MONASH University

Scalable and Accurate Time Series Classification

Ahmed Shifaz

Doctor of Philosophy

Supervisors:

Geoffrey I. Webb

François Petitjean

Charlotte Pelletier

Matthieu Herrmann

A Thesis Submitted for the Degree of Doctor of Philosophy at **Monash University** in 2021

Department of Data Science and Artificial Intelligence,
Faculty of Information Technology,
Monash University,
Melbourne, Australia

Copyright notice

Abstract

Research on time series analysis has seen enormous progress over the last few decades. One reason for that is the exponential growth of time series data and the wide range of their applications. These include the analysis of medical data, stock market data, audio data, activity and gesture recognition data, data from sensor networks used to monitor biological, chemical or industrial processes, and time series generated from satellite images and many more. The time dimension can be added to virtually any type of data. As the global market for machine learning research grows to billions of dollars, time series analysis is a critical component of this research.

My PhD research particularly focuses on Time Series Classification (TSC). It addresses two main challenges: 1) lack of scalability of prior state-of-the-art TSC classifiers, 2) a paucity of TSC techniques for multivariate time series.

To address the first challenge, I contributed to the development of two novel TSC algorithms. The first is the Proximity Forest, which is an algorithm that leverages 30 years of research to develop time series specific similarity measures by utilizing them as splitters at the nodes of purpose-built decision trees. Proximity Forest can train 1 million time series from Earth observation data in 17 hours, while the prior state of the art, FLAT-COTE, was estimated to require more than 200 years. This is a 100,000 times speedup while delivering state-of-the-art accuracy. The second algorithm, called TS-CHIEF, augmented Proximity Forest by adding splitters using dictionary and interval techniques. This integration significantly raised the accuracy of TS-CHIEF making it highly competitive with the then state-of-the-art HIVE-COTE (successor to FLAT-COTE). TS-CHIEF can be trained on 130k time series in 2 days, a data quantity that was beyond the reach of any TSC algorithm of the time with comparable accuracy. Comparatively, on the same dataset, HIVE-COTE takes 8 days to train 1,500 time series.

The second focus of my research is to develop methods for multivariate TSC. I developed multivariate versions of seven commonly used elastic similarity measures for time series data analytics. Elastic similarity measures are a class of similarity measures that can compensate for misalignments in the time axis of time series data. I adapted these seven measures by employing two existing strategies used in a multivariate version of the well-known Dynamic Time Warping (DTW), namely, Independent and Dependent DTW. These measures can be applied to various time series data mining tasks such as classification, clustering, regression, forecasting, anomaly detection, indexing and segmentation. Detailed analysis indicates that there are some multivariate TSC tasks that are inherently best tackled as either independent or dependent across dimensions.

I also researched ways to ensemble these multivariate measures for classification and explored ways of selecting subsets of dimensions to improve the speed and effectiveness of such classifiers.

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: _____

Print Name: Ahmed Shifaz

Date: 25th September 2021

Publications during enrolment

Published Journal Articles:

- **Co-author, 2019:** Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., Petitjean, F. and Webb, G.I., 2019. Proximity Forest: an effective and scalable distance-based classifier for time series. Data Mining and Knowledge Discovery, 33(3), pp.607-635.
- **Author, 2020:** Shifaz, A., Pelletier, C., Petitjean, F. and Webb, G.I., 2020. TS-CHIEF: a scalable and accurate forest algorithm for time series classification. Data Mining and Knowledge Discovery, 34(3), pp.742-775.

Journal Articles Under Review:

- **Author, 2021:** Shifaz, A., Pelletier, C., Petitjean, F. and Webb, G.I., 2021. Elastic Similarity Measures for Multivariate Time Series Classification. arXiv preprint arXiv:2102.10231.

Acknowledgements

First and foremost, I begin by thanking and praising God for innumerous blessings in my life. Since childhood, I have always wanted to be an academic, driven by curiosity and a desire to learn and understand the world. Today, as I complete one chapter of my life as a graduate student and begin the next chapter of my life as a researcher, I am filled with gratitude and excitement to reach a long-awaited goal. Completing a PhD is a long and challenging journey, one that requires the recognition of many people.

I thank my parents and family for their encouragement and sacrifices throughout my journey for education. I was born on a small island in the Maldives with extremely limited opportunities. In my early childhood, my parents migrated to the overcrowded capital, Malé City, in search of a better life and education for their children. I cannot express how thankful I am for the sacrifices they made and for continuously supporting me and encouraging me to excel at everything I do.

Next, I would like to express my gratitude to all my supervisors, Prof. Geoffrey I. Webb, Dr. François Petitjean, Dr. Charlotte Pelletier, and Dr. Matthieu Herrmann for their continuous support, guidance and friendship throughout the PhD. I am extremely happy to have been given this opportunity to work with such a brilliant, inspiring and encouraging team. They have always pushed me to achieve the best while lifting me up every time I make a mistake and felt discouraged. They taught many valuable lessons and helped me to learn and grow as a researcher.

I also would like to thank my PhD progress review panel members Dr.Mahsa Salehi, Dr. Christoph Bergmeir, Dr. Mario Boley and Prof. Wray Buntine, for their suggestions and feedback during the progress reviews. I am especially grateful to Prof. Anthony Bagnall and Assoc. Prof. Jessica Lin for accepting the invitation to be examiners of my PhD thesis. I also thank many other academics I met throughout my tertiary education, including my undergraduate honors supervisors Dr. Peter Tischer and Dr. David Albrecht who encouraged and supported me to be a machine learning researcher.

I thank the Australian Government and its people for supporting financially challenged students such as myself in their pursuit of higher education. I am extremely blessed to have studied both my undergraduate degree and PhD under fully funded Australian Government scholarships. This research was supported by an Australian Government Research Training Program (RTP) Scholarship.

In addition, I thank all my friends, both in Australia and Maldives, who supported me during the candidature, especially my housemates. And finally my loving partner, Hafeeza Ibrahim, who has been the most wonderful person in my life, continuously supporting me despite being separated in two countries and time zones for so long.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A time series is a sequence of observations of either one variable (*i.e.* univariate) or multiple variables (*i.e.* multivariate) made over time. Examples of time series data sources include audio data [10], stock market data [11, 12], human activity data (e.g. tracking movement from a device like Microsoft Kinect) [13], medical data such as Electrocardiogram (ECG), Electroencephalogram (EEG) and Electromyography (EMG) [14], data from sensors networks used to monitor biological, chemical or industrial processes, and satellite images taken over time [3].

The ubiquitous nature of time series data has made time series data analytics an important and growing area of machine learning research. Data mining from time series includes tasks such as time series classification [9, 15], clustering [16–18], regression [19], forecasting [20, 21], anomaly detection [22, 23], indexing [24], subsequence search [25] and segmentation [26].

My research focuses on Time Series Classification (TSC). However, the fundamental contributions are directly applicable to many other time series analysis tasks.

TSC is a fast developing field, especially in the last few years. My research began in 2018. At that time the recently developed FLAT-COTE[1] algorithm defined the state of the art in terms of accuracy, attaining significantly better average rank on accuracy relative to other leading algorithms of the time [9], as reflected in Figure 1.1.

However, while setting new standards in accuracy, FLAT-COTE had several limitations including that its time complexity made it impractical to apply to quantities of data beyond a few hundred series and that it only supports univariate time series. FLAT-COTE's design goal was to achieve high accuracy by ensembling a large number of TSC algorithms, and scalability was not a major consideration.

[1] Flat Collective of Transformation-Based Ensemble. It is also abbreviated as COTE.

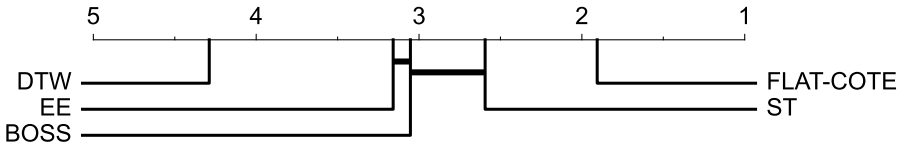


FIGURE 1.1: This diagram shows the mean rankings (on error rate) of five well known TSC algorithms over 85 benchmark datasets [1]. Algorithms to the right are more accurate. In this case, FLAT-COTE is the most accurate algorithm. Mean ranking diagrams will be explained in Section 2.6.

In addition to the limitations of the state-of-the-art algorithm FLAT-COTE, in general, the following challenges were present in the TSC field when I commenced my PhD in 2018:

- The majority of the research on TSC was evaluated on the University of California Riverside (UCR) Time Series Data Mining Archive (abbreviated hereafter as *the UCR Archive*) with relatively small datasets [1]. Until 2018, the largest dataset in the repository (*ElectricDevices*) only had 8,926 time series. 63 out of 85 of the datasets in the repository had fewer than 500 time series. This does not reflect the quantity of time series data available in many "real world" applications. For example, the Phoneme dataset [27] has 370,000 time series, and the Satellite Image Time Series (SITS) dataset [3] has 1,000,000 time series. Growth of time series data analytics show that the size of time series datasets can be in orders of several millions. Many of the TSC algorithms developed in the past two decades have been tuned by using UCR datasets, so there is concern of potential over-fitting of the algorithms' hyperparameters to the small datasets in the UCR Archive.

- One of most extensive reviews of TSC field to that date — dubbed "the great TSC bakeoff" — was published in 2017 [9]. It identified four algorithms as state of the art: Flat Collective of Transformation-based Ensembles (FLAT-COTE) [28], Elastic Ensemble (EE) [15], Shapelet Transform (ST) [29], and Bag of SFA Symbols (BOSS) [30]. As mentioned before, FLAT-COTE was identified as the most accurate algorithm, with significantly higher accuracy than EE, ST, and BOSS (see Figure 1.1). Since FLAT-COTE is an ensemble, its time complexity is bounded by the slowest of it components: EE, ST, and BOSS. ST is extremely slow with its shapelet search with a training time complexity of $O(n^2\ L^4)$, where n is the number of series in the training set and L is the length of the series. Since EE uses several similarity measures with parameters selected using leave-one-out cross-validation, its training time complexity is $O(n^2 \quad L^2)$. Furthermore, since EE and BOSS are

nearest neighbor based algorithms, they require a scan of the whole dataset during testing as well. These factors limit the scalability of FLAT-COTE to very few and short time series in any reasonable time frame.

- There were many studies conducted on how to speed up time series nearest neighbor search with elastic similarity measures such as Dynamic Time Warping (DTW) [31, 32] to large quantities of data [33]. A highly optimized 1-nearest neighbor (1-NN) using DTW with lower bounding and early abandoning can be used to classify millions of time series data relatively quickly [3, 34]. However, comparisons show that 1-NN accuracy is significantly lower than state-of-the-art TSC algorithms [9].

- Variants of dictionary algorithms (see Section 2.4.4), such as BOSS, had been developed for scalability. These include BOSS-VS and WEASEL (explained in Section 2.4.4). However they also compromise accuracy for scalability, or do not scale well in terms of other factors such as memory usage.

- The majority of TSC algorithms had been developed for univariate time series classification [6]. For example, leading classifiers prior to 2018 such as EE, ST, BOSS, FLAT-COTE do not support classification from multivariate time series. Multivariate time series data are very common in many application domains [6], so this limitation forced either multivariate data to be converted to a univariate form using some feature extraction technique or for these methods to be applied separately in each dimension. This hindered capturing potentially valuable information in the dependencies between multiple dimensions.

These challenges highlight important shortcomings in the TSC field that motivated us to explore and investigate new scalable and accurate TSC algorithms. A short summary of the contributions made in this thesis include (a detailed list is given in the conclusion of this thesis in Section 7.2):

- Contribution to the development of a novel TSC algorithm called Proximity Forest [8]. Proximity Forest uses the divide and conquer strategy of decision trees, employing a novel similarity-based splitting mechanism. It was the first scalable algorithm with accuracy competitive with state-of-the-art TSC classifiers of the time, such as EE, ST, BOSS or FLAT-COTE. For example, using an Earth observation dataset with one million series, we showed that Proximity Forest can be trained in 17 hours, while then state-of-the-art FLAT-COTE is estimated to require 200 years. My contribution, a Gini-based evaluation criteria to select between multiple candidate splits at the nodes of the trees boosted its accuracy significantly.

- Development of the novel TSC algorithm TS-CHIEF[2] [35]. TS-CHIEF extends Proximity Forest by adding dictionary-based (see Section 2.4.4) and interval-based (see Section 2.4.2) splitters. While rapid progress has been made in the last three years (recent advances are presented in Chapter 2), TS-CHIEF maintained a consistent competitive edge with newer state-of-the-art classifiers such as HIVE-COTE[3] [36, 37] (FLAT-COTE's successor, see Section 2.4.7). We empirically showed that it is scalable to more than 150k time series within a time frame that was far beyond the reach of any other competitor of the time. Since publication, the main results of TS-CHIEF have been independently replicated [38], and TS-CHIEF has been independently recognized as one of four TSC algorithms that define the state of the art [7, 39].
- Development of multivariate versions of seven commonly used univariate elastic similarity measures. This was achieved by adopting two strategies — using multiple dimensions independently and dependently [40]. Univariate versions of these seven measures have been widely used in time series analysis tasks such as classification, clustering, indexing, segmentation, anomaly detection and subsequence search. Therefore, this work is expected to be of great benefit to the time series analysis community.
- Further comparative studies using multivariate similarity measures that give insight into important distinctions between the two strategies of using dimensions independently or dependently.
- Development of multivariate versions of Elastic Ensemble and Proximity Forest and exploring the effectiveness of using random subsets of dimensions from multivariate time series.

1.1 Research Aims and Objectives

The primary aim of my research has been to develop both **scalable** and **accurate** time series classification algorithms. A secondary aim has been to extend similarity-based TSC to multivariate time series.

I identified the following Research Objectives (RO) to achieve these aims.

- *RO-1*: **To develop a scalable, similarity-based time series classification algorithm that is scalable to relatively large datasets, while being competitive in accuracy with existing similarity-based methods.** When I

[2] Time Series Combination of Heterogeneous and Integrated Embeddings Forest
[3] Hierarchical Collective of Transformation-Based Ensemble

started, Proximity Forest was being developed and its accuracy was still not competitive with FLAT-COTE. The project was handed over to me to further improve its accuracy while maintaining its scalability. In order to do this, my initial objective was to re-implement Proximity Forest with a Gini-based split selection criteria at the nodes of purpose built decision trees, giving them the ability to evaluate more than one candidate split at each node.

- *RO-2*: **To integrate dictionary-based time series classification techniques to Proximity Forest.** Dictionary-based techniques convert time series into a bag-of-word model. This splitter would be based on BOSS [41]. BOSS converts the time series into the frequency domain, and discretize them to build a database of histograms, which is then compared via a special distance measure with an ensemble of 1-nearest neighbors. The goal was to use ideas from BOSS and built our own dictionary-based splitter.

- *RO-3*: **To integrate interval-based time series classification techniques to TS-CHIEF**. The goal would be to implement a technique similar to the Random Interval Spectral Ensemble (RISE) [36], which was the leading interval-based classifier of the time. The original RISE algorithm extracts random intervals of subseries and applies a set of transform functions (e.g. autocorrelation, power spectrum) on the intervals. It then trains an ensemble of binary decision trees on each interval. The goal was to use ideas from RISE to built our own interval-based splitter.

- *RO-4*: **To extend the similarity measures used in Proximity Forest to multivariate time series**. Techniques for multivariate time series classification were largely neglected at the time. For the similarity measure DTW, there were two strategies to combine the dimensions, either independently or dependently [42]. Our goal is to adapt these strategies to other similarity measures to develop multivariate versions of commonly used similarity measures. Another goal is to study how these two strategies work for different datasets, for different similarity measures.

- *RO-5*: **To investigate ways of ensembling multivariate similarity measures and exploring ways to select which dimensions to use** The goal is to use multivariate measures to implement a Multivariate Elastic Ensemble and a Multivariate Proximity Forest in order to evaluate how ensembles of multivariate similarity measures perform. In addition, another goal is to investigate strategies to select which dimensions to use since, in most cases, using all dimensions together is unlikely to result in the optimal performance.

1.2 Thesis Overview

I have organized this thesis into seven chapters. The remaining chapters are organized as follows:

Chapter 2 presents a detailed literature review of the field of time series classification. After basic notations and introductory concepts, I present a detailed literature review of the univariate time series classification research followed by research on multivariate time series classification. In this chapter, I present different techniques used in TSC such as similarity, dictionary, and interval-based approaches, which was briefly mentioned in the research objectives. I conclude the chapter by presenting a common methodology for benchmarking, followed by a benchmark of univariate classifiers and then multivariate classifiers.

Chapter 3 focuses on a novel, highly accurate and scalable forest algorithm for time series classification called Proximity Forest. This chapter is based on the co-authored paper Lucas et al. [8] with a focus on my specific contributions. It also includes additional research that I carried out post-publication.

Chapter 4 extends the similarity-based Proximity Forest algorithm by adding dictionary and interval-based TSC techniques to the algorithm. It achieves state-of-the-art accuracy while maintaining a similar level of scalability with Proximity Forest. This chapter adds an introduction, then embeds the journal paper for TS-CHIEF as it was published [35], and finally adds a list of contributions.

Chapter 5 extends the set of similarity measures used in Proximity Forest to multivariate time series and investigates two strategies used to extend these measures to the multivariate case. This chapter also adds an introduction, then embeds a journal paper currently under review [40], and finally adds a list of contributions.

Chapter 6 ensembles the multivariate similarity measures presented in Chapter 5 to form two classifiers Multivariate Elastic Ensemble and Multivariate Proximity Forest.

Chapter 7 concludes this thesis with a summary of research, a list of overall contributions to knowledge, followed by a discussion of limitations and future work.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, I present a literature review of important methods used in time series classification (TSC). I start by presenting key definitions. I then introduce time series classification. Next, I summarize the main methods developed for univariate TSC and then multivariate TSC. Then I describe the benchmarking methodology used throughout the thesis. Finally, I conclude by presenting a benchmark of univariate classifiers, followed by a benchmark of multivariate classifiers.

2.2 Definitions

Definition 2.1. Time Series

A time series T of length L is an ordered sequence of L time-value pairs $T = h(t_1, x_1), \quad , (t_L, x_L)i$, where t_i is the timestamp at sequence index i, $i \ 2 \ f1, \quad , Lg$, and x_i is a D-dimensional point representing observations of D real-valued variables or features at timestamp t_i. Note that for the sake of this thesis we assume that all datasets have the same length. Each time point $x_i \ 2 \ \mathrm{R}^D$ is defined by $fx_i^1, \quad , x_i^d, \quad , x_i^D g$. Usually, timestamps t_i are assumed to be equidistant, and thus omitted, which results in a simpler representation where $T = hx_1, \quad , x_L i$.

