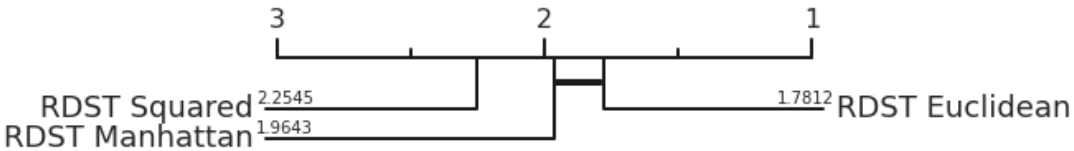

Figure 3.17: Critical difference diagram of the accuracy rank for RDST using the squared Euclidean, Euclidean and Manhattan distances.

3.9.3 Impact of individual contributions

We compare with critical diagrams the accuracy gain of each of our contributions by implementing multiple variations of RDST, each using different contributions, as presented by Table 3.3. Each variation, independently of the component it uses, follows the same algorithm as RDST, which can be summarized in 4 steps as:

1. Generate the set of input shapelets $\mathcal{S}$ using a random selection, which for some variations will be influenced by α similarity.
2. Compute all the distance vectors $V_{S,X}$ and extract features from it. The variations not using the Shapelet Occurrence feature will only extract min and argmin.
3. Standardize each feature independently and train a Ridge classifier.

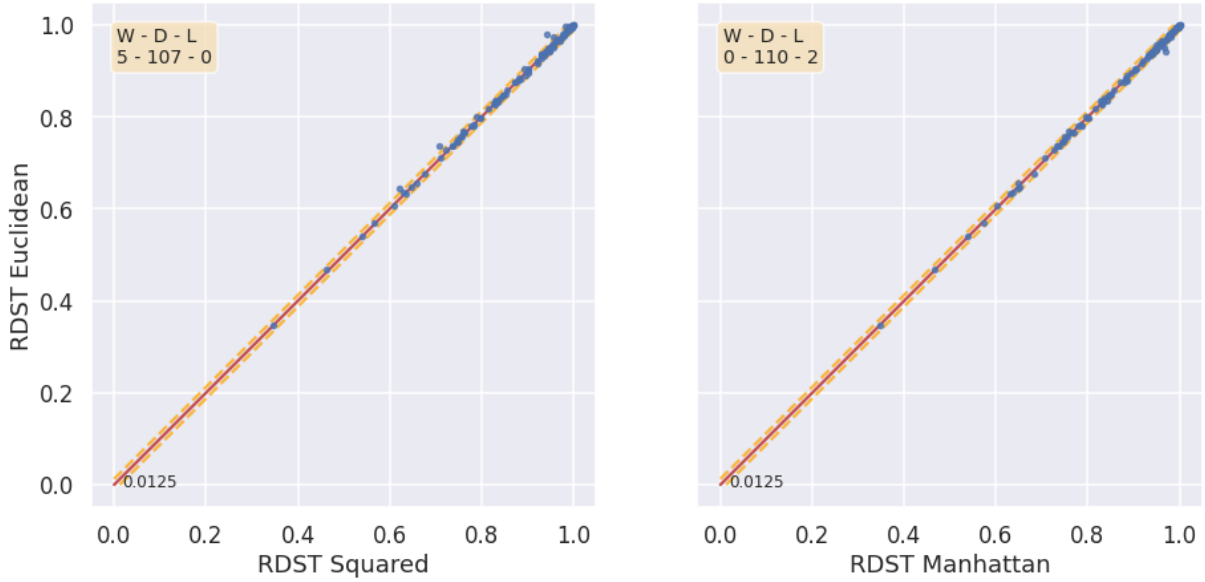

Figure 3.18: Pairwise accuracy plot for RDST using the Euclidean distance against RDST with the squared Euclidean and Manhattan distances, a margin of +/- 1.25% of accuracy to count draws. Both figures share the y-axis.

4. When predicting, standardize the features using the mean and standard deviation used on the training data, and use the fitted Ridge classifier.

We define a baseline method, denoted RST baseline, which do not include our contribution. It randomly samples shapelets from the input and extract the min and argmin from each distance vectors, and uses a Ridge classifier. Figure 3.19 shows how the RST baseline, compares to the state of the art. The critical diagrams we use in this section are built on the result of a 10-resample validation on the 112 datasets of the UCR archive. We also give pairwise accuracy plots to look at the performance on individual datasets against RST baseline and RDST.

Table 3.3: What are the contributions used by each variation of RDST that are used to study the impact of individual contributions

Name	Dilation	Shapelet Occurrence	α similarity	input subsampling
RST baseline				
RST + dilation	✓			
RST + lambda		✓		
RDST	✓	✓		
RDST + alpha=x	✓	✓	✓	
RDST + sub=y + alpha=x	✓	✓	✓	✓

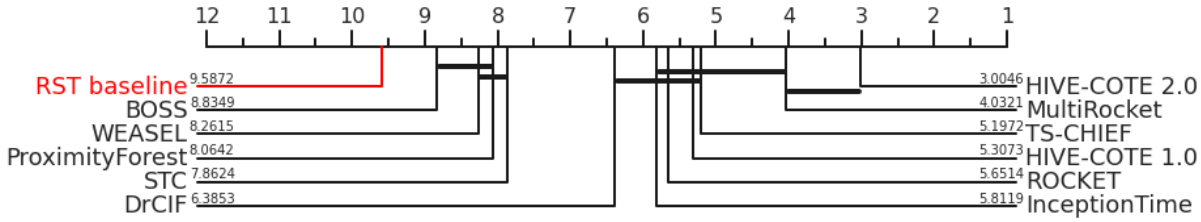

Figure 3.19: How does the RST baseline perform against the state of the art for accuracy on 10-fold resamples on the UCR archive.

Dilation

Adding the notion of dilation to the baseline already provides significative improvements on most dataset types, as shown by Figure 3.20. The highest increase between the baseline and the version with dilation is on the PigCVP dataset, with a mean gain of 36% accuracy, other datasets with a difference of at least 25% accuracy are shown in Figure 3.21, with the Win-Draw-Loss count in the top right made with a margin of 2.5% for draws.

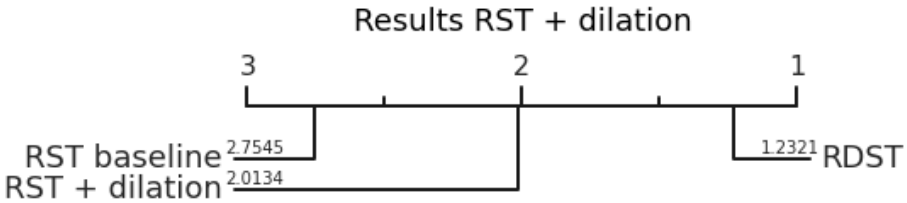

Figure 3.20: Critical difference diagram for a version with dilation against both baselines

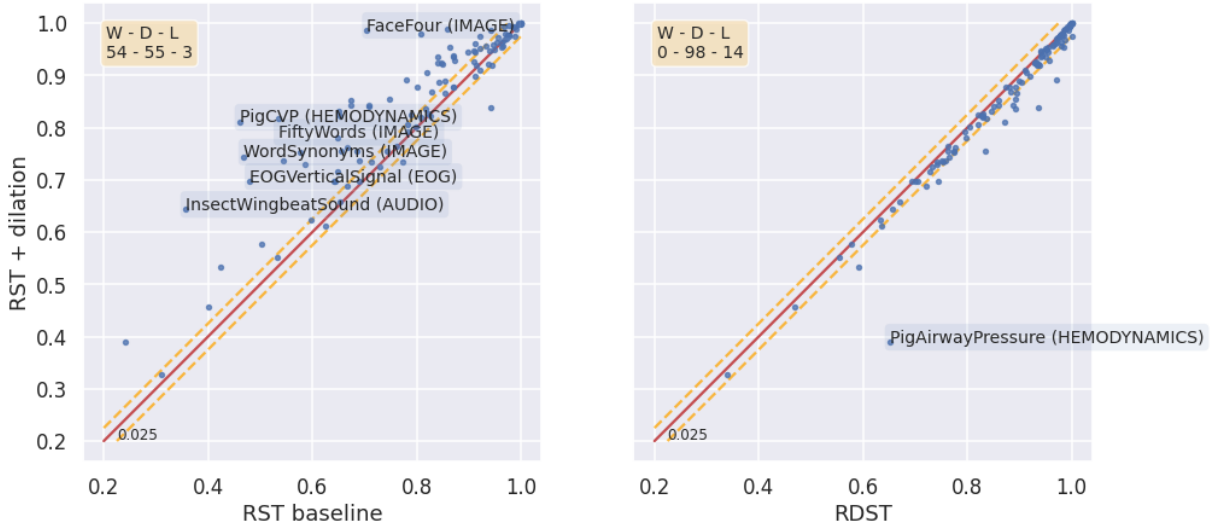

Figure 3.21: Pairwise accuracy plots for adding dilation against (a) RST baseline, which doesn't use dilation and (b) against RDST which use dilation and the Shapelet Occurrence feature, with a margin of +/- 2.5% of accuracy to count draws. Figure (b) share the y-axis of Figure (a).

Shapelet occurrence feature

The addition of the shapelet occurrence feature with the lambda threshold also represent a significative improvement compared to the simple baseline, as shown by Figure 3.22. Though, looking at the pairwise accuracy plot on Figure 3.23, we see that it has more important losses against RDST compared to the addition of dilation. The highest increase between the simple baseline and the version with lambda threshold is again on the PigCVP dataset, this time with a gain of 16.9% accuracy.

Figure 3.22: Critical difference diagram for a version with the lambda threshold against both baselines

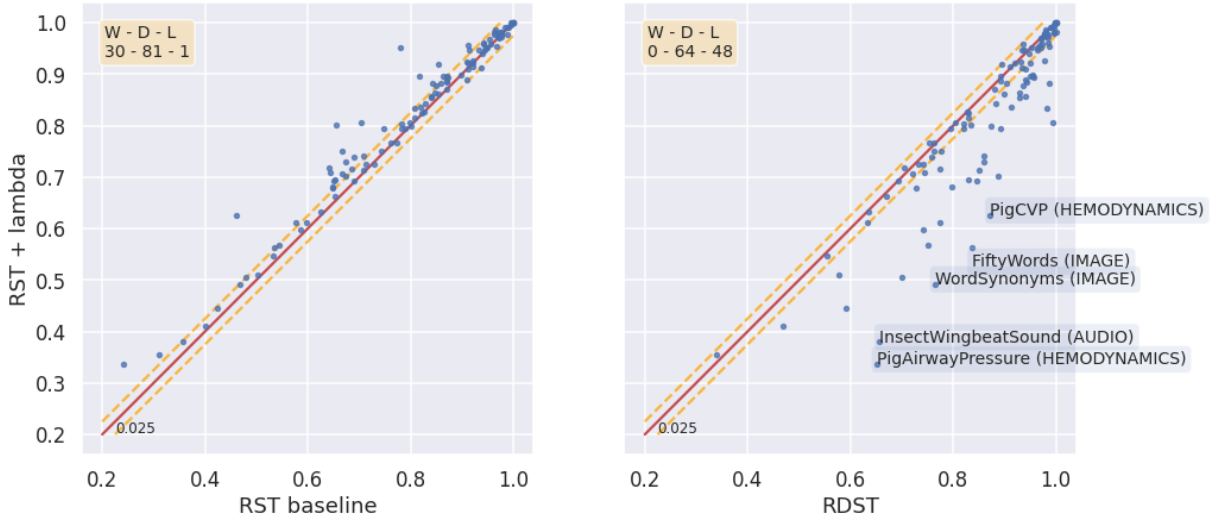

Figure 3.23: Pairwise accuracy plots for adding the lambda threshold to compute shapelet occurrence compared to the two baselines, with a margin of +/- 2.5% of accuracy to count draws. Both figures share the y-axis.

3.9.4 Shapelet sampling

We compare the notion of α-similarity with $\alpha = 1$, which is equivalent to the definition of self-similarity, and with $\alpha = 0.5$. Figure 3.24 shows that both variations give similar results, with $\alpha = 1$ and $\alpha = 0.5$, having a lower mean rank than RDST. Looking at the pairwise accuracy plot of Figure 3.25, built with a very small margin of 0.5% accuracy for draws, we see that the higher mean rank for RDST is only due to gains below this 0.5% threshold, confirming that both approaches are similar in terms of accuracy.

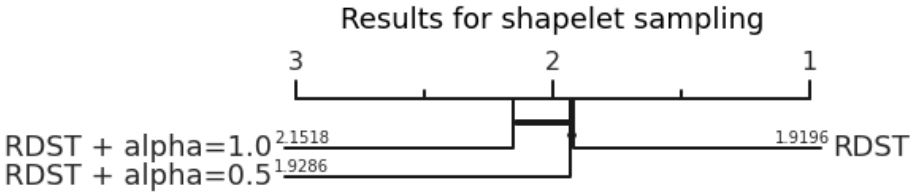

Figure 3.24: Critical difference diagram for RDST using shapelet sampling, with $\alpha = 0.5$ and $\alpha = 1.0$

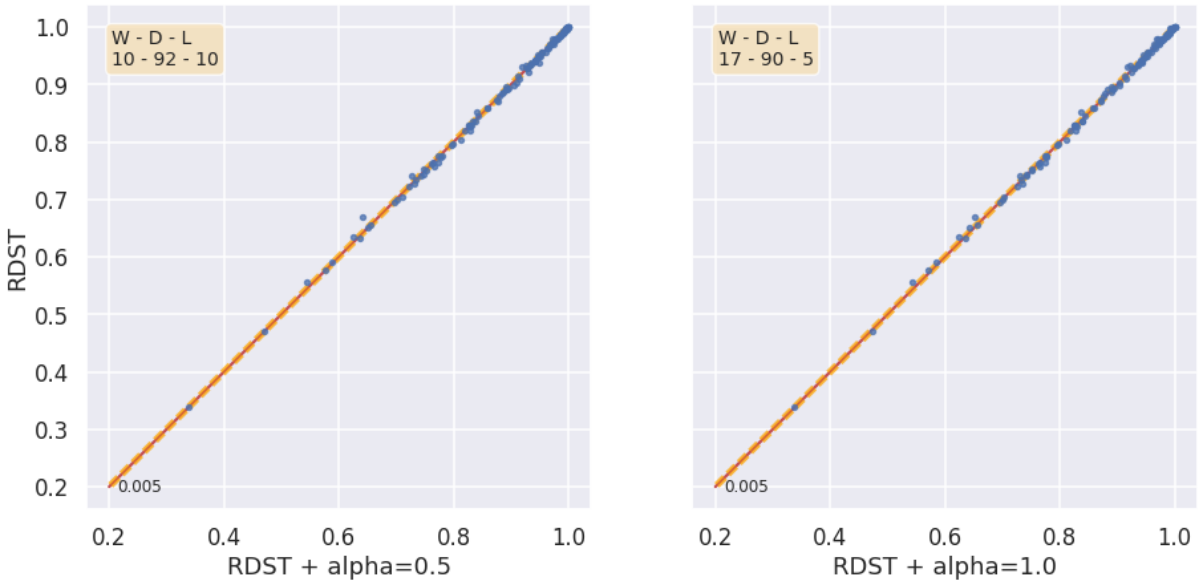

Figure 3.25: Pairwise accuracy plots for adding shapelet sampling, occurrence compared to the two baselines, with a margin of +/- 0.5% of accuracy to count draws.

Input subsampling

Similarly to shapelet sampling, we compare a stratified input subsampling (on the input time series) of 50% with shapelet sampling with $\alpha = 1$ and $\alpha = 0.5$. Using subsampling with $\alpha = 0.5$ do not produce any significant changes over the accuracy of RDST, as shown in Figure 3.26, but $\alpha = 1$ appears to be worse. The lower mean rank for a subsampling of 50% with $\alpha = 0.5$ is due to minor gains around 0.5% accuracy, as shown by the pairwise accuracy plot in Figure 3.27.

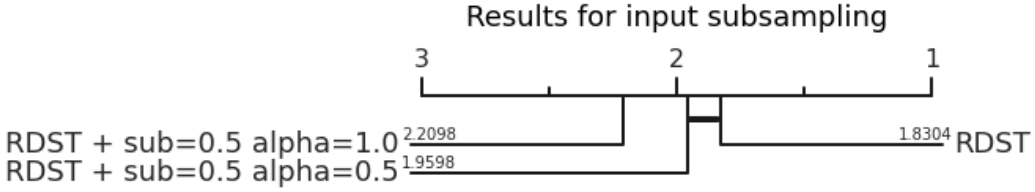

Figure 3.26: Critical difference diagram for RDST using input subsampling, with $\alpha = 0.5$ and $\alpha = 1.0$

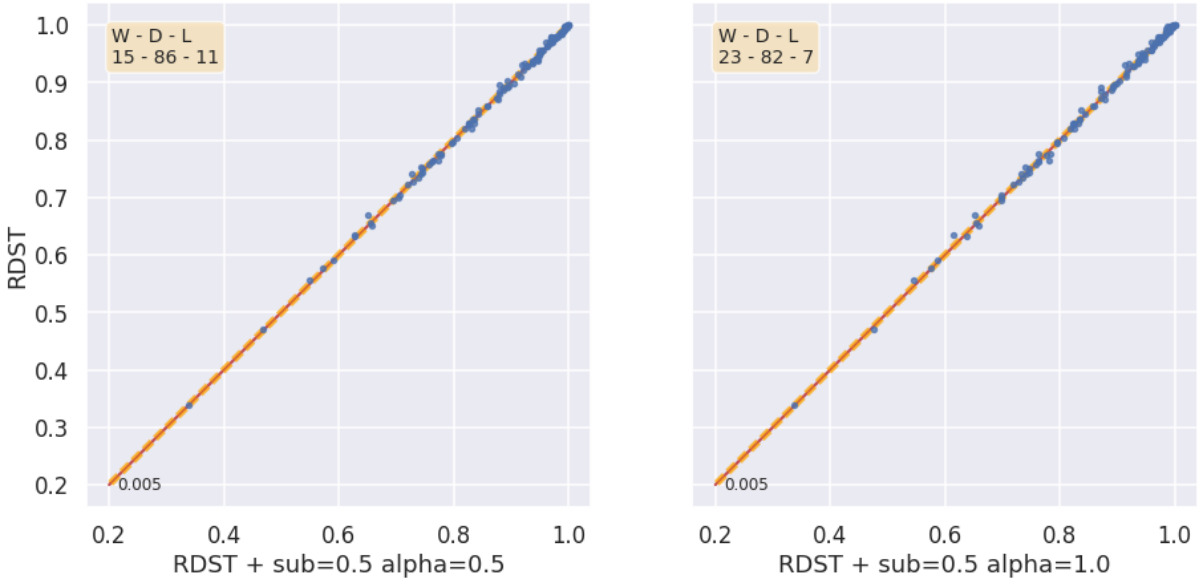

Figure 3.27: Pairwise accuracy plots for adding input subsampling, occurrence compared to the two baselines, with a margin of +/- 0.5% of accuracy to count draws.

Impact on run-time

While the addition of shapelet sampling and input subsampling does not improve the average accuracy, they have an impact on run-time. Figure 3.28 shows the sum of the total time needed to evaluate one resample of all datasets, which are grouped in bins on the total number of points ($n \times m$).

For small datasets, using α-similarity makes the method faster, this is due to a smaller number of shapelets being extracted compared to RDST. In small datasets, the search space of some combination of dilation and normalization is small enough for the whole Boolean mask used by α-similarity to be marked as False (i.e. no more sampling points are available), thus leading to the total number of shapelets being less than 10000. These advantages do not hold for bigger datasets, as the number of shapelets sampled will come closer to RDST as the whole Boolean mask cannot be False, and the added time complexity of the sampling techniques will not be compensated.

An intuition arising from those results might be that the sampling variations only have an advantage of accuracy for bigger datasets, due to their higher chance of sampling a more diversified set of shapelets. Table 3.4 gives accuracy results on the 3 biggest (for $n \times m$) datasets of the UCR archive for each sampling variation, and there is no difference in accuracy. In conclusion, α-similarity is a parameter to be tuned depending on the length of the data and on the number of shapelets to sample. Using input subsampling could help to further reduce the number of candidates, in order to have a higher chance of randomly sampling good shapelets, for big datasets without subclasses. While the positive impact of those parameters cannot be felt on the UCR archive, it might be worth considering other datasets.

Considering the above results, we use $\alpha = 0.5$ without input subsampling by default for RDST in the following. It does not significantly decrease the accuracy, but prevent

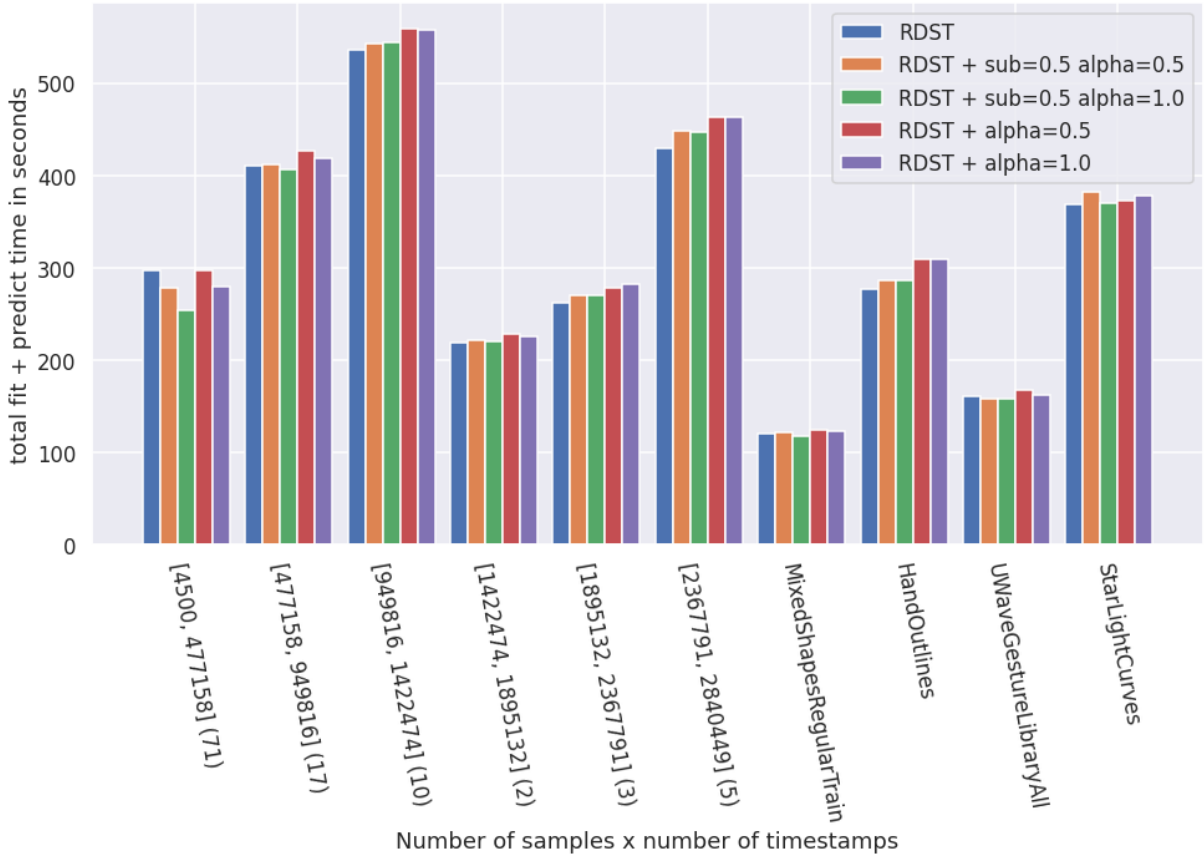

Figure 3.28: Total run times (fit+predict) for one resample of all UCR datasets for each sampling and subsampling variations. Dataset are grouped by bins depending on their number of samples times their number of timestamps. Each bin contains the sum of all run-times for datasets in this bin. The number in parentheses are the number of dataset in the bin.

Table 3.4: Mean accuracy results for 10 resamples for the 3 biggest (for $n \times m$) datasets of the UCR archive for all sampling and subsampling variations

	HandOutlines	StarLightCurves	UWaveGestureLibraryAll
RDST	0.955(+/-0.01)	0.980(+/-0.01)	0.981(+/-0.01)
$\text{RDST}_{sub=0.5\ alpha=0.5}$	0.952(+/-0.01)	0.980(+/-0.01)	0.981(+/-0.01)
$\text{RDST}_{sub=0.5\ alpha=1.0}$	0.952(+/-0.01)	0.980(+/-0.01)	0.981(+/-0.02)
$\text{RDST}_{alpha=1.0}$	0.952(+/-0.01)	0.980(+/-0.01)	0.981(+/-0.01)
$\text{RDST}_{alpha=0.5}$	0.952(+/-0.01)	0.980(+/-0.01)	0.981(+/-0.01)

the bad case scenarios of the fully random search of RDST, which could, considering the worst outcome, even if it is unlikely, sample the same point for all shapelets.

3.9.5 Invariance properties and robustness

We now look at the impact of adding either phase invariance (Phase) or robustness to complexity (CID) to our formulation. In terms of performance, Figure 3.29 does not show any significative change in the global accuracy for both additions. Locally, Figure 3.30 shows that phase invariance has a mean accuracy at least 2.5% higher on 3 datasets, which are Lightning7, BME and SemgHandMovementCh2.

BME is a synthetic data set with three classes: one class is characterized by a small positive bell at the beginning, the other at the end, and one does not have any bell. All series have a central plate, which may be positive or negative. Phase invariance may allow to sample shapelets that ignore the shape of the central plate and only focus on the start and the end of the signal. The other two datasets are power spectrums, which give the amplitudes associated with the frequencies composing the time series. For power spectrums, we suppose phase invariance helps by identifying discriminative patterns composed of both high and low frequencies (i.e. a pattern "looping" from the end to the beginning of the power spectrum).

Let us recall that the robustness to complexity multiplies the Euclidean distance between a shapelet and a subsequence by a correction factor, which for each vector computes the length of the "flattened" first order difference, and then computes a ratio between those two values, giving an estimate of the difference in complexity between the two vectors.

Adding this correction factor makes results worse by 4% on the SmoothSubspace dataset, which is a simulated dataset where each class is distinguished by a pattern at either the beginning, the middle, or the end, and the rest of the series are filled with random values. The length of the time series of this dataset being of 15 with discriminant patterns of size 5. As we use shapelets of size 11, the use of the correction factor may give worse results when the discriminant pattern is less represented than noise inside the shapelets. In the case of SmoothSubspace, the contribution of the noise (6 points) to the correction factor can be greater than the discriminant pattern (5 points).

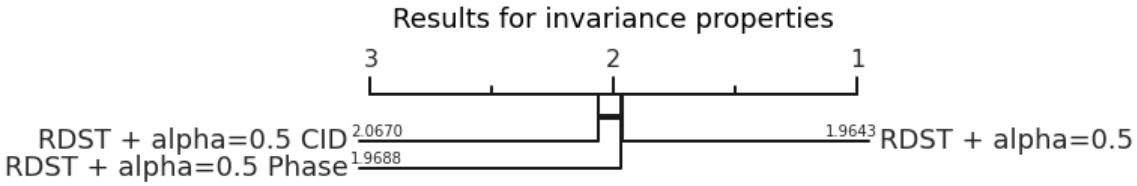

Figure 3.29: Critical difference diagram for RDST with phase invariance and RDST with complexity invariance against RDST.

In terms of impact on the run-time, given the total time needed to evaluate one resample of the 112 univariate datasets of the UCR archive, using RDST with this correction factor is 2.44 times slower than RDST, and phase invariance is 1.43 times slower. As robustness to complexity does not give any significative improvement on accuracy and adds noticeable time complexity, we choose not to include it.

Concerning phase invariance, it may be interesting when dealing with some types of data. Even if it does not show significative improvement on accuracy on the UCR archive, we think that increasing the diversity of cases that can be handled by the method is important. Additionally, the fact that phase invariance makes all distance vector $V_{S,X}$ of

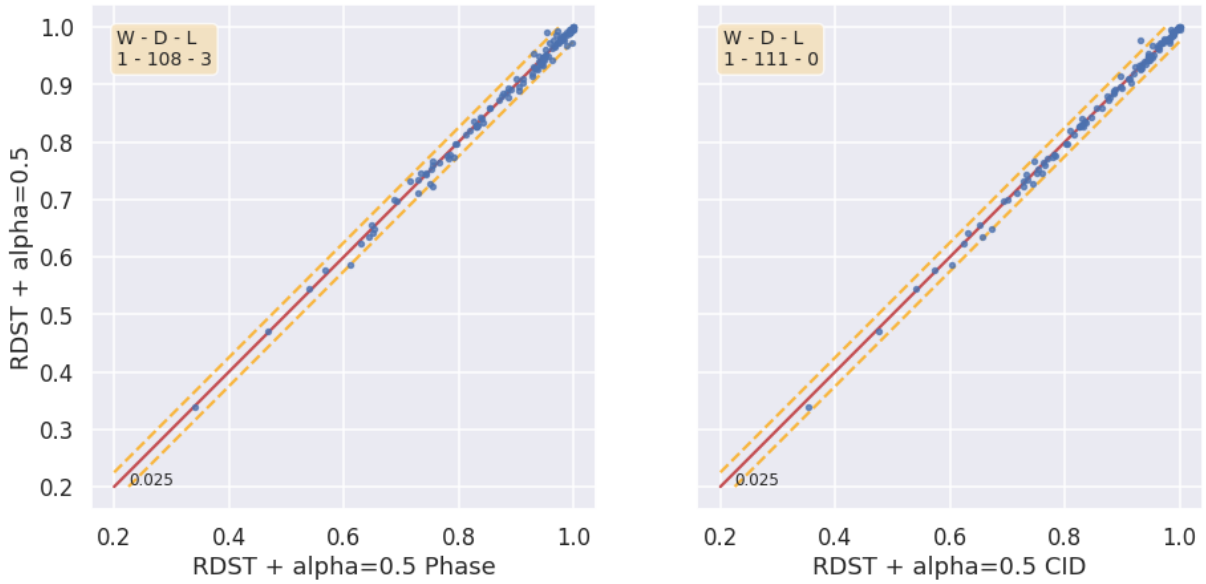

Figure 3.30: Pairwise accuracy plot for both phase and complexity invariance against RDST, with a margin of +/- 2.5% of accuracy to count draws.

size m, independently of the parameter of S, could be exploited in future works to provide further optimization of run-time.

In the following, we include **phase invariance** and **α-similarity**, with $\alpha = 0.5$, as **default** when using the term **RDST**.

3.9.6 Optimizing the λ threshold

To evaluate the performance of our heuristic to set the λ threshold, we compare it with a version of RDST that sets the λ threshold using information gain computed with the Gini impurity, testing all possible values and selecting the one that maximizes the information gain. Figure 3.31 shows the pairwise accuracy plot for the version optimizing the λ threshold (RDST + IFG) and RDST (both using phase invariance and α-similarity).

The use of information gain particularly increases the performance for one dataset where RDST fails to perform correctly compared to the state of the art, which is PigAirwayPressure. Although it has a better mean rank over all the datasets, the difference in accuracy is not statistically significative.

The impact on run-time, on the other hand, is extremely significant. Table 3.5 shows the increase in time needed to fit and predict a single resample of all 112 datasets, compared to the base formulation of RDST without phase invariance or α-similarity, denoted RDST_{BASE}. Using RDST + IFG cause an increase of 2111% compared to RDST_{BASE}, to only gain on the mean accuracy rank of the method, without significative differences. Considering those results, we consider our heuristic for the λ threshold as satisfying, and we keep it as the default option for RDST to preserve scalability.

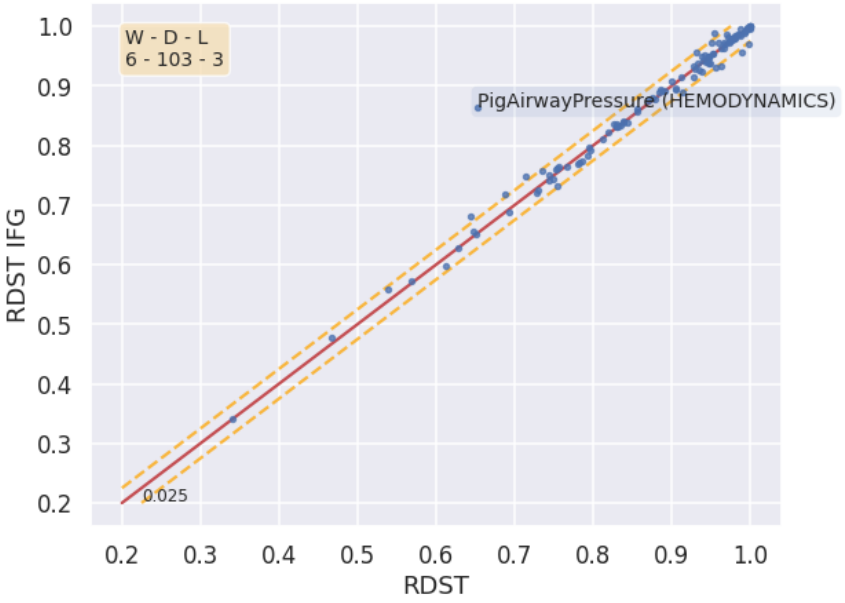

Figure 3.31: Pairwise accuracy plot for RDST using Information gain to set the λ threshold (RDST + IFG) against RDST, with a margin of +/- 2.5% of accuracy to count draws.

Table 3.5: Increase in time needed to fit and predict one resample for the 112 datasets of the UCR archive, compared to the base formulation of RDST without phase invariance or α-similarity, denoted RDST_{BASE}

	Run time	Compared to RDST_{BASE}
RDST_{BASE}	3081.43	+0.00%
RDST	4412.57	+43.1%
RDST_{IFG}	65076.28	+2111%

3.9.7 Time complexity

In this section, we study the impact of material acceleration against early abandon methods, and compare the scalability of RDST (with phase invariance and $\alpha = 0.5$), against state-of-the-art algorithms.

Impact of material acceleration

We compare the impact of early abandon, which stops the evaluation of the distance for a point $v_i \in V_{S,X}$ if at any step of the sum of pointwise differences, $v_i > \min\{v_1, \ldots, v_{i-1}\}$, against material acceleration, by using a vectorized sum as shown in Section 3.8.3. Figure 3.32 shows the total time needed to compute $V_{S,X}$ using a randomly sampled shapelet on 43 datasets of the UCR archive, averaged over 10 runs for 100 different shapelets and for different shapelet sizes (relative to the input length).

This experiment was performed with multiple variations, which all are implemented in pure Python, and with Numba [43], the Python compiler we introduced in Section 3.8. The variations we compare are :

- "no speed-up" : $V_{S,X}$ is fully computed,
- "early abandon" : the evaluation of $v_i \in V_{S,X}$ can be stopped if at any step of the distance computation $v_i > \min\{v_1, \ldots, v_{i-1}\}$
- "early abandon + random order": Similar to early abandon, but the point of $V_{S,X}$ are not computed in order as $v_1, \ldots, v_n$, but in a random order, for example $\{v_9, v_n, v_3, \ldots v_2\}$.

The idea behind early abandon with random ordering is that if we have $v_{i+1} < v_i \ \forall i \in [1, n[$, we will never skip any distance computation without random ordering. In other words, on average, it increases the number of operations skipped by early abandon, as shown by [10]. Vectorization can only be used in "no speed-up Numba", as early abandon require a check on each step of the distance computation, which makes it impossible to simultaneously compute part of the sum with vectorization as each step depends on the previous ones.

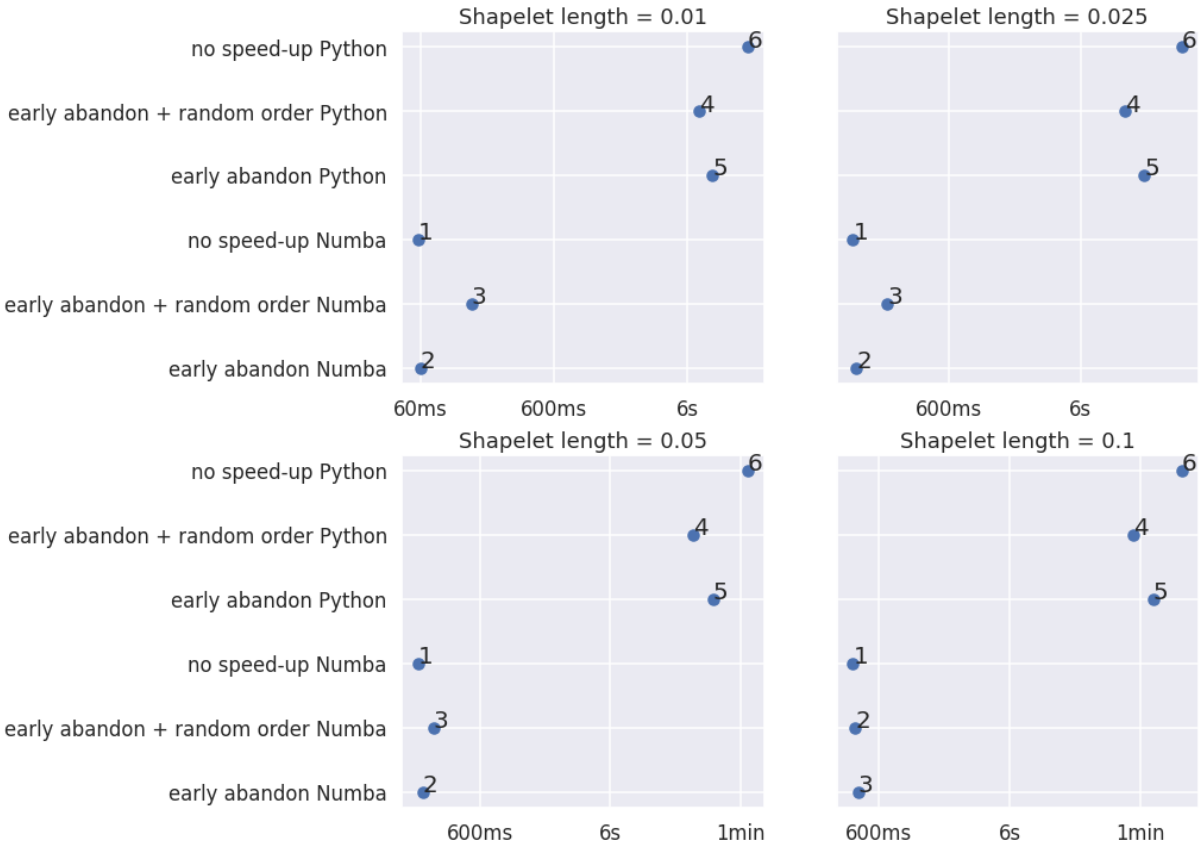

Figure 3.32: Total time to slide a randomly sampled shapelet, for different shapelet sizes (relative to the input length as $\lfloor m \times l \rfloor$), on 43 datasets of the UCR archive, averaged over 10 runs for 100 different shapelets. The number next to each point is its rank.

We see that, for the Python implementation, the early abandon with random order is always faster than the two other Python methods, but using any early abandon technique does not seem to be faster than using vectorization, as the "no speed-up Numba" is always

faster for Numba methods. This is an important result, because the more diverse the set of features extracted from the distance vector, the harder it becomes to apply early abandon techniques. This result shows that we can ignore early abandon in our formulation without loosing performance for our experiments on the UCR archive, as we rely on optimization made at the compiler level.

Another trend in these results is that the early abandon with random order becomes more efficient as the shapelet size grows. This does not affect the result of our experiments, as we use a fixed length for all dataset.

Scalability of RDST

We perform a comparison of the scalability of RDST (with $\alpha = 0.5$ and phase invariance) against Hive-Cote 1.0 (HC1), Hive-Cote 2.0 (HC2), DrCIF, ROCKET, and the Shapelet Transform Classifier (STC). Note that when used as a component in HC1 and HC2, STC is by default subject to a time contract of two hours. Except from this default configuration in HC1 and HC2, we are not setting any time contract in other algorithms. Both STC and RDST are by default sampling 10000 shapelets, and ROCKET use 10000 kernels.

We are aware that the runtime of HC1, HC2 and STC could be reduced with time contracts. But, as our goal in this section is to contextualize the gain in classification accuracy against the time complexity of each method, we present these results with the parameters used to generate the accuracy results of the next section.

We use the Crop Dataset and the Rock Dataset of the UCR archive for evaluating the scalability respectively on the number of time series and their length. As all competing algorithms implemented in the sktime package [54] can use parallel processing, we set each algorithm to use 90 cores. Figure 3.33 reports the mean training time over 10 resamples, showing the very competitive scalability of RDST, even with the added time complexity of the α-similarity and phase invariance.

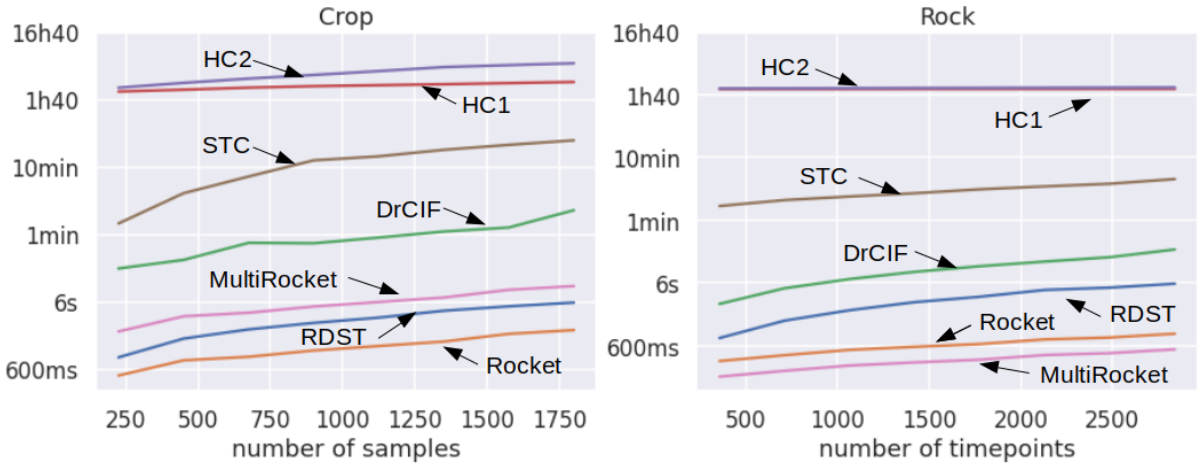

Figure 3.33: Result of the scalability study of the competing algorithms for current state-of-the-art, for (a) number of time series and (b) time series of increasing length. The Y-axis uses log-scale.

3.9.8 Comparison to the state of the art

Results for univariate datasets

We present the results of our comparative study using the mean accuracy over 30 resamples for each of the 112 datasets, and compare our approach against the state of the art. Figure 3.34 gives the critical diagram resulting from this experiment, and Figure 3.35 gives a pairwise accuracy plot for RDST against Hive Cote 2.0 (HC2) and MultiRocket. Figures splitting the critical difference diagram in two, for datasets used in the sensitivity analysis and the others, are given in Appendix A.

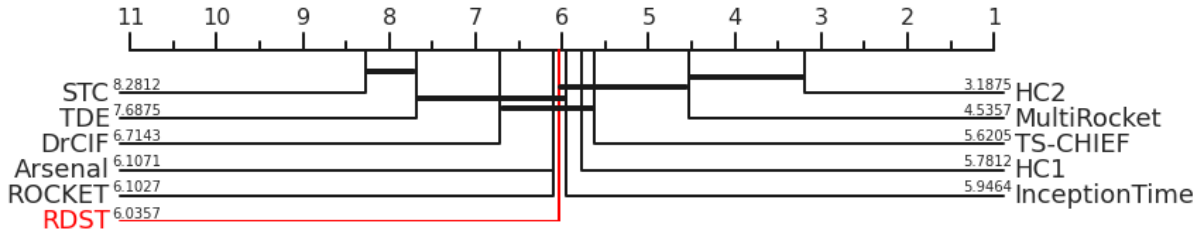

Figure 3.34: Mean accuracy ranks of each method for the 112 datasets of the UCR archive, including RDST.

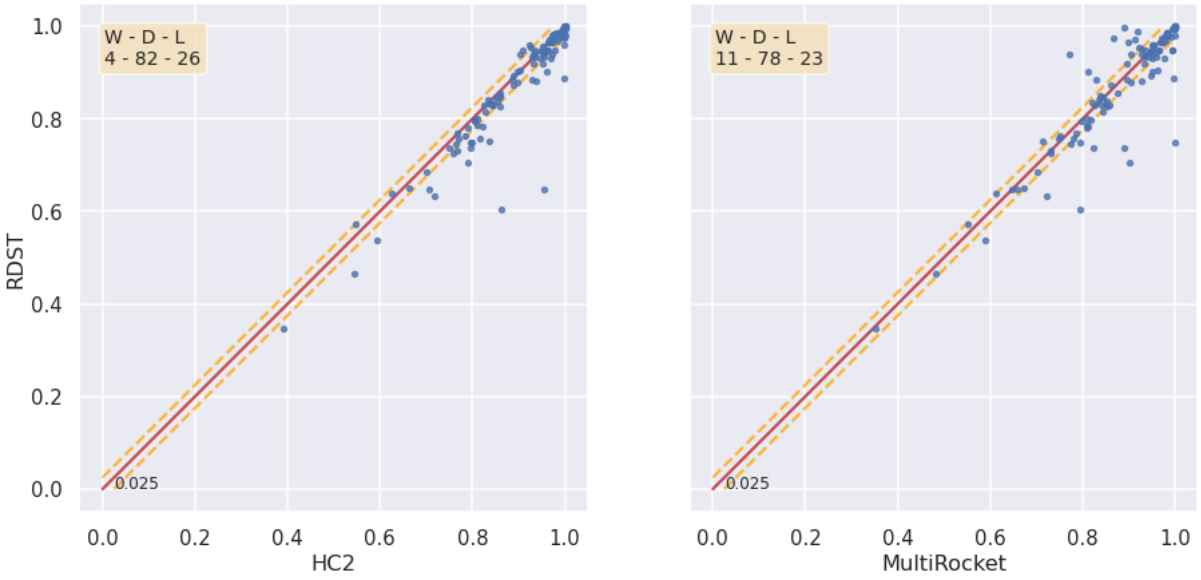

Figure 3.35: Pairwise accuracy plot for the 112 datasets, for HC2 and MultiRocket against RDST, with a margin of +/- 2.5% of accuracy to count draws.

Given the scalability of RDST, having an accuracy comparable to the prior developments of HC2, ROCKET and deep learning approaches (InceptionTime) is a promising result for a shapelet-based method. For reference, using RDST without any distance normalization is equivalent to STC in terms of mean accuracy rank, with the same protocol as above.

Ensemble scheme

Similarly to Supervised Time Series Forest [14], we can create a simple ensemble scheme by using different representations of the inputs, in order to enhance the range of discriminative features we can extract from the data. To adapt RDST as an ensemble scheme, we apply RDST independently to the following representations:

- Base representation: to focus on patterns already existing in the original representation of the inputs.
- First order difference: the rate of changes between points can give additional information such as the slope of the series. It may also highlight some outliers or patterns compared to the base representation.
- Periodogram: a periodogram is an estimate of the spectral density of a signal, which by relying on Fourier analysis, quantify the relative strengths (i.e. power) of the various frequencies composing the signal.

For the voting scheme, we use the weighted majority voting system described in HIVE COTE 2.0 [58], which weights each classifier by its accuracy using a cross validation setting on the training data. This accuracy score (between 0 and 1) is then raised to the power 4 to penalize classifiers with low accuracy. We denote this method by "RDST Ensemble" in the following, and use a leave-one-out cross validation on the training data to estimate the weight of each representation. Figure 3.36 gives a visual representation of how RDST ensemble perform class prediction for a time series X

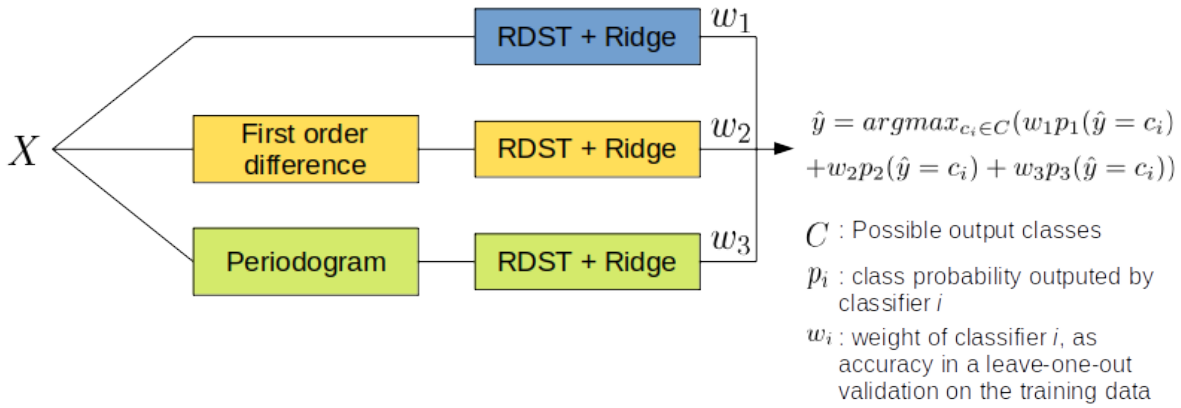

Figure 3.36: Visual representation of RDST ensemble, and how it predicts the class $\hat{y}$ of a time serie X.

Figure 3.37 gives the critical diagram for the 112 univariate datasets of the UCR archive, and 3.38 the pairwise accuracy plot for RDST Ensemble against MultiRocket and HC2. They show that the ensemble scheme improves the performance of the method, up to the point where it is considered to be equivalent to HC2 and MultiRocket by the statistical test of the critical diagram.