A univariate (or single-dimensional) time series is a special case where a single variable is observed ($D = 1$). Therefore, x_i is a scalar, and consequently, $T = hx_1, \quad , x_L i$.

Definition 2.2. Labeled Time Series Dataset

A labeled time series dataset S consists of N labeled time series indexed by n, where n 2 f1, , Ng. Each time series T_n in S is associated with a label y_n 2 f1, , cg, where c is the number of classes.

Definition 2.3. Time Series Classification

In a Time Series Classification (TSC) task, a time series classifier is trained on a labeled time series dataset, and then used to predict labels of unlabeled time series. The classifier is a predictive mapping function that maps from the space of input variables to discrete class labels.

2.3 Time Series Classification

To understand time series classification more intuitively, let us consider two examples of applications.

As a first example, consider using classification to tell apart a normal Electrocardiogram (EGG) from an abnormal one (Figure 2.1). The depicted series come from the *ECG200* dataset obtained from a commonly used time series dataset repository called the UCR time series archive [2, 43]. This dataset contains 100 time series (*i.e.* size $n = 100$) in the training set, half of them labeled as "normal", and the other half as "Myocardial Infarction". Each time series consists of 96 measurements (*i.e.* length $L = 96$) of electrical activity (measuring one variable, hence a univariate time series) during one heart beat. A time series classification algorithm is trained using such labeled data, and the goal is to predict the label of an unlabeled time series, in this case, either a "normal" or "Myocardial Infarction" class of heart beat.

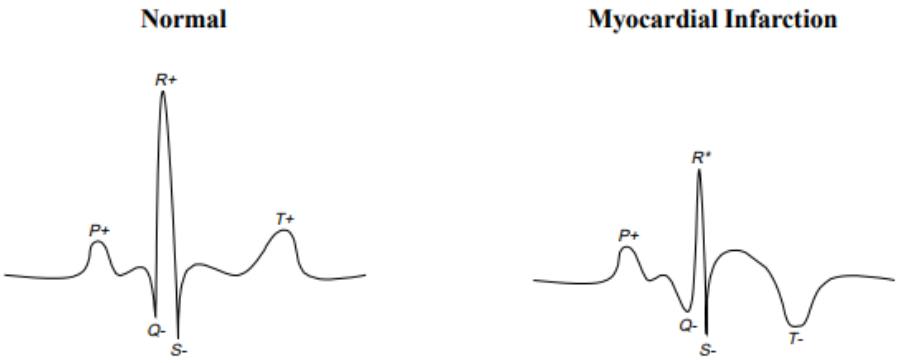


FIGURE 2.1: Example of two time series from the ECG200 dataset in the UCR archive representing a normal heart beat on the left, and an abnormal heart beat on the right. This image is taken from [2].

Another example application of time series classification is remote sensing from earth observation satellites. Consider the problem of producing land cover maps from satellite images (see Figure 2.2). Earth observation satellite Sentinel-2 takes 10 m to 60 m resolution of images of Earth's surface every 5 days in multiple-spectral bands (13 bands specifically) [44]. In such images, every pixel consists of a sequence of multivariate values representing the evolution of the reflection from the land use features of a particular geographic location. These land use features may be labeled to classes such as *wheat*, *corn*, *deciduous forest* or *urban*. Because each pixel forms a time series, research has shown that using time series classification techniques produce accurate land cover maps [45]. Since such satellite images can be gigabytes or terabytes of data with millions of time series, existing TSC algorithms were not able to handle this amount of data with high accuracy.

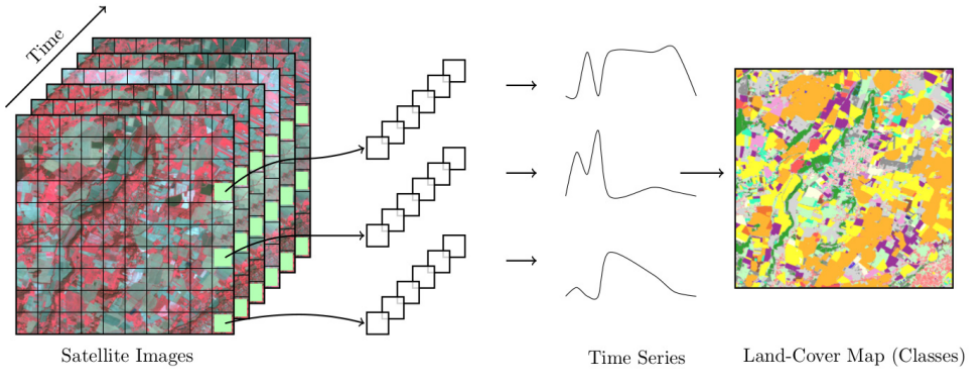


FIGURE 2.2: An illustration showing how land cover maps are produced. Time series are extracted from multiple multi-spectral satellite images. This image is taken from Tan et al. [3] with author's permission.

In time series classification we aim to develop specialized classification techniques for time series data. Research indicates that most general purpose classification algorithms - such as Random Forests, SVMs - applied on raw time series data are not as accurate as the state of the art. This is because such algorithms are not developed to specifically extract information from the sequential nature of time series data [9]. In a time series, often the previous time points are highly correlated with the current time point (this is called autocorrelation). Ideally, we want techniques that can extract information from these sequential dependencies. Due to this, a variety of time series specific techniques has been developed [9, 46], which will be summarized in this Chapter.

2.4 Univariate TSC

Most of the existing work on TSC has been conducted on univariate time series [5, 47]. These methods have been categorized in many different ways in the literature. In this

thesis, I follow a traditional method of categorizing them [9], with some modifications to accommodate recent algorithms in the field. Since some methods combine more than one technique for TSC (such as using time domain information from the original time series or using the frequency domain after a transformation), there is no definitive way to categorize them. The presented taxonomy of TSC techniques is a pragmatic choice, as it eases the explanation of the TS-CHIEF algorithm, which will be presented later in Chapter 4.

As mentioned before, general purpose classifiers applied on raw time series do not perform well [9]. Generally, two kind of approaches are used to address this issue. The first kind is to develop techniques able to accommodate the dependencies in the temporal dimension of time series. Examples of such techniques include classifiers that use time series specific similarity measures. Section 2.4.1 presents an overview of similarity-based methods in the univariate context. In Chapter 5, I will present more details of similarity measures when presenting my contributions to multivariate similarity measures for time series classification. The second kind of approach is to transform the series into a new feature space before applying general purpose classifiers. Once it has been transformed into another space, the data is no longer sequential in the original space, hence, any technique that works on tabular data can be applied. Many of the methods described below (Section 2.4.2 to 2.4.6), such as interval-based, shapelet-based, dictionary-based methods use some type of transformation either on the full series or a subseries.

2.4.1 Similarity-based Methods

A similarity measure computes a real value that quantifies the degree of similarity between two series. For the measures we consider, a smaller value means that the two series are more similar, or "closer" to each other. Similarity-based TSC methods use similarity measures for classification.

Many traditional TSC algorithms use 1-Nearest Neighbor (1-NN) with a class of similarity measures specifically developed for time series data called *elastic* similarity measures. Elastic measures are designed to compensate for local distortions, miss-alignments or warpings in time series that might exist due to stretched or shrunken subsections within the time series. For example, consider two time series of heart beats. The *systolic* pressure - maximum pressure as heart contracts - and *diastolic* pressure - minimum pressure as heart relaxes - may not be aligned in the time-axis between two time series for various reasons such as the data is collected separately, they are from different patients, etc... A non-elastic similarity measure such as the Euclidean distance cannot compensate for this

misalignment. Whereas, an elastic similarity measure can accommodate the misaligned two systolic maximums and two diastolic minimums from the two series.

The most well known elastic similarity measure used in TSC is Dynamic Time Warping (DTW) [31, 32]. Figure 2.3 shows an example of Euclidean distance and DTW aligning the points between two time series.

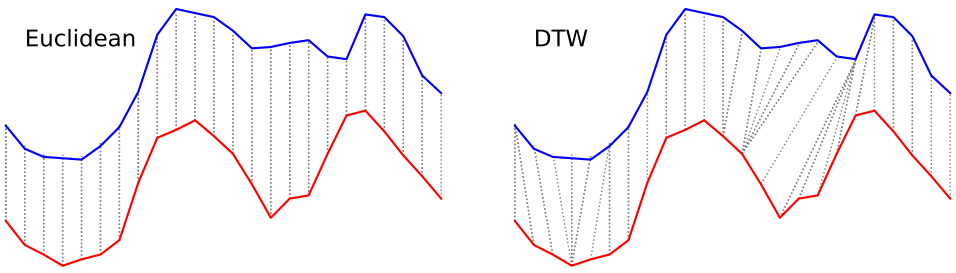


FIGURE 2.3: An example showing how a non-elastic similarity measure Euclidean distance and an elastic similarity measure DTW aligns the points between two time series. Euclidean distance use one-to-one alignments only. By contrast, DTW supports one-to-many alignments.

DTW is efficiently solved using a dynamic programming technique. Let $Q = \hbar q_1, \quad , q_L \imath$ and $C = \hbar c_1, \quad , c_L \imath$ be two univariate series of length L, and let Δ_{DTW} be an L-by-L cumulative cost matrix in which the element (i, j) is defined as the squared Euclidean distance between the two corresponding points q_i and c_j, plus the minimum cost of the previous points. Equations 2.1 to 2.3 defines the initial conditions of the cost matrix (matrix indexing starts at 0). Equation 2.4 defines the elements $\Delta_{DTW}(i, j)$ of the cumulative cost matrix.

$$\Delta_{DTW}(0,0) = 0 \tag{2.1}$$

$$\Delta_{DTW}(i,0) = + \mathit{1} \tag{2.2}$$

$$\Delta_{DTW}(0,j) = + \mathit{1} \tag{2.3}$$

$$\Delta_{DTW}(i,j) = (q_i \quad c_j)^2 + min \begin{cases} \Delta_{DTW}(i \quad 1, j \quad 1) & \textbf{if } i, j \quad 1 \\ \Delta_{DTW}(i, j \quad 1) & \textbf{if } j \quad 1 \\ \Delta_{DTW}(i \quad 1, j) & \textbf{if } i \quad 1. \end{cases} \tag{2.4}$$

Two series Q and C can be aligned using the warping paths that traverse through this cost matrix (within the constraints defined by DTW - *i.e.* all paths must be monotonically increasing, within the warping window (explained shortly), starts at $\Delta_{DTW}(0, 0)$ and ends at $\Delta_{DTW}(L, L)$). DTW represents the accumulated cost of the optimal warping path. Therefore, in the dynamic programming case, DTW between two series Q and C is the cost accumulated in the last element of the cost matrix (*i.e.* $i, j = L$) as

defined in Equation 2.5. Figure 2.4 shows how DTW alignment is solved using the cost matrix[1].

$$DTW(Q, C) = \Delta_{DTW}(L, L). \tag{2.5}$$

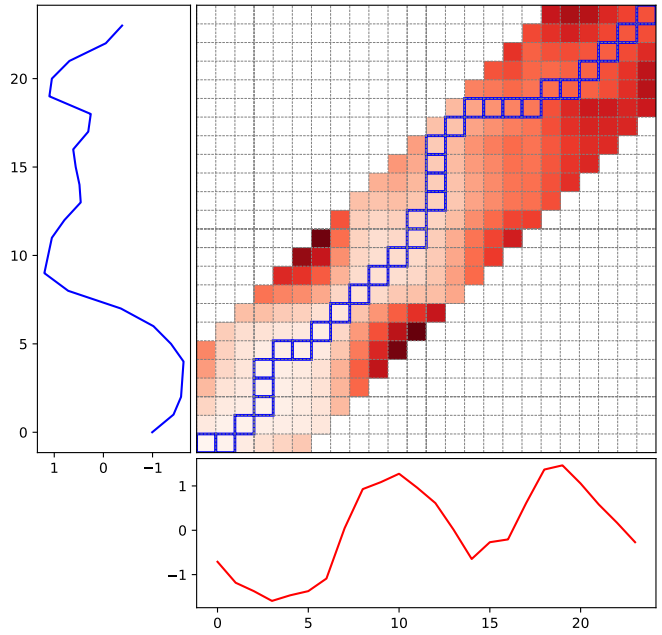


FIGURE 2.4: DTW is efficiently solved using a dynamic programming technique. The shaded region shows the maximum deviations the warping path of DTW can follow when constrained using a window. Cells are shaded based on the cumulative cost and the cells with blue outline mark the minimum cost path (*i.e.* DTW path).

DTW has a parameter called "window size", w, which helps to prevent pathological warpings by constraining the maximum allowed warping distance. For example, when $w = 0$, DTW produces a one-to-one alignment, which is equivalent to the Euclidean distance. A larger warping window allows one-to-many alignments where points from one series can match points from the other series over longer time frames. Therefore, w controls the *elasticity* of the similarity measure. For many decades, 1-NN with DTW [31] and cross validated warping window size [34] was the standard benchmark for TSC.

Parameter w also improves the computational efficiency, since in most cases, the ideal w is much less than the length of the series [49]. When w is small, DTW runs relatively fast, especially with lower bounding, and early abandoning techniques [3, 49–54]. The time complexity to calculate DTW with a warping window is $O(L\ \ w)$, instead of $O(L^2)$ for the full DTW. Where there is a need to distinguish the two variants we use DTWF for the full DTW with no window and DTW to mean DTW with windowing.

[1] Figures 2.3 and 2.4 are generated using the *dtw-python* package available at https://dynamictimewarping.github.io [48].

Commonly used similarity measures include variations of DTW such as Derivative DTW (DDTW) [55, 56], Weighted DTW (WDTW) [57], Weighted DDTW (WDDTW) [57], and measures based on edit distance such as Longest Common Subsequence (LCSS) [58], Move-Split-Merge (MSM) [59], Edit Distance with Real Penalty (ERP) [60], and Time Warp Edit distance TWE [61]. Most of these measures have additional parameters that can be tuned. Details of these measures can be found in [9, 15], and in Chapter 5.

Ensembles formed using multiple 1-NN classifiers with a diversity of similarity measures have proved to be significantly more accurate than 1-NN with any single measure [15]. Such ensembles help to reduce the variance of the model and thus help to improve the overall classification accuracy. For example, Elastic Ensemble (EE) combines 11 1-NN algorithms, each using one of the 11 elastic measures [15]. For each measure, the parameters are optimized with respect to accuracy using leave-one-out cross-validation [9, 15]. Although EE is a relatively accurate classifier [9], it is slow to train due to the high computational cost of the leave-one-out cross-validation used to tune its parameters – $O(n^2 \ L^2)$. Furthermore, since EE is an ensemble of 1-NN models, the classification time for each time series is also high – $O(n \ L^2)$. EE was the overall most accurate similarity-based classifier on the UCR benchmark until our contribution Proximity Forest [9]. It was used as a component of FLAT-COTE and early versions of HIVE-COTE as well.

Our Proximity Forest [8] (PF) is currently the most accurate and the fastest similarity-based time series classifier available. PF is a tree-based ensemble that uses the set of similarity measures used in EE. By contrast to the 1-NN method used in EE, PF takes advantage of the divide-and-conquer strategies of trees to train in $O(n \ log(n))$. In addition, unlike EE which uses slow leave-one-out cross-validation to select the parameters, PF uses a randomization strategy, which helps to speed up the algorithm considerably. For a dataset that EE is estimated to require 200 years to train, PF can train in just 17 hours as we will see in Chapter 3.

2.4.2 Interval-based Methods

These algorithms select a set of intervals from the whole series and apply transformations to these intervals to generate a new feature vector. For a series of length L, there are $L(L \ \ 1)/2$ possible intervals. Therefore, many of these methods generate a large number of intervals (often random), hoping that the subset of generated intervals will produce an accurate representation of interval-based features available (see Figure 2.5). Transformed feature vectors are finally used to train a classifier such as decision trees. Many methods rely on ensembling and use randomization to increase the diversity among individual classifiers, in order to reduce the overall error.

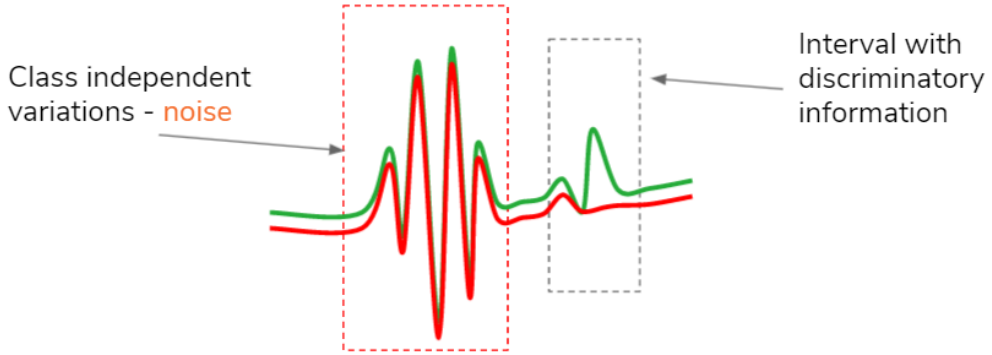


FIGURE 2.5: An example showing interval-based features in time series. Intervals are phase-dependent features, which means that their location in the series is useful for discrimination.

For instance, Time Series Forest (TSF) samples random intervals and applies three time domain transformations – mean, standard deviation and slope – at the nodes of the trees and trains an ensemble of a customized decision tree called Time Series Tree [62]. Other notable interval-based algorithms are Time Series Bag of Features (TSBF) [63], Learned Pattern Similarity (LPS) [64], and the HIVE-COTE's interval-based component called Random Interval Spectral Ensemble (RISE) [37]. Note that RISE selects a single interval for each base whereas TSF and TSBF select multiple intervals for each base classifier.

RISE computes four different transformations for each random interval selected: Autocorrelation Function (ACF), Partial Autocorrelation Function (PACF), and Autoregressive model (AR) which extracts features in time domain, and Power Spectrum (PS) which extracts features in the frequency domain [37, 64]. Coefficients of these functions are used to form a new transformed feature vector. After these transformations have been computed for each interval, a Random Tree (similar to trees used in the Random Forest algorithm [65]) is trained on each of the transformed intervals. The average training complexity of RISE is $O(k \quad n \quad log(n) \quad L^2)$ [37], and the test complexity is $O(k \quad log(n) \quad L^2)$, where k is the number of trees.

More recently introduced interval-based classifiers include Supervised Time Series Forest (STSF) [66], Randomized STSF (r-STSF) [67], Canonical Interval Forest (CIF) [68], and HIVE-COTEv2's new interval-based component Diverse Representation CIF (DrCIF) [7]. r-STSF (an improved version of STSF) explores a supervised method of sampling the intervals, and then uses simple summary statistic features, spectral features and features from first derivative of the series over those intervals. It also compares Random Trees with Extremely Randomized Trees [69] to design a more stochastic but more accurate ensemble. DrCIF (an improved version of CIF) uses 29 features: mean, standard deviation, slope, median, inter-quartile range, min, max and 22 features from

the "catch-22" [70] features for time series analysis. The "catch-22" features were derived from a clustering and filtering of the 7658 *hctsa* features used in the time series analysis *hctsa* toolbox [71]. At the time of its publication, DrCIF was benchmarked as the overall most accurate interval-based algorithm [7][Figure 4]. However, there has been no comparison of DrCIF relative to r-STSF, which was released after DrCIF.

2.4.3 Shapelet-based Methods

Rather than extracting intervals, where the location of sub-sequences are important, shapelet-based algorithms seek to identify sub-sequences that allow discrimination between classes irrespective of where they occur in a sequence [72]. Ideally, a good shapelet candidate should be a sub-sequence that is common to time series from the same class, and not found in time series from other classes (see Figure 2.6). The similarity measure derived from a shapelet is the minimum Euclidean distance between it and any sub-sequence of the same length from the target series.

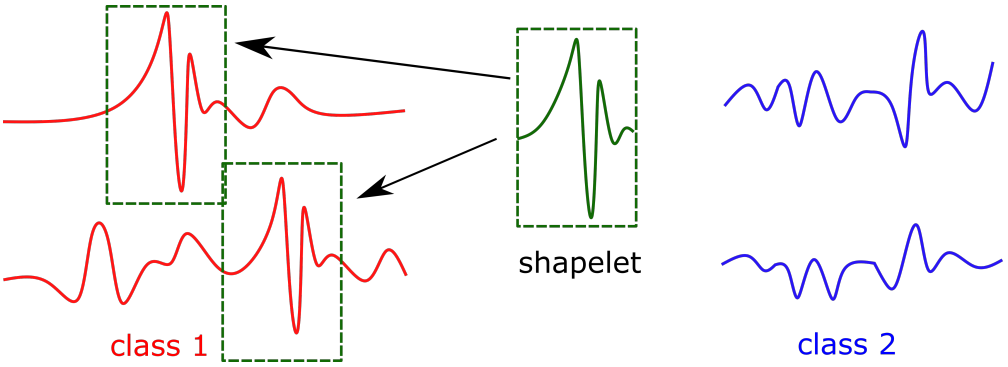


FIGURE 2.6: An example showing shapelet-based features in time series. Shapelets are phase-independent features, which means that location of the subseries is not important when selecting a shapelet. Shapelets are selected based on their ability discriminate between classes.

The original version of the shapelet algorithm [72, 73] enumerates all possible subsequences among the training set to find the "best" possible shapelets. It uses the Information Gain criterion to assess how well a given shapelet candidate can split the data. The selected shapelet candidate and a distance threshold is used as a decision criterion at the node of a binary decision tree. The search for the "best" shapelet is then recursively repeated until obtaining pure leaves. Despite some optimizations proposed in the paper, it is still a very slow algorithm with training complexity of $O(n^2 \quad L^4)$.

Much of the research about shapelets has focused on ways of speeding up the shapelet discovery phase. Instead of enumerating all possible shapelet candidates, researchers have tried to come up with ways of quickly identifying possible "good" shapelets. These include Fast Shapelets (FS) [74] and Learned Shapelets (LS) [75]. Fast Shapelet proposed

to use an approximation technique called Symbolic Aggregate Approximation (SAX) [76] to discretize the time series before the shapelet discovery process in order to speed up by reducing the number of shapelet candidates. LS attempted to "learn" the shapelets rather than enumerate all possible candidates. FS is faster than LS, but it is less accurate [9].

One algorithm that speeds up the shapelet-based techniques is Generalized Random Shapelet Forest (GRSF) [77]. GRSF selects a set of random shapelets at each node of a decision tree and performs the shapelet transformation at the node level of the decision tree. GRSF is fast because it is tree-based and uses random selection of shapelets instead of enumerating all shapelets. However, results available in [77] was carried out on a subset of the 85 UCR datasets making it difficult to compare against other classifiers.

Another notable shapelet algorithm is Shapelet Transform (ST) [29]. In ST, the 'best' k shapelets are first extracted based on their ability to separate classes using a quality measure such as Information Gain, and then the distance of each of the "best" k shapelets to each of the samples in the training set is computed [29, 78, 79]. The distance from k shapelets to each time series forms a matrix of distances which defines a new transformation of the dataset. This transformed dataset is finally used to train an ensemble of eight traditional classification algorithms including 1-Nearest Neighbor with Euclidean distance and DTW, C45 Decision Trees, BayesNet, NaiveBayes, SVM, Rotation Forest and Random Forest. Although accurate, ST also has a high training-time complexity of $O(n^2 \quad L^4)$ [29, 37].

More recent advances to shapelet-based algorithms were introduced in HIVE-COTEv1 which uses a time contracted version of ST called Shapelet Transform Classifier (STC) [39]. Research indicates that exhaustively searching all shapelets can lead to overfitting [80], and does not necessarily perform better than random search. Contracted version takes a time limit as an input to the algorithm and optimizes a random shapelet search within the time frame. It also replaces the heterogeneous ensemble of classifiers used in ST with a Rotation Forest [81]. Currently, HIVE-COTEv2 also uses this new version of the shapelet-transform to speed up its shapelet-based TSC component.

2.4.4 Dictionary-based Methods

Dictionary-based algorithms transform time series data into bag of words [41, 82, 83]. Dictionary based algorithms are good at handling noisy data and finding discriminatory information in data with recurring patterns [41]. Usually, an approximation method is first applied to reduce the length of the series [76, 84, 85], and then a quantization method is used to discretize the values, and thus to form words [41, 83]. Each time series

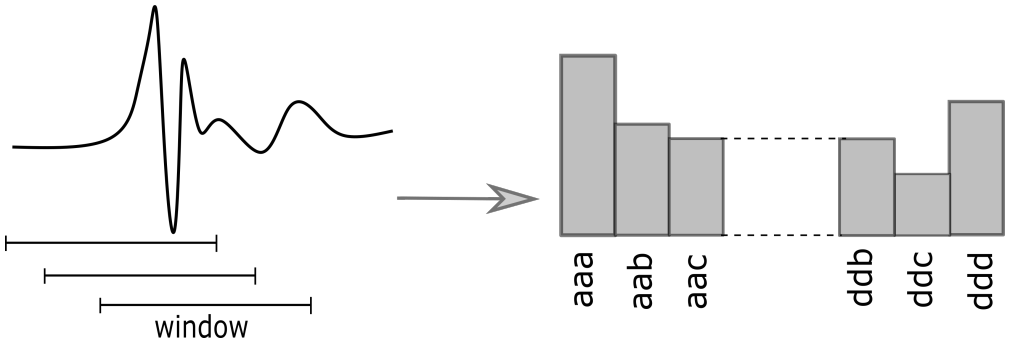


FIGURE 2.7: Dictionary-based methods convert a set of windows over the series (either sliding windows or disjoint windows) to a set of symbols. These symbols are then counted to form a histogram of word frequencies to represent the original time series.

is then represented by a histogram that counts the word frequencies (see Figure 2.7). 1-NN with a similarity measure, that compares the similarity between histograms, can then be used to train a classification model. Notable dictionary based algorithms are Bag of Patterns (BoP) [86], Symbolic Aggregate Approximation-Vector Space Model (SAX-VSM) [82], Bag-of-SFA-Symbols (BOSS) [41], BOSS in Vector Space (BOSS-VS) [87] and Word eXtrAction for time SEries cLassification (WEASEL) [88].

To compute an approximation of a series, BOP and SAX-VSM use a method called Symbolic Aggregate Approximation (SAX) [76]. SAX uses Piecewise Aggregate Approximation (PAA) [84] which concatenates the means of consecutive segments of the series and uses quantiles of the normal distribution as breakpoints to discretize or quantize the series to form a word representation. By contrast, BOSS, BOSS-VS, and WEASEL use a method called Symbolic Fourier Approximation (SFA) [85] to compute the approximated series. SFA applies Discrete Fourier Transformation (DFT) on the series and uses the coefficients of DFT to form a short approximation, representing the frequencies in the series. This approximation is then discretized using a data-adaptive quantization method called Multiple Coefficient Binning (MCB) [41, 85].

The most commonly used algorithm in this category is Bag-of-SFA-Symbols (BOSS), which is an ensemble of dictionary-based 1-NN models [41]. BOSS is a component of HIVE-COTE and our algorithm TS-CHIEF also has a component inspired by BOSS. BOSS has a training time complexity of $O(n^2 \ L^2)$ and a testing time complexity of $O(n \ L)$ [41]. A variant of BOSS called BOSS-VS [87] has a much faster train and test time while being less accurate. The more recent variant WEASEL [88] is more accurate but has a slower training time than BOSS and BOSS-VS, in addition to high space complexity [8, 88, 89].

More recent advances in dictionary-based splitters include time contractable BOSS

(cBOSS) [89], BOSS with Spatial Pyramids (S-BOSS) [90], Temporal Dictionary Ensemble (TDE) [91]. cBOSS introduces time contracting to BOSS and combines various research progress made on dictionary-based classifiers since 2015. It is much faster and improves the accuracy. S-BOSS adds a form of location information to the extracted words by extracting words at multiple levels. At the first level all words are extracted from the whole series, at the second level words are extracted separately from each of the halves, and at the third level words are extracted from 1/4th of the series, and so on. S-BOSS is ranked more accurate than cBOSS and similar to WEASEL. But similar to WEASEL, S-BOSS also has high memory requirements. TDE combines features from both cBOSS and S-BOSS, and adds additional improvements to make it the most accurate dictionary-based classifier available at the moment [7][Figure 3]. It is also a component of the HIVE-COTEv2.

2.4.5 Deep Learning-based Methods

Deep learning is interesting for time series both because of the structuring dimension offered by time (deep learning has been particularly good for images and videos) and for its linear scalability with training size. Most related research has focused on developing specific architectures based mainly on Convolutional Neural Networks (CNNs) [46, 92], coupled with data augmentation, which is required to make it possible for them to reach high accuracy on the relatively small training set sizes present in the UCR archive [46, 93]. The most comprehensive empirical study of deep learning architectures used for TSC was conducted by Fawaz et al. in 2019 [46]. They reviewed nine architectures and benchmarked them on the UCR Archive. While the approaches are computationally efficient, the two leading algorithms in the review, Fully Connected Network (FCN) [92] and Residual Neural Network (ResNet) [92], was found to be less accurate than FLAT-COTE and HIVE-COTE (alpha) [46].

Inspired by the success of Convolution Neural Networks (CNN) for two dimensional image recognition, one dimensional CNNs have been studied for time series classification in Temporal CNN (tempCNN) [4] (see Figure 2.8). They have been successfully applied in the area of remote sensing for TSC.

InceptionTime is a deep learning algorithm that was developed later by Fawaz et al. [94], which incorporates features from Inception modules [95] and ResNets [96]. InceptionTime is an ensemble of deep learning models where each model is initialized with a random set of weights. This is done to reduce variance by averaging the errors. Each model consists of three blocks of Inception modules which are connected by residual connections, followed by global average pooling and softmax layers. InceptionTime is

currently the most accurate deep learning architecture available [94]. Since InceptionTime runs on GPUs, the available runtime information is not directly comparable to benchmarks run on CPUs, however, as according to recent benchmarks it is the second fastest state-of-the-art TSC classifier after ROCKET [7][Table 4].

2.4.6 Random Convolution-based Methods

One of the most important recent advances in TSC was made by a method called Random Convolution Kernel Transform (ROCKET) [97]. Rocket transforms time series using a large number of random convolutional kernels. These kernels are parameterized with random lengths, weights, biases, dilations, and paddings. By contrast to regular CNN architectures where convolutional kernels are learned, random kernels provides a way to transform a series into a high dimensional feature vector very quickly. A large number of random kernels are convolved with the input series, resulting in a transformed output series for each filter. Two summary statistics are extracted from each of these output series — the *maximum value* and the *proportion of positive values* (PPV) (the number of values that are positive as a proportion of all values). These extracted features are then used to train a linear classifier. For small datasets, it uses a ridge classifier with built in cross-validation for the regularization. And for large datasets, it uses a logistic regression trained using stochastic gradient descent.

ROCKET provides state-of-the-art accuracy using a fraction of time required by other scalable TSC classifiers. A recent benchmark shows that ROCKET can train 112 datasets from the UCR Archive 2018 version in under 3 hours on a single thread [7][Table 4]. By Comparison, HIVE-COTEv2 took 341 hours, HIVE-COTEv1 took 428 hours, InceptionTime (run on GPUs) took 67 hours and TS-CHIEF took 1000+ hours (see Section 2.4.7 for details on HIVE-COTE and TS-CHIEF)[2]. ROCKET is also scalable for large datasets, with training complexity linear in both time series length and the

[2] TS-CHIEF was published earlier than these algorithms hence is not as optimized as HIVE-COTE 1.0 and HIVE-COTE 2.0. Most of the total time taken to train TS-CHIEF is taken by only a few datasets.

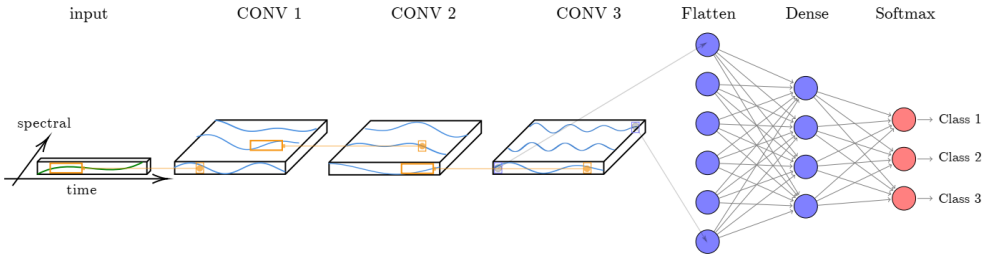


FIGURE 2.8: An example of a deep learning architecture (TempCNN) used for TSC. This image is taken from [4]

number of training examples. ROCKET can learn from 1 million time series of SITS dataset [3] in 1 hr 3 min, to a similar accuracy as Proximity Forest, which requires more than 16 hrs to train on the same quantity of data.

Two recently introduced improvements to this algorithm are MiniROCKET [98] and MultiROCKET [99]. MiniROCKET maintains similar accuracy to ROCKET while making the algorithm 75 times faster for large datasets. This optimization exploits various properties of the kernels and summary feature PPV. The algorithm is made almost deterministic by using a small set of fixed kernels and proving that it can produce essentially similar performance as using a large number of random kernels. According to the authors, using MiniROCKET, 109 of the UCR Archive 2018 datasets can be trained in less than 10 minutes [98]. MultiROCKET improves the accuracy even more with a small increase in the computation overhead. MultiROCKET uses 26 summary features including the features from "catch-22" [70] as well.

2.4.7 Combination Methods

This is a class of algorithms that ensemble other TSC algorithms to form "meta-ensembles". They have been mostly used as a target for benchmarking rather than optimizing for speed.

Two most important algorithms in this category are Flat Collective of Transformation-Based Ensembles (FLAT-COTE, also known as COTE) [28] and Hierarchical Vote COTE (HIVE-COTE, also known as HIVE-COTE alpha) [36, 37]. FLAT-COTE is a meta-ensemble of 35 different classifiers that use different time series classification methods such as similarity-based, shapelet-based, and interval-based techniques. In particular, it includes other ensembles such as EE and ST. The label of a query time series is determined by applying weighted majority voting, where the weighting of each constituent depends on the training leave-one-out cross-validation (LOO CV) accuracy. HIVE-COTE works similarly, and it includes EE, ST, TSF, BOSS and RISE. The other main change was grouping the constituent classifiers into modules that use different data representations and then applying the weightings at module level to provide more balance between data representations. These modifications result in a major gain in accuracy, and it is currently considered as one of the state-of-the-art algorithm in TSC for accuracy. Both variants of COTE have high training complexity, lower bounded by the slow cross-validation used by EE – $O(n^2 \ L^2)$ – and exhaustive shapelet enumeration used by ST – $O(n^2 \ L^4)$.

The two most recent updates are HIVE-COTEv1 (HC1) [39] and HIVE-COTEv2 (HC2) [7]. HC1 dropped EE from the ensemble due to its slow speed. It then replaced ST and BOSS with contractable versions of the algorithms. HC1 gives similar accuracy to HIVE-COTE while being magnitudes faster. HC2 improves the accuracy of HIVE-COTE significantly more than any other algorithm making it currently the most accurate TSC classifier. While HC2 uses most of the features from HC1, it replaces contract BOSS with the new dictionary-based classifier TDE and integrates an ensemble of ROCKETS called Arsenal.

One of my main contribution TS-CHIEF also combines similarity, interval and dictionary-based techniques. But unlike the COTE algorithms, TS-CHIEF combines them in a uniquely different way at the node level of purpose built decision trees. It achieves state-of-the-art accuracy with quasi-linear training time with respect to the number of series. I present its details in Chapter 4.

2.4.8 Univariate Datasets

The majority of the work in TSC field has been conducted on a collection of datasets called the University of California Riverside (UCR) Time Series Data Mining Archive (shortly called the UCR Archive). The initial version of this repository was published in 2002 [100]. Two important versions of the archive are: the 2015 version with 85 fixed-length datasets [1] and the most recent 2018 version with 128 datasets [43] (112 of which are fixed-length). Since the publications contained in this thesis was evaluated on the 85 datasets from the 2015 version, unless explicitly stated, in this thesis I use UCR Archive to refer to the 2015 version.

Using a common archive for benchmarking solves many problems that existed in the TSC community. An early survey conducted in 2003 [47] showed that the majority of the papers at the time used artificial data created by authors to evaluate their own algorithms. This creates biased algorithms that may not generalize well to real-world data. This survey also highlighted the difficulties in reproducing the results of some experiments due to inability to obtain the data used in published papers. Introduction of UCR Archive to the TSC community helped to minimize some of these problems. More than a thousand papers have used these datasets to evaluate their algorithms using a standard benchmark set [43].

The UCR Archive contains datasets from a very diverse range of real-world applications. These include time series data obtained from images, motion sensors, mass spectrometry, synthetic examples, electrocardiograms. For both the 2015 and 2018 versions, a standard train/test split - obtained with stratified sampling while making

sure that all classes are represented in both train and test split - is provided in the https://www.timeseriesclassification.com. While 2018 version has few datasets with missing data and variable length, all datasets in the 2015 version has fixed-length and no missing data. All datasets in the 2015 archive are z-normalized with zero mean and unit standard deviation, while 2018 version have some unnormalized datasets as well [43].

Even though the archive has many benefits, it is not without criticism [43]. Some of them include: 1) z-normalizing will have a negative impact on algorithms that use the shape of the time series as discriminatory features, 2) most of the datasets are small to medium size, with most of them being less than few hundred time series, 3) some datasets have labeling issues (for example *ElectricDevice* dataset contains multiple time series with exactly the same numeric values that are labeled as different classes), 4) datasets are obtained from domains that the contributors of the archive have previously worked with, creating a bias in representation of real-world data.