In terms of impact on run-time, the non-ensemble version of RDST takes 4412 seconds to fit and predict a single split of the 112 datasets, while RDST Ensemble takes 5470

seconds. This represents an increase of about 24% in runtime, which still keeps it amongst the fastest algorithms in the state of the art. This low increase in runtime between the two is due to the fact that we launch 3 independent processes each managing their own thread pool, and that we re-use the results from the Ridge Classifier which internally uses a leave-one-out validation to choose the strength of the L2 regularization.

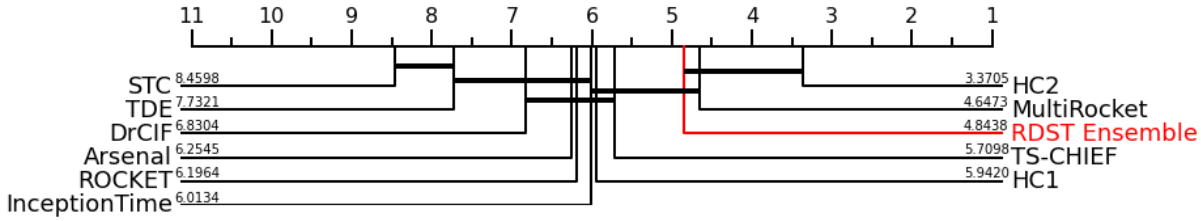

Figure 3.37: Mean accuracy ranks of each method for the 112 datasets of the UCR archive, including RDST Ensemble.

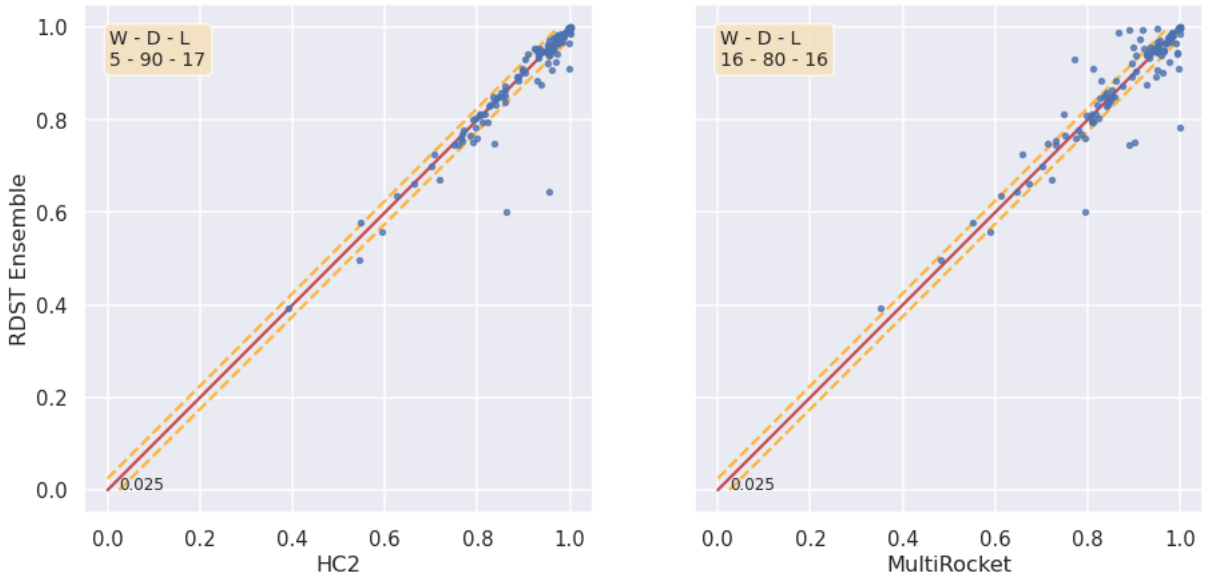

Figure 3.38: Pairwise accuracy plot for the 112 datasets, for HC2 and MultiRocket against RDST Ensemble, with a margin of +/- 2.5% of accuracy to count draws.

Results for multivariate datasets

The results of our comparative study for multivariate datasets, also using the mean accuracy over 30 resamples for each of the 26 datasets, rank RDST ensemble similarly to the univariate comparison. Figure 3.39 gives the critical diagram resulting from this experiment, and Figure 3.40 gives a pairwise accuracy plot for RDST against Hive Cote 2.0 (HC2) and Arsenal.

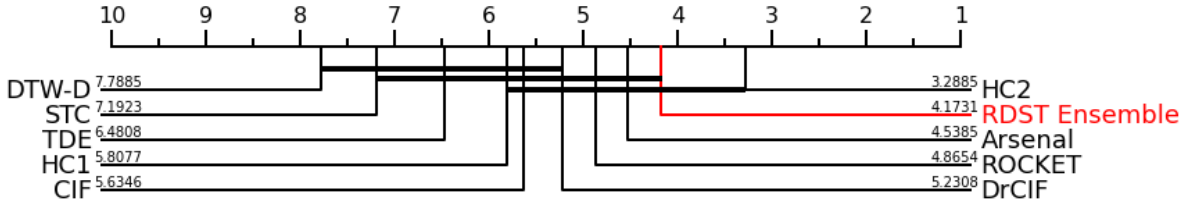

Figure 3.39: Mean accuracy ranks of each method for the 26 multivariate datasets of the UEA archive, including RDST Ensemble.

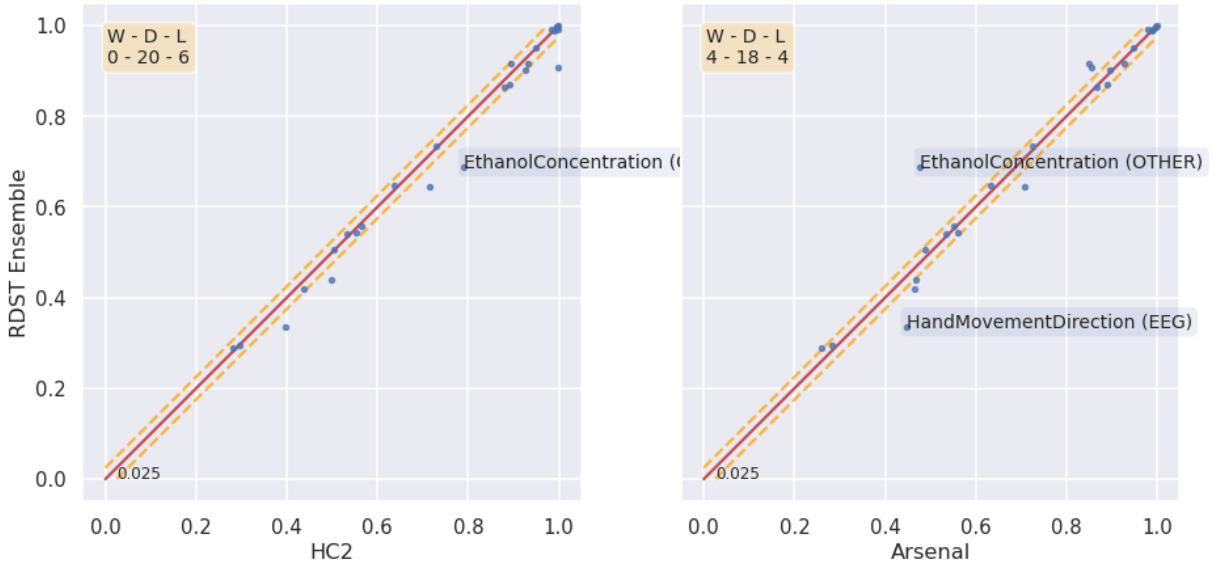

Figure 3.40: Pairwise accuracy plot for RDST Ensemble Against HC2 and Arsenal, for the 26 multivariate dataset of the UEA archive.

3.10 Conclusion

In this chapter, we introduce a new shapelet based classification method, the Random Dilated Shapelet Transform (RDST), as well as potential improvements for it through the addition of phase invariance, input sampling techniques and an ensemble scheme, leveraging different representations of the input, to further increase the performance of the method. We showed, based on our experimental results, that this method is comparable to state-of-the-art classifiers for time series classification, while also being amongst the fastest. We also released our method as part of a Python package[1] available to the public, allowing all our experiments to be reproduced.

Many other improvements, notably on the scalability of the method, can be considered for future works due to the addition of the phase invariance. As it makes all distance vector $V_{S,X}$ of size m, independently of the parameters of S, additional optimization could be explored in future works, such as transitioning to large matrix operations on GPUs for extracting the features.

Other directions for increasing the accuracy of the method should focus on the shapelet

[1] https://github.com/baraline/convst

generation part. Exploring the search space for good candidate shapelets includes searching for a combination of length, dilation, normalization, λ threshold, and the value of the shapelet itself, which would result in an extremely slow method using an exhaustive search. In future works, we want to focus on heuristics that would be able to estimate the value of some of these parameters adaptively from the data.

More work is also needed on the adaptation to the multivariate context, particularly on the shapelet generation and sampling, where solutions, rather than to be adapted from the univariate context to more dimensions, must be specifically designed for the multivariate context, which is more complex.

Chapter 4

Predictive Maintenance

Résumé du chapitre en français
Dans ce chapitre, nous introduisons le domaine de la maintenance prédictive, et de montrons comment on peut extraire un jeu de donnée adapté à une tâche de maintenance prédictive, à partir des données émises par une machine. Ensuite, nous montrons comment la tâche d'apprentissage est formulée pour ce type d'application, aussi bien pour les approches de régression que de classification. Après cela, nous explorons les différentes approches utilisées dans la littérature pour créer un modèle de maintenance prédictive, ainsi que les différences entre les applications utilisant des données de capteurs et des journaux d'événements. Nous définissions ensuite un protocole expérimental adapté aux problématiques de la maintenance prédictive, comprenant différents contextes d'évaluations pour les modèles, ainsi qu'une méthode d'estimation du coût d'un système de maintenance prédictive. Enfin, nous introduisons les domaines de la classification précoce et de l'analyse de survie, qui seront utilisés dans nos contributions.

Chapter summary
In this chapter, we introduce the field of predictive maintenance and define a methodology to extract datasets adapted to predictive maintenance tasks, from the data emitted by a machine. We start with an introduction of the predictive maintenance problem and how we can extract a predictive maintenance dataset from the data emitted by a machine. We then give the formulation of the learning task related to the problem, for both regression or classification methods. We then present the different approaches used in the literature to create a predictive model for maintenance, as well as the differences between applications using sensor data and event logs. We then define an experimental protocol adapted to predictive maintenance applications, the different contexts of evaluation for predictive models, as well as a method to estimate the cost of a predictive maintenance system. We then introduce the fields of early classification and survival analysis, which will be used in our contributions.

The main goal of predictive maintenance is to increase machine availability by minimizing unplanned maintenance caused by machine failures. The knowledge of when and why a machine is going to fail offers many benefits, such as a better maintenance planning, spare part management, and other cost optimizations related to the maintenance process. Other benefits can be obtained depending on the use case: for example, in a

factory predictive maintenance can lead to increased productivity, whereas in a hospital it can reduce the downtime of medical equipments that may be critical to patient health.

The use case that motivated the thesis was a case of predictive maintenance for a fleet of Automated Teller Machines (ATM). Given a set of machines from which we can only collect software logs, we want to build a predictive maintenance model, capable of predicting when and which module of the machine is going to fail. In this chapter, we introduce the problem formulation and present the field of predictive maintenance, with the different approaches used in the literature to solve predictive maintenance problems. We then present the field of Early classification and Survival analysis, which both present an interest for predictive maintenance applications and will be used in our contribution.

4.1 Problem formulation

In this section, we first introduce a problem formulation for predictive maintenance, which tries to unify the formulations used across the literature. Although it may need to be adapted for some specific tasks, we believe it is general enough to give the reader a good comprehension of the problematics behind predictive maintenance applications.

Given a set of monitored machines, either with sensors to measure physical phenomenons or by extracting software logs, the objective of a predictive maintenance model is to raise an alert at a time t when a machine is going to fail at a time $T > t$. To make this decision, the model uses the data emitted by a machine up to the time t. Ideally, the alert should be given early enough for the maintenance team to perform maintenance before the actual failure, but not too early, to avoid performing an unnecessary maintenance.

The ideal time for a maintenance alert to be raised depends on multiple factors, such as the maintenance costs, the costs linked to the immobilization of the machine, etc. We introduce a methodology to evaluate the cost of a maintenance alert in Section 4.3. This cost can then be used to estimate the optimal time for a maintenance alert and to estimate the cost of a predictive maintenance system during the evaluation of a model.

Note that in this chapter, we will consider time series X to be of the same length m to simplify the notations. Due to the nature of the application, predictive maintenance data will most often be of variable length. Every step of the data processing we present can nevertheless be adapted to variable length time series without much difficulty.

4.1.1 Extracting a predictive maintenance dataset

Given the raw data emitted by a monitored machine, the first step toward building a model to predict failures is to extract a dataset suited for this task. Let X a time series $X = \{x_1, \ldots, x_m\}$, representing the data emitted by a machine up to time m. If the machine had at least one failure during the time interval $[1, m]$, we can use X to extract time series for a predictive maintenance dataset. Before extracting these series, we need to answer some questions about the task we are trying to solve:

1. Should we predict which module will be responsible for the failure?

2. If X is multivariate, for example with a sensor measuring temperature and another measuring pressure for multiple modules, are these features independent ?

3. Do we have detailed maintenance logs allowing to know the cause of the failures and the operations performed during the maintenances?

We call life cycles the subsequences extracted from X, that contain the data emitted by a machine that is correctly functioning, starting after the end of the last failure, up to the next failure. A life cycle is defined by a machine from which it is extracted, a time interval, and if the cause of failure is known, a module. Depending on the characteristics of the data, the process used to extract life cycle will vary. For example, if we have sensors measuring temperature for neighbouring modules in a machine, it is likely that a rise of temperature caused by one module will influence the value of the other sensors. In such a case, isolating each module in independent life cycles only contain the data emitted by each module could lead to false positives, due to changes in temperature possibly caused by other modules.

Consider $X = \{x_1, \ldots, x_m\}$ the data emitted by a machine, that had two failures, each defined by a start and end time (i.e. the moment of failure and the end of the maintenance), as $[t_1, t_2[$ for the first failure and $[t_3, t_4[$ for the second, we can extract three life cycles. The first one would be $X_1 = \{x_1, \ldots, x_{t_1-1}\}$, the second $X_2 = \{x_{t_2}, \ldots, x_{t_3-1}\}$, and the third $X_3 = \{x_{t_4}, \ldots, x_m\}$. In this case X_3 does not end by a failure, and may not be useable by some approaches due to the uncertainty of the failure time. An example could be a regression model that would need to know the time remaining before failure at each timestamp to use X_3 during the training phase. Then, if the cause of failure is known, we can affect it to its corresponding life cycles.

Finally, the formulation of the learning task will depend on the need to specify the module responsible for the failure when raising maintenance alerts and of the type of model used. For example, if we need to specify the cause of failure and we use a classification model, we will need to formulate a multiclass problem or divide it into multiple binary one versus all problems.

In the following, we show how the learning task can be formulated for both classification and regression tasks, in the case where we **do not need to** predict the module responsible for failure. We will show in Chapter 5 how the learning task can be adapted to the case where the module causing the failure need to be identified. In the next sections, we will use the notion of moving windows, which are subsequences extracted from a time series given a length parameter l. For example, given a time series $X = \{x_1, \ldots, x_m\}$, we can extract a set of moving windows $\mathcal{W} = \{W_1, \ldots, W_{m-(l-1)}\}$, with $W_i = \{x_i, \ldots, x_{i+(l-1)}\}$.

4.1.2 Regression tasks

In a regression context, the goal is either to estimate the remaining time before the next failure event, called the remaining useful life (RUL), or to estimate a health indicator, which can be seen as a probability of failure. To raise predictive maintenance alerts, such systems have to define thresholds on the predicted quantity below which the maintenance process should be initiated.

For the case of RUL estimation, given a set of life cycles, the data used to learn the estimator is most often extracted using a moving window scheme, with l the length of each window W_i. A window W_i is then labelled by the remaining time before failure, as shown by Figure 4.1.

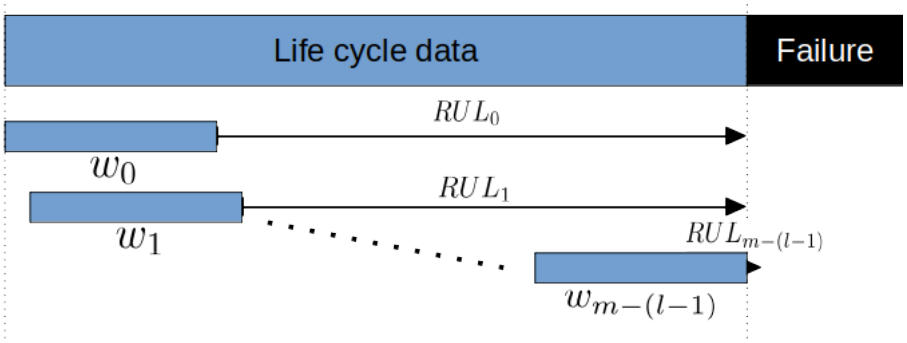

Figure 4.1: How a time series is sliced into multiple windows to create a dataset for a regression task. Each window has for target value the remaining time before failure and is of size l.

More formally, given a life cycle $X = \{x_1, \ldots, x_m\}$, with the machine failing at time t_{m+1}, and a window size l, a set of moving windows $\mathcal{W} = \{W_1, \ldots, W_{m-(l-1)}\}$ is extracted, with $W_i = \{x_i, \ldots, x_{i+(l-1)}\}$ one window and its target value $y_i = (m+1) - (i + (l-1))$, which represents the remaining time before failure. Multiple approaches exist to build an estimator based on this data, one example could be to extract statistics from each window and fit a linear regression. Once a regression model has been fitted, when new data is received from a machine (either online or in batch), the most l recent data points are used to extract the same features and to make a prediction.

4.1.3 Classification tasks

In a classification context, the goal is to estimate whether, at time t, a predictive maintenance alert should be raised. While a classification model for predictive maintenance typically does not need to define thresholds as a regression model does, it must label data in such a way that the model learns to identify data that is linked to a soon-to-happen failure. Given a set of moving windows $\mathcal{W} = \{W_1, \ldots, W_{m-(l-1)}\}$ extracted from a life cycle X, the objective then becomes to choose how a window W_i should be labelled relative to its position to the end of X. This problem is usually solved by defining one or multiple time intervals, which take into account the time needed to perform maintenance once an alert is raised.

There does not seem to exist a consensus for the definition of those intervals in the literature, we define those intervals based on the work of [73]. Let us consider a life cycle $X = \{x_1, \ldots, x_m\}$ of length m, with a failure occurring at time t_{m+1}. Similarly to regression, a set of moving windows of size l, $\mathcal{W} = \{W_1, \ldots, W_{m-(l-1)}\}$ is extracted. To process $\mathcal{W}$, let us first consider three time intervals related to the predictive maintenance problem we are trying to solve:

- **Responsive Duration** (rd): a time interval reflecting the real-life time needed to perform maintenance from the moment an alert is raised. This includes the time needed to get on site and the time needed to perform maintenance.
- **Predictive Padding** (pp): a time interval reflecting how early, relative to the occurrence of the failure, it is acceptable to raise a maintenance alert. An alert

raised in this interval should give enough time to plan a maintenance operation at an optimal time (e.g. when the machine is not used).

- **Infected Interval** (ii): a time interval that reflects the time needed to return to normal behaviour after the end of the maintenance process. This interval is used to remove potentially noisy data caused by a restart process.

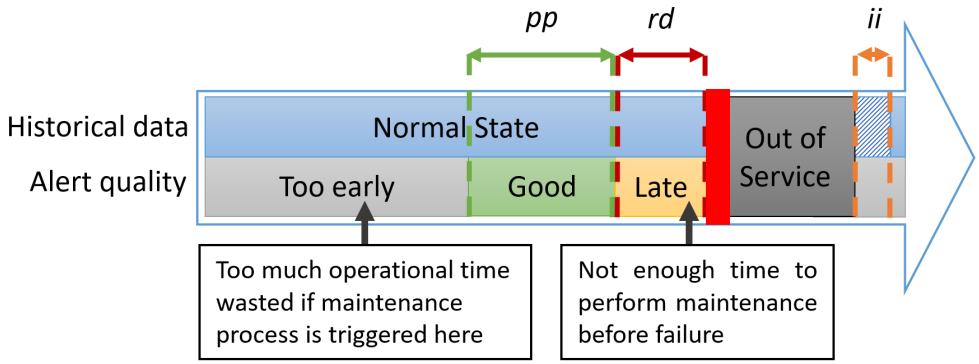

Figure 4.2: Visualization of the three time intervals used by classification models for predictive maintenance. The red bar represents a failure.

Figure 4.2 gives a visual representation of those intervals on an abstract timeline. The Responsive Duration (rd) is placed right before the failure to illustrate that, if an alert is raised in this interval, it is unlikely that we will be able to schedule a maintenance operation before the failure happens.

Given those three intervals, we can now define how we process the set of moving windows $\mathcal{W} = \{W_1, \ldots, W_{m-(l-1)}\}$ extracted from a life cycle X with a failure occurring at time t_{m+1}. In a binary classification context, where we predict failure independently of the cause, and where class 1 represents windows where we should raise an alert, the class y_i of window $W_i = \{x_i, \ldots, x_{i+(l-1)}\}$ is defined as:

$$y_i = \begin{cases} 1 & \text{if } i + (l-1) \geq (m+1) - (pp + rd) \\ 0 & otherwise \end{cases} \tag{4.1}$$

Figure 4.3 illustrates this labelling process for one life cycle. Windows ending in the interval $[(m+1)-(pp+rd), (m+1)-rd]$ can be used to train the model if they may contain failure patterns that could occur before $(m+1)-(pp+rd)$. But when evaluating the model, an alert raised in this interval should not be considered successful, as it does not give enough time to perform maintenance. Concerning the infected interval ii, windows touching the interval $[1, ii]$ should be discarded if the restart processes of the machine produce noisy data.

As for regression, multiple approaches exists to train a classifier from this set of labelled moving windows. A common practice in the literature is to extract features from each window and use a tabular classifier, such as a random forest. Then, when new data is received from a machine (either online or in batch), the most l recent data points are used to extract the same features as those used to train the model, and make a prediction.

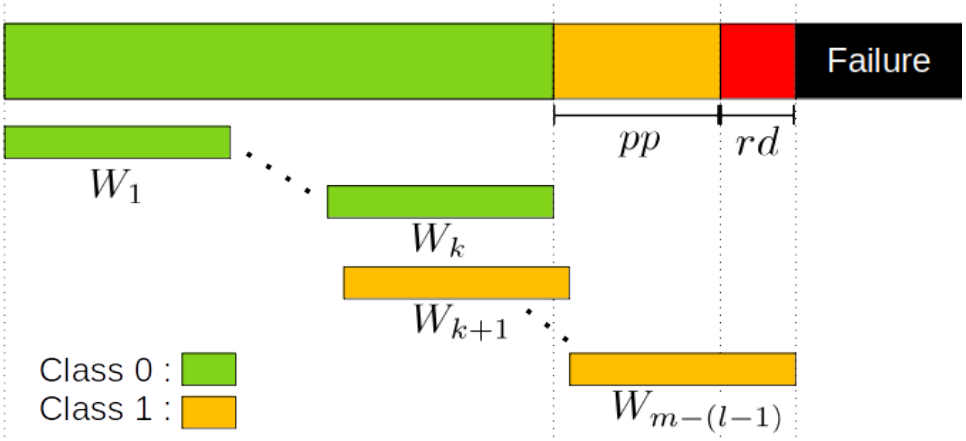

Figure 4.3: Illustration of the problem formulation commonly used in the literature for classification tasks. W_i is the i^{th} window of size l extracted from a life cycle, rd is the time needed by the maintenance team to perform maintenance and pp is the acceptable interval for maintenance alerts.

4.2 Related work

The field of predictive maintenance has been very active in the past decade. Multiple recent surveys on predictive maintenance [42, 28, 61] show the diversity of solutions used to solve predictive maintenance problems. The common point of those approaches is that they consider temporal data, which are the most common inputs for predictive maintenance problems.

The predictive maintenance literature being mainly composed of applied papers, most proposed approaches are designed and fine-tuned specifically for the use case studied in the papers. Those methods nevertheless share similarities within families of methods. Hence, in the following, we do not aim at giving a review of all existing use-case specific models, but we rather introduce the three main approaches used to solve predictive maintenance use cases in the literature. As the use case we present in the next chapter relies on log data, we also discuss the differences between sensor and log data for predictive maintenance.

4.2.1 Pattern mining approaches

Pattern mining methods for predictive maintenance aims at finding patterns that are relevant to predict a soon-to-happen failure. This could be, for example, a particular sequence of warning and error messages. Once good candidate patterns have been found, if they are found again in recent data emitted by a machine, a predictive maintenance alert can be raised.