2.5 Multivariate TSC

One of the biggest challenges in TSC is that most of the research focuses on univariate time series. However, it is common to encounter datasets with multivariate time series [6, 101]. Some of the reasons for a lack of work in this area could be the assumption that extending the existing methods to multivariate case is a trivial problem [42] and lack of a standard set of benchmark datasets for multivariate problems. However, recently there has been much more emphasis on adapting algorithms for multivariate datasets. Two important recent advances include the release of multivariate UEA benchmark datasets in 2018 [5] and the recent review of multivariate TSC methods by Ruiz et al. in 2020 [6]. Using the same taxonomy I used before, next I will present different techniques used for multivariate TSC.

2.5.1 Similarity-based methods

The most commonly used similarity measure for time series, DTW, was extended to the multivariate case in [42]. They identified two key strategies for such extension. The first strategy uses dimensions independently (*independent DTW*) and applies the univariate DTW to each dimension and then sums the resulting distances. Therefore, warping across time points is allowed in each dimension separately. The second strategy

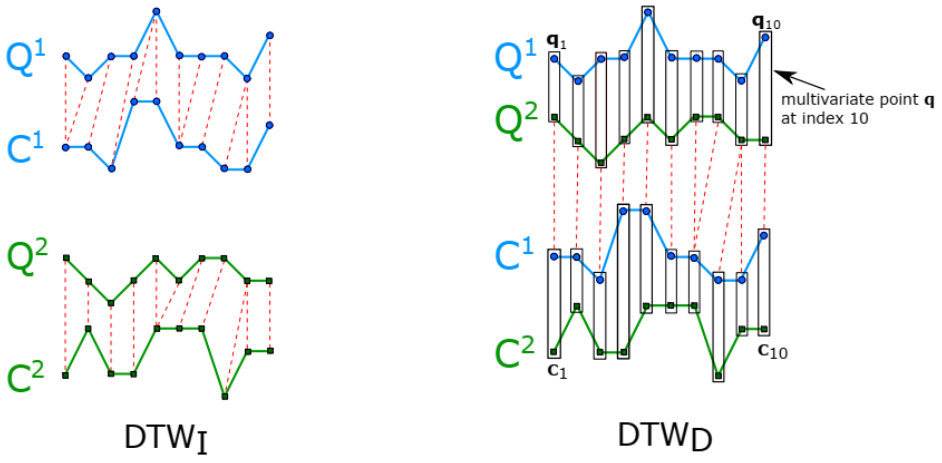


FIGURE 2.9: Independent DTW (DTW_I, on the left) and dependent DTW (DTW_D, on the right). Dimension 1 in series Q and C is shown in blue color, and the dimension 2 is shown in green color.

uses dimensions dependently (*dependent DTW*) and treats each time step as a multidimensional point. DTW is then applied on the Euclidean distances between these multidimensional points. In this case, DTW can warp across multiple dimensions together, capturing possible dependencies across dimensions. Figure 2.9 shows an illustration of independent DTW (DTW_I) and dependent DTW (DTW_D).

Shokoohi et al. showed that for DTW these two strategies are theoretically as well as empirically different [42]. One of the main contributions of this thesis is that we extend this idea from DTW to seven other measures and show that this hypothesis holds across several other common measures as well. We present formal definitions of these two strategies and other multivariate measures in Chapter 5.

2.5.2 Interval-based methods

Interval-based methods that extract location dependent subseries include RISE [36] and TSF [62]. Multivariate versions of RISE and TSF introduced in HIVE-COTEv1 supports multivariate TSC by ensembling over models built on each dimension independently [39]. Accuracy of these two methods are below DTW_D [6][Figure 7c]. Recently introduced classifier CIF [68] has shown promising results for multivariate classification. CIF selects a random dimension at each node of the trees to sample the random intervals. Even more recently, CIF has been further improved as DrCIF to be included in HIVE-COTEv2 [7]. Currently it is one of the top multivariate classifiers [6][Figure 7a], performing better than more complex HIVE-COTEv1.

2.5.3 Shapelet-based methods

Shapelet-based methods include Learned Pattern Similarity (LPS) [64], Ultra-fast Shapelets (UFS) [102], gRSF [77], and STC [39]. Since gRSF is a tree-based algorithm, it selects a random dimension at each node from which random shapelets are extracted. STC ensembles over models built on each dimension independently [39]. According to a recent review, STC is the current most accurate multivariate method that uses shapelets (and it is ranked below ResNet) [6][Figure 7a].

2.5.4 Dictionary-based methods

Dictionary-based methods include WEASEL with a Multivariate Unsupervised Symbols and dErivatives (MUSE) (a.k.a WEASEL+MUSE, or simply MUSE) [88], cBOSS [89], MrSEQL [103] and TDE [89]. WEASEL+MUSE is the multivariate version of WEASEL, which builds its dictionary of word frequencies using all series and dimensions together. To a degree, it is able to capture the dependencies between dimensions by adding dimension identifiers to the words, and reduce the effects of less relevant dimensions by using feature selection with a Chi-squared test to remove the least relevant words. cBOSS simply forms an ensemble over models built on each dimension independently [39]. Many dictionary-based classifiers are memory intensive due the storage of large amount of transformed data. For instance, WEASEL+MUSE use over 500 GB of memory required on some datasets [6]. Recent classifier TDE made improvements to its memory usage. While I was not able to obtain its memory usage on the UEA multivariate archive, it was reported to use a maximum of 6.5 GB to run 112 datasets of UCR 2018 univariate datasets [7][Table 4]. Since extracting words from all dimensions are memory intensive, using leave-one-out cross validation on bags of words created from disjoint windows per dimension, TDE samples a subset of dimensions to use. Dimensions with accuracy estimate less than 85% of the highest accuracy are not retained in the classifier (with a maximum limit of 20 dimensions).

2.5.5 Deep Learning-based methods

Nine deep learning methods for TSC that supports multivariate time series were evaluated in the review by Fawaz et al. in 2018 [46]. This review used 12 datasets published by Baydogan in [101]. The statistical test did not find any significant difference between the nine classifiers which could be due the small number of datasets. In the study, a Fully Connected Network was ranked highest, just above ResNet [46][Figure 9].

It is also possible to extend InceptionTime to the multivariate case [94]. According to more recent benchmarking results for multivariate classifiers, InceptionTime outperforms ResNet, as reported in Ruiz et al. [6][Figure 12a].

2.5.6 Random Convolution-based methods

Multivariate versions of ROCKET and MiniROCKET is also available in the github repository of *sktime* package for time series analysis[3]. In the multivariate versions, each kernel is assigned a random subset of dimensions (up to the length of the kernel, so one kernel of length 9 might be assigned between 1, 2, ..., 9 dimensions). That kernel is therefore a 2-d convolutional kernel, with its length (e.g., 9) in the time dimension, and its 'depth' equal to the size of the subset of dimensions. For ROCKET, all the weights are random. For MiniROCKET, each dimension gets the same set of weights (whatever the weights are for that kernel). The max value and PPV is then calculated across all dimensions for each kernel, producing a 20,000 attribute instance. Note that unlike DTW_D which uses all dimensions, multivariate ROCKET is able to capture the dependencies between random subsets of dimensions.

2.5.7 Combination Methods

HIVE-COTEv1 and its subsequent extensions all support multivariate TSC. Essentially they combines multivariate versions of STC, TSF, cBOSS and RISE (or TDE and DrCIF for HC2). These classifiers build a separate model for each dimension and ensembles them.

2.5.8 Multivariate Datasets

One of the reasons for slow progress of multivariate TSC research is due to small size and number of benchmark datasets available [5]. Mustafa Baydogan maintained a useful archive [4] for multivariate datasets. The review of deep learning methods by Fawaz et al. [46] also used this archive to compare multivariate deep learning methods. However, it had many limitations, as described by Bagnall et al. "The datasets are all very small, are not independent and are not representative of many important multivariate time series classification (MTSC) domains" [5].

[3] https://github.com/alan-turing-institute/sktime/blob/master/sktime/transformers/series_as_features/rocket.py

[4] http://www.mustafabaydogan.com/multivariate-time-series-discretization-for-classification.html

In 2018, in collaboration of many TSC researchers, a benchmark repository for multivariate time series was released by University of East Anglia (UEA) TSC research team [5]. This archive has 30 multivariate datasets (26 of them are fixed length), with standard train/test splits provided on their website: `timeseriesclassification.com`. In Chapter 5 where multivariate similarity measures are presented, I provide additional information about these datasets.

2.6 Benchmarking

Both the univariate UCR Archive and the multivariate UEA Archive are important drivers of progress in the field of TSC (see Section 2.4.8 and 2.5.8). It has become the norm to benchmark algorithms against one another by their performance on these archives [6, 9, 46, 47].

For benchmarking, I use a common technique used by many TSC researchers to compare multiple algorithms over the multiple datasets [104, 105]. First a null hypothesis is assumed which states that there is no significant difference in the mean accuracy ranks of the multiple algorithms (at a statistical significance level $\alpha = 0.05$). Then, a Friedman test is performed to reject or accept the null hypothesis. In cases where the null-hypothesis is rejected, a Wilcoxon signed rank test (as advised in [105]) is used to compare the pair-wise difference in ranks between algorithms, and then the Holm–Bonferroni method is used to adjust for family-wise errors [104–107].

Results of these tests can be visualized using mean ranking diagrams as in Figure 1.1 (on page 2). Traditionally, they are drawn based on mean error ranks, hence the algorithms to the left most with lowest ranks are the most accurate. Algorithms that do not differ to a level of statistical significance are grouped using thick horizontal bars (see the horizontal bars connecting EE and BOSS, and BOSS and ST in Figure 1.1). When many classifiers are compared, it may become difficult to read these groupings. In such cases, we report the pairs that are statistically different in the text based on the adjusted p-value tables. In order to perform these statistical tests and generate the diagrams, I used the source code freely available at `https://github.com/hfawaz/cd-diagram` [46].

Note that, these diagrams are based on the widely used Critical Difference (CD) diagrams [104]. I use the term *mean ranking diagrams* because the diagrams used in this thesis differ from the diagrams used in the original CD paper [104]. The difference is that in CD diagrams, a constant "critical difference" value is used to test for statistical difference between groups of algorithms. This is usually depicted using a reference horizontal bar on the diagram. However, since we use the Holm–Bonferroni method to

adjust for the family-wise errors, a constant "critical difference" value is no longer used. A number of recent papers also follow this notation [97, 98].

2.6.1 Comparing Univariate Classifiers

In addition to the UCR datasets, detailed results of several benchmarks of TSC algorithms can also be obtained from https://www.timeseriesclassification.com. The results available in this website are trained on 30 re-samples 109 of the new UCR 2018 datasets. The implementations are based on two packages for time series analysis: Java based *tsml* package[5] and Python based *sktime* package[6].

Figure 2.10 displays the mean ranks (in this case, on error) between a set of TSC classifiers (selected as best of class exemplars of each category). We observe that while the most recently released algorithm HIVE-COTEv2 is significantly more accurate than all other algorithms, other leading algorithms such as versions of ROCKET, HIVE-COTEv1, InceptionTime and TS-CHIEF are highly competitive and not statistically different to each other.

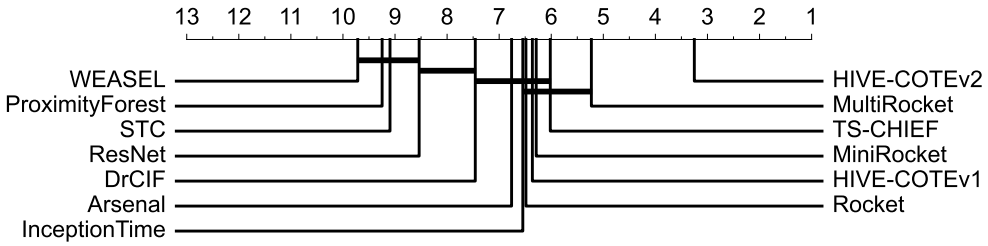


FIGURE 2.10: Mean error ranks of common TSC classifiers on 30 resamples of 109 datasets of the UCR 2008 archive. Data for this diagram is obtained from www.timeseriesclassification.com.

2.6.2 Comparing Multivariate Classifiers

Figure 2.11 displays the mean ranks (on error rate) of the leading multivariate TSC methods as identified by Ruiz et al. [6] and Middlehurst et al. [7]. In addition, DTW_I and DTW_D are added for historical reasons. The most accurate algorithm is HIVE-COTEv2 followed by ROCKET. Possibly due to the small number of datasets, most algorithms are difficult to distinguish statistically (grouped by the two thick horizontal lines).

[5] https://github.com/uea-machine-learning/tsml
[6] https://github.com/alan-turing-institute/sktime

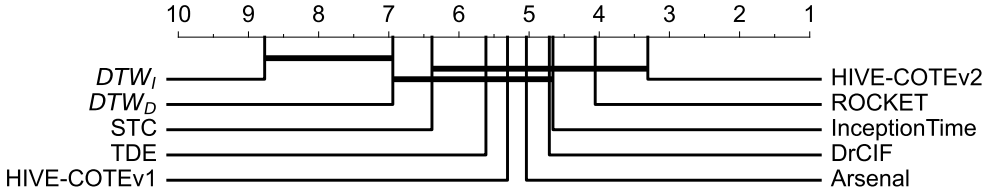


FIGURE 2.11: Mean error ranks of common multivariate TSC classifiers on 26 datasets of the UEA multivariate time series archive [5]. Data for this diagram is obtained from Ruiz et al. [6] and Middlehurst et al. [7]

.

Chapter 3

Proximity Forest

3.1 Introduction

In this Chapter I present a novel tree-based TSC algorithm called Proximity Forest. Proximity Forest is designed to address the scalability issue of TSC algorithms while maintaining high accuracy. It is the first TSC algorithm that was able to scale to millions of time series while maintaining accuracy comparable to the state of the art.

This chapter is based on Lucas et al. [8] that I co-authored during my PhD. When I started my PhD in 2018, Proximity Forest was being developed by a team of researchers at Monash University. Their goal was to develop an alternative algorithm to Elastic Ensemble that can be scaled to large quantities of data while also making the most out of decades of research on similarity-measures. On the commencement of my PhD, I took over the project as its main developer and re-implemented the system[1]. I also further developed the algorithm. Since I did not develop the core of Proximity Forest, I have written this chapter to highlight my contributions to the research, and added research that was done since the publication of the Proximity Forest paper [8]. As my research was included in the paper, some of this chapter uses the paper's content directly. I explicitly specify my contributions to this work in Section 3.5.1. I have also add a copy of the original publication to Appendix A of this thesis.

- **This chapter is based on the published paper:** Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., Petitjean, F. and Webb, G.I., 2019. Proximity Forest: an effective and scalable distance-based classifier for time series. Data Mining and Knowledge Discovery, 33(3), pp.607-635.

[1] It is now maintained in the TS-CHIEF Github repository at https://github.com/dotnet54/TS-CHIEF

3.2 Background

One of the most important challenges in TSC is scaling existing algorithms to large quantities of data. Most of the algorithms published prior to 2018 had significant short comings with respect to scalability and high accuracy. An extensive review conducted in 2017 [9] revealed that none of the state-of-the-art TSC algorithms are scalable beyond a few hundred time series.

In Chapter 1, I highlighted several challenges in TSC field, especially prior to 2018, with respect to accuracy and scalability. To reiterate briefly here, they include:

- The majority of the research was conducted on the UCR Archive with small to medium size datasets, while real-world problems may have hundreds of thousands of series or more.
- In the 2017 review [9], four algorithms were identified as state-of-the-art algorithms for TSC: FLAT-COTE [28], EE [15], ST [29], and BOSS [30]. Neither the then most accurate classifier, FLAT-COTE, nor the then most accurate similarity-based classifier, EE, scales to large quantities of data due to their prohibitive time complexities of $O(n^2 \ L^4)$ and $O(n^2 \ L^2)$ respectively [2].
- There were TSC algorithms that can be scaled to large quantities of data. However they compromise accuracy in order to achieve that goal. These include classic methods such as DTW with lower bounding and early abandoning, and later methods such as BOSS-VS and WEASEL which are further limited by high memory usage.

3.3 Proximity Forest

In this section, we present our novel algorithm for TSC: Proximity Forest. We present our design decisions and the rationales behind them, then present how to train a Proximity Forest and how to classify using a Proximity Forest. We conclude this section with a description of its complexity.

[2] When we submitted the Proximity Forest for peer revision, HIVE-COTE's results were not publicly available, so we were not able to compare PF with HIVE-COTE.

3.3.1 Design of Proximity Forest

Proximity Forest is a TSC algorithm that was intended to provide a scalable similarity-based alternative to EE. To this end it employs the divide and conquer approach of classical decision trees, together with a novel similarity-based split mechanism.

Proximity Forest uses several important design choices to make the algorithm efficient:

- Proximity Forest uses 11 existing similarity measures. This leverages 30 years of research into designing similarity measures for time series.
- Proximity Forest makes several stochastic choices during the training process. This makes the algorithm fast and introduces high variance between individual classifiers in the ensemble. In ensembling theory, averaging the predictions of multiple high variance and low bias models is expected to produce lower error in total than any of the individual classifiers [65]. This has been demonstrated in many well known ensembles, including Random Forest [65] and Extremely Random Forests [69]. Each tree in Random Forest learns from a subset of the data and a subset of the features. This makes each tree have low bias and high variance, but the error rate of the overall Random Forest is low.
- Using simple and fast random choices helps to improve speed immensely. This also contrasts with the slow leave-one-out cross validation used by Elastic Ensemble to find the "optimal parameters" for similarity measures. Essentially, using a large amount of compute resources to find the "best" parameters does not necessarily lead to the best solution overall since such models overfit more easily.
- Proximity Forest is designed to leverage the benefits of the divide and conquer strategy of tree algorithms. This makes it extremely scalable with an average-case training complexity of $O(n\ log\ (n)\ \ L^2)$ and a classification complexity of $O(log\ (n)\ \ L^2)$ per tree for n training time series of length L. This contrasts with the state of the art, which learns in $O(n^2\ \ L^2)$ (EE) or $O(n^2\ \ L^4)$ (ST, FLAT-COTE). Ensembles such as EE and BOSS uses 1-nearest neighbors as their base classifiers which have high classification time compared to tree algorithms.