In recent years, pattern-based predictive maintenance has switched from the sequential pattern mining paradigm to chronicle mining, introduced by [26] and more recently, in a generalized version by [20]. One of the main shortcomings of sequential patterns is that they only inform about the occurrence of a series of events, but they do not take into account the time between those events. Chronicle mining algorithms are able to search for rules taking into account a set of time constraints, specified by time intervals, between

each event.

Many applications have been published using chronicles as a way of extracting features or building statistical models to solve predictive maintenance problems. For example, [70] defines the notion of failure chronicles as time-constrained sequences of events which frequently appear before a failure. It then proposes an algorithm to find such patterns in a dataset with labelled failures. Other approaches [16] combine chronicles with domain ontologies, which provide insight on how the data is related to the degradation process of the machine, and perform ontology reasoning (e.g. rules based on the nodes and links of the ontology) to detect anomalies and predict failures. There also exist semi-supervised approaches for anomaly detection, such as the one of [21], where using logs from data centers, the authors propose a system to manage in real-time an evolving rule-based system capable of identifying deviation from the normal behaviour of the machines.

To summarize, the main goal of pattern based methods is to identify a set of rules, possibly with temporal constraints, which are significative of either an abnormal behaviour, or are known to happen before failures. Predictive alerts are then raised when a sequence of events known to lead to a failure is detected, or with a system of thresholds based on an anomaly score.

4.2.2 Statistical model approaches

The second family of methods focuses on building statistical models, either to estimate a remaining time before failure, called Remaining Useful Life (RUL), or to estimate a health indicator by modelling the degradation process of monitored machines. A popular technique often seen in industrial applications is survival analysis, also known as hazard-rate models [55], which, based on historical data on the duration of life cycles, is able to estimate the survival probability or the RUL of a life cycle at a given time. The most common approach used by such a system is to define a threshold on the RUL, below which a maintenance process will be initiated. We present the field of survival analysis more in depth in Section 4.5.

We also find in the literature the use of autoregressive models [7] which, in addition to a set of statistics extracted from moving windows, use ARMA models to forecast the evolution of those statistics in future windows. The extracted statistics and the forecasted ones for each window are then used to build a regression model to estimate the RUL of a machine in a regression context. Change point detection models [37] are also used for predictive maintenance, based on the assumption that a change in the data distribution emitted by a machine can be significative of a soon-to-happen failure.

In general, statistical approaches will try to model the behaviour of the monitored machine to identify deviations from normal behaviour or to estimate the remaining useful life (RUL) of a machine based on historical lifespans of machines. Those models may also include the influence of physical factors (e.g. temperature, vibration, etc.) on the expected RUL.

4.2.3 Machine learning approaches

In the last family of methods, we consider approaches which use machine learning algorithms in a supervised context. A majority of methods, independently of using a regression

or a classification context, use sliding windows to extract features and then use machine learning algorithms such as Random Forests, or to extract same length windows on which time series classification or regression algorithms can be applied.

For example, [80] slices time series of event logs using a moving window, and then extract statistics on the number of occurrences of events and the occurrences of error patterns before using an ensemble of tree-based classifier to predict whether the current window is close to a failure event. Similarly, [6] uses LSTM Auto-Encoders to predict the current state of a machine, between an "initial", an "intermediate" and an "bad" state, which indicates a need for a maintenance operation.

The choice of the right approach, and more importantly the formulation of the problem, depend on multiple factors such as the knowledge of the degradation process of the machines (e.g., linear, exponential, random, unknown), the properties of the time series, the business constraints, and the quantity and quality of labelled data.

4.2.4 Predictive maintenance for log data

Log data are records of events, along with timestamps, which happen on a system or a software application (i.e. a host). They are automatically produced by the host in order to provide information for either maintenance or audit purposes, so that we can understand the activity and actions taken by the host and diagnose problems.

When used in the context of predictive maintenance, logs are often designed around severity levels, such as OK, Warning and Error, indicating respectively a normal or expected activity, a non-critical error, which does not directly impact the host activity, and a critical error or an anomaly.

Compared to sensor data, which directly record physical phenomenon such as temperature, the quality of log data is much lower, as it is not intended for physical maintenance, but rather for software analysis. Using logs for predictive maintenance is often a question of cost, as installing or acquiring the outputs from sensors can be costly, or depending on the use case, simply impossible. When comparing the literature for log and sensor data, we can find the following differences:

- Only a fraction of the predictive maintenance literature covers the case of predictive maintenance on log data [34, 15, 51, 21, 73, 80, 41], and even less public datasets are available, as data mostly comes from private companies.
- Predicting the remaining useful life or a health indicator using a regression context is less frequent in the log data literature, which relies more heavily on classification [51, 73, 80, 41].

Independently of using a classification or a regression context, most studies convert the log data into a quantitative format, for example by counting the number of occurrences of each event in a fixed time windows, performing a temporal sampling, or by feature engineering with expert knowledge. Compared to sensor-based predictive maintenance, a lot more efforts are being put into feature engineering to compensate for the quality of the data. The type of features resulting from the feature engineering steps are often very diverse (i.e. discrete, continuous, categorical), which might influence the choice of classification over regression.

4.3 Experimental protocol for predictive maintenance

Consider an application where we want to predict failures, independently of the module that will fail. Consider a set of life cycles $\mathcal{X} = \{X_1, \ldots, X_n\}$ with $X_i = \{x_1, \ldots, x_m\}$. We extract a set of moving windows $\mathcal{W} = \{W_1, \ldots, W_{n\times(m-(l-1))}\}$ of size l, with the label of each window as $Y = \{y_1, \ldots, y_{n\times(m-(l-1))}\}$ obtained by the process described in Section 4.1.2 or Section 4.1.3.

The first point to rule out is that we should not use a f-fold cross validation on the set of moving windows $\mathcal{W}$. Consider two windows $W_i = \{x_i, \ldots, x_{i+(l-1)}\}$ and $W_{i+1} = \{x_{i+1}, \ldots, x_{i+1+(l-1)}\}$ extracted from a life cycle X. Using a f-fold cross validation without any constraints might cause W_i to be in the training set, and W_{i+1} in the testing set. As they both uses the points $\{x_{i+1}, \ldots, x_{i+(l-1)}\}$, a f-fold cross validation on $\mathcal{W}$ can cause data leaks from the training set into the testing set.

Hence, the validation process must be made on the set of life cycles $\mathcal{X}$. The life cycles will then be sliced into moving windows independently for the training and testing sets, before training the model and before predicting the classes.

4.3.1 Micro and Macro metrics

We have as input for our experimental protocol a set of life cycles $\mathcal{X} = \{X_1, \ldots, X_n\}$ with $X_i = \{x_1, \ldots, x_m\}$ a life cycle of length m. In the context of cross validation, $\mathcal{X}$ is divided into two disjoint ensembles, $\mathcal{X}_{Train}$ and $\mathcal{X}_{Test}$, for each validation fold.

A model is first learned on the moving windows extracted from $\mathcal{X}_{Train}$. Then, for each life cycle $X_i \in \mathcal{X}_{Test}$, the model is used to predict the class of each moving windows of size l extracted from X_i. We obtain the vector $\hat{Y}_i = \{\hat{y}_1, \ldots, \hat{y}_{m-(l-1)}\}$, containing the predicted class for each window. Given the vector of true classes $Y_i = \{y_1, \ldots, y_{m-(l-1)}\}$, the goal is to estimate the performance of the model, and of the predictive maintenance system that will use it.

We can then define micro metrics as metrics between Y_i and $\hat{Y}_i$, and macro metrics as metrics between the first occurrence of $\hat{y}_j = 1 \in \hat{Y}_i$ and $y_j \in Y_i$. This first occurrence represents the first alert that would be raised for the life cycle X_i. Micro metric measure the performance of the model on the learning task, while macro metric measure the performance of the predictive maintenance system. When using a macro metric, we take into consideration the fact that, when used in a real context, if an alert is raised, a maintenance is always performed. This maintenance will also modify the behaviour of the machine, and there is no guarantee that the data following the first occurrence of $\hat{y}_j = 1$ would be seen by the model.

To give the intuition of why micro metrics alone would not be reliable to estimate the performance of a predictive maintenance system, let us consider a classification model with an accuracy of 99% in the micro context. While this score may seem satisfying, it hides the fact that the missing 1% could be caused by a unique false positive, raised at the beginning of all the tested life cycles. As a positive prediction means that a maintenance operation will be scheduled, if the model is deployed, it may raise unwanted early maintenance alerts at the beginning of new unseen life cycles. Figure 4.4 illustrates this example for one life cycle.

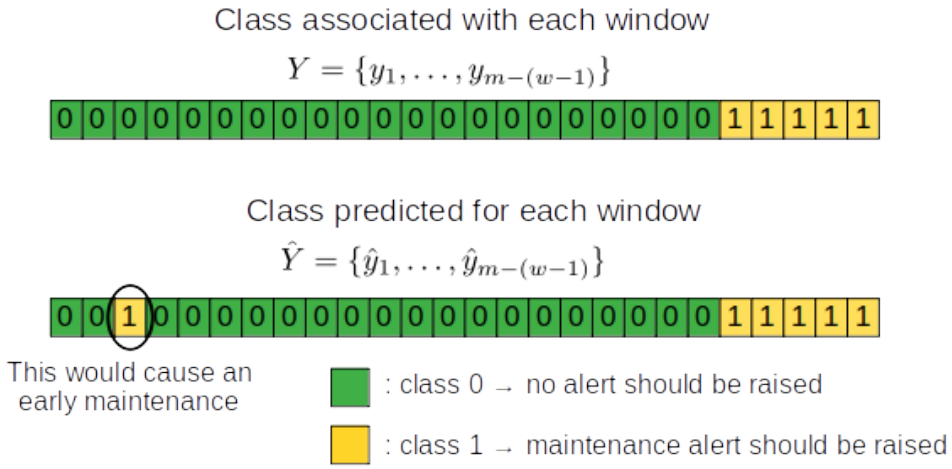

Figure 4.4: Despite a high accuracy, a unique early false positive will trigger an early maintenance process, making the accuracy in a micro context not trustworthy for the application.

4.3.2 Example with a classification model

As in chapter 5 we address a classification problem, let us detail now how the protocol is applied to a case of classification. We have as input a life cycle dataset $\mathcal{X} = \{X_1, \ldots, X_n\}$ with X_i a life cycle and a parameter f, which gives the number of cross validation folds. We create f different folds of $\mathcal{X}$, using $\frac{n}{f}$ life cycles in each testing set, such as each X_i appears in only one testing set across the f folds.

For each validation fold, the life cycles in the training set are sliced into moving windows, on which a classifier is fitted. The life cycles in the testing set are then also sliced into moving windows, and evaluated using the fitted classifier. Any necessary preprocessing steps needed by the model, such as slicing the life cycles into windows or normalization, should be made independently for each life cycle to avoid any data leak. Then, given the definition of predictive padding (pp) and reactive duration (rd) given in Section 4.1.3, we define the macro and micro metrics that are used to evaluate the classification model.

Let $X_i = \{x_1, \ldots, x_m\} \in \mathcal{X}$ a life cycle used in the testing set of a cross-validation step and $\hat{Y}_i = \{\hat{y}_1, \ldots, \hat{y}_{m-(l-1)}\}$ the prediction made by the classifier for each window extracted from X_i, with l the length of the moving windows. Given $Y_i = \{y_1, \ldots, y_{m-(l-1)}\}$ the target class for each window, the target class for $\{y_{(m-((l-1)+pp+rd))}, \ldots, y_{m-((l-1)+rd)}\}$ is set to 1, all other cases are set to 0, as defined in Section 4.1.3.

We can then define **micro metrics** as classification metrics between Y_i and $\hat{Y}_i$ and **macro metrics** as classification metrics between the first occurrence of $\hat{y}_j = 1 \in \hat{Y}_i$ and $y_j \in Y_i$. $\hat{y}_j = 1$ represents the first maintenance alert raised for X_i, which is the one we should care about, as discussed in Section 4.3.1. In the macro context, we can also define an error metric as $error_i = \max(0, m - ((l-1) + pp + rd) - j)$, giving the time separating the first alert $\hat{y}_j = 1$ from the desired alert interval. Figure 4.5 illustrates how the metrics and the error are computed for a life cycle.

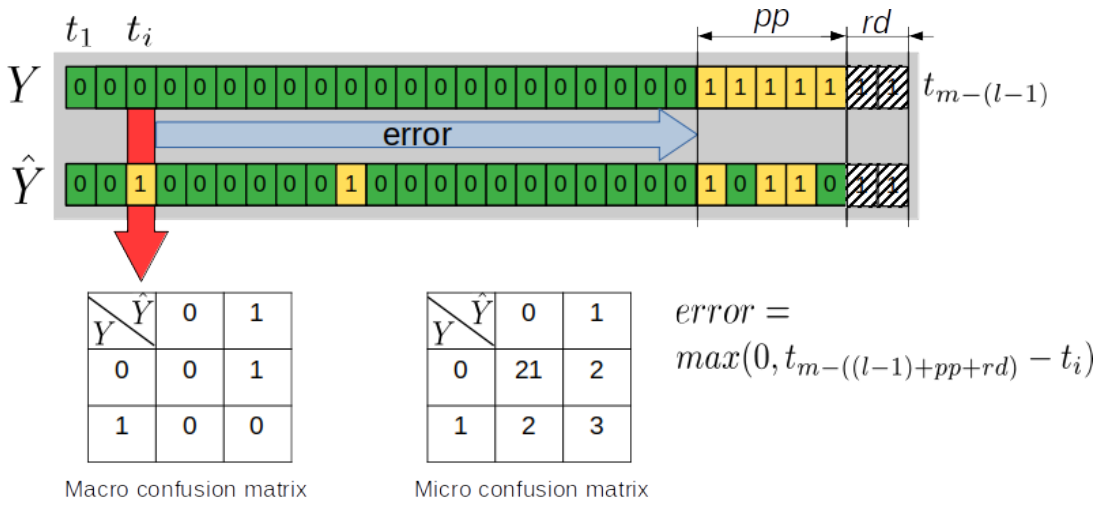

Figure 4.5: The difference between macro and micro metrics, illustrated with confusion matrices on the labels associated with each increasing window of a life cycle. Y contains the expected classes of each window and $\hat{Y}$ contains the predicted classes.

4.3.3 Estimating the cost of a predictive maintenance system

While the use of micro and macro metrics allow for a better interpretation of the evaluation results, they do not consider the costs linked to the maintenance process. To fill this gap, we propose a metric to estimate the cost of an alert for a life cycle, which can then be used to estimate if the predictive maintenance system is cost-efficient. In order to compute this metric, we first define the costs associated to a maintenance operation for a life cycle, using the following variable:

- t_f: the time when the life cycle ends
- t_m: the time when the predictive maintenance system raised an alert.
- $C_{fail}(t)$: the cost associated with the machine not functioning at time t. This cost depends on time in order to differentiate between times when the machine should, for example, be producing goods, and times when it is not operated (e.g. a weekly planning).
- C_{tech}: the cost of sending the technical team on site. We suppose this is a constant.
- $C_{replace}(t_f - t_m)$: the cost of the early maintenance operation, which depends on how early this replacement was performed compared to the time when the machine would have failed. In the following, we assume that this cost cast be computed as the difference $t_f - t_m$ multiplied by a constant.
- Δ_m: the time elapsed between the departure of the technical team, the arrival on site, and the time needed for the maintenance process.
- Δ_a: the time elapsed between the moment the machine fails, the failure is noticed, an alert is raised and the departure of the technical team.

For simplicity, we do not take into account variables measuring the impact of maintenance team availability and spare part management, and we consider that both the team and the necessary hardware are available at the time when the alert is raised. We can first define a baseline cost, which corresponds to the sum of the cost of failures if a predictive maintenance system was not used. The cost of an unexpected failure is defined by C_{neg} as:

$$C_{neg} = C_{tech} + \int_{t_f}^{t_f+\Delta_m+\Delta_a} C_{fail}(dt) \tag{4.2}$$

The cost of a predictive maintenance operation, justified or due to a faulty decision of the model, is described by a cost C_{pos}:

$$C_{pos} = C_{tech} + C_{replace}(t_f - t_m) + \int_{t_m}^{t_m+\Delta_m} C_{fail}(dt) \tag{4.3}$$

Independently of the model, be it regression, classification, anomaly detection, etc..., the system will either perform a successful alert, where $C_{neg} \geq C_{pos}$, or an unsuccessful alert, where $C_{neg} < C_{pos}$. Looking at the difference between these two costs for all life cycles, defined by Equation 4.4, we can then estimate if the predictive maintenance system is worth using, independently of the type of model used to predict failures. Note that the notion of cost is not necessarily linked to a monetary cost, and depends on the application and the function of the machine. A negative value for C_{global} would indicate a higher overall cost compared to the baseline, which is to not use a predictive model and let failure happen.

$$\begin{aligned} C_{global} &= \sum_{i=1}^{n\_failures} (C_{neg_i} - C_{pos_i}) \\ C_{global} &= \sum_{i=1}^{n\_failures} \left(\int_{t_{f_i}}^{t_{f_i}+\Delta_m+\Delta_a} C_{fail}(dt) \right) - \left(C_{replace}(t_{f_i} - t_{m_i}) + \int_{t_{m_i}}^{t_{m_i}+\Delta_m} C_{fail}(dt) \right) \end{aligned} \tag{4.4}$$

As C_{tech} is considered constant for both unexpected and planned maintenance, it can be simplified when developing the equation. If the exact costs are unknown, we can replace each component of the equation by variables quantifying these costs relative to the cost of an unexpected error. Considering $\int_{t_{f_i}}^{t_{f_i}+\Delta_m+\Delta_a} C_{fail}(dt)$ as the cost of an unexpected failure, if $C_{fail}(t)$ is unknown, we can replace the integral by $\lambda_{neg} = 1$. If we estimate that a planned maintenance is on average half the cost of an unexpected one, we then can replace $\int_{t_{m_i}}^{t_{m_i}+\Delta_m} C_{fail}(dt)$ by $\lambda_{pos} = 0.5$.

Then, we use the error metric described in section 4.3.2 to estimate $C_{replace}(t_{f_i} - t_{m_i})$ as $error_i \times \lambda_\epsilon$. The λ_ϵ will quantify the cost of early replacement per timestamp. For example, $\lambda_\epsilon = 0.01$ would mean that each timestamp separating the alert from the ideal maintenance interval cost one percent of the cost of an unexpected maintenance. We can then produce an estimate $\hat{C}_{global}$ of the global cost C_{global} by using the error and a macro confusion matrix, and the three λ values as :

$$\hat{C}_{global} = \sum_{i=1}^{n} \lambda_{neg} - \sum_{i=1}^{n} \begin{cases} \lambda_{pos} + error_i \times \lambda_\epsilon & \text{if } \max(\hat{Y}_i) = 1 \\ \lambda_{neg} & \text{if } \max(\hat{Y}_i) = 0 \end{cases} \tag{4.5}$$

Given the problem formulation we introduced for predictive maintenance use cases, an interesting addition would be to impose constraints on the model when making prediction, in order to avoid early maintenance alerts. This idea of constraining the prediction has already been developed in the field of early classification, which we introduce in the following section.

4.4 Early classification

Early classification of time series (ECTS) as a distinct field first appeared in [83]. It differs from time series classification by considering the notion of earliness of the prediction as a factor of success of a model.

While the training procedure between ECTS models can vary, the evaluation of such models remains the same. First, given a time series $X = \{x_1, \ldots, x_m\}$ to evaluate, it is transformed into increasing windows as $W_t = \{x_1, \ldots, x_{t+(l-1)}\}$ and the model then will iteratively make predictions for each window from 1 to $m - (l - 1)$ until some constraint, defined by a trigger system, estimates that at a window W_t, the model has seen enough data to make a good prediction. The output of the model for the series X will then be the prediction made on W_t, with its earliness being quantified as $\frac{t+(l-1)}{m}$.

The quality of an ECTS model is defined by how accurate its predictions are, but also by how early they are made, with the goal being to balance the trade-off between accuracy and earliness, based on the importance of both factors in the task to solve. The idea motivating the field is that, for some task, the earliness of the prediction can be highly valuable to, for example, be able to perform preventive actions before a problem occurs.

4.4.1 Related work

Early works on ECTS proposed to solve the problem by using shapelets [85], with an extension to the multivariate context by [35]. In their method, given a time series dataset $\mathcal{X} = \{X_1, \ldots, X_n\}$ with $X_i = \{x_1, \ldots, x_m\}$ and a shapelet $S = \{s_1, \ldots, s_l\}$, the quality of S is not only evaluated by the information gain resulting from the minimum of the distance vector (i.e. $\min V_{S,X_i}$) for all $X_i \in \mathcal{X}$, but also by its position (i.e. $\operatorname{argmin} V_{S,X_i}$) in the series. A good shapelet candidate must then be discriminative toward the class of the problem, but also be as close as possible to the start of the time series, in order to maximize the notion of earliness when applied to unseen series. A selected candidate S is then associated to the class of the series from which it is extracted and to a threshold λ that maximizes the information gain of computed on $minV_{S,X_i}\ \forall X_i \in \mathcal{X}$.

The trigger system used by such an approach is implicitly defined by the criterion of quality of the shapelets. When evaluating increasing windows of X, to know that W_t should be the one used to give the prediction for X, a simple rule based system is used: if for one of the extracted shapelet S we have $minV_{S,W_t} < \lambda$, then the class associated with S will be returned as a prediction for X. Note that more than one shapelet could be associated with a class, the sensibility of the trigger system can then be tweaked by choosing the proportion of shapelets of a class below their threshold that are necessary for a class to be predicted. To optimize earliness, this method assumes that a shapelet occurring early in the series of the training data, will also appear early in unseen data.

Other works in ECTS use ensemble of classification models, with each model affected to a different timestamp t, and trained only on the increasing windows $W_t = \{x_1, \ldots, x_t\}$ extracted from all time series $X \in \mathcal{X}$, with the class of W_t equal to the class of the series X from which it is extracted. The main differences between this kind of ECTS algorithm reside in how they manage this ensemble of models, and how they design the trigger system to decide that t is the time when the prediction should be made. When the trigger system considers that t is the right time to make a prediction, the output of the model trained on the increasing windows W_t is used.

Early work on trigger systems used heuristics, often based on the confidence of the prediction at time t, which was expressed as the class probabilities output by the model trained with subsequences up to t. If the certainty of the prediction was above a threshold, the prediction was given as output. More recent approaches [18, 78, 1] formalized the notion of earliness of prediction as an optimization problem, where two user-provided cost functions are defined:

- $C_m(\hat{y}|y)$: The misclassification cost function, defining the cost of predicting $\hat{y}$ when y is the true class
- $C_d(t)$: The delay cost function, defined as a non-decreasing function of time, representing the cost associated with delaying the prediction.

Given these two functions, the cost of making a prediction at timestamp t using $W_t = \{x_1, \ldots, x_t\}$ and the classifier h_t associated to timestamp t, with Y the possible classes of the problem, is expressed as:

$$f(W_t) = \sum_{y \in Y} P_t(y|W_t) \sum_{\hat{y} \in Y} P_t(\hat{y}|y, W_t) C_m(\hat{y}|y) + C_d(t) \tag{4.6}$$

Where $P_t(\hat{y}|y, W_t)$ is the misclassification probability, estimated using a confusion matrix obtained from the evaluation of the training data with the classifier h_t, and $P_t(y|W_t)$ the posterior probability of class y given W_t. The ideal timestamp t to raise a decision then becomes the one which minimize Equation 4.6, but, to known which timestamp t is optimal, we need to observe the whole series. This is why ECTS models aims at estimating this cost for future timestamps, modifying Equation 4.6 by adding a constant τ to t.

Multiple methodologies have been proposed to estimate the cost of yet unseen data (i.e. $P_{t+\tau}(\hat{y}|y, W_{t+\tau})$, but as we only use the idea behind trigger systems in our contribution, we report the reader to the reviews of [33] and [9] for a more detailed view of the field. We do not consider ECTS models as they rely on some assumptions that are not applicable to our case. We discuss these assumptions in the next section, the proposed solutions of the literature, and why they are not yet sufficient to directly apply ECTS models to predictive maintenance.

4.4.2 Early classification for predictive maintenance

The first remark one could make is that, for predictive maintenance, we are not necessarily interested in optimizing the earliness of the alerts, as early maintenances result in additional cost due to the early replacement of a functioning piece. We believe that this

issue can be tackled by changing the costs used in the function to be optimized by the model. For example, the prediction delay cost could be changed to favour predictions made in an interval near the failure, such as the one defined by the predictive padding and reactive duration in Section 4.1.3.

While the formulation used by ECTS models, where a cost is estimated iteratively on increasing subsequences to know if the prediction should be made, shares similarities with the evaluation protocol we defined for predictive maintenance, the fact that it requires an ensemble of models, each affected to a specific timestamp, is problematic. In the ATM use case that we present in Chapter 5, we are dealing with time series of variable length with a very skewed distribution, the later timestamps will have fewer windows to learn from, possibly with only one class being represented. This also raises the question of how such models deal with time series lengths that are superior to the ones seen in the training set. The impact of time series which are not temporarily aligned (e.g. starting at the same time), is also not clearly addressed.