3.3.2 Training Proximity Forest

Proximity Forest is an ensemble of k Proximity trees. Algorithms 1 and 2 describe the process of training a single Proximity Tree. Algorithm 1 takes a time series dataset D, number of candidate splits to be tried at each node C_e, and outputs a Proximity Tree

T. To train each Proximity Tree T, we start at the root node and partition the input data into multiple branches and then pass this data recursively down the tree hierarchy. At each node, one branch is created for each class of data reaching the node. Recursion is stopped to create a leaf node when data reaching a node is pure *i.e.* all series belongs to the same class (or $Gini(D) = 0$, see Equation 3.1). Leaf nodes are labeled with the class of data at the node to be used for classification later. In Equation 3.1, p_i is the proportion of data from class c indexed by i.

$$Gini(D) = 1 \quad \sum_{i=1}^{c} (p_i)^2 \tag{3.1}$$

Algorithm 1: build_proximity_tree (D, C_e)

Input: D: a time series dataset
Input: C_e: no. of similarity-based candidates
Output: T: a Proximity Tree

```
1  if is_pure(D) then
2  |  return create_leaf(D)
3  T    create_node()                    // Create tree represented by its root node
4  S_e  generate_similarity_splitters(D, C_e)    // set of candidate splitters
5
6  δ?   arg max_{δ ∈ S_e} WeightedGini(δ)    // select the best splitter using Gini
7
8  T    δ?                               // store the best splitter in the new node T
9  T_B  ;                                // store the set of branch nodes in T
10 // Partition the data using δ? and recurse
11 // δ?_E is the set exemplars of the best similarity-based splitter δ?
      selected by Gini
12 foreach e ∈ δ?_E do
13 |  // δ?_M is the distance measure of the best similarity-based
   |     splitter δ? selected by Gini
14 |  D+   {d ∈ D | δ?_M(d, e) = min_{x ∈ E*}(δ?_M(d, x))}
15 |  t_e  build_tree(D+, C_e)
16 |  Add new branch t_e to T_B
17 end
18 return T
```

At each internal node, Proximity Tree generates a set of splitting functions S_e (line 4 in Algorithm 1) and selects the best splitting function $\delta^?$ that partitions the data into multiple branches at the node (line 6). Each splitting function δ consists of one time series exemplar δ_E randomly selected from each class reaching the node and a randomly parameterized similarity measure δ_M selected at random from the same set of eleven similarity measures used in Elastic Ensemble. Each splitting function partitions the data into branches based on the similarity or "proximity" of each time series at the node

Algorithm 2: generate_similarity_splitters(D, C_e)

Input: D: a time series dataset.
Input: C_e: no. of similarity-based candidates
Output: S_e: a set of similarity-based splitting functions

```
1  S_e    ;                          // set of candidate similarity splitters
2  for i = 1 to C_e do
3      // sample a parameterized measure M uniformly at random from Δ
4      M    Δ      // Δ is the set of 11 similarity measures used in [15]
5
6      // Select one exemplar per class to constitute the set E
7      E    ;
8      foreach class c present in D do
9          D_c    fd 2 D j class(d) = cg            // D_c is the data for class c
10         e    D_c       // sample an exemplar e uniformly at random from D_c
11         Add e to E
12     end
13     // Store measure M and exemplars E in the new splitter δ
14     (δ_M, δ_E)    (M, E)
15     Add splitter δ to S_e
16 end
17 return S_e
```

to the set of exemplars using the parameterized similarity measure (lines 12-15). Ties in similarity computation are broken uniformly at random. We refer each possible way to split the data into branches as one candidate split.

Note that this way of splitting data at the node contrasts with classic decision trees (such as C4.5 or CART [108, 109]) which split data into two branches based on an attribute and a value (for example, if weight is < 50 kg follow the left branch, or else follow the right branch). Since time points may not be aligned in time series data (as in the columns of regular tabular data), attribute-value based splitting cannot be used in a meaningful way when using raw time series data.

Evaluating multiple candidate splits (My Contribution): Since we test multiple splitting functions at each node, we calculate the weighted sum of Gini index of each candidate split, and select the splitting function that produces the minimum weighted sum of Gini as the best splitting function to be retained at the node. The weighting is applied based on the partition size by using Equation 3.2 to make sure that splits with more pure and larger partitions are favored. In Equation 3.2, D_i is the data sent to the ith branch out of B branches, n is the number of series at the node, and n_c is the number of series in the partition for class c. Algorithm 2 describes the procedure

to generate multiple candidate splits. By default, we test $C_e = 5$ candidate splits per node.

$$WeightedGini(D_1, \quad , D_B) = (n_c/n\ ) \ \sum_{i=1}^{B}(Gini_i(D_i)) \tag{3.2}$$

Selection of similarity measures and their parameterization Similarity measures are also selected uniformly at random from the set of eleven elastic similarity measures used in Elastic Ensemble. They are also parameterized randomly using the same parameter space used in Elastic Ensemble. Random choices improve the speed and diversity of the ensemble. This helps to reduce the overall error rate by reducing the variance of the ensemble.

The set of similarity measures we use are: Euclidean Distance (ED); Dynamic Time Warping using the full window (DTWF) [31, 32]; Dynamic Time Warping with a restricted warping window (DTW); Weighted Dynamic Time Warping (WDTW) [57]; Derivative Dynamic Time Warping using the full window (DDTWF) [55, 56]; Derivative Dynamic Time Warping with a restricted warping window (DDTW); Weighted Derivative Dynamic Time Warping (WDDTW); Longest Common Subsequence (LCSS) [58]; Edit Distance with Real Penalty (ERP) [60]; Time Warp Edit Distance (TWE) [61]; and Move-Split-Merge (MSM) [59]. Details of their parameterization can be found in Section 3.2 of Proximity Forest paper (a copy is included in Appendix A).

3.3.3 Classifying Using Proximity Forest

Algorithm 3 describes the procedure to classify a query time series using a single Proximity Tree. Classification begins at the root node and the similarity of query series is compared against each of the exemplars at the node using the selected similarity measure at the node. Measure uses the parameters selected at training time. The query is

Algorithm 3: classification(Q, T)

Input: Q: Query Time Series
Input: T: Proximity Tree
Output: a class label c

```
if is_leaf(T) then
    return majority class of T
(e, T^?)   arg min_{(e^0;T^0)2T_B}  δ_M(Q, e^0)
// recursive call on subtree T^?
return classification(Q, T^?)
```

then sent down the hierarchy of exemplar of the branch to which it is closest to. This process stops when the query reaches a node labeled as leaf. The query is then labeled with the label of the associated with the leaf node. Majority voting among trees is used within the ensemble to decide the final prediction of the query.

3.3.4 Complexity Analysis

A detailed complexity analysis of Proximity Forest can be found in Section 3.4 of the paper [8] (see Appendix A). For brevity, I summarize the analysis here.

Training time Proximity Forest is a tree-based algorithm, so in the worst case the tree depth will be $O(n)$, while on average it is expected to be $O(log(n))$. During training, n series have to traverse the depth of the tree until it reaches a leaf. At each internal node, C_e candidate splits are evaluated and for each candidate split, each series is compared with at most c exemplars from each class and the slowest similarity measure (e.g. WDTW) takes at most $O(L^2)$ time. Therefore, at worst each tree has a training time of $O(n^2 \ C_e \ c \ L^2)$. However, this is rare case in practice for tree algorithms. Due to shallower average depths, on average they can be trained using $O(n \ log(n) \ C_e \ c \ L^2)$ time. Since there are k trees in the ensemble, average training time complexity for Proximity Forest is $O(k \ nlog(n) \ C_e \ c \ L^2)$.

Classification Time During classification, each query will traverse a depth of $O(log(n))$ on average. Each query is compared against at most c exemplars using a similarity measure that takes at most $O(L^2)$ time to compute the similarity between exemplar and query. Majority voting is performed for across k trees in the ensemble for final classification. So the average classification time complexity for Proximity Forest is $O(k \ \log n \ c \ L^2)$.

3.4 Experiments

This section presents experiments conducted to evaluate the performance of Proximity Forest. First I present the scalability of Proximity Forest with respect to the size of training dataset using experiments conducted using a satellite image time series dataset. This is followed by experiments conducted using 85 datasets from 2015 version of the UCR archive. Results for scalability tests are obtained from Lucas et al. [8], while I reproduced the Proximity Forest results on UCR Archive with more experiments for this thesis.

3.4.1 Experiments Using the SITS dataset

To demonstrate the scalability of Proximity Forest algorithm we used a satellite image time series dataset called SITS [3] with 1 million time series of length 46. It has a train-test split ratio of 90%-10%. The dataset is labeled with 24 classes (e.g. "wheat", "corn") and has been converted to a univariate time series dataset using Normalized Difference Vegetation Index (NDVI) (see Figure 2.2).

The following experiments were performed on this dataset, comparing the performance of Proximity Forest against three competitors: BOSS-VS (a dictionary-based classifier designed for scalability), WEASEL (a dictionary-based classifier designed for speed and accuracy), and EE (a similarity-based classifier designed for accuracy). We used BOSS-VS and WEASEL in our comparisons because they are designed to be more scalable than BOSS (one of the four state-of-the-art classifiers as identified by [9]). We repeated each experiment of Proximity Forest 5 times and each of the competitors once. This is because Proximity Forest is a stochastic algorithm while others are deterministic. This section focuses on demonstrating scalability while showing that its accuracy is better than its competitors. We used 100 trees and one candidate split per node. We study parameter selection in Section 3.4.2.

Training Scalability Figure 3.1(a) shows the scalability of Proximity Forest and three other competitors with respect to the training set size. Vertical axes shows the training time in hours while horizontal axis shows increasing training size. We observe that Proximity Forest can be trained on one million time series from SITS dataset in approximately 17 hours. In comparison we estimated that EE will take approximately 200 years to train this amount of data. Even for a small training set of about 2000 time series, EE took 10 hours to train compared to just 79 seconds for Proximity Forest. Fitting a quadratic curve through EE and Proximity Forests gives a quadratic component of 6.3 for EE and $8.10 \quad 10^{6}$ for Proximity Forest showing that observed results matches to theoretical quadratic training complexity of EE, and quasi-linear training complexity of Proximity Forest.

WEASEL was found to be very fast but we were not able to scale it beyond 8000 time series due to its memory complexity even with 64 GB of RAM. As for BOSS-VS, it trains faster than Proximity Forest but shows much lower accuracy as we can see from Figure 3.2.

Testing Scalability Figure 3.1(b) shows the test time per query of Proximity Forest and three other competitors with respect to the training set size. Proximity Forest scales

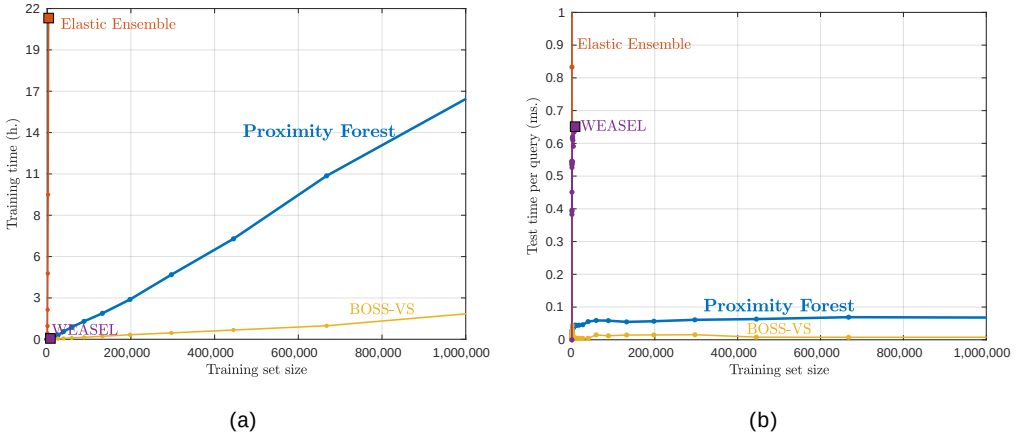


FIGURE 3.1: Training time (a) and testing time per query (b) as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS. This image is taken from [8].

logarithmically with training set size, while EE must scan the full database many times. WEASEL is infeasible to train even with relatively small quantities of training data. Proximity Forest and BOSS-VS require respectively 0.0679 ms and 0.0077 ms to classify a time series with a model trained on 1M time series.

Accuracy Scalability Figure 3.2 shows the accuracy as a function of training set for the four algorithms we compared. Proximity Forest performs better than its competitors, especially when data quantity grows higher. EE and WEASEL does not scale to large quantities of data, while BOSS-VS do not perform very well on this dataset.

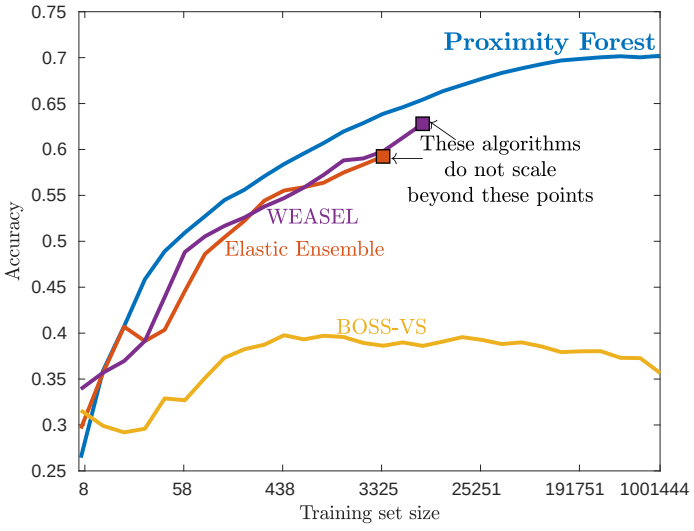


FIGURE 3.2: Accuracy as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS. This image is taken from [8].

3.4.2 Experiments Using UCR Archive

Experiments in this section are performed on the 85 datasets from UCR Archive 2015 version [1]. They compare Proximity Forest to the state of the art circa 2017 as identified by [9]. These experiments use the standard train/test splits available at www.timeseriesclassification.com. Accuracy results reported for of all TSC algorithms except Proximity Forest are obtained from the same website. For Proximity Forest, experiments were run on a cluster of mixed hardware with Intel Xeon-E5-2680-v3 and Intel Xeon-E5-2680-v4 CPUs. Each dataset was run using 16 threads with at least 64GB of memory. Since Proximity Forest is a stochastic algorithm, each experiment was repeated 5 times and the mean of each performance metric is reported (e.g. accuracy, train time, test time, depth).

3.4.2.1 Proximity Forest *vs* Elastic Ensemble

We first compare the accuracy of Proximity Forest with its closest relative Elastic Ensemble, since they are both based on the same set of similarity measures. Figure 3.3 shows a scatter plot comparing the accuracy of Proximity Forest to Elastic Ensemble. Here, each point represents one dataset from the UCR 2005 Archive. For Proximity Forest, we used 100 trees ($k = 100$) and set the number of candidate splits to 5 ($C_e = 5$) (hyperparameters selection is discussed later). The similarity measures used in Proximity Forest select the parameters randomly from the same parameter space used in the Elastic Ensemble. Elastic Ensemble is used with the default parameters. Points above the diagonal line indicate when Proximity Forest is more accurate than Elastic Ensemble. Out of the 85 datasets, Proximity Forest is more accurate or wins 61 times, ties 2 times and loses 22 times when compared to Elastic Ensemble. When Proximity Forest wins, most of the points are well above the diagonal line, indicating that Proximity Forest has a larger margin when winning on average. Datasets with more than 10% difference in accuracy are labeled (dataset names have been truncated for visual aesthetics). For Proximity Forest, 4 datasets have more than 10% wins, compared to only one dataset, *ShapeletSim*, for Elastic Ensemble.

Figure 3.4 shows training and test time of Proximity Forest and Elastic Ensemble in log scale. Proximity Forest trains several magnitudes faster than Elastic Ensemble. As for test time, Proximity Forest has greater test time per query than EE for 12 datasets, the majority of which are small datasets (i.e. less than 50 training instances). The largest such difference is observed for the *Phonemes* dataset for which Proximity Forest takes about 17 sec per query compared to 13 sec per query for EE. In contrast, the test time for Proximity Forest is much smaller than EE for the biggest datasets (i.e. more than 800

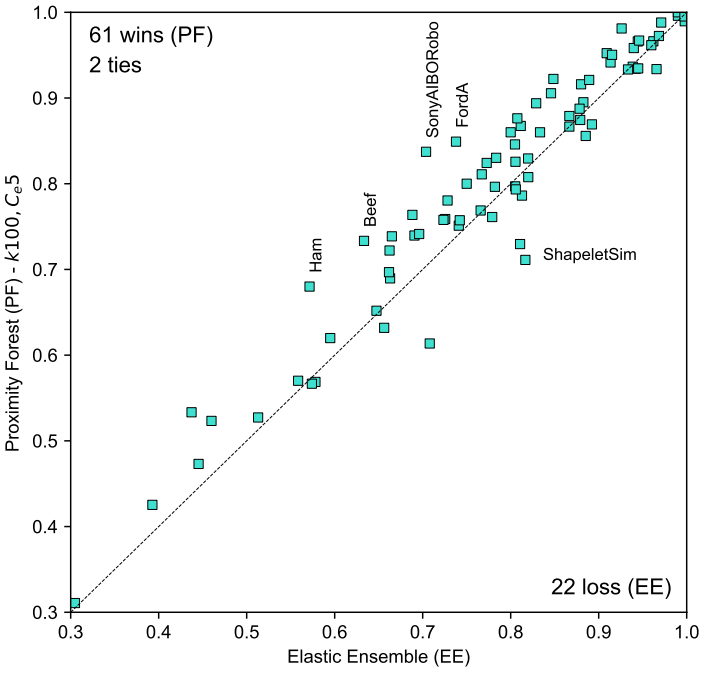
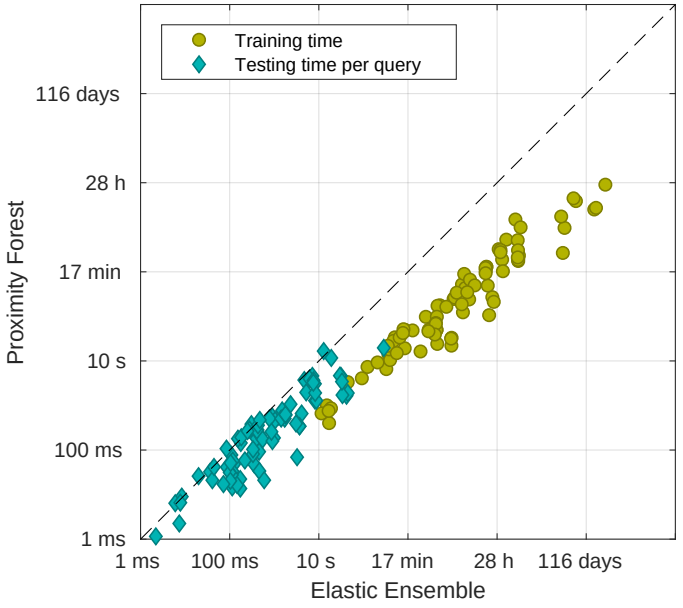


FIGURE 3.3: Comparison of Accuracy between Proximity Forest and Elastic Ensemble. Each point represents a dataset from the UCR Archive. Points above the diagonal indicates datasets where Proximity Forest is more accurate (61 datasets) and points below the diagonal line indicates datasets where Elastic Ensemble is more accurate (22 datasets).



(a)

FIGURE 3.4: Comparison of training and testing times in log scale of Proximity Forest and Elastic Ensemble on 85 UCR datasets. This image is taken from [8].

training instances). For example, the biggest test time difference is for the *HandOutlines* dataset for which Proximity Forest takes about 19 sec per query compared to 286 sec per query for EE.

3.4.2.2 Proximity Forest *vs* State-of-the-art TSC algorithms

Figure 3.5 depicts an accuracy ranking diagram (explained in Section 2.6) showing a comparison of two configurations of Proximity Forest and 5 other classifiers. The first configuration, PF (initial, $k = 100$), is the version of Proximity Forest before I started working on the project. It does not use Gini index to evaluate between multiple candidate splits. The second configuration, PF (with Gini, $C_e = 5$, $k = 100$), is after my improvements. This version evaluates multiple candidate splits at the nodes (5 in this case) using Gini index. We choose EE, BOSS, ST and FLAT-COTE because they were found to be the leading classifiers by the review in Bagnall et al. 2017 [9]. We also added DTW (with a leave-one-out cross validated window size) to the comparison as it has been the traditional benchmark TSC classifier. We used Holm-Bonferroni correction with ($alpha = 0.05$) in this test. The average ranks in order are: 6.0059 for DTW, 4.5000 for EE, 4.3118 for PF (initial), 4.1647 for BOSS, 3.4235 for ST, 3.1824 for PF (with Gini), and 2.4118 for FLAT-COTE. Pairs that cannot be statistically significantly differentiated are grouped using the thick horizontal line on the diagram. We observe that the improvements I made is statistically different from the initial version - significantly boosting its accuracy compared to before. Proximity Forest ranks higher than leading classifiers of the time in each category (*i.e.* EE, ST and BOSS). Proximity Forest is also significantly different than Elastic Ensemble. When compared with FLAT-COTE, this results shows that there is a significant difference between them, but the adjusted p-value is very small 0.0001 [3].

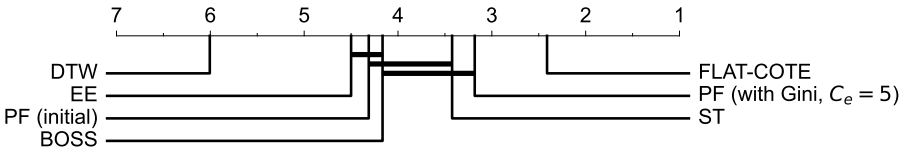


FIGURE 3.5: Average accuracy rank of Proximity Forest compared with other leading TSC algorithms circa 2017 as identified in [9].

[3] Note that the results reported in [8] shows that FLAT-COTE and PF are signi cantly not di erent. This minor di erence can be explained by the variance of the results.

3.4.2.3 Comparing ensemble sizes

Figure 3.6 depicts an accuracy ranking diagram showing a comparison of 8 ensemble sizes (where k is the number of trees). The number of candidate splits is kept constant at 5 ($C_e = 5$) throughout the experiments. Ensembling theory tells us that larger ensembles will have lower variance and lower error rates. In addition, as the ensemble size grows, reduction in error should be a diminishing return. From Figure 3.6, we observe that all configurations are statistically different from each other except the largest three ensembles - $k = 100$, $k = 200$ and $k = 500$. While we expect $k = 500$ to be ranked higher than $k = 200$, this small difference may be due to the variance arising from small datasets or possible overfitting on small datasets. We showed that the variance of our ensemble decreases overall as the number of trees increase in Figure 9 of [8] (see also Appendix A).

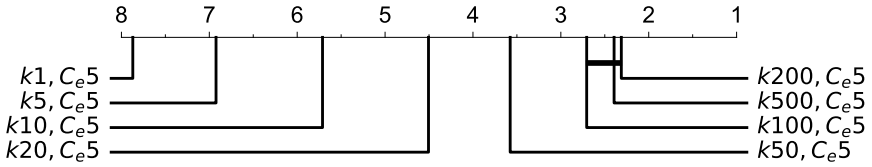


FIGURE 3.6: Average accuracy rank of Proximity Forest with various number of trees k in the ensemble (with number of candidate splits kept constant at $C_e = 5$).

To better understand the comparison between $k = 100$ vs $k = 200$, and $k = 200$ vs $k = 500$, these two configurations are shown in the scatter plots in Figure 3.7. When observing the scatter plot for $k = 100$ vs $k = 200$ we see that there are 47 wins for $k = 200$ and 29 losses. Most of the datasets are still very close to the diagonal line. In the case of $k = 200$ vs $k = 500$ we see that $k = 500$ wins on 34 datasets and losses on 39 datasets. In this case, most of the datasets are even more closer to the diagonal line.

Figure 3.8 shows accuracy comparison of three configurations using the ratios of the accuracy between configurations $k = 200$ / $k = 100$ and $k = 500$ / $k = 100$ on the horizontal and vertical axis, respectively. It shows that points in the shaded region are much further away from the diagonal line than the points above the line in the white region, indicating that $k = 200$ trees will give a good level of accuracy if there is a good amount of computing resources available. However, as most of the points are scattered close around the location (1.0, 1.0) the benefits we get from increasing ensemble size is not substantially large (this is supported from Figure 3.7 as well).

Table 3.1 indicates the training time and test time of 85 UCR datasets with increasing ensemble size. This experiment was done on a cluster with 16-threads. Figure 3.9 shows

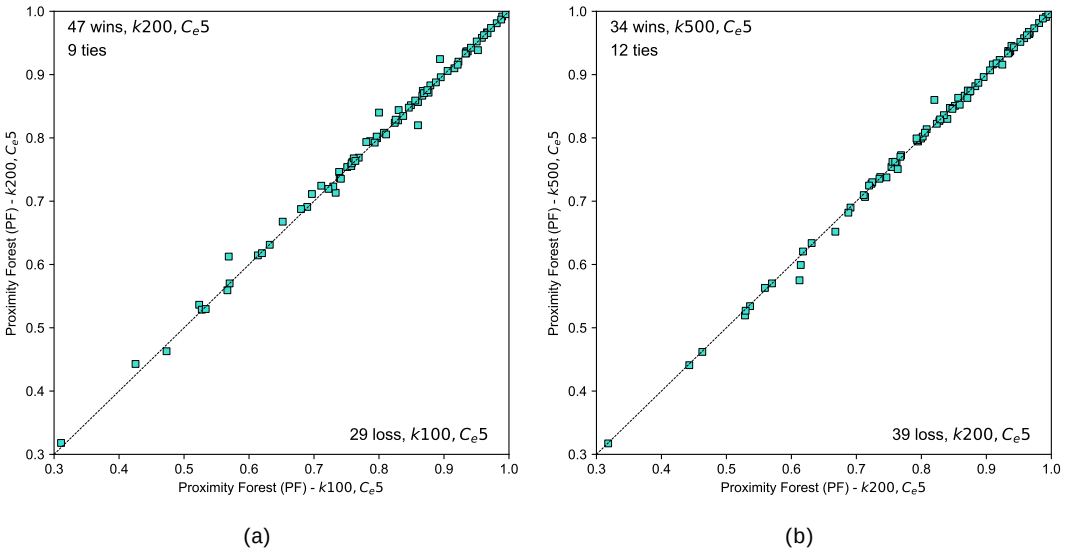


FIGURE 3.7: Scatter plots showing comparisons between different ensemble sizes (with $C_e = 5$) a) $k = 100$ vs $k = 200$ b) $k = 200$ vs $k = 500$.

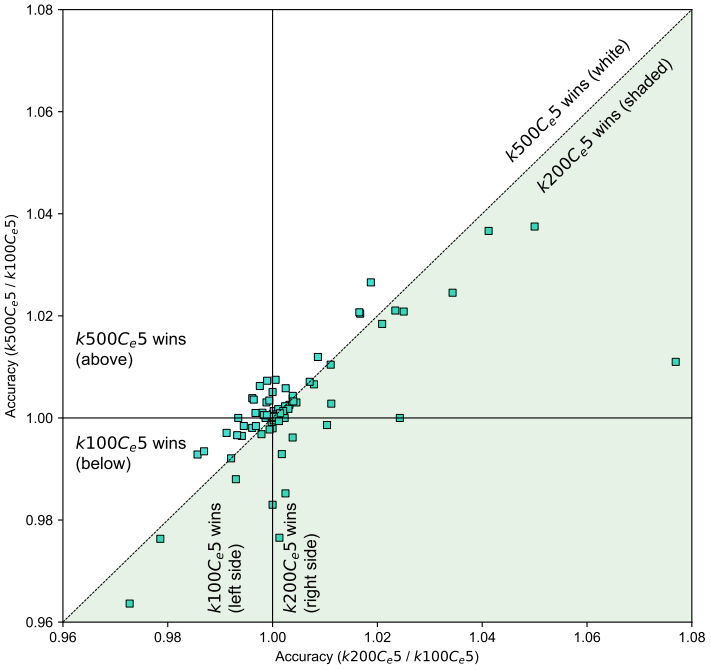


FIGURE 3.8: Accuracy comparison of three configurations of PF using ratio of accuracy between: $k = 200$ / $k = 100$ *vs* $k = 500$ / $k = 100$, with $C_e = 5$.

the training time of the slowest 10 datasets with increasing ensemble size. Using $k = 100$ trees is approx. 1.7 times faster than $k = 200$ trees and approx. 2.3 faster than $k = 500$ trees. It also shows that when increasing from $k = 100$ to $k = 500$, training time increases by roughly 4.2 times (theoretically we would expect about a 5 fold increase). We choose $k = 100$ as the default configuration for most of the experiments as it is a good trade off between accuracy and training time.

Settings	Train Time (hr)	Test Time (hr)	Total Time (hr)
$k1, C_e5$	3.09	0.80	3.52
$k5, C_e5$	1.49	0.52	1.85
$k10, C_e5$	2.00	0.70	2.46
$k20, C_e5$	2.40	0.70	2.84
$k50, C_e5$	4.59	1.26	5.33
$k100, C_e5$	9.56	3.73	12.29
$k200, C_e5$	17.52	4.83	20.49
$k500, C_e5$	41.30	9.74	46.43

TABLE 3.1: Training time and test time of of Proximity Forest on 85 UCR datasets with increasing ensemble size. This experiment was done on a cluster with 16-threads.

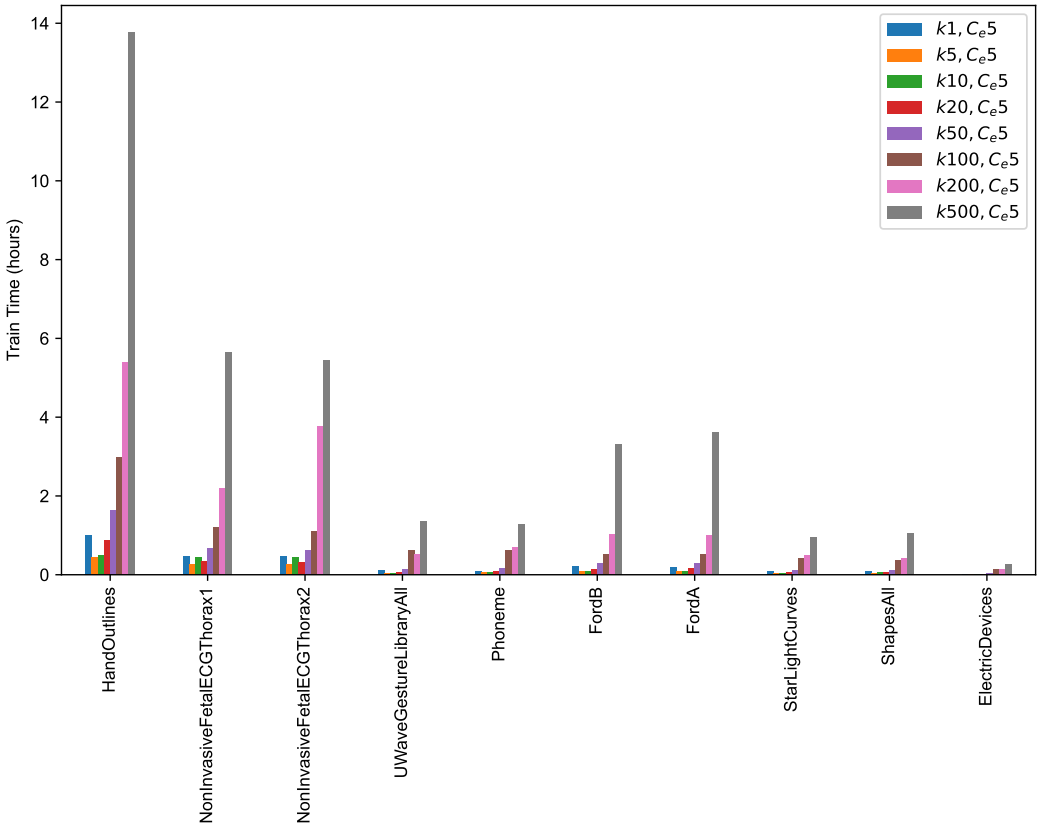


FIGURE 3.9: Training time of the slowest 10 datasets with increasing ensemble size.

3.4.2.4 Comparing the number of candidates splits

Adding the feature to evaluate and select between multiple candidates is my main algorithmic contribution to the Proximity Forest. We saw in Figure 3.5 that my contributions improved the accuracy of Proximity Forest significantly. In this section, to further compare the improvements I made to Proximity Forest, first of all I start with Figure 3.10 which shows a scatter plot showing accuracy comparison of Proximity Forest before and after my contributions. We can observe that Proximity Forest with best candidate split selection ($k = 100, C_e = 5$) wins on 67 datasets, and ties on 6 datasets, while losing only

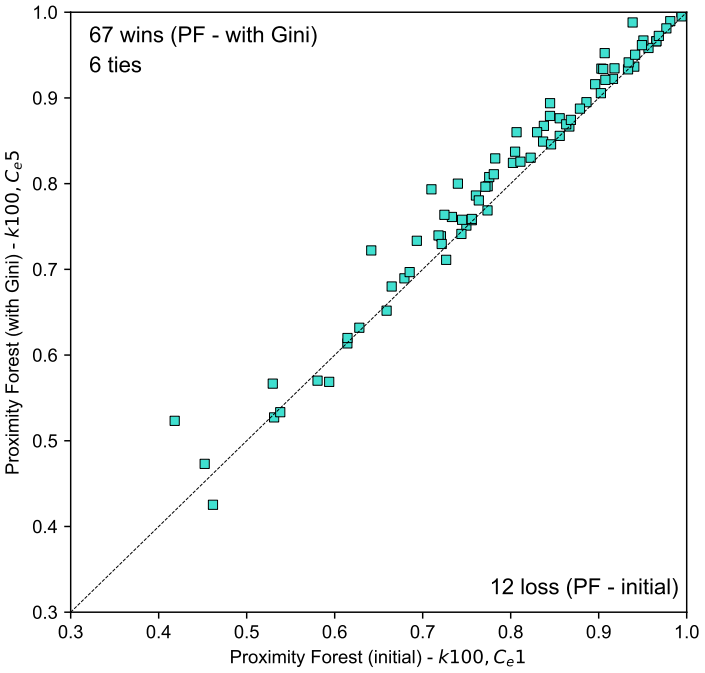


FIGURE 3.10: A scatter plot showing accuracy comparison of Proximity Forest before and after my contributions

on 12 datasets to the initial Proximity Forest without best candidate split selection(*i.e.* $k = 100, C_e = 1$). Most of the points above the diagonal line are well above the line, while the points below the diagonal line are close to it, indicating that the margin of wins are much higher on average compared to the margin of losses.

The experiments published in Proximity Forest paper [8] (see Appendix A), did not explore beyond $C_e = 5$. After the publication of Proximity Forest, I conducted further experiments and I include following new experimental data and observations in this thesis.

Figure 3.11 shows the accuracy rankings of five configurations of Proximity Forest with various number of candidate splits at the node, while maintaining the ensemble size constant at $k = 100$. All configurations are significantly different in accuracy, with rankings in order as follows: $C_e = 1 = 4.3000$, $C_e = 2 = 3.5824$, $C_e = 5 = 2.7294$, $C_e = 10 = 2.3176$, and $C_e = 20 = 2.0706$.

Figure 3.12 shows two comparisons between $C_e = 5$ *vs* $C_e = 10$ and $C_e = 10$ *vs* $C_e = 20$. Once again, each point represents a single dataset. $C_e = 10$ is a significant improvement over $C_e = 5$, with 53 wins, 6 ties and 26 losses. $C_e = 20$ also wins on 52 datasets, ties on 5 and loses on 28 datasets when compared to $C_e = 10$. However, in this case the points are much closer to the diagonal line than Figure 3.12(a).

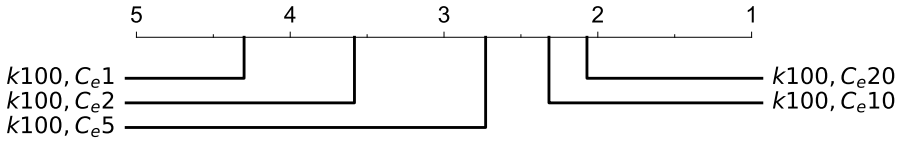


FIGURE 3.11: Average accuracy rank of Proximity Forest with various number of candidate splits per node C_e.

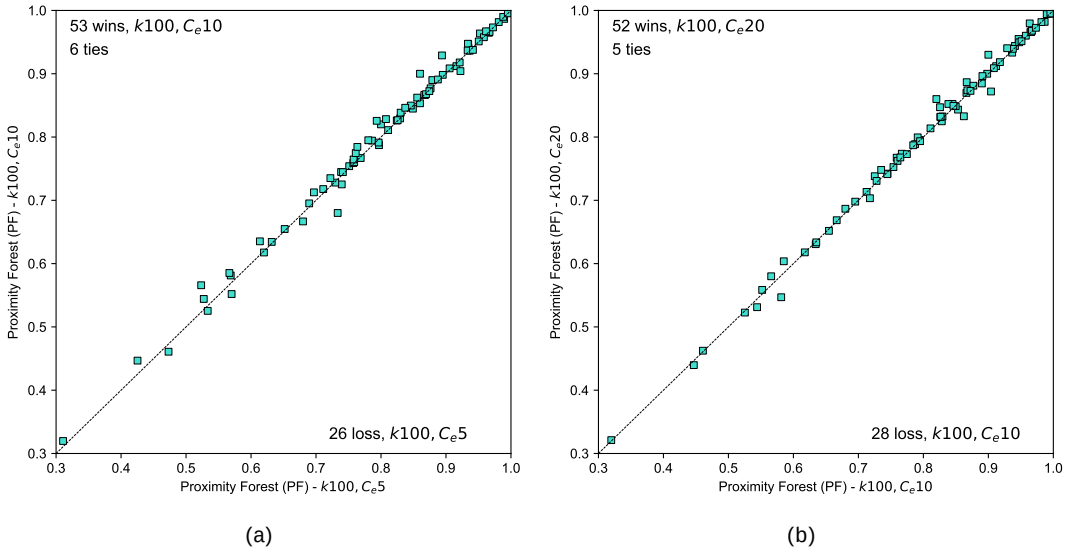


FIGURE 3.12: Scatter plots showing comparisons between different number of candidate splits, while ensemble size is kept constant at $k = 100$ a) $C_e = 5$ vs $C_e = 10$ b) $C_e = 10$ vs $C_e = 20$.