Another incompatibility is introduced by our problem formulation for predictive maintenance in a classification context. A life cycle $X = \{x_1, \ldots, x_m\}$, independently of its class (i.e. the cause of failure), has two "subclasses" defined by the predictive padding (pp) and the reactive duration (rd). The first one in $[x_1, x_{m-(pp+rd)}[$, which is the area characterizing early alerts, and the second in $[x_{m-(pp+rd)}, x_{m-rd}]$, which gives the target area for maintenance alerts. It is yet unclear to us how an ECTS model could deal with these constraints.

4.5 Introduction to survival analysis

To include the idea of trigger systems developed in early classification into our contribution, we use the field of survival analysis to provide an estimate of the probability of survival of a given life cycle at time t, and build a trigger system around this probability. In this section, we give an introduction to survival analysis and the components we use in our contribution.

Survival analysis is an ensemble of statistical methods used to analyse the expected time to event for an ensemble of test subjects, often called a population. It was traditionally used in clinical trials, where the survivals of groups under different treatments (or against a control group) were compared in order to estimate the efficiency of these treatments. Due to sharing a lot of common definitions and approaches with reliability theory in engineering, it also has been used in predictive maintenance [80].

4.5.1 Survival data

In the simplest scenario, survival data consist of a table that, for each subject, records if the event (e.g. a failure for a machine) has occurred, and at what time. Quite often in survival analysis, not all the studied subjects will have experienced the event yet, some data might be missing, or new subjects might enter the study after other subjects. All those concepts are related to the notion of data censoring. Censored data are missing information about the subject time to event data. We can distinguish three distinct case of censoring:

- Right censored data correspond to subjects that have not yet experienced the event at time t, for example, machines that are still functioning at time t, t being the time point where we wish to perform survival analysis.

- Left censored data correspond to missing data at the beginning of the observation time. This could be machines for which we do not have the start time, meaning that they were already functioning since an unknown time when we started collecting the data.

- Interval censored data correspond to the case where an event is happening between two observations. If we look at a machine state every hour, and that at observation $i + 1$ the machine has failed, then the event occurred at an unknown time before i and $i + 1$.

An example of survival data with right censoring is given by Table 4.1.

Table 4.1: Example of a survival dataset with right censoring, with subject 1 and 2 not yet having experienced the event.

Subject ID	Observed lifetime	Time of event
0	220 days	day 220
1	310 days	NA
2	330 days	NA
3	250 days	day 250

4.5.2 Survival and Hazard Function

An interesting feature of survival analysis is that it takes into account censored data when modelling the chance of survival of a population. This notion of chance of survival is formalized with the survival function $S(t)$, which is one of the main components of survival models. In an engineering context, the survival function is often referred to as the reliability function.

Given T a random variable representing the time of occurrence of the event and t a specific time, for example the last observed lifetime of a subject, we can define $P(T > t)$, the probability that the event has not occurred yet at time t, or equivalently, the probability of surviving after time t. The survival function is then defined as:

$$S(t) = P(T > t) \text{ with } S(t + 1) \leq S(t)$$

Another important aspect of survival analysis is the hazard function, or conditional failure rate in engineering, often noted $\lambda(t)$. It represents the chance of occurrence of the event between time t and $t + dt$, at the condition that event has not occurred before t. It is often referred to as the instantaneous rate of event and is expressed as:

$$\lambda(t) = \lim_{dt \to 0} \frac{P(t \leq T \leq t + dt \mid T > t)}{dt}$$

When plotted over time, the shape of the hazard function gives hints about the event occurrence mechanism. For example, an increasing hazard function could highlight the influence of age for subjects in a cancer study. A decreasing one corresponds to subjects having some kind of early setup conditions, an example of this could be disease that only affect children.

In many applications, the effect of some variables linked to the subjects can also be taken into account, such as the age and the sex. In this case, the effect of each variable X is quantified by a coefficient β using survival regression, which computes the β coefficient using the maximum likelihood method. The hazard of a subject at time t will then be modelled as a function of βX and of a baseline hazard $\lambda_0(t)$. A popular model, called the Proportional Hazards model, expresses this function as $\lambda(t|X) = \lambda_0(t)exp(\beta X)$.

4.5.3 Survival function estimators

As we usually only observe a sample of the population (e.g. a group of patients with cancer is observed in a study, not all existing patients with cancer), we can rarely know the true survival function of a population, and need to estimate the true survival function using only a fraction of it. To do that, a multitude of estimators have been proposed in the literature. Survival estimators are often grouped into three families [40]:

- Non-parametric estimators: These estimators are mainly used when there is no variable associated with subjects, and that we only wish to estimate the survival function based on the observed lifetime. A popular one is the Kaplan-Meier estimator [38], which estimates $S(t)$ at each time step t as:

$$S(t) = \prod_{i:t_i<t} \left(1 - \frac{d_i}{n_i}\right) \tag{4.7}$$

  with t_i a time when at least one event occurred, d_i the number of subjects that had event at time t_i and n_i the number of subjects that did not experience event before time t_i.

- Parametric estimators: These estimators make an assumption on the distribution of survival times or hazard of the population and use the available data to fit a baseline hazard or survival function to this distribution. For example, an estimator assuming a normal distribution will use the mean μ and the variance σ of the observed lifetime of the population as parameters to estimate the baseline survival function as, $S_0(t) = 1 - \Phi\left[\frac{t-\mu}{\sigma}\right]$ with Φ the cumulative distribution function of the standard normal distribution.

- Semi-parametric estimators: Semi-parametric estimators are used when the underlying distributions are unknown or difficult to estimate. They provide a way to estimate the regression coefficients without the need of a specified baseline hazard or survival function. The most popular being the Cox Proportional Hazards Model [17].

Another interesting methodology developed in the literature are Survival trees. They were developed to provide a non-linear approach to survival analysis, as other models,

such as the Cox Proportional Hazards model are considered to be linear due to the fact that they affect a weight to each variable. Similarly to decision trees, survival trees create partitions of the input space, by iteratively choosing a variable and a threshold. Previous works on survival trees [13] use the log-rank test as a splitting criterion to evaluate the quality of a split.

The log rank test is a hypothesis test that, given two survival functions, S_1, S_2 (i.e. those of the child nodes) formulates the null hypothesis as $S_1(t) = S_2(t)$ against the alternative hypothesis that $S_1(t) \neq S_2(t)$, for any time t. If the null hypothesis is rejected by at least one candidate split, the split that maximizes the statistical test is selected. If the null hypothesis cannot be rejected, the current node is considered as a leaf. Since the log-rank test compares survival probabilities, one of its drawback is that it does not directly consider the size of the populations that were used to estimate those probabilities. Although alternative tests were developed [45], for the predictive maintenance use case we present in the next chapter, we will present in our contribution a simple alternative, as we have no form of censoring in the training data.

Having introduced the predictive maintenance domain, and how we plan on evaluating predictive models, we can now introduce the use case that motivated this thesis. We will then use some ideas developed by the field of early classification and survival analysis in our contribution, to propose a new classification model using a trigger system based on survival probability of a life cycle.

Chapter 5

The ATM use case

Résumé du chapitre en français
Dans ce chapitre, nous présentons l'application industrielle de maintenance prédictive pour ATMs proposée par l'entreprise Worldline. Dans un premier temps, nous présentons le contexte entourant l'application et le processus de maintenance des ATMs, et détaillons la création du jeu de données ATM. Nous donnons une description des journaux d'événements que nous avons extraits des ATMs, comment nous avons obtenu les données de pannes et comment nous avons transformé ces journaux d'événements en cycle de vie. Nous présentons ensuite les prétraitements appliqués aux cycles de vie et décrivons les caractéristiques du jeu de donnée obtenu. Ensuite, en utilisant le protocole expérimental introduit au chapitre précédent et des approches existantes, nous présentons les premiers résultats expérimentaux, que nous utiliserons comme base de comparaison. Nous présentons ensuite nos contributions pour cette application, comprenant un nouveau modèle de classification utilisant l'analyse de survie pour créer un système de déclenchement, inspiré des idées introduites par le domaine de la classification précoce. Nous comparons ensuite les résultats de ce nouveau modèle avec les résultats de base que nous avons obtenus.

Chapter summary
In this chapter, we present the industrial use case of predictive maintenance for ATMs proposed by Worldline. After presenting the ATM application context and the maintenance process, we show how we created the ATM dataset. We give a description of the event logs that we extracted from ATMs, how we obtained the failure data and how we transformed these event logs into life cycles. We then present the preprocessing steps applied to the life cycles to obtain the ATM dataset, and describe its characteristics. We then conduct experiments on this dataset, based on the experimental protocol introduced in the previous chapter, and using existing approaches in order to obtain baseline results. We then present our contributions for this use case, including a new classification model using survival analysis to create a trigger system, following the ideas introduced by the field of early classification. We then compare the results of this new model with the baselines results we obtained.

5.1 Application context

Before presenting the context, we define some domain specific terms:

- **Cash-In-Transit (CIT)**: Security companies, which are in charge of moving cash and emptying or refilling ATMs when needed.
- **ATM module**: An ATM is an ensemble of connected modules, each composed of several components directly interacting with each other. For example, the distribution module is composed of storage boxes for the banknotes, linked to an ensemble of rolling or suction cup mechanisms to route banknotes to the exit slot.
- **ATM life cycles**: A time period in which an ATM is functioning properly, ended by any kind of failure (i.e. mechanical or not). It starts after the end of a maintenance process and ends when a failure occurs. This is similar to the definition of a life cycle given in the previous chapter.
- **Withdrawal**: This term simply denotes the action, initiated by a user, of withdrawing cash from an ATM. While a log is linked to this action due to the change in the amount of cash in the safe, no transaction or user data is present in the dataset.

The inside of an ATM can be seen as an ensemble of connected objects, which are managed by a computer using a specialized software. Figure 5.1 gives a view of the inside of an ATM. When a user introduces its card, the card reader reads the information stored in it and security checks are performed. Once the user has been authenticated by its PIN code, it can choose the amount of cash to withdraw. After the choice of the amount, the distribution module gathers the banknotes in their specific storage boxes and groups them into a storage area. Once done, the user is prompted to remove his card from the card reader to receive its cash.

If a failure occurs on a module of the ATM, the owner (generally a bank) initiates a maintenance ticket. A technical maintainer will then be dispatched on site to evaluate the problem. From there, we can distinguish three cases:

- If it is a hardware issue and if the necessary spare parts are available in the maintainer truck, the maintenance can be performed immediately.
- If the maintainer does not have all the required parts, they will be ordered and the maintenance is delayed until the parts are received.
- If no hardware issue is found (i.e. possibly a software or network issue), an inspection of the activity of the ATM is needed to find the cause of failure.

In all cases, the maintainer will send a report to the helpdesk that will change the ticket information based on the maintainer report. Note that if a technical intervention requires the opening of the cash safe, for example to access to the distribution module, the CIT has to empty the ATM before the maintainer intervention for security reasons, which can add even more delay before the ATM is fixed.

The interest of predictive maintenance for ATMs is two-folds: first it can allow for spare parts planning in the maintainer truck if the cause of failure is known in advance, and secondly, it can allow the synchronization of maintenance operations with regularly

planned interventions of CIT to refill the ATM, and considerably reduce ATM unavailability. The objective for this project was to make a proof of concept on a subset of the ATM fleet (more than 12000 ATMs across the world) managed by Worldline before adapting the product for the whole fleet.

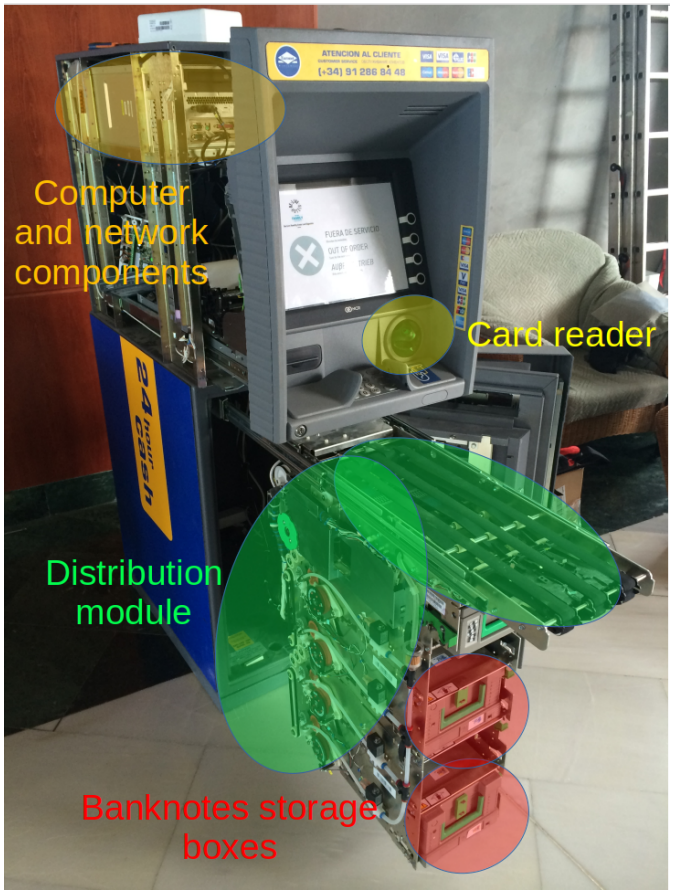

Figure 5.1: Inside view of an ATM with highlighted components. Modification of the original image from Bjoertvedt, CC BY-SA 4.0, via Wikimedia Commons (https://commons.wikimedia.org/wiki/File:ATM_inside_tenerife_IMG_8732.JPG)

5.2 The ATM dataset

The ATM dataset is composed of 1707 life cycles, gathered from 460 ATMs over a period of two years, representing more than 14 millions logs. Due to the nature of the application, life cycles have different lengths and are ended by different kinds of failure, such as the distribution module, the card reader, a network problem, an empty cash-safe or external damages.

Before being preprocessed and sliced into life cycles, the ATM dataset is a log file composed of 4 columns, as presented in Table 5.1:

- ATM ID: identifier of the ATM that produced the log
- Timestamp: when the log was produced

- Event code: a logical event encoded by an integer
- Complement (optional): additional information about the event

Table 5.1: An example of log data from the ATM dataset

ATM ID	Timestamp	Event Code	Complement
41	31/08/2019 12:33:57	40000	51400
75	31/08/2019 12:33:57	6000	
53	31/08/2019 12:34:05	1001	
41	31/08/2019 12:39:41	40000	51300

With a few exceptions, event codes follow a severity level for each hardware and software component. Given a particular component we have three codes, namely OK, warning and error. For example, a code 6000 indicates an "OK" event on the distribution module, which can (not exclusively) happen during a restart process or when the ATM recovers from a problem, whereas the code 6001 indicates an error event and the code 6002 a warning event. An error recovery might for example happen when a banknote is damaged when manipulated by a mechanism. Normally, the ATM is able to reject such damaged banknotes into a designated dedicated box to avoid further problems. The complement field is optional, it provides additional information on the event, and is notably used to indicate an estimation of the remaining amount of cash in the safe of the ATM when a withdrawal code (40000) is generated. One of the main challenge of the data is that some event codes can be triggered by multiple physical events, which makes the interpretation of the data difficult.

5.2.1 Extracting life cycles

In this section, we detail how we extract ATM life cycles from the event logs. As input, we have a log file containing all logs emitted by all the ATMs, with each log being identified by a timestamp and an ATM identifier, which indicates which ATM generated the log. To extract ATM life cycles from the logs, we must first know when (and why) an ATM entered a failure state. Once we have the time intervals in which an ATM was in a failure state, we can then extract the corresponding life cycles of this ATM.

Obtaining failure data

As we presented in Section 4.1.3, knowing when failures happen is essential to extract life cycles. The first difficulty we encountered with the use case was that ATMs managed by the company are not all under the same type of contracts with the clients. While we do have event log data for all ATMs independently of the contract, we only have maintenance information for some of them. When we do have some information about maintenance operations, the quality is not ideal as the company is not directly responsible for maintenance, but rather serves as an intermediate between the client and the maintenance companies.

As those companies also have commercial interests in building predictive maintenance systems, we could not obtain any information from them. What we have at our disposal are support tickets raised by the clients, which, for some, indicate a hardware issue. A ticket contains a start and an end date, a subject, as well as the discussions between the different parties involved in the ticket. We extract tickets related to failures, as well as the identified cause of failure.

The second issue with early work on the project arose from the fact that the failure indicated by the tickets did not always correspond to abnormal events in the log data, withdrawal were still performed and there was no anomaly or spike of errors. The timestamps indicated in the tickets were also sometimes inaccurate. This motivated the creation of a tool to identify possible failures directly from the event logs in historical data, which can also be used as an online anomaly detection tool to detect possible ongoing failures.

The reasoning behind this tool is the following: if there is no withdrawal activity during a time period where activity is normally seen, it may mean that the ATM is in a failure state. Looking at the event codes happening in this interval of abnormal non-activity then gives us hints about the possible cause of failure. ATMs are subject to a very important daily and weekly seasonality for the number of withdrawals performed, different for each ATM. For example, an ATM located in a night-life area will have most of its withdrawals during the night, with a higher volume on Fridays and Saturdays. Another example could be an ATM located in a mall closed during the night, which cause withdrawals to be performed during the opening hours and days of the mall.

Definition of the ATM failure extraction tool

In the following, we consider a time series $T_i = \{t_1, \ldots, t_m\}$ with t_j the timestamp where ATM i generated an event log. Due to the high variability in seasonality and volumes of withdrawal, the tool processes each ATM independently. Based on a time series T_i, we aim at obtaining a set of triplets $(t_{start}, t_{end}, cause)$, each containing the start and end time of the suspected failure, along with the identified cause.

The first step performed by this tool is to sample T_i given a frequency f, such as, for each time interval of length f in a week, we compute the probability of at least one withdrawal event log being made amongst all events in this time interval in T_i. For example, with $f = 1$ hour, we have 7×24 intervals, with the first one being Monday from 00h00 to 01h00 (excluded) and the last being Sunday from 23h00 to 00h00 (excluded). If we have 10 occurrences of the interval Monday from 00h00 to 01h00, with 8 out of 10 having one or more withdrawal events, the probability affected to this interval will be of 0.8. We denote p_j the probability that at least one withdrawal occurs in the period j. Figure 5.2 gives an example of the resulting probabilities for two ATMs with a frequency of one hour.

Given these probabilities and the time series $T_i = \{t_1, \ldots, t_m\}$, we sample a new time series A_i, with frequency f, as follows: for each $a_j \in A_i$, we affect $a_j = 1$ if at least one withdrawal was performed during the time interval designated by j, else, we affect $a_j = 1 - p_k$, with p_k the probability of having at least one withdrawal in the period k related to j in the weekly seasonality (e.g. Monday 9:00 to 9:59).

To compute the anomaly score Q_i, for each interval $[j_{start}, j_{end}] \in A_i$ where successive

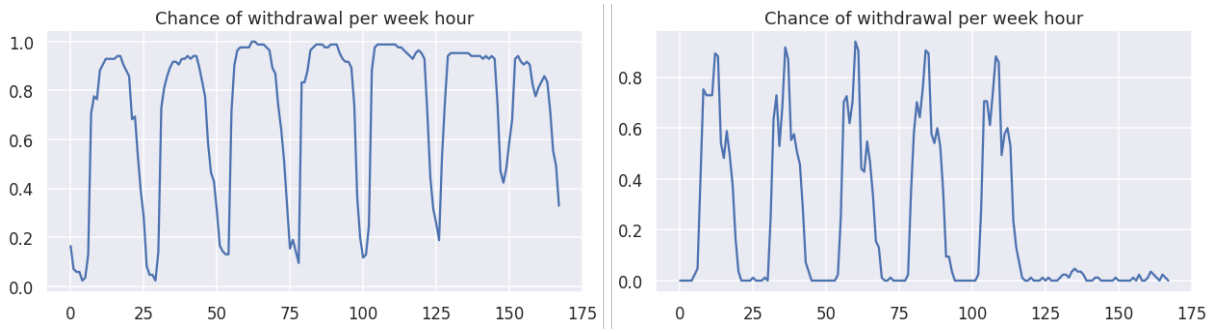

Figure 5.2: Example of weekly withdrawal seasonality for two ATMs. Y-axis is the probability of having at least one withdrawal in a period. Index 0 is Monday from 00:00 to 00:59, index 1 from 01:00 to 01:59, ..., and index 167 is Sunday from 23:00 to 23:59

values of a_j are inferior to 1, we compute a cumulative product, as defined by Equation 5.1. In the other cases, we affect $q_j = a_j$. In this context, the lower the value of q_j, the higher the risk of anomaly.

$$q_j = \prod_{p=j_{start}}^{j} a_p \mid \forall j \in [j_{start}, j_{end}] \tag{5.1}$$

Given a threshold λ and a number of hours H, an interval $[j_{start}, j_{end}] \in A_i$ is considered as a failure interval if there exists $a_j \in [j_{start}, j_{end}] \leq \lambda$ with $t_{end} - t_{start} \leq H$. The supposed cause of failure is then extracted by looking at the event logs generated by the ATM in the period $[t_{start}, t_{end}]$: if the ATM still has cash available in its safe, the cause of failure will be determined by the type of error code which occurs the most, else it will be assumed that the absence of withdrawal is due to an empty safe. Figure 5.3 gives a visualization of the anomaly score. The $[j_{start}, j_{end}]$ interval kept to locate a failure is always the complete interval where we have successive value of $a_j < 1$.

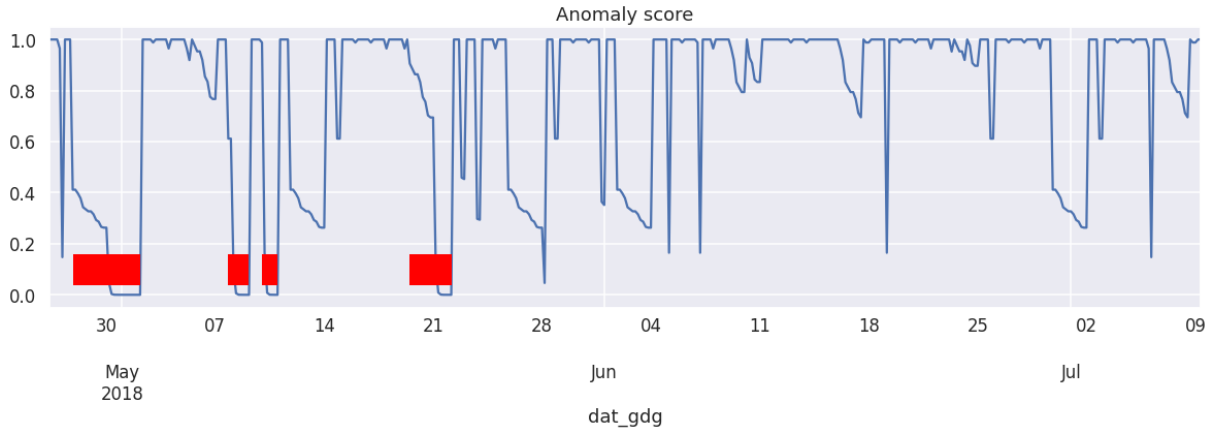

Figure 5.3: Example resulting anomaly score for an ATM using $\lambda = 0.0005$ and $H = 8$ hours. Red areas are those identified as possible failures.

Running this process for all ATMs gives as output a set of time intervals as presented in Table 5.2. The tool and its parameter were then validated with the existing mainte-

nance tickets by manually inspecting results from the tool and comparing them to the maintenance tickets and the log data.

Table 5.2: Example of failure data used to generated life cycles

ATM ID	Start	End	Cause
40	21/04/2018 ...	28/04/2018 ...	Distribution
76	04/05/2018 ...	07/05/2018 ...	Card Reader
74	14/06/2018 ...	16/08/2018 ...	Connection
40	24/08/2018 ...	25/08/2018 ...	Empty Safe

Failure data for the ATM dataset

To obtain the failure data we use for the ATM dataset, we combined the information from the maintenance tickets and the time intervals generated by the tool. Given a time interval $T^{tool} = [t_a, t_b]$ found by the tool for an ATM i, and any time interval $T^{ticket} = [t_c, t_d]$ from the maintenance ticket of ATM i, we apply the following rules :

1. If an intersection exists between the two time intervals T^{tool} and T^{ticket}, then we keep the interval $[min(t_a, t_c), max(t_b, t_d)]$ as the final value for the time interval. Else, we discard T^{tool}.

2. Then, if the causes of failure mentioned by T^{tool} and T^{ticket} are the same, we keep this cause to be associated with the time interval $[min(t_a, t_c), max(t_b, t_d)]$. If there is a difference, we affect the value "Unknown" to the cause.

If there is no intersection between T^{tool} and T^{ticket}, it is possible that the ATM is unreachable by customers due to an exceptional event (e.g. road work), and although no withdrawal would appear (hence triggering the tool) the ATM would not in reality be in a failure state. The result of this operation is a table similar to the one represented by Table 5.2, which we use to extract life cycles foo each ATM. A life cycle has as label the cause of the failure happening at its end. We do not include in life cycles the data from the period in which the ATM was in a failure state, as it is noisy and irrelevant to the task of predicting failure before they happen.

5.2.2 Preprocessing

To avoid introducing a bias based on the number of total events in a day, we removed from life cycles the complete day when a failure occurs (e.g., if failure occurs on 14/06/2018 08:00:00, the last data point of the resulting life cycle will be on 13/06/2018 23:59:59). The notion of infected interval, that we present in Section 4.1.3 is also applied, and we removed the first day of data (e.g., a life cycle starting on 01/05/2018 16:00:00 would now start on 02/05/2018 00:00:00). We have then discarded life cycles with a lifespan inferior to 7 days, to obtain the 1707 life cycles of our dataset. Those very short life cycles are mainly due to human errors during previous maintenances (e.g., a failure happens a few days after maintenance, on the same module that was supposed to be repaired).

We have chosen to transform the problem into a numeric representation, similarly to the preprocessing employed by [51]. Thus, in next sections, we will focus on numerical methods rather than on approaches such as chronicle mining [20] that may have handled categorical series of event codes. Given a set of life cycles composed of logs as in Table 5.1, we perform the following operations for each life cycle:

1. Create a table, which uses the timestamps of the logs as index and create as many columns as event codes. Fill each row of this table with a 1 for the event code (i.e. column) that occurred at this timestamp, 0 for the others. This gives us 189 features.

2. Remove the event codes that only appear in less than 5% of samples of all classes and that are purely made for software maintenance or audit purposes (i.e. no relation with a physical event). This reduces the number of features to 54.

3. Given a resample frequency, resample the table to count the number of occurrences of each code during the period. We use a frequency of 8 hours, counting event codes occurrences from [0h00 to 08h00[, [08h00 to 16h00[, and [16h00 to 0h00[for each day in the life cycle.

The result of this preprocessing is an uneven and multivariate time series dataset with 1707 samples and 54 variables, with each life cycle labelled with its cause of failure. In the ATM dataset, a life cycle is then represented by a time series such as the one presented in Table 5.3.

Table 5.3: Example of time series data representing a life cycle. Each column represents the number of occurrences of an event code in a period of 8 hours, starting at the timestamp and ending before the next timestamp.

Timestamp	1000	1001	...	11001	51000
21/04/2018 00:00:00	0	1	...	1	0
21/04/2018 08:00:00	3	0	...	3	1
21/04/2018 16:00:00	0	0	...	0	5
22/04/2018 00:00:00	1	2	...	0	1

5.2.3 ATM dataset characteristics

The first particularity is the distribution of the length of the life cycles, which, as shown by Figure 5.4 is very skewed. This is due to the fact that the data, as some failures are not rare (e.g. empty safe or network problem), and can create a lot of life cycles for one ATM. While, as discussed in Section 4.1.1, other ways to extract life cycles exists, the quality of the data did not allow them to be applied. The total number of failures for each failure type are: 551 for network issue, 505 for empty safe, 442 unknown (tool and ticket do not indicate the same failure type), 159 for the distribution module and 50 for the card reader.

Another important fact about the data is that it is sparse, with 58% of the values being zeros. To smooth the values and reduce sparsity, we use a rolling mean of a week (the

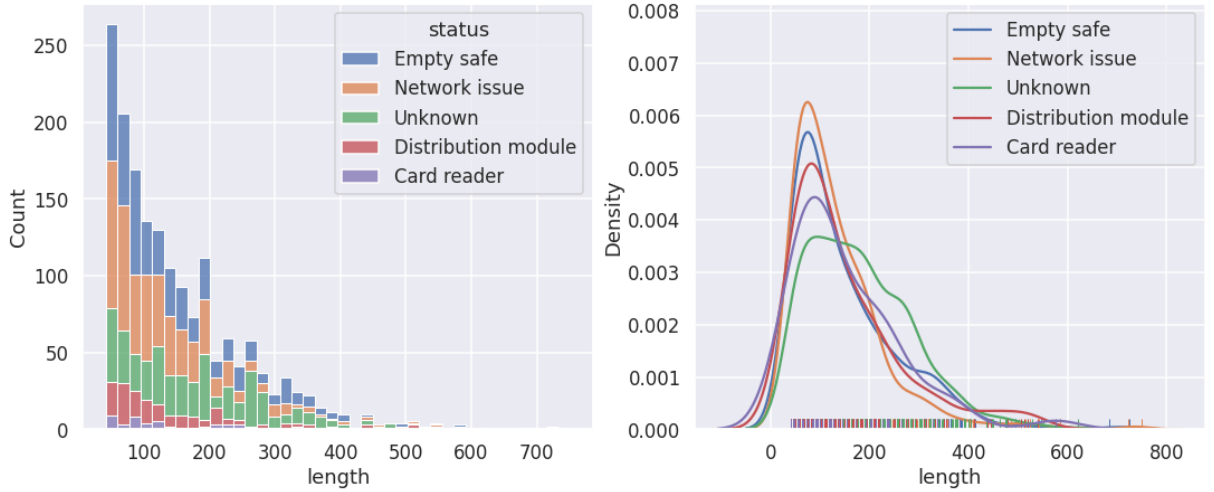

Figure 5.4: Distribution of life cycle duration per failure cause. One unit of length is 8 hours.

minimal data length), which gives us a sparsity of 26% and a smooth signal. Figure 5.5 gives an example of two time series without smoothing, and Figure 5.6 with smoothing for life cycles ending with a distribution module failure. These two examples also illustrate another difficulty of the data, the absence of clearly defined patterns that are linked to a type of failure. The high dimensionality of the dataset may also prevent us to identify such patterns by visualization. There are also some anomalies happening in the data, that should not be picked up by the model as reason to raise a predictive maintenance alert. Figure 5.7 gives an example of such anomalies.

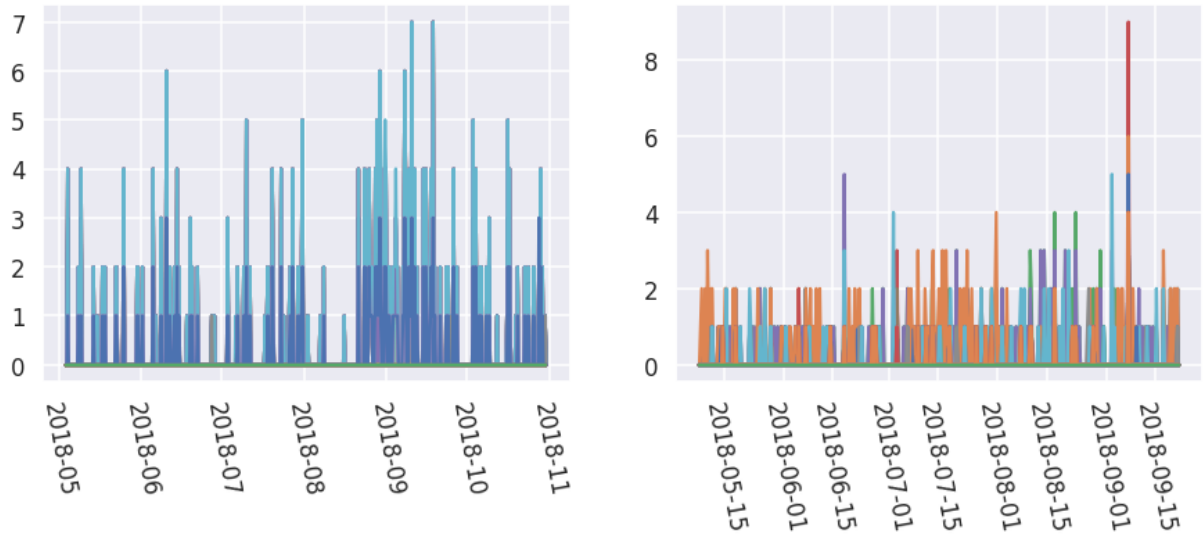

Figure 5.5: Two (left and right plot) raw time series ending with a distribution failure (colour legend is disabled due to the high number of features).

In this case study we are interested in failure of the distribution module, but as other types of failure are also occurring on the same machines, we need to take them into account when learning a predictive model, as the anomalies or patterns linked with these

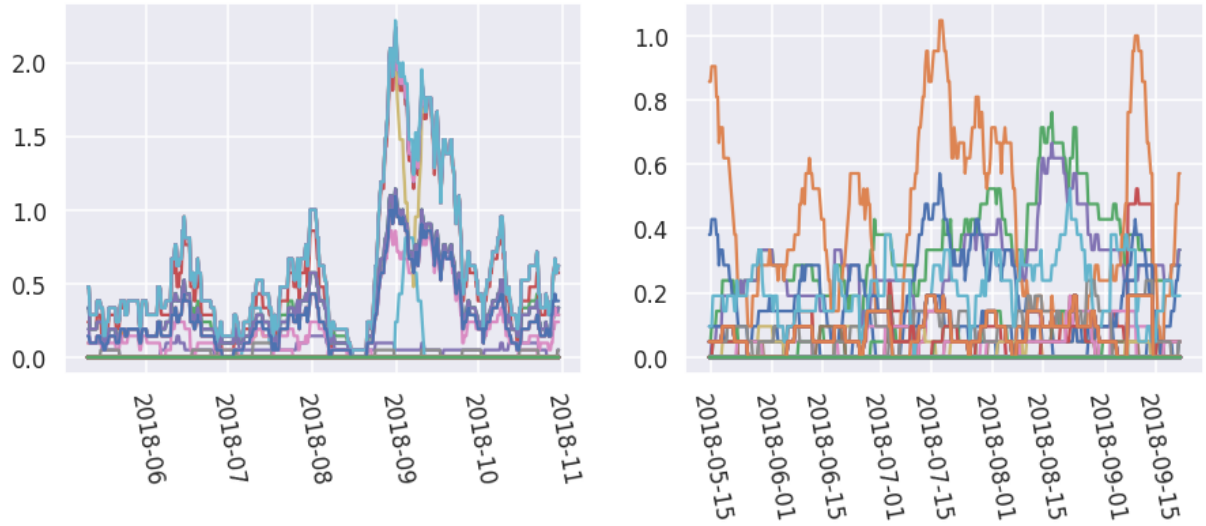

Figure 5.6: Two time series smoothed by a rolling mean, ending with a distribution failure (colour legend is disabled due to the high number of features).

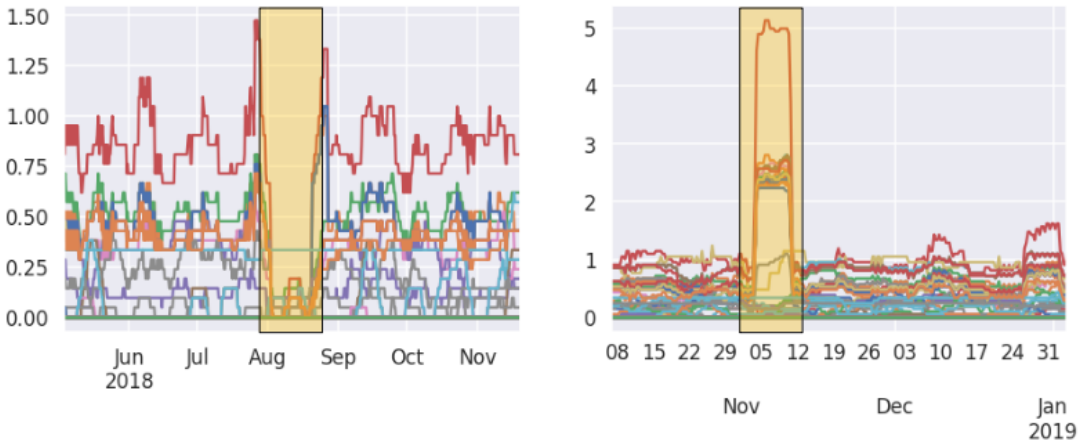

Figure 5.7: Two life cycles with early anomalies that should not raise maintenance alerts.

other failures can have an influence on the event codes that are kept as features for the ATM dataset.

In the following, we choose to focus on predicting the failures of the distribution module, which is the first and most costly cause of hardware failures. We use a binary context, where class 1 is a life cycle ending by a failure of the distribution module, and class 0 is a life cycle ending by any other kinds of failure (e.g., card reader, empty cash safe, connexion problems, unknown causes ...). The objective of the classification problem then becomes to find patterns that are specific to life cycles ending with a distribution module failure, or that only occurs in life cycles ending with other types of failures.

5.3 Experiments with existing approaches

In this section, we adapt the experimental protocol described in Section 4.3, which considers that prediction are made independently of the cause of failure, to the ATM use case. We then compute baseline results using the most common approaches in the literature, which is to slice time series into moving windows.

In these experiments, we are interested in predicting the failure of the distribution module and must thus change how the class of each moving window is affected compared to Section 4.1.3 where we predicted failure independently of the cause. Given a window W_i, class 1 represents a window in the interval $[m-(pp+rd), m-rd]$ (i.e. where we should raise alerts) **only** for life cycles ending with a distribution module failure. All other cases are given class 0, as illustrated by Figure 5.8. Using this formulation, we are interested in how accurately a predictive model can raise alerts, not only at the right moment, but also for the right cause.

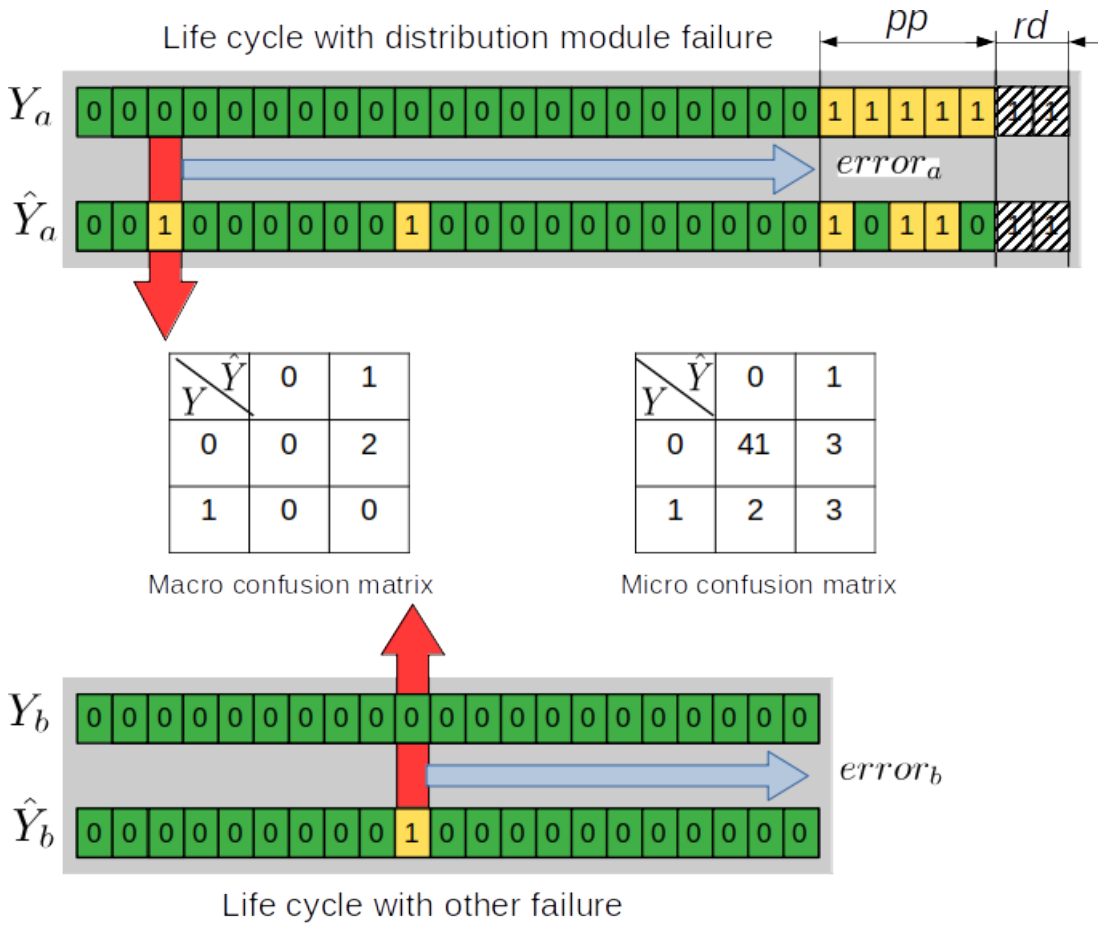

Figure 5.8: How micro and macro metrics are computed for the ATM dataset given two life cycle a, b, focusing on the distribution module failure only.

The estimation of the global cost $\hat{C}_{global}$ must then be defined in the context of a predictive maintenance system that **must only** predict failures of the distribution module.

We redefine $\hat{C}_{global}$ in Equation 5.2, where z_i represent the true class of life cycle X_i ($z_i = 1$ for distribution module failure, else $z_i = 0$) and $\hat{Y}_i$ is the predicted class for each window of X_i.

When $\max(\hat{Y}_i) = 0$, it means that no alert was raised for all the windows of X_i, which is the desired behaviour when $z_i = 0$. Similarly, when $\max(\hat{Y}_i) = 1$, it means that at least one alert was raised in the windows of X_i, which is the desired behaviour when $z_i = 1$. In this last case, the location of the first alert will be used to estimate $error_i$, as the number of timestamps before the acceptable maintenance interval.

$$\hat{C}_{global} = \sum_{i=1}^{n} \begin{cases} \lambda_{neg} & \text{if } z_i = 1 \\ 0 & \text{if } z_i = 0 \end{cases} - \sum_{i=1}^{n} \begin{cases} \lambda_{pos} + error_i \times \lambda_\epsilon & \text{if } \max(\hat{Y}_i) = 1 \\ \lambda_{neg} & \text{if } z_i = 1 \text{ and } \max(\hat{Y}_i) = 0 \\ 0 & \text{if } z_i = 0 \text{ and } \max(\hat{Y}_i) = 0 \end{cases} \tag{5.2}$$

To create baselines for the ATM dataset, we use the following approaches:

- Using a sliding window approach, we apply Catch22 [52], introduced in Section 2.2, to extract features for each window. It will extract the set of 22 discriminative features for each feature of the life cycles. Then we use a Ridge Classifier (Catch22 + Ridge) and a Random Forest Classifier (Catch22 + RF) to experiment with linear and non-linear classifiers.

- Similarly, we use the MultiRocket [75] classifier with a sliding window approach.

In terms of parameters, we use a predictive padding (pp) of 7 days and a responsible duration (rd) of 1 day, with a window length of 7 days (giving windows of size 21 as we have a frequency of 8 hours) and a step of 1 day between each window, which correspond to the desired parameter of the ATM predictive maintenance application. Due to the numerous constraints behind the maintenance process of the distribution module, we are going to assume $\lambda_{pos} = 0.25$ and $\lambda_{neg} = 1$. We then present our experimental results with different values of λ_ϵ. We use a 10-fold validation and present the sum of $\hat{C}_{global}$ and of the confusion matrices for both the micro and the macro context.

Experimental results

To fix the parameters of each baseline, we use an inner cross validation for each validation split. This means that for each of the 10 validation split, we perform a 5-fold validation using only the training data for each set of parameters, and keep the one that maximizes the global cost. Once the best parameters have been found for a split, the model is trained on the whole training set and evaluated on the testing set, which was never seen in the parameter optimization step. The parameters subject to this optimization are:

- For Catch22 with a Random Forest classifier: the number of trees, the number of features to evaluate per tree split, the number of samples per trees, the strength of the cost complexity pruning, and whether to use balanced class weights.

- For MultiRocket and Catch22 with a Ridge classifier: the strength of the regularization, and whether to use balanced class weights.

Figure 5.9 shows the global cost with varying error cost on the x-axis. The approaches based on Catch22 perform better than MultiRocket, and both stays at a positive cost for all tested values of the error cost, which means that a system using these methods is likely to result in an overall cost-reduction for maintenance operations. Table 5.4 gives the micro confusion matrices, and Table 5.5 gives the number of maintenance operations (i.e. first alerts in the macro context) that would have been performed with the system, distinguishing alerts raised early, in the $[m-(pp+rd), m-rd]$ interval, and for life cycles with a failure cause other than the distribution module. Considering the mediocre results of MultiRocket, we do not include it as a baseline.

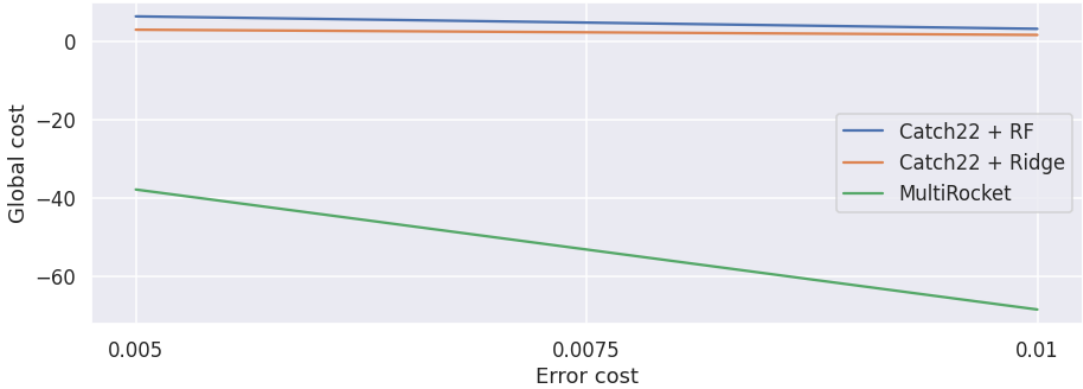

Figure 5.9: Global cost for all baselines with a varying error cost on the x-axis.

Table 5.4: Micro confusion matrices for each baseline on the ATM dataset.

	TN	FP	FN	TP
Catch22 + Ridge	63121.0	92.0	1629.0	12.0
Catch22 + RF	63170.0	43.0	1629.0	12.0
MultiRocket	62488.0	725.0	1547.0	94.0

Table 5.5: Number of first alerts raised early, in the $[m-(pp+rd), m-rd]$ interval (Good), and alerts raised for life cycle with a failure other than the distribution module. The total number of life cycles of class 0 (Other) is 1548, and of class 1 is 159.

	Distribution Early	Distribution Good	Other
Catch22 + Ridge	2	4	9
Catch22 + RF	11	5	10
MultiRocket	33	26	215

5.4 Contributions

We propose a new ensemble classification model, which includes a trigger system based on the survival probability of a given sample. Let us recall that we use as input a multivariate and variable length time series dataset with k features $\mathcal{X} = \{X_1, ..., X_n\}$, with time series $X_i = \{(x_{1,1}, \ldots, x_{1,k}), ..., (x_{m_i,1}, ..., x_{m_i,k})\}$ and y_i its associated class.

5.4.1 Survival RoTation Forest (SRTF)

In this section, we describe a new ensemble classification algorithm, the Survival RoTation Forest (SRTF). First, we show how we use the Random Dilated Shapelet Transform (RDST), introduced in Section 3.7, to transform the multivariate and variable length time series into a tabular representation. Then, given this tabular data and the information about the length of the time series from which features are extracted, we build a Rotation Forest using a multi-objective criterion for choosing tree nodes. This criterion uses both the notion of information gain and the survival probability of the life cycle in the parent and child nodes to decide the best split, composed of a variable and a threshold, for the current node.

Life cycles to features

Few time series classification models are adapted to the case of multivariate and variable length time series, and while survival models can handle such data and predict a remaining useful life, they are not designed to discriminate between different kinds of failures. To the best of our knowledge and given our application constraints, no solution was available in the time series classification literature without applying a moving window extraction.

We use the generalized version of the Random Dilated Shapelet Transform (RDST) defined in Section 3.7 with the following modification to tune it to the characteristics of the use case:

- Instead of extracting argmin from the distance vector between a shapelet and a time series, we extract the maximum. The notion of dissimilarity with a pattern seems more important and simpler to exploit than the location of the minimum given the variable length context and the sparsity of the data. Thus, the three features we extracts are min, max and Shapelet Occurrences.
- We sample the dilation parameter of shapelets based on the minimum length of all inputs.

Given a set of shapelets $\mathcal{S} = \{S_1, \ldots, S_q\}$ and a set of life cycles $\mathcal{X} = \{X_1, \ldots, X_n\}$, RDST outputs a feature matrix of size $(n, 3q)$, which is used as input for our classification model. The model also takes as input a vector $\mathcal{T} = \{T_1, \ldots, T_n\}$ with T_i the length of the life cycle X_i, that information will be used as survival data. In our experiments, we will try two methods, one where we sample shapelets from the whole series, and one where we only sample them in the target interval for maintenance alerts (i.e. in $[m - (pp + rd), m]$.

Note that the usage of a variable length time series classification method instead of a moving window scheme necessitate an adaptation of the experimental protocol. We show how the experimental protocol for the ATM dataset, described in Section 5.3, can be adapted to the case of variable length time series in Section 5.5.1.