Figure 3.13 shows a scatter plot showing the accuracy of $C_e = 5$ / $C_e = 1$ *vs* $C_e = 10$ / $C_e = 1$. Most of the points are above the diagonal line, indicating that $C_e = 10$ is better than $C_e = 5$. Also, the majority of the points are well above the $y = 1$ line and right of the $x = 1$ line, showing that there is a clear improvement to the accuracy when multiple candidates are evaluated using the Gini (when *i.e.* $C_e > 1$).

Table 3.2 indicates the training time and test time on 85 UCR datasets with increasing increasing number of candidate splits. This experiment was done on a cluster with 16-threads. The results show that when increasing from $C_e = 1$ to $C_e = 5$ (with $k = 100$), training times increases by roughly 4.3 times (total time increases by 2 times, while test time decreases slightly). Interestingly, when increasing from $C_e = 5$ to $C_e = 10$ training time increases by 2 times and from $C_e = 10$ to $C_e = 20$ it increases roughly 2 times as well. In Proximity Forest paper [8] we did not explore beyond $C_e = 5$. Given these new results ($C_e = 10$ and $C_e = 20$), for a researcher using Proximity Forest, who is interested in boosting the accuracy of a particular dataset, it would be worth while to increase C_e

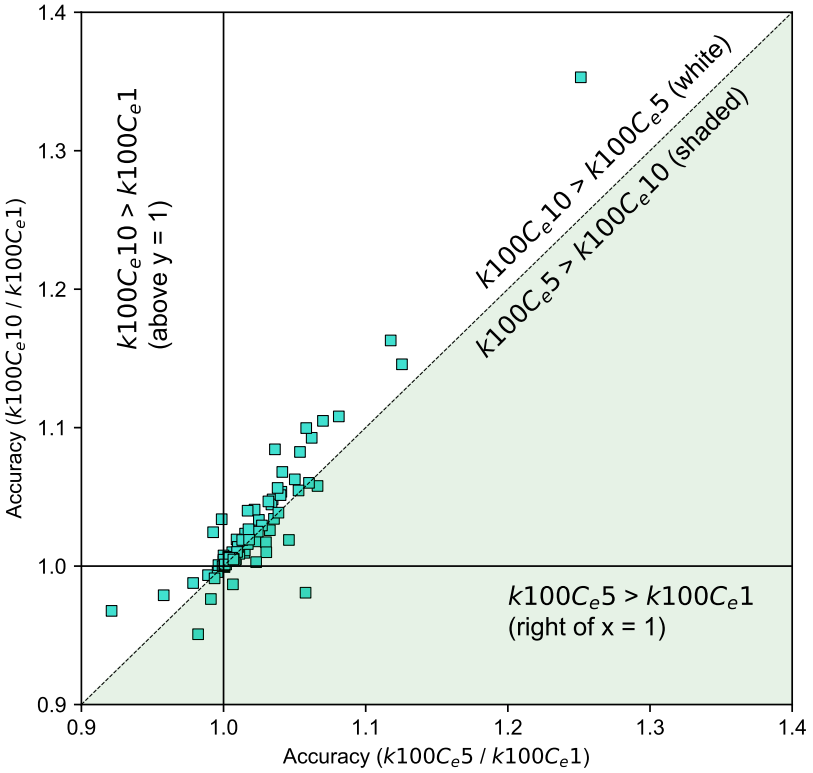


FIGURE 3.13: Accuracy comparison of three configurations of PF using ratio of accuracy between: $C_e = 5$ / $C_e = 1$ *vs* $C_e = 10$ / $C_e = 1$, with $k = 100$.)

Settings	Train Time (hr)	Test Time (hr)	Total Time (hr)	Mean Depth
$k100, C_e1$	2.22	4.63	6.85	8.64
$k100, C_e2$	3.76	3.57	7.32	8.24
$k100, C_e5$	9.56	4.48	14.04	7.56
$k100, C_e10$	18.88	3.55	22.44	7.07
$k100, C_e20$	32.99	2.82	35.81	6.64

TABLE 3.2: Training time, test time and mean depth of Proximity Forest on 85 UCR datasets with increasing number of candidates. This experiment was done on a cluster with 16-threads.

higher than its "default" 5 candidates if computational time is available.

Figures 3.14, 3.15 and 3.16 shows the training time, test time and mean tree depth of the slowest 10 datasets to train with increasing number of candidate splits. First observation is that out of the 32 training hours for $C_e = 20$, the slowest dataset *Handoutline* takes 12 hours to train. We also observe that testing time and mean depth decreases as C_e increases. This is because when C_e is large, Gini can select the best candidate split from a larger pool of candidate splits. Thus, the chance of selecting a function that partitions the data well is increased, reducing the depth of the trees. However, we do observe a very distinct exception in test time for *Phoneme* dataset in Figure 3.15. This is because

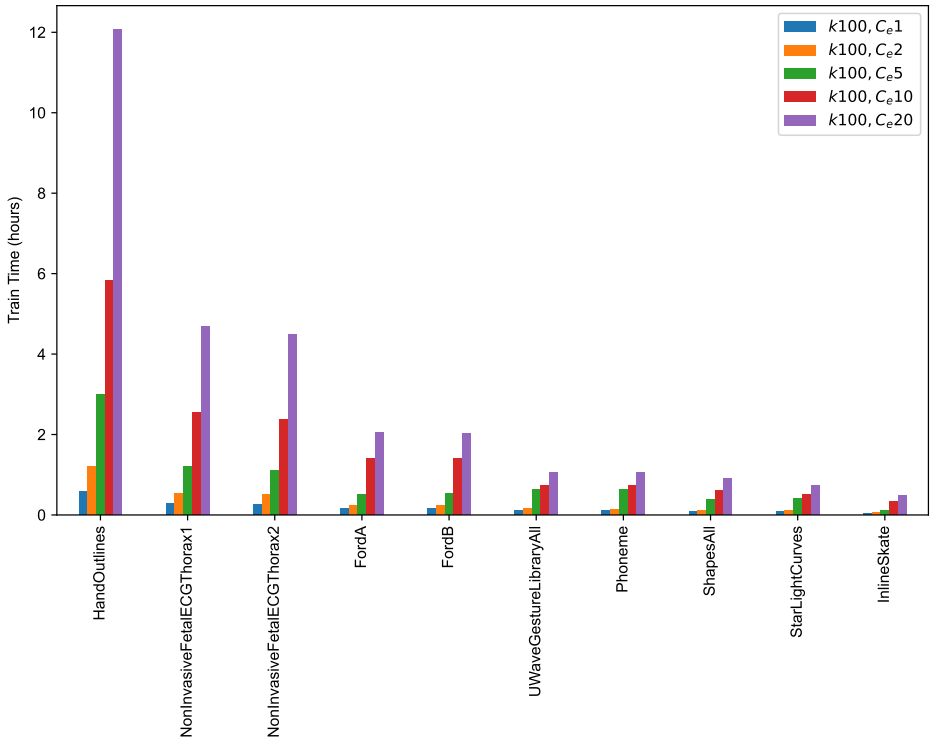


FIGURE 3.14: Training time of the slowest 10 datasets to train with increasing number of candidate splits.

it has 39 classes and the number of classes affect the test time of Proximity Forest. Note that the mean depth for *Phoneme* follows the general trend in Figure 3.16 as expected.

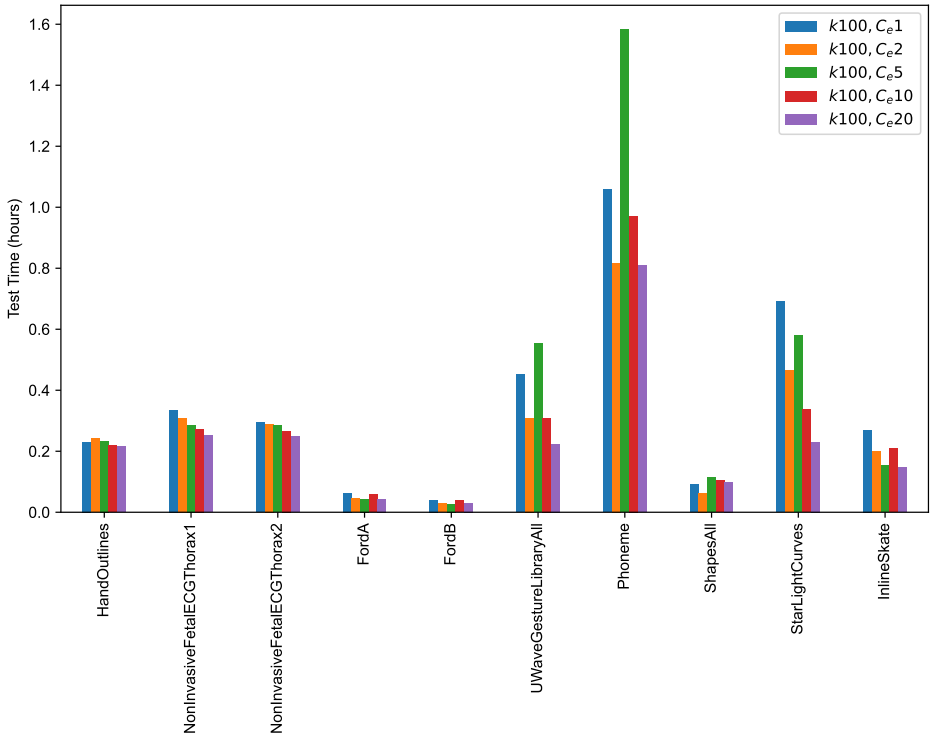


FIGURE 3.15: Test time of the slowest 10 datasets to train with increasing number of candidate splits.

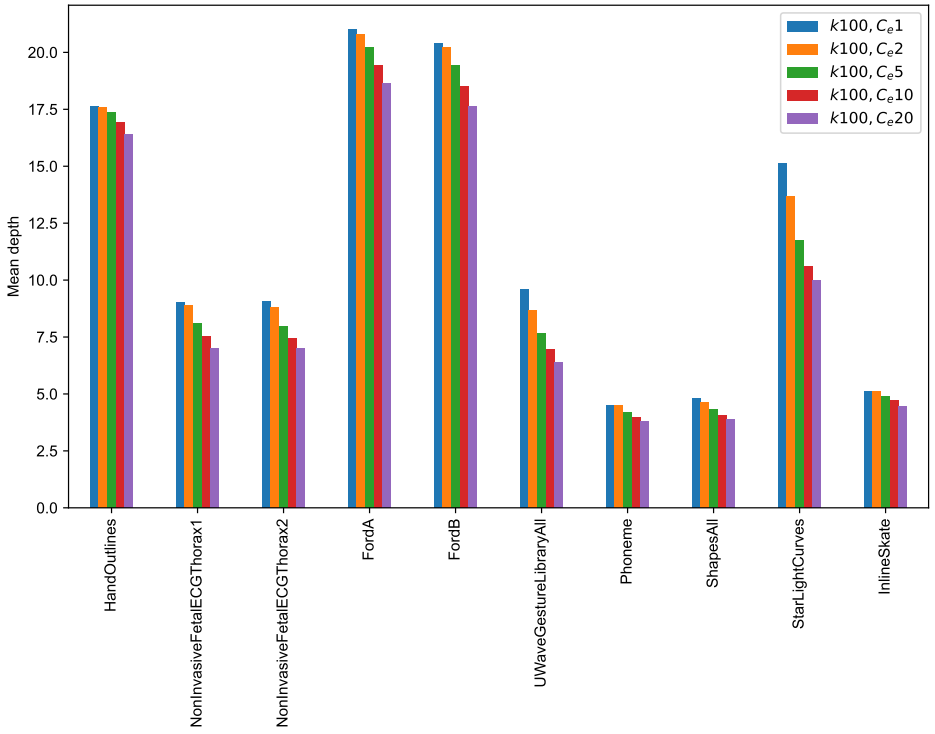


FIGURE 3.16: Mean depth of the slowest 10 datasets to train with increasing number of candidate splits.

3.4.2.5 What if the number of candidates are selected differently for each dataset?

The results we observed suggests that the best number of candidate will be different for each dataset based on its characteristics such as the series length. One reason that fewer candidates might result in better accuracy is overfitting. The more candidates that are considered, the greater the chance that a candidate is selected whose accuracy on the training data is substantially higher than its accuracy on subsequent holdout data.

Figure 3.17 is a bar chart showing the number of datasets for which each configuration of C_e attains the highest accuracy out of the 85 UCR datasets. If there is a tie, the lowest C_e is counted as it has the fastest training time. We observe that out of 85 datasets, the most accurate configuration, $k = 100, C_e = 20$, has the highest accuracy on 39 datasets. The next configuration, $k = 100, C_e = 10$, wins on 18 datasets. The "default configuration", $k = 100, C_e = 5$, wins on 14 datasets. Finally, $k = 100, C_e = 1$, wins on 8 datasets and $k = 100, C_e = 2$, on 6 datasets. $k = 100, C_e = 1$ wins more because we the lowest C_e when there is a tie.

Figure 3.18 shows average accuracy rankings of 5 different number of candidates with one additional configuration, labeled as PF ($k = 50, C_e = best$), which selects the most

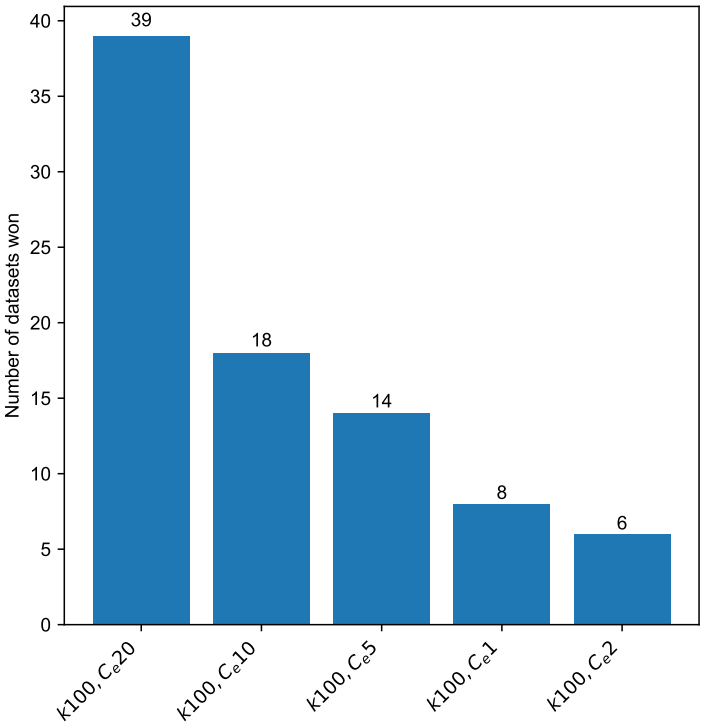


FIGURE 3.17: Number of times each configuration of C_e attains the highest accuracy for 85 UCR datasets. If there is a tie, the lowest C_e is counted as it has the fastest training time.

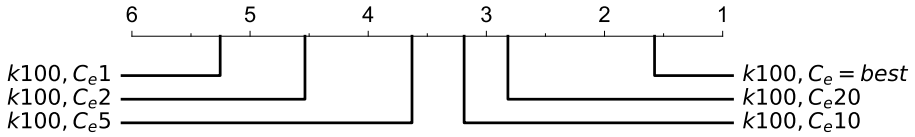


FIGURE 3.18: Average accuracy rankings of 5 different number of candidates run on 85 UCR datasets and "$k = 50, C_e = best$" configuration which selects the best performing setting out of the 5 settings for each dataset.

accurate C_e for each datasets from the 5 choices. This suggests that if a cross-validation strategy is used to select the best hyperparameter C_e for each particular dataset, the accuracy of the Proximity Forest can be improved even further. To improve the overall accuracy, an initialization step can be added to Proximity Forest to train a smaller number of trees on various C_e to find the best performing configuration before training the final ensemble with more trees. I note that this observation was not explicitly stated in the original publication [8], and thus is an additional contribution in this thesis.

3.4.2.6 Trade-off between ensemble size and the number of candidate splits

There is a trade off between selecting the number of candidate splits C_e and the ensemble size k. Increasing both of these hyperparameters together will improve the accuracy while increasing the training time. It is difficult to have a direct comparison between these parameters because the number of trees, k, influences the algorithm at the ensemble level and C_e influences at the node level. To compare these parameters, I kept the total number of candidate splits at the root nodes at 500 using the following four configurations: $k = 50, C_e = 10$, $k = 100, C_e = 5$, $k = 250, C_e = 2$ and $k = 500, C_e = 1$.

Figure 3.19 shows the average accuracy ranking of Proximity Forest with these four different configurations. We observe that the least accurate configuration is $k = 500, C_e = 1$, followed by $k = 250, C_e = 2$. The highest ranked configuration is $k = 100, C_e = 5$, even though there is no significant difference between the last two configurations $k = 100, C_e = 5$ and $k = 50, C_e = 10$.

Figure 3.20 shows three scatter plots comparing the difference between these four configurations. Between $k = 100, C_e = 5$ *vs* $k = 50, C_e = 10$ in Figure 3.20(a) we observe that most of the datasets are evenly dispersed on two sides of the diagonal line (41 wins and 39 loss). Magnitude of wins and losses also relatively small on two sides. Before, in Figure 3.19 we observed that there is no significant difference between them. Note that the smaller ensemble $k50$ is also expected to have a higher variance than $k = 100$. In Figure 3.20(b) when comparing $k = 100, C_e = 5$ *vs* $k = 250, C_e = 2$, we observe

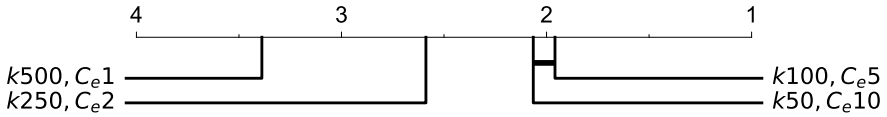


FIGURE 3.19: Average accuracy rank of Proximity Forest with at least a total of 500 candidate splits evaluated at the root nodes using different configurations of C_e and k.

that $k = 100, C_e = 5$ has 57 wins and 20 losses. More points below diagonal are now further away from the line indicating that the magnitude of wins for $k = 100, C_e = 5$ is higher than the wins observed in Figure 3.20(a). From Figure 3.20(c) we see that $k = 100, C_e = 5$ has most of the points well below the diagonal line (69 wins/12 loss) when compared with $k = 500, C_e = 1$.