Survival Classification Tree (SCTree)

From now on, we denote survival functions as U to avoid a conflict of notations with shapelets. Before introducing the ensemble method, we present its individual component, the Survival Classification Tree. The idea behind its design is to have a splitting criterion that favour features that are both discriminative for classification and that isolate subpopulations based on their survival (e.g. life cycles with a duration inferior to a month and the others). The goal is to be able to isolate short life cycles from long ones in the tree, as we expect the cause of failure to be different (e.g., random problem for short life cycles against wear of mechanical equipment for long ones). We view this as a multi-objective problem where we aim at:

- maximizing the information gain using the Gini impurity criterion to select discriminant features,
- maximizing the difference between the survival functions, which are estimated from the duration of life cycles in the child nodes resulting from a split.

Given a node and the vector $\mathcal{T}$ of life cycle durations of the samples in this node, we can estimate the survival function U of the node by using the Kaplan-Meier estimator, presented in Section 4.5. By using $\mathcal{T}$ and a value t_i, we set d_i as the number of life cycle failing at time t_i and n_i as the number of life cycles of duration superior to t_i to fit a Kaplan-Meier estimator as:

$$U(t) = \prod_{i:t_i<t} \left(1 - \frac{d_i}{n_i}\right) \tag{5.3}$$

Then, for a node and a candidate split, we have a survival function U and the class distribution Y of the node, along with the survival functions and the class distributions of the child nodes (U_l, Y_l for the left child, U_r, Y_r for the right child). We evaluate the quality of a candidate split as follows. First, the information gain IF_{Gain}, using Gini impurity, is computed as usual as:

$$IF_{Gain} = Gini(Y) - \frac{||Y_l||}{||Y||}Gini(Y_l) - \frac{||Y_r||}{||Y||}Gini(Y_r) \tag{5.4}$$

with $Gini(Y)$ being equal to one minus the sum of each class probability squared. Then, using the mean squared Euclidean distance $d(U, U') = \frac{1}{\max(\mathcal{T})}\sum_i^{\max(\mathcal{T})}(u_i - u'_i)^2$, between two survival functions, we compute the survival gain U_{Gain} as:

$$U_{Gain} = \frac{1}{2}\left(\frac{||Y_l||}{||Y||}d(U, U_l) + \frac{||Y_r||}{||Y||}d(U, U_r)\right) \tag{5.5}$$

A squared Euclidean distance is used to favour larger differences between survival curves. Note that when using sample weights (e.g. in an unbalanced class scenario), $||Y||$ becomes the sum of weights of all samples.

As all survival functions take theirs values in $[0, 1]$, and we consider the mean of the squared Euclidean distance, which for each point is also bounded between $[0, 1]$, the U_{Gain} is divided by two in Equation 5.5 to match the bounds of the IF_{Gain}, as $d(U, U') \in [0, 1]$. Both U_{Gain} and IF_{Gain} are then bounded between $[0, 0.5]$ with a higher value meaning a

higher quality split. Our goal is then to find a split that maximizes the sum of those two quality measures, weighted by μ. The quality of the node is then computed as a sum of the U_{Gain} and IF_{Gain} weighted by a parameter $\mu \in [0, 1]$, which controls the importance of the survival gain during training:

$$Quality(Node) = (1 - \mu) \times IF_{Gain} + \mu \times U_{Gain} \tag{5.6}$$

Figure 5.10 gives a visual example of the resulting split of a survival function. To build a tree, we recursively select the best candidate split from the root node until we obtain pure leaves. Then a cost complexity pruning is applied to control over-fitting.

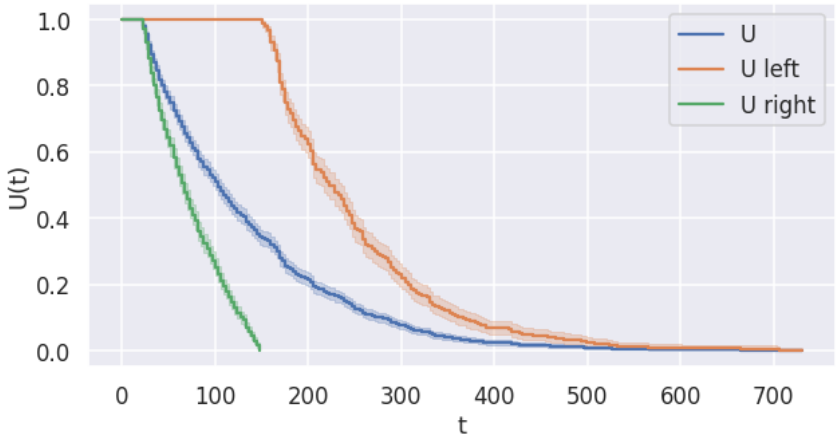

Figure 5.10: Example of a survival function split resulting from the maximization of Equation 5.6

Finally, in each leaf of the tree, a survival function is estimated using the samples of this leaf. Figure 5.11 gives an example of the result of this process for a tree. When doing prediction, the survival function of a leaf allows us to estimate the survival probability given the current length of a life cycle, along with the predicted class.

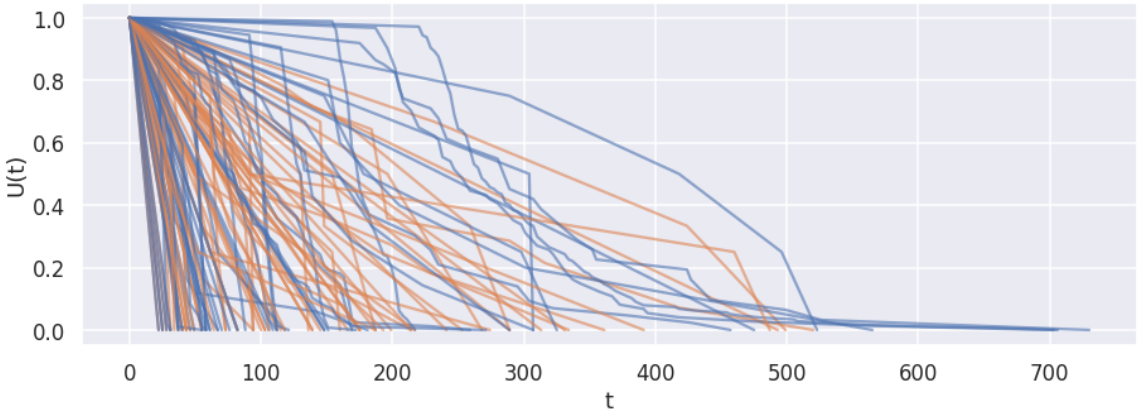

Figure 5.11: Survival functions for all leaves of a non-pruned SCTree fitted on the ATM dataset, with samples of class 1 in oranges and of class 0 in blue.

Trigger system

Inspired by the strategy used in early classification, we want to add a trigger system to optimize the time when an alert is raised. The two intervals of interest for the trigger system are the responsive duration rd, which gives the time necessary to perform maintenance and the predictive padding pp, which indicates an acceptable maintenance time.

Given the survival function S of a leaf, the length of the sample at this leaf $\mathcal{T} = \{T_1, ..., T_n\}$ and the two intervals pp, rd, we propose to define a threshold for the survival probability, below which it is acceptable to raise a predictive maintenance alert. For each sample i of the leaf, $T_i - (pp + rd)$ gives a lower bound for the acceptable maintenance interval, which correspond to the start of the green area shown in Figure 4.2. We define the threshold λ_S as the mean of these lower bounds for all samples in the leaf:

$$\lambda_S(S, \mathcal{T}) = \frac{1}{||n||} \sum_{i}^{n} S(T_i - pp + rd) \tag{5.7}$$

For the ATM use case, an alert can of course only be raised if the majority class of the leaf is the class 1 (i.e. the class characterizing failures of the distribution module). Then for leaves where the majority class is 1, given an unseen sample of length T, if we have $S(T) <= \lambda_S$, then class 1 will be returned, otherwise we return class 0 as prediction (i.e. no alert) and wait for the next batch of data which will increase the value of T.

Ensemble method

To build the Survival RoTation Forest (SRTF), we extend the Rotation Forest classifier [62] with the Survival Classification Trees (SCTree) we just defined. We choose to use a Rotation Forest because, on average, for continuous variables such as the ones generated by shapelet based algorithms, a Rotation Forest is a better classifier compared to other approaches such as Random Forest, XGBoost or support vector machines [4].

A Rotation Tree takes as input a feature matrix X of size (n, m), a vector Y of size n containing the labels for all samples, a parameter K for the number of random feature subsets to generate, and a parameter P for the size of the subsamples affected to each subset. For each feature subset (of size m/K), a random subset of the classes is removed, and a random subsampling of P samples of the remaining class is made. A PCA transformation is then fitted on this specific feature subset and samples, and then applied to all samples in X, for all feature subsets. This transformed data is then used to fit a decision tree, and the transformation is performed again on new samples when predicting. In our case, the decision tree is simply replaced by the Survival classification tree.

The Survival RoTation Forest (SRTF) method, described in Algorithm 6, takes as inputs:

- a set of multivariate time series $\mathcal{X} = \{X_1, \ldots, X_n\}$ with their respective classes $Y = \{Y_1, \ldots, Y_n\}$ and their lengths $\mathcal{T} = \{T_1, \ldots, T_n\}$.
- the parameters for the shapelet algorithm (RDST): a set of possible lengths (shp_len), the number of shapelets to initialize (n_shp), and the minimum length of the series (min_len).

- the parameters of the rotation forest: the number of distinct PCA transformation (K) and the proportion of samples used to fit each PCA(P) and the number of trees to build (n_trees).

Given each component of the method, namely, RDST and a set of SCTrees with their associated PCA transforms, we can then predict the state of a life cycle as follows:

1. Transform the input time series with all the shapelets previously initialized by RDST, to extract features and remove the time component.

2. For each SCTree, transform the data using the fitted PCA transformations (recall that one feature is linked to only one of the PCA transforms for each tree), and predict the leaves on which the samples will end up.

3. For a sample, give as prediction the majority class out all predictions of the SCTrees.

Algorithm 6 Survival RoTation Forest (SRTF)

Require: $\mathcal{X}$, Y, T, n_shp, shp_len, K, P, n_trees
  RDST ← RDST(n_shp, shp_len).fit($\mathcal{X}$, Y)
  X_shp ← RDST.transform($\mathcal{X}$)
  Trees_PCAs ← empty_list()
  Trees ← empty_list()
  for i in range(n_trees) **do**
    Trees_PCAs[i] ← fit_PCAs_subsets(X_shp, Y, K, P)
    X_tree ← Trees_PCAs[i].transform(X_shp)
    Trees[i] ← SCTree().fit(X_Tree, T, Y)
  end for
  return RDST, Trees_PCAs, Trees

5.5 Experiments

In this section, we use the same protocol as we used to establish the baseline results in Section 4.3, with the difference that our approach uses increasing windows, rather than sliding windows. By performing experiments with our proposed model, we aim at answering the following questions:

- What is the impact of using the survival gain during the construction of the survival trees?

- Is using a Rotation Forest worth it compared to a Random Forest?

- Does our contribution improve on the baseline results?

- Is sampling shapelets near the end of the life cycles a better approach than on the whole length?

Similarly to the baseline experiments, we set the parameters of each method using an inner cross validation for each validation split, with the addition of the parameters of the Rotation Forest (i.e. proportion of samples for each PCA and number of PCA) and the parameter μ, which controls the importance of the survival gain in SRTF. Table 5.6 names and summarizes the characteristics of the algorithms we compare in this section. When Survival gain is not used, μ is set to 0 in Equation 5.6. If Rotation Forest is not used, no PCA transformations are applied on the features given to the forest, and if shapelets are sampled at the end of the life cycles, we only give the data located in $[m_i - (pp + rd), m_i]$ for all life cycles as input to fit RDST.

Name	Survival gain	Rotation Forest	Shapelet sampled at the end
SRTF	✓	✓	
SRTF Gini		✓	
SRF	✓		
SRTF End	✓	✓	✓
SRTF Gini End		✓	✓
SRF End	✓		✓

Table 5.6: Characteristics used by each variation of SRTF for the ATM dataset experiments.

5.5.1 Experimental protocol for variable length methods

The experimental protocol for variable length methods is very similar to the one used for the baseline results. Consider a validation fold with $\mathcal{X}_{Train}$ the training set and $\mathcal{X}_{Test}$ the testing set, both composed of life cycles. The first change compared to the baseline protocol is to use $\mathcal{X}_{Train}$ directly for training the model, as it can handle variable length time series. We rely on the trigger system to avoid early alerts, and let the model focus on discriminating the failure of the distribution module of the others.

The second change is on the use of the testing set. Using the whole life cycle would not be fair compared to the moving window scheme, as we need to simulate the stream of data that would be emitted by the machine in the real use case. To do this, we use an expanding window scheme, such as for a life cycle $X_i \in \mathcal{X}_{Test}$ of size m, we extract a set of expanding windows $\mathcal{W}_i = \{W_1, \ldots, W_m - (l-1)\}$, with l the starting length of the expanding window. An expanding window $W_j \in \mathcal{W}_i$ is defined as $W_j = \{x_1, \ldots, x_{j+(l-1)}\}$.

By using the same value for the l parameter for both moving and expanding window create the same number of windows, which allow for a fair comparison between the baseline results and the variable length method. The way we compute the cost and extract micro and macro metrics does not change for expanding windows. Figure 5.12 illustrates the difference between the two protocols.

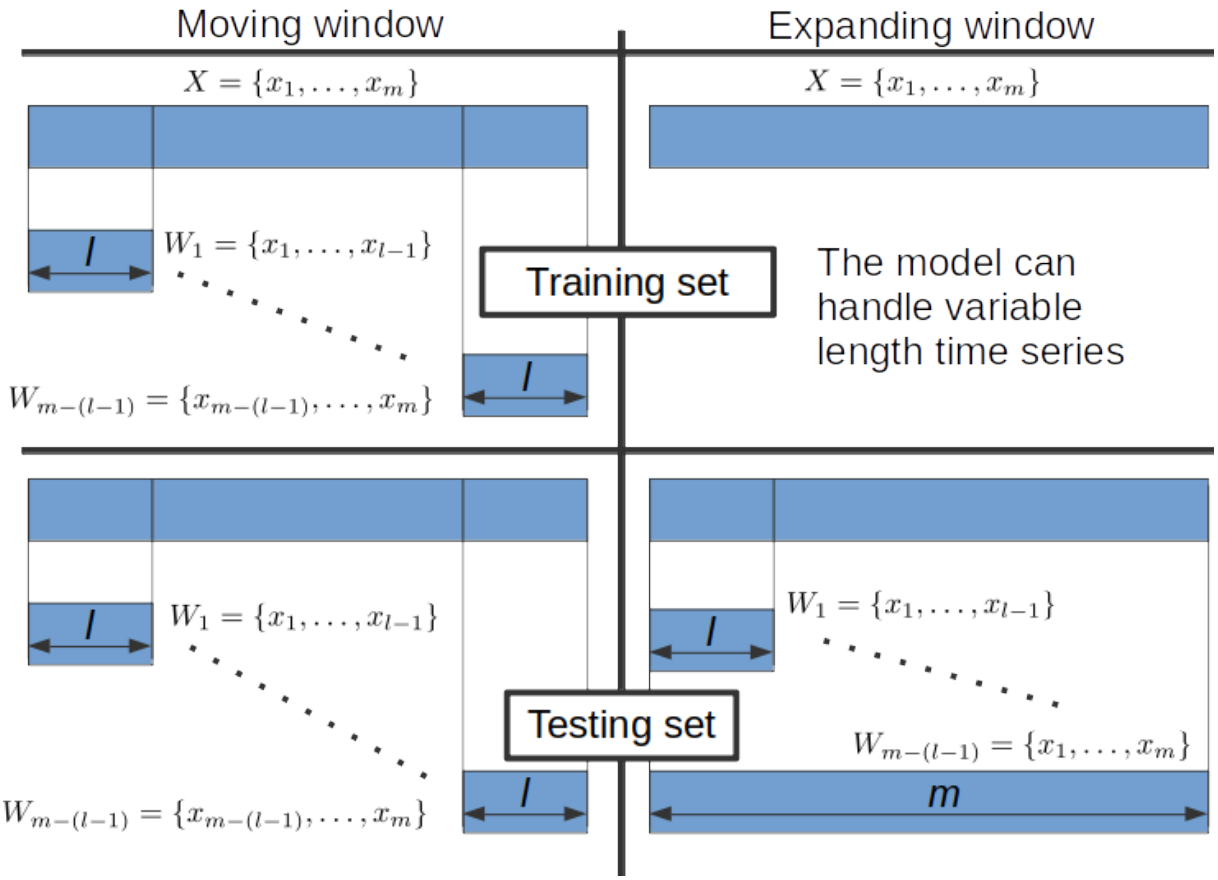

Figure 5.12: The differences between the experimental protocol used for the baseline results that are using moving window, and for our variable length time series method. We use expanding windows to simulate the stream of data emitted by the machine.

5.6 Experimental results

The results for the global cost are given in Figure 5.13, the micro confusion matrices in Table 5.7 and the maintenance operations caused by the first alert in all life cycles in Table 5.8. We see that Catch22 with a Random Forest remains a very strong baseline, with only SRTF End having a better global cost for all tested values of the error cost. While methods based on shapelets raise significantly more true and false positive in the micro context compared to Catch22, they do not perform significantly more maintenance operations, meaning that they raise more alerts for each life cycle. With more early alerts raised, shapelet methods are also less valuable the more the error cost grows, the exception being SRTF End, which, compared to Catch22, balance its higher number of early maintenance by a higher number of successful ones.

The use of the rotation forest clearly improves the results for shapelet methods, with the worst global cost being SRF and SRF End, the two variations not using the PCA transformations. Despite a higher number of alerts raised for life cycles with distribution module failures, the two methods also raise a lot of alerts for other life cycles. In this regard, the superior results of methods using the PCA transformations may be due to the fact that the PCAs are fitted on a unique class, which highly reduce the number of alert

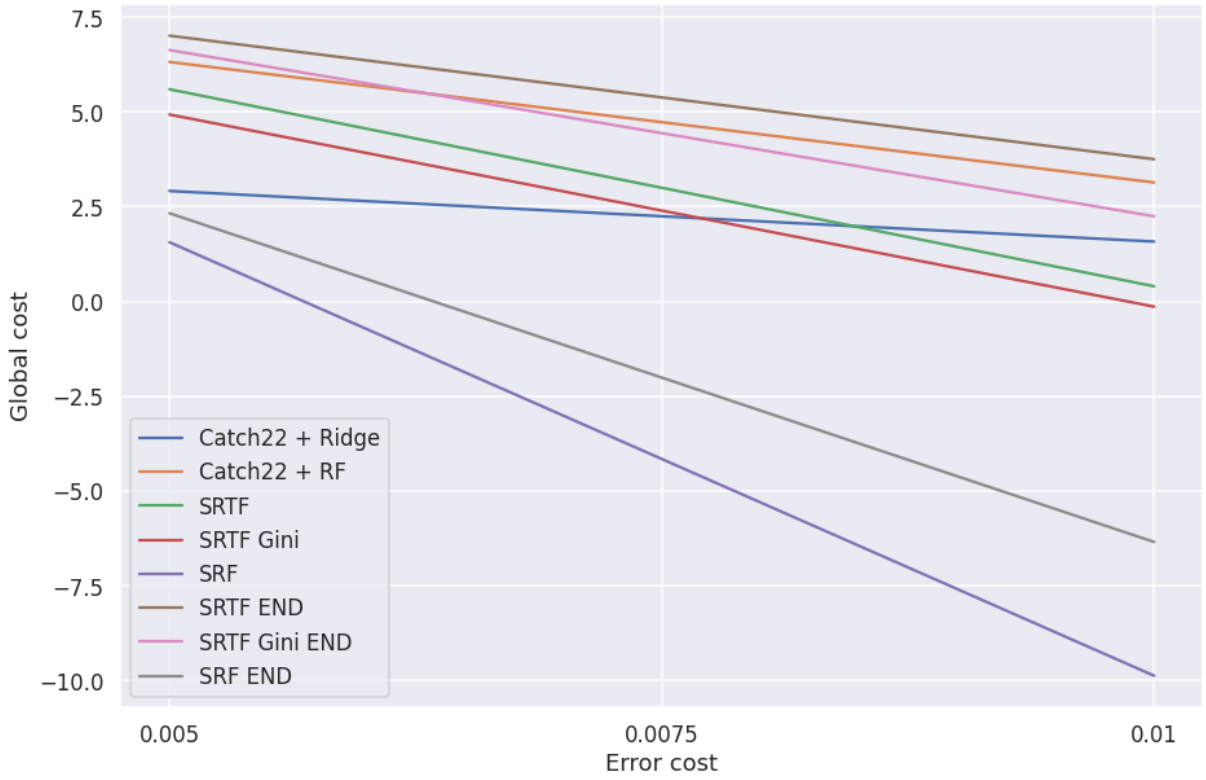

Figure 5.13: Global cost for all methods with a varying error cost on the x-axis.

raised for life cycles with other causes of failure as shown in Table 5.8.

	TN	FP	FN	TP
Catch22 + Ridge	63121	92	1629	12
Catch22 + RF	63170	43	1629	12
SRTF	62712	501	1522	119
SRTF Gini	62524	689	1514	127
SRF	62314	899	1520	121
SRTF End	62832	381	1540	101
SRTF Gini End	62682	531	1502	139
SRF End	62512	701	1522	119

Table 5.7: Micro confusion matrices for all methods.

Sampling shapelets at the end of the life cycles, in the target interval fixed by the predictive padding and the responsive duration, also seems to have a positive effect on the results, as the global cost of methods using the whole life cycles to sample shapelets is always inferior to its counterpart. Considering the quality of the data, sampling shapelets using only the end of life cycles may help to increase the quality of the extracted set of shapelets, notably by avoiding sampling shapelets on early or irrelevant anomalies.

	Distribution Early	Distribution Good	Other
Catch22 + Ridge	2	4	9
Catch22 + RF	11	5	10
SRTF	20	3	19
SRTF Gini	16	2	17
SRF	24	7	40
SRTF End	14	7	14
SRTF Gini End	17	4	15
SRF End	20	6	35

Table 5.8: Number of maintenance operations that would have been performed by each model, considering only the first alert raised for each life cycle.

5.7 Conclusion

In this chapter, we presented the ATM predictive maintenance use case proposed by Worldline. We showed how we obtained the log and failure data for a set of 460 ATMs located in France over a period of two years, and how we create the ATM dataset from these data sources. We then conducted experiments using existing approaches, based on a moving window approach to deal with the variable length of the series. The first results showed that a predictive maintenance model built using this approach could be cost-effective under our proposed experimental protocol.

Then, we presented our contribution for this use case, including a new ensemble classification model using a trigger system inspired by the early classification field and based on the estimated survival probability of life cycles. Another set of experiments show that it is possible to improve on the baseline results using this new approach.

Despite our best effort, while some models achieve a cost reduction, we estimate that the results obtained on the ATM dataset are not satisfying enough yet for a product release. This is due to the quality of the data, which is clearly lacking in both the log and maintenance data. An effort to change the data source is currently ongoing at Worldline, which could drastically increase the quality of the log data, and thus, the performance of the predictive models. While obtaining maintenance data is still a challenge due to the commercial interest of maintainers, with higher quality log data, the tool that we presented in Section 5.2.1 to extract failure data may give better results, without having to rely on maintenance tickets for validation.

Chapter 6

Conclusion and Future Works

Initially, this thesis was motivated by a predictive maintenance use case, which, given the temporal nature of the data, pushed us toward studying time series algorithms. After experimenting with the state-of-the-art algorithms, we wanted to create an accurate and scalable method, similarly to the results of ROCKET [22], while also being easily interpretable.

We began by studying shapelets [86] due to their natural interpretability. The issue was that the accuracy of shapelet methods was far behind recent time series methods, such as ROCKET [22], InceptionTime [36] or HIVE-COTE [50, 58]. While some shapelet algorithms achieved reasonable scalability, it was at the cost of accuracy, notably by using random approaches for generating shapelets.

To solve those issues, our objective was first to enhance the discriminative power of shapelets, in order to rely on the scalability of random approaches to achieve our goals. We presented those improvements in Chapter 3, the first one being the addition of dilation into the shapelet formulation. Dilation allows shapelets to match with non-contiguous subsequences, which increase the diversity of patterns that can be represented. A very intuitive example of the benefit of dilation can found on data influenced by the heart beats of a subject (e.g. PigCVP dataset), where a dilated shapelet can "synchronize" with the cardiac frequency and represent a sequence of points, each spaced by the duration of a heart-beat. More generally, for any cyclic component in a time series and supposing that the frequency does not change, dilated shapelets can represent relations between points of successive cycles. For example, in the heart-beat case, it could represent a successive decrease in the blood pressure for l successive heart-beats, with l the length of the shapelet.

The second major addition to the shapelet formulation was the Shapelet Occurrences (SO) feature. To produce features, a shapelet computes a distance vector between itself and a time series, and then extracts the minimum of this vector as a feature. Following the work of [32], the argmin is also extracted to discriminate cases where the location of a pattern is discriminant, rather than its presence or absence in the series. One case that is not covered by extracting the minimum and argmin is the case where a pattern is present in all classes, but occurs a different number of times in each. To solve this issue, we added another feature, which given a threshold λ, count the number of points in the distance vector that are below this threshold. The intuition being to count the number of times a shapelet is λ-close to the time series.

To use these contributions, we presented the Random Dilated Shapelet Transform

(RDST), a time series classification algorithm that randomly extracts shapelets from the input time series, given lengths and dilations parameters, also randomly initialized. We introduced multiple optimizations to further increase the scalability of the method, and some additional parameters to tune the shapelet sampling process, notably with α-similarity, to reduce the number of similar shapelets sampled by the random algorithm.

As previous research in the field of time series classification [50, 14, 75] have shown, considering multiple representations of the input is beneficial to find discriminative properties between the classes of a problem. Following this observation, we introduced the RDST Ensemble, which further improved the accuracy of RDST by applying it to different input representations, and combining the predictions made on each representation using a weighted majority voting scheme defined by [58]. This allowed the method to achieve comparable accuracy to the most accurate time series classification algorithms, while keeping its scalability and interpretability.

After focusing on time series classification algorithms, we studied the field of predictive maintenance. First, we looked at how the problem is formulated, for both classification and regression tasks, with the common goal of predicting when a machine is going to fail. Different kind of approaches can then be implemented to solve the problem: rule-based systems, anomaly or change point detection, supervised models, etc. One common pitfall in the machine learning literature is that, when dealing with predictive maintenance applications, models are evaluated using metrics such as accuracy or mean absolute error, which fail to consider the costs related to the maintenance process, and the particularity of the application. We defined a specific experimental protocol for predictive maintenance applications, which, given a predictive maintenance model to evaluate and the costs associated with the maintenance process, estimate how worth it is compared to a baseline maintenance system.

We then introduced the ATM predictive maintenance use case, and detailed how we extracted the ATM dataset from the event log data generated by a fleet of ATM. To extract life cycles from the event logs, we first presented a tool to identify failures based on the seasonality of withdrawal operations for an ATM. If no withdrawal was performed during a period, and that we expected withdrawal to happen during this period, based on the known seasonality, an anomaly score proportional to the expected amount was affected. For each successive period without withdrawals, a cumulative product was used to compute the final anomaly score. Failure periods were then selected using a threshold on the anomaly score.

Once the ATM dataset was extracted, we conducted some experiments to obtain baseline results. To improve on these results, we proposed a classification model, the Survival Rotation Forest (SRTF). It modifies a classification tree algorithm by adding the notion of survival gain during candidate split evaluation, which quantify the ability of a candidate split to isolate short and long life cycles. We also added a trigger system based on a survival function estimated at each leaf of a tree. Given the constraints of the maintenance process, we estimate a threshold on the survival probability of ongoing life cycles to classify, above which the prediction would be delayed. Using this method, with RDST as a feature extraction step, and by restricting the shapelet sampling to the end of the life cycles, we slightly improved on the baseline results.

6.1 Future Works

Considering our contributions, multiple axis for future works can be considered :

- Additional optimization for dilated shapelets, notably on the distance vectors computation. For example, consider a time series $X = \{x_1, \ldots, x_m\}$ and the set of shapelets $\mathcal{S}^{l,d}$ with parameters (l, d). Currently, we extract the subsequences $X^{i,l,d}$ and compute the distance between each shapelet and subsequences in a loop. Existing methods to optimize distance matrix computations, such as the ones presented in [2], could be adapted to this context, by replacing the loop with matrix operations, which could be computed more efficiently.

- New approaches for multivariate shapelets. In our current formulation, the multivariate case is treated as the sum of the distance vectors obtained from a set of univariate shapelets and time series. There is no explicit relation measured between the features. A solution could be to modify the definition of a multivariate shapelet by adding a vector $F = \{f_1, \ldots, f_l\}$ that indicates the feature targeted by the value s_i, as well as a vector $T = \{t_1, \ldots, t_l\}$ that indicates the progression in time between each point s_i. Then, by modifying the computation of a point i of the distance vector as:

$$v_i = \sum_{j=1}^{l} s_j - x_{i+(t_j \times d), f_j} \tag{6.1}$$

  We could first compare a point x_{i,f_1} and then a point x_{i,f_2} of another feature, but at the same timestamp, supposing we have $t_1 = 0$.

- In our current formulation, the dilation parameter remains fixed between all values of a shapelet. It may be interesting to make the dilation vary between each value to represent more complex patterns. By adding another parameter $D = \{d_1, \ldots, d_l\}$, the dilation applied at each step of the distance computation could vary. We hope that it would allow matching a cyclic component which change its frequency in the series. For example, discriminating two classes composed of sinusoidal functions, which change their frequency after each cycle, but with the values of those changes being specific to each class.

Other works could also be carried out on shapelet algorithm. Rather than being divided in two steps: the shapelet candidates generation and the extraction of the features, multiple agents could simultaneously conduct a search for the best candidates. They would then orient their search based on the quality of the shapelets extracted by other agents. In other words, adapting reinforcement learning techniques to the context of shapelets candidates generation.

Concerning our works on predictive maintenance, we assumed a particular modelization of the problem by using life cycles to slice the data. We would like to explore other approaches, and see how they would influence the results and the experimental protocol we defined. The cost estimation of a maintenance alert could also be further refined to be as close as possible to the reality of a predictive maintenance system. Another interesting idea would be to adapt the cost of an alert as a loss function to optimize during training, similarly to the works of the field of early classification on trigger systems. Ideally, the

experimental protocol should be consolidated into a more formal framework, to allow for the evaluation of any type of predictive maintenance model and problem modelization. Obtaining comparable results across multiple kinds of approaches and models would be a great start toward building a benchmark for predictive maintenance tasks, similarly to the UCR and UEA archive for time series classification.

Appendix A

Additional results for RDST experiments

All the results table, for all experiments, including the standard deviation for each dataset are available online on the project repository at https://github.com/baraline/convst/.

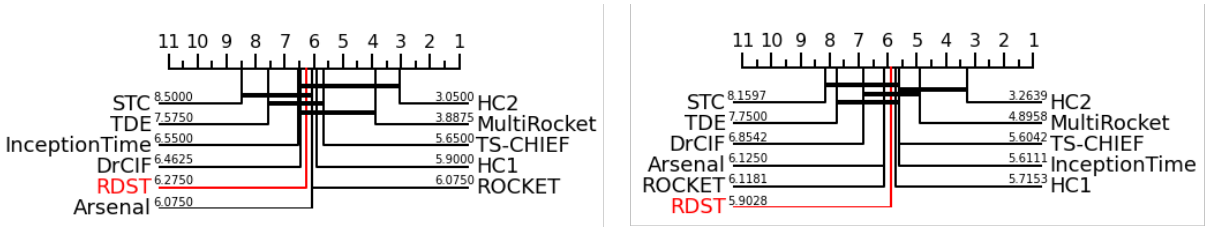

Figure A.1: Critical diagrams for the accuracy of RDST on the 112 UCR univariate datasets. The left diagram is for the 40 datasets used in the sensitivity analysis of RDST, the right diagram is for the 72 others.

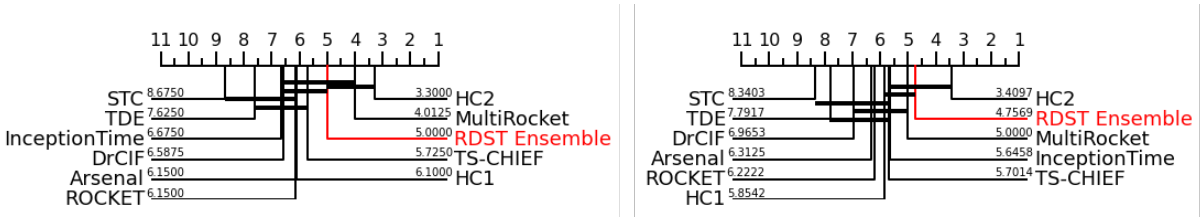

Figure A.2: Critical diagrams for the accuracy of RDST Ensemble on the 112 UCR univariate datasets. The left diagram is for the 40 datasets used in the sensitivity analysis of RDST, the right diagram is for the 72 others.

Bibliography

[1] Youssef Achenchabe, Alexis Bondu, Antoine Cornuéjols, and Asma Dachraoui. Early classification of time series: Cost-based optimization criterion and algorithms. *Mach. Learn.*, 110(6):1481–1504, jun 2021.

[2] Mélodie Angeletti, Jean-Marie Bonny, and Jonas Koko. Parallel euclidean distance matrix computation on big datasets. 2019.

[3] A. Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron George Bostrom, Paul Southam, and Eamonn J. Keogh. The uea multivariate time series classification archive, 2018. *ArXiv*, abs/1811.00075, 2018.

[4] A. Bagnall, M. Flynn, J. Large, J. Line, A. Bostrom, and G. Cawley. Is rotation forest the best classifier for problems with continuous features? *arXiv*, 2020.

[5] Anthony Bagnall, Eamonn Keogh, Jason Lines, Aaron Bostrom, James Large, and Matthew Middlehurst. The UEA & UCR Time Series Classification Repository.

[6] Xanthi Bampoula, Georgios Siaterlis, Nikolaos Nikolakis, and Kosmas Alexopoulos. A deep learning model for predictive maintenance in cyber-physical production systems using lstm autoencoders. *Sensors*, 21(3), 2021.

[7] Márcia Baptista, Shankar Sankararaman, Ivo de Medeiros, Cairo Nascimento Jr, Helmut Prendinger, and Elsa Henriques. Forecasting fault events for predictive maintenance using data-driven techniques and arma modeling. *Computers & Industrial Engineering*, 115, 2017.

[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 Mining and Knowledge Discovery*, 28:634–669, 5 2014.

[9] Alexis Bondu, Youssef Achenchabe, Albert Bifet, Fabrice Clérot, Antoine Cornuéjols, Joao Gama, Georges Hébrail, Vincent Lemaire, and Pierre-François Marteau. Open challenges for machine learning based early decision-making research, 2022.

[10] Aaron George Bostrom. *Shapelet Transforms for Univariate andMultivariate Time Series Classification.* PhD thesis, University of East Anglia, 2018.

[11] Aaron George Bostrom and A. Bagnall. Binary shapelet transform for multiclass time series classification. *Trans. Large Scale Data Knowl. Centered Syst.*, 32:24–46, 2017.

[12] Aaron George Bostrom and A. Bagnall. A shapelet transform for multivariate time series classification. In *Proceedings of the 3rd ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 2018.

[13] Imad Bou-Hamad, Denis Larocque, and Hatem Ben-Ameur. A review of survival trees. *Statistics Surveys*, 5(none):44 – 71, 2011.

[14] Nestor Cabello, Elham Naghizade, Jianzhong Qi, and Lars Kulik. Fast and accurate time series classification through supervised interval search. pages 948–953, 11 2020.

[15] Matteo Calabrese, Martin Cimmino, Francesca Fiume, Martina Manfrin, Luca Romeo, Silvia Ceccacci, Marina Paolanti, Giuseppe Toscano, Giovanni Ciandrini, Alberto Carrotta, Maura Mengoni, Emanuele Frontoni, and Dimos Kapetis. Sophia: An event-based iot and machine learning architecture for predictive maintenance in industry 4.0. *Information*, 11(4), 2020.

[16] Qiushi Cao, Cecilia Zanni-Merk, Ahmed Samet, Christoph Reich, François de Bertrand de Beuvron, Arnold Beckmann, and Cinzia Giannetti. Kspmi: A knowledge-based system for predictive maintenance in industry 4.0. *Robotics and Computer-Integrated Manufacturing*, 74:102281, 2022.

[17] D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972.

[18] Asma Dachraoui, Alexis Bondu, and Antoine Cornuéjols. Early classification of time series as a non myopic sequential decision making problem. In Annalisa Appice, Pedro Pereira Rodrigues, Vítor Santos Costa, Carlos Soares, João Gama, and Alípio Jorge, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 433–447, Cham, 2015. Springer International Publishing.

[19] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

[20] Yann Dauxais and Thomas Guyet. Generalized chronicles for temporal sequence classification. In *AALTD 2020 - 5th ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, pages 1–16, 2020.

[21] L. Decker, D. Leite, L. Giommi, and D. Bonacorsi. Real-time anomaly detection in data centers for log-based predictive maintenance using an evolving fuzzy-rule-based approach. In *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8, 2020.

[22] Angus Dempster, François Petitjean, and Geoffrey Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34, 09 2020.

[23] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the*

27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, KDD '21, page 248–257, New York, NY, USA, 2021. Association for Computing Machinery.

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

[25] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

[26] Christophe Dousson and Thang Vu Duong. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'99, page 620–626, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[27] Ralph Duncan. A survey of parallel computer architectures. *Computer*, 23:5–16, 1990.

[28] Aurora Esteban, Amelia Zafra, and Sebastián Ventura. Data mining in predictive maintenance systems: A taxonomy and systematic review. *WIREs Data Mining and Knowledge Discovery*, page e1471.

[29] Michael Flynn, James Large, and Tony Bagnall. The contract random interval spectral ensemble (c-rise): The effect of contracting a classifier on accuracy. In *Hybrid Artificial Intelligent Systems: 14th International Conference, HAIS 2019, León, Spain, September 4–6, 2019, Proceedings*, page 381–392, Berlin, Heidelberg, 2019. Springer-Verlag.

[30] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. pages 392–401. Association for Computing Machinery, 2014.

[31] Antoine Guillaume, Christel Vrain, and Wael Elloumi. Random dilated shapelet transform: A new approach for time series shapelets. In *Pattern Recognition and Artificial Intelligence*, pages 653–664, Cham, 2022. Springer International Publishing.

[32] Mael Guillemé, Simon Malinowski, Romain Tavenard, and Xavier Renard. Localized random shapelets. In Vincent Lemaire, Simon Malinowski, Anthony Bagnall, Alexis Bondu, Thomas Guyet, and Romain Tavenard, editors, *Advanced Analytics and Learning on Temporal Data*, pages 85–97, Cham, 2020. Springer International Publishing.

[33] Ashish Gupta, Hari Prabhat Gupta, Bhaskar Biswas, and Tanima Dutta. Approaches and applications of early classification of time series: A review. *IEEE Transactions on Artificial Intelligence*, 1(1):47–61, 2020.

[34] Clemens Gutschi, Nikolaus Furian, Josef Suschnigg, Dietmar Neubacher, and Siegfried Voessner. Log-based predictive maintenance in discrete parts manufacturing. *Procedia CIRP*, 79:528–533, 2019. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy.

[35] Guoliang He, Yong Duan, Rong Peng, Xiaoyuan Jing, Tieyun Qian, and Lingling Wang. Early classification on multivariate time series. *Neurocomputing*, 149:777–787, 2015.

[36] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 2020.

[37] Baihong Jin, Yuxin Chen, Dan Li, Kameshwar Poolla, and Alberto Sangiovanni-Vincentelli. A one-class support vector machine calibration method for time series change point detection. In *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–5, 2019.

[38] E. L. Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282):457–481, 1958.

[39] Evgenios Kladis, Charilaos Akasiadis, Evangelos Michelioudakis, Elias Alevizos, and Alexandros Artikis. An empirical evaluation of early time-series classification algorithms. In *EDBT/ICDT Workshops*, 2021.

[40] John P. Klein and Melvin L. Moeschberger. *Survival Analysis : Techniques for Censored and Truncated Data*. Statistics for biology and health. Springer New York : Imprint: Springer, New York, NY, 1st ed. 1997. edition, 1997.

[41] Panagiotis Korvesis, Stephane Besseau, and Michalis Vazirgiannis. Predictive maintenance in aviation: Failure prediction from post-flight reports. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1414–1422, 2018.

[42] Christian Krupitzer, Tim Wagenhals, Marwin Züfle, Veronika Lesch, Dominik Schäfer, Amin Mozaffarin, Janick Edinger, Christian Becker, and Samuel Kounev. A survey on predictive maintenance for industry 4.0, 2020.

[43] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[44] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis and transformation. In *CGO*, pages 75–88, San Jose, CA, USA, Mar 2004.

[45] Huimin Li, Dong Han, Yawen Hou, Huilin Chen, and Zheng Chen. Statistical inference methods for two crossing survival curves: A comparison of methods. *PLOS ONE*, 10(1):1–18, 01 2015.

[46] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: A novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15:107–144, 08 2007.

[47] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29, 2015.

[48] Jason Lines, Luke M. Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. pages 289–297, 2012.

[49] Jason Lines and George Oastler. Ts-quad: A smaller elastic ensemble for time series classification with no reduction in accuracy. In Mounîm El Yacoubi, Eric Granger, Pong Chi Yuen, Umapada Pal, and Nicole Vincent, editors, *Pattern Recognition and Artificial Intelligence*, pages 221–232, Cham, 2022. Springer International Publishing.

[50] Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Trans. Knowl. Discov. Data*, 12(5), jul 2018.

[51] Myriam Lopez, Marie Beurton-Aimar, Gayo Diallo, and Sofian Maabout. Log data preparation for predicting critical errors occurrences. In Álvaro Rocha, Hojjat Adeli, Gintautas Dzemyda, Fernando Moreira, and Ana Maria Ramalho Correia, editors, *Trends and Applications in Information Systems and Technologies*, pages 224–233, Cham, 2021. Springer International Publishing.

[52] Carl Henning Lubba, Sarab Sethi, Philip Knaute, Simon Schultz, Ben Fulcher, and Nick Jones. catch22: Canonical time-series characteristics: Selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery*, 33, 08 2019.

[53] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, 33, 05 2019.

[54] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz Király. sktime: A unified interface for machine learning with time series. 09 2019.

[55] Z. Ma and A. W. Krings. Survival analysis approach to reliability, survivability and prognostics and health management (phm). In *2008 IEEE Aerospace Conference*, pages 1–20, 2008.

[56] Matthew Middlehurst, James Large, and Anthony Bagnall. The canonical interval forest (cif) classifier for time series classification. pages 188–195, 12 2020.

[57] Matthew Middlehurst, James Large, Gavin Cawley, and Anthony Bagnall. The temporal dictionary ensemble (tde) classifier for time series classification. In Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 660–676, Cham, 2021. Springer International Publishing.

[58] Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110, 12 2021.

[59] Usue Mori, Alexander Mendiburu, Sanjoy Dasgupta, and Jose Lozano. Early classification of time series by simultaneously optimizing the accuracy and earliness. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–10, 11 2017.

[60] Thanawin Rakthanmanon and Eamonn Keogh. *Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets*, pages 668–676. 05 2013.

[61] Yongyi Ran, Xin Zhou, Pengfeng Lin, Yonggang Wen, and Ruilong Deng. A survey of predictive maintenance: Systems, purposes and approaches, 2019.

[62] Juan Rodríguez, Ludmila Kuncheva, and Carlos Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28:1619–30, 11 2006.

[63] Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 35(2):401–449, mar 2021.

[64] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[65] Patrick Schäfer. Scalable time series classification. *Data Min. Knowl. Discov.*, 30(5):1273–1298, sep 2016.

[66] Patrick Schäfer and Ulf Leser. Fast and accurate time series classification with weasel. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 637–646, New York, NY, USA, 2017. Association for Computing Machinery.

[67] Roy Schwartz, Jesse Dodge, Noah Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63:54–63, 11 2020.

[68] Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29, 11 2015.

[69] Patrick Schäfer and Ulf Leser. Teaser: early and accurate time series classification. *Data Mining and Knowledge Discovery*, 34, 09 2020.

[70] Chayma Sellami, Ahmed Samet, and Mohamed Anis Bach Tobji. Frequent chronicle mining: Application on predictive maintenance. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1388–1393, 2018.

[71] Mit Shah, Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning dtw-shapelets for time-series classification. Association for Computing Machinery, Inc, 3 2016.

[72] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey Webb. Ts-chief: a scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, 34, 05 2020.

[73] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. Log-based predictive maintenance. pages 1867–1876. Association for Computing Machinery, 2014.

[74] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[75] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I. Webb. Multirocket: Multiple pooling operators and transformations for fast and effective time series classification, 2021.

[76] Chang Wei Tan, François Petitjean, Eamonn J. Keogh, and Geoffrey I. Webb. Time series classification for varying length series. *ArXiv*, abs/1910.04341, 2019.

[77] Chang Wei Tan, François Petitjean, and Geoffrey I. Webb. Fastee: Fast ensembles of elastic distances for time series classification. *Data Mining and Knowledge Discovery*, 34:231–272, 2019.

[78] Romain Tavenard and Simon Malinowski. Cost-aware early classification of time series. In Paolo Frasconi, Niels Landwehr, Giuseppe Manco, and Jilles Vreeken, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 632–647, Cham, 2016. Springer International Publishing.

[79] Gilles Vandewiele, Femke Ongenae, and Filip De Turck. Gendis: Genetic discovery of shapelets. *Sensors*, 21(4), 2021.

[80] J. Wang, C. Li, S. Han, S. Sarkar, and X. Zhou. Predictive maintenance based on event-log analysis: A case study. *IBM J. Res. Dev.*, 61, 2017.

[81] Yichang Wang, Rémi Emonet, Élisa Fromont, Simon Malinowski, and Romain Tavenard. Adversarial regularization for explainable-by-design time series classification. *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1079–1087, 2020.

[82] Martin Wistuba, Josif Grabocka, and Lars Schmidt-Thieme. Ultra-fast shapelets for time series classification, 2015.

[83] Zheng zheng Xing, Jian Pei, and Philip Yu. Early prediction on time series: A nearest neighbor approach. pages 1297–1302, 01 2009.

[84] Zheng zheng Xing, Jian Pei, Philip Yu, and Ke Wang. Extracting interpretable features for early classification on time series. pages 247–258, 04 2011.

[85] Zheng zheng Xing, Jian Pei, Philip Yu, and Ke Wang. Extracting interpretable features for early classification on time series. pages 247–258, 04 2011.

[86] Lexiang Ye and Eamonn Keogh. Time series shapelets: A new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 947–956, New York, NY, USA, 2009. Association for Computing Machinery.

[87] Jesin Zakaria, Abdullah Mueen, and Eamonn Keogh. Clustering time series using unsupervised-shapelets. In *2012 IEEE 12th International Conference on Data Mining*, pages 785–794, 2012.

[88] Hanbo Zhang, Peng Wang, Zicheng Fang, Zeyu Wang, and Wei Wang. Elis++: a shapelet learning approach for accurate and efficient time series classification. *World Wide Web*, 24, 03 2021.

Antoine GUILLAUME

Classification de séries temporelles avec les Shapelets : Application à la maintenance prédictive via journaux d'évènements

Dans cette thèse, nous nous intéréssos à l'apprentissage automatique, et plus particulièrement à la classification supervisée de séries temporelles et son application à la maintenance prédictive. Notre premier objectif est d'utiliser les shapelets, des motifs extraits des séries temporelles, pour construire un algorithme de classification supervisé, permettant de prédire la classe d'une série en fonction de la présence de ces motifs dans la série. Nous proposons plusieurs contributions pour améliorer leurs performances, telles que l'ajout de la notion de dilatation et l'ajout d'un nouveau descripteur qui, à partir d'un seuil de distance, compte le nombre d'occurrences d'une shapelet dans une série. Nous présentons ensuite un algorithme de classification intégrant ces contributions et évaluons ses performances par rapport aux méthodes existantes sur les archives de données de l'Université d'East Anglia (UEA) et de California Riverside (UCR). Nous étudions ensuite les méthodes de classification des séries temporelles pouvant être utilisées pour la maintenance prédictive. Nous formalisons d'abord la tâche d'apprentissage, puis présentons les méthodes utilisées dans la littérature pour produire des modèles adaptés à la maintenance prédictive. Ensuite, nous introduisons un cas d'utilisation industrielle de maintenance prédictive sur des journaux d'événements, issus de distributeurs automatique de billets. Enfin, nous présentons un protocole expérimental, incluant une métrique pour estimer le coût du système de maintenance, et un nouveau modèle de classification spécifiquement conçu pour cette tâche.

Mots clés : Classification des séries temporelles, Shapelets, Maintenance prédictive

Time series classification with Shapelets: Application to predictive maintenance on event logs

In this thesis, we are interested in machine learning, and more specifically in supervised classification of time series and its application to predictive maintenance. Our first objective is to use shapelets, patterns extracted from time series, to build a supervised classification algorithm, allowing to predict the class of a series based on the presence of these patterns in the series. We propose several contributions to improve their performance, such as the addition of the notion of dilation and a new descriptor which, given a distance threshold, counts the number of occurrences of a shapelet in a series. We then present a classification algorithm using these contributions and evaluate its performance against existing methods on the University of East Anglia (UEA) and California Riverside (UCR) data archives. Then, we study time series classification methods that can be used for predictive maintenance. First, we formalize the learning task, then present the methods used in the literature to learn models suitable for predictive maintenance. Then, we introduce an industrial use case of predictive maintenance on event logs, from ATMs. Finally, we present an experimental protocol, including a metric to estimate the cost of the maintenance system, and a new classification model specifically designed for this task.

Keywords : Time series classification, Shapelets, Predictive maintenance

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

LIFO, 6 Rue Léonard de Vinci, 45067 Orléans

WORLDLINE