Figure 3.21 shows the ratio between accuracy of PF with three significantly different configurations (without $k = 50, C_e = 10$). We observe that most points are above the line $y = 1$ and are above the diagonal line (white region) indicating that the best trade-off among the choices is $k = 100, C_e = 5$.

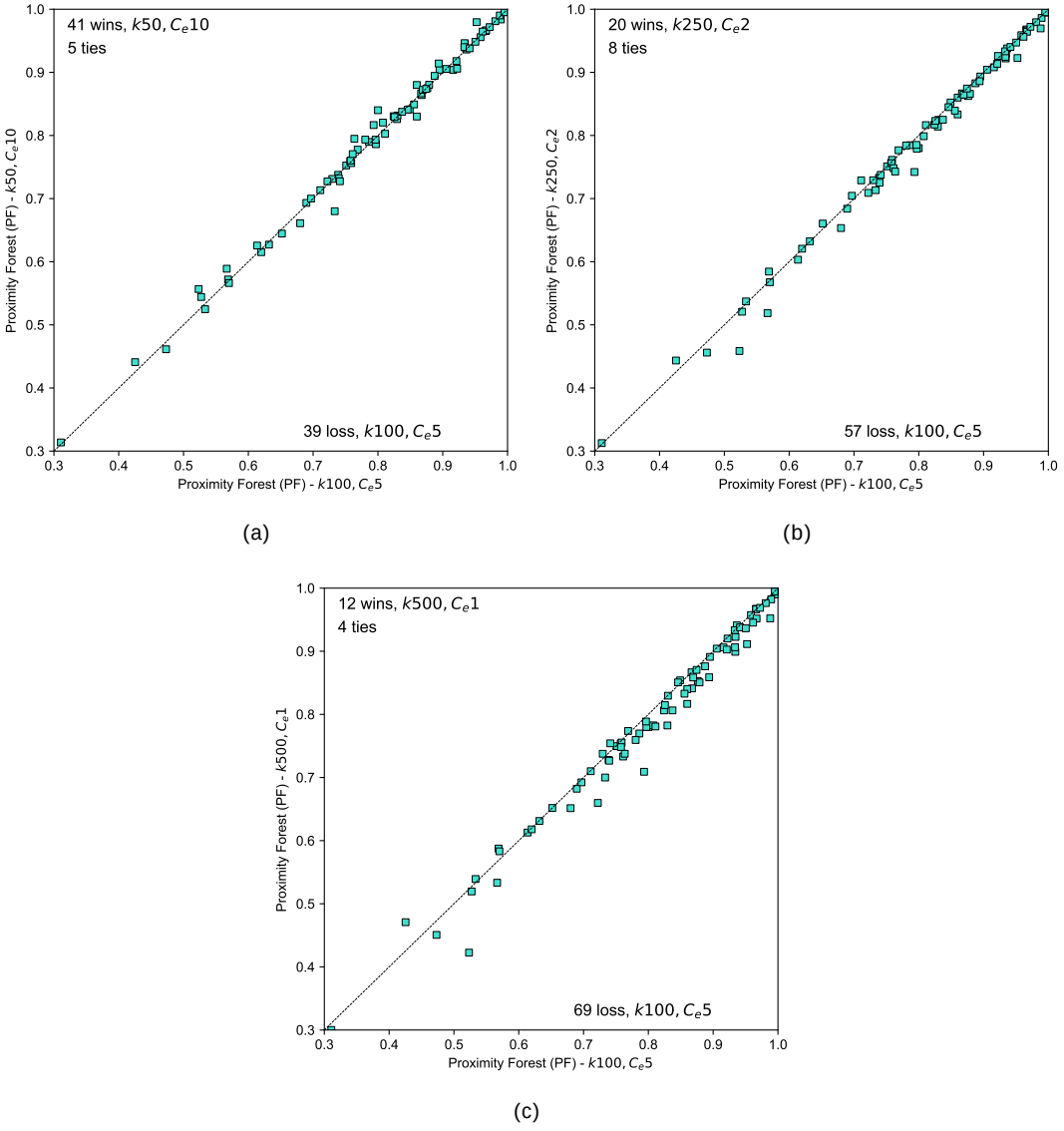


FIGURE 3.20: Scatter plots showing the accuracy of Proximity Forest with a) PF $(k100, C_e5)$ *vs* PF $(k50, C_e10)$ b) PF $(k100, C_e5)$ *vs* PF $(k250, C_e2)$ c) PF $(k100, C_e5)$ *vs* PF $(k500, C_e1)$.

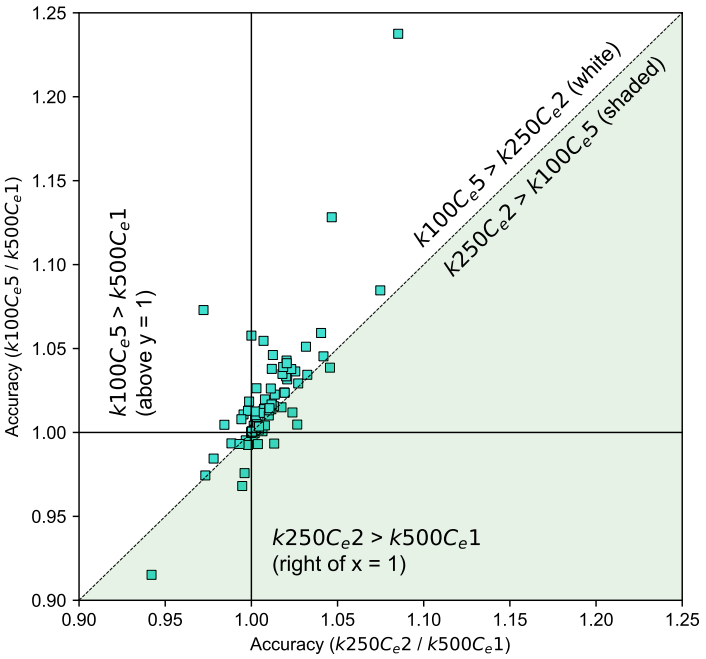


FIGURE 3.21: Accuracy comparison of three configurations of PF using ratio of accuracy between: $k = 250, C_e = 2$ / $k = 500, C_e = 1$ *vs* $k = 100, C_e = 5$ / $k = 500, C_e = 1$.

Table 3.3 shows the training time, test time, total time and mean depth of Proximity Forest on 85 UCR datasets with 4 configurations of Proximity Forest. We observe that increasing k, while decreasing C_e, is not ideal as it increases training time in three cases – except when $k = 500, C_e = 1$. Increasing C_e, does reduce test times (because mean depth decreases), however, accuracy is lower as we observed from Figure 3.19 and Figure 3.21. Therefore, these results suggest that $k = 100, C_e = 5$ as default configuration is an overall good choice to make.

Settings	Train Time (hr)	Test Time (hr)	Total Time (hr)	Mean Depth
$k50, C_e10$	7.90	1.22	9.12	10.96
$k100, C_e5$	8.56	3.73	12.29	11.58
$k250, C_e2$	9.03	6.13	15.16	12.34
$k500, C_e1$	7.35	10.34	17.69	12.70

TABLE 3.3: Training time, test time, total time and mean depth of Proximity Forest on 85 UCR datasets with 4 configurations of Proximity Forest. This experiment was done on a cluster with 16-threads.

3.5 Conclusion

This chapter introduced Proximity Forest: a novel, scalable algorithm for accurate time series classification. Motivated by a need for an accurate algorithm that could learn from millions of time series, it addresses the scalability issues of then state-of-the-art classifiers including FLAT-COTE and its nearest neighbor-based component Elastic Ensemble. Proximity Forest uses divide and conquer strategy of trees and randomization to create a fast and a diverse ensemble. It also leverages decades of work to develop similarity measures for time series and delivers a novel similarity-based splitting criterion for trees. In our case study, we demonstrated that Proximity Forest scales quasi-linearly with the quantity of training data, whereas most state-of-the-art algorithms scale quadratically. Our experiments on the UCR datasets show that Proximity Forest is not only very fast. It also has highly competitive accuracy relative to the current state of the art, and is significantly more accurate than EE.

There are many more strategies that can be explored to improve Proximity Forest while maintaining its quasi-linear complexity. This includes improving the randomized selection of parameters for the similarity measures – which is currently designed to be similar to EE. In addition, since EE, research on similarity measures have also progressed. This includes new measures and progress in developing speed up techniques such as early abandoning. A set of new measures that are less correlated with such speed up techniques could improve Proximity Forest much further.

3.5.1 Contributions

Contributions I made in this chapter include:

1. Co-authoring Proximity Forest, which is the first algorithm with accuracy that competes with state-of-the-art TSC classifiers of the time such as EE, ST, BOSS or FLAT-COTE, while impressively more scalable than any of its other competitors. It is the first TSC classifier that was able to learn from millions of time series within a reasonable time period with such competitive accuracy.

2. Adding the capability to evaluate multiple candidate splits using Gini Index to Proximity Forest. Before that, first version of Proximity Forest selects a random set of exemplars, a similarity measure and its parameters at each node. Evaluating multiple candidates and selecting the best splitting function using the weighted Gini Index boosted its accuracy significantly more than the first version. It is also a fundamental contribution that helped us to develop the state-of-the-art classifier TS-CHIEF (see Chapter 4).

3. Re-implementing the Proximity Forest project in Java with support for multi-threading, with bug fixes and optimization of similarity measures, many features to export a large number of run-time statistics, support for multiple input file formats, and better command-line handling of user input.

4. Helping to prepare a new revision of the paper by conducting a new set of experiments using the implementation of Proximity Forest that evaluates multiple candidate splits, analysis of the new results, generating new figures from the data, proof reading and assisting the authorship of the new versions of the manuscript.

5. Conducting further experiments after the publication of the paper (original copy of the publication is in Appendix A). I have redone the experiments from Section 3.4.2 (except for experiments in Figure 3.2 and 3.4). This is because since the publication, I have fixed some implementation bugs in similarity measures and some of the research conducted in the published paper includes a limited set of experiments (for example, $k = 5, 10, 50$ and 100 to study ensemble sizes, and $C_e = 1, 2$ and 5 to study the number of candidates). I have added more configurations to explore up to $k = 500$ and $C_e = 20$ as well. New research in this thesis that was not included in the published paper include Sections 3.4.2.6 and 3.4.2.5.

Chapter 4

TS-CHIEF

4.1 Introduction

In this chapter I present one the main contributions of my thesis, the TS-CHIEF algorithm, which has been independently assessed as one of the four current state-of-the-art time series classifiers [7, 39]. TS-CHIEF extends Proximity Forest, presented in Chapter 3, by adding dictionary-based and interval-based splitting mechanisms to the nodes of the Proximity Trees.

I include the published journal paper as it is without any modifications. I present a summary of contributions made in this paper at the end of this chapter in Section 4.2.

- **Based on the publication:** Shifaz, A., Pelletier, C., Petitjean, F. and Webb, G.I., 2020. TS-CHIEF: a scalable and accurate forest algorithm for time series classification. Data Mining and Knowledge Discovery, 34(3), pp.742-775.

Data Mining and Knowledge Discovery (2020) 34:742–775
https://doi.org/10.1007/s10618-020-00679-8

TS-CHIEF: a scalable and accurate forest algorithm for time series classification

Ahmed Shifaz[1] · **Charlotte Pelletier**[1,2] · **François Petitjean**[1] · **Geoffrey I. Webb**[1]

Received: 25 June 2019 / Accepted: 24 February 2020 / Published online: 5 March 2020

Abstract

Time Series Classification (TSC) has seen enormous progress over the last two decades. HIVE-COTE (Hierarchical Vote Collective of Transformation-based Ensembles) is the current state of the art in terms of classification accuracy. HIVE-COTE recognizes that time series data are a specific data type for which the traditional attribute-value representation, used predominantly in machine learning, fails to provide a relevant representation. HIVE-COTE combines multiple types of classifiers: each extracting information about a specific aspect of a time series, be it in the time domain, frequency domain or summarization of intervals within the series. However, HIVE-COTE (and its predecessor, FLAT-COTE) is often infeasible to run on even modest amounts of data. For instance, training HIVE-COTE on a dataset with only 1500 time series can require 8 days of CPU time. It has polynomial runtime with respect to the training set size, so this problem compounds as data quantity increases. We propose a novel TSC algorithm, TS-CHIEF (Time Series Combination of Heterogeneous and Integrated Embedding Forest), which rivals HIVE-COTE in accuracy but requires only a fraction of the runtime. TS-CHIEF constructs an ensemble classifier that integrates the most effective embeddings of time series that research has developed in the last decade. It uses tree-structured classifiers to do so efficiently. We assess TS-CHIEF on 85 datasets of the University of California Riverside (UCR) archive, where it achieves state-of-the-art accuracy with scalability and efficiency. We demonstrate that TS-CHIEF can be trained on 130 k time series in 2 days, a data quantity that is beyond the reach of any TSC algorithm with comparable accuracy.

Keywords Time series · Classification · Metrics · Bag of words · Transformation · Forest · Scalable

Responsible editor: Eamonn Keogh.

Extended author information available on the last page of the article

1 Introduction

Time Series Classification (TSC) is an important area of machine learning research that has been growing rapidly in the past few decades (Keogh and Kasetty 2003; Dau et al. 2018b; Bagnall et al. 2017; Fawaz et al. 2019; Yang and Wu 2006; Esling and Agon 2012; Silva et al. 2018). Numerous problems require classification of large quantities of time series data. These include land cover classification from temporal satellite images (Pelletier et al. 2019), human activity recognition (Nweke et al. 2018; Wang et al. 2019), classification of medical data from Electrocardiograms (ECG) (Wang et al. 2013), electric device identification from power consumption patterns (Lines and Bagnall 2015), and many more (Rajkomar et al. 2018; Nwe et al. 2017; Susto et al. 2018). The diversity of such applications are evident from the commonly used University of California Riverside (UCR) archive of TSC datasets (Dau et al. 2018a; Chen et al. 2015).

A number of recent TSC algorithms (Lucas et al. 2019; Schäfer and Leser 2017; Schäfer 2016) have tackled the issue of ever increasing data volumes, achieving greater efficiency and scalability than typical TSC algorithms. However, none has been competitive in accuracy to the state-of-the-art HIVE-COTE (Hierarchical Vote Collective of Transformation-based Ensembles) (Lines et al. 2018).

Our novel method, TS-CHIEF (Time Series Combination of Heterogeneous and Integrated Embedding Forest), is a stochastic, tree-based ensemble that is specifically designed for speed and high accuracy. When building TS-CHIEF trees, at each node we select from a random selection of TSC methods one that best classifies the data reaching the node. Some of these classification methods work with different representations of time series data (Schäfer 2015; Bagnall et al. 2017). Therefore, our technique combines decades of work in developing different classification methods for time series data (Lucas et al. 2019; Lines and Bagnall 2015; Schäfer 2015; Bagnall et al. 2015; Lines et al. 2018; Bagnall et al. 2017) and representations of time series data (Bagnall et al. 2012, 2015; Schäfer 2015), into a hetereogenous tree-based ensemble, that is able to capture a wide variety of discriminatory information from the dataset.

TS-CHIEF achieves scalability without sacrificing accuracy. It is orders of magnitude faster than HIVE-COTE (and its predecessor, FLAT-COTE) while attaining a rank on accuracy on the benchmark UCR archive that is almost indistinguishable, as illustrated in Fig. 1.

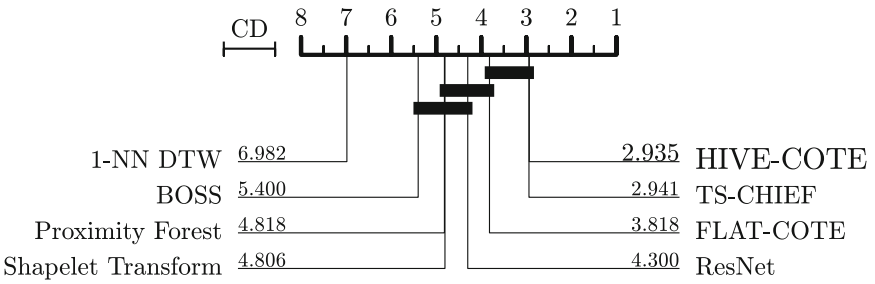


Fig. 1 Critical difference diagram showing the average ranks on error of leading TSC algorithms (described in Sect. 2) across 85 datasets from the benchmark UCR archive (Dau et al. 2018a). The lower the rank (further to the right) the lower the error of an algorithm relative to the others on average

In addition, Fig. 3 shows an experiment that demonstrates the scalability of TS-CHIEF using the Satellite Image Time Series (SITS) dataset (Tan et al. 2017). It is 900x faster than HIVE-COTE for 1500 time series (13 min *versus* 8 days).

Moreover, the relative speedup grows with data quantity: at 132 k instances TS-CHIEF is 46,000x faster. For a training size that took TS-CHIEF 2 days, we estimated 234 years for HIVE-COTE.

Overall, the following strategies are the key to attaining this exceptional efficiency without compromising accuracy: (1) using stochastic decisions during ensemble construction, (2) using stochastic selection instead of cross-validation for parameter selection, (3) using a tree-based approach to speed up training and testing, and (4) including improved variants of HIVE-COTE components Elastic Ensemble (EE) (Bagnall et al. 2015), Bag-of-SFA-Symbols (BOSS) (Schäfer 2015) and Random Interval Spectral Ensemble (RISE) (Lines et al. 2018), but excluding its computationally expensive component Shapelet Transform (ST) (Rakthanmanon et al. 2013) (see Sect. 2.3).

The rest of the paper is organized as follows: Sect. 2 discusses related work. Section 3 presents our algorithm TS-CHIEF, and its time and space complexity. In Sect. 4, we compare the accuracy of TS-CHIEF against state-of-the-art TSC classifiers and investigate its scalability. In Sect. 4, we also study the variance of the ensemble, and the relative contributions of the ensemble's components. Finally, in Sect. 5 we draw conclusions.

2 Related work

Time Series Classification (TSC) aims to predict a discrete label $y \in \{1, \ldots, c\}$ for an unlabeled time series, where c is the number of classes in the TSC task. Although our work could be extended to time series with varying lengths and multi-variate time series, we focus here on univariate time series of fixed lengths. A univariate time series T of length ℓ is an ordered sequence of ℓ observations of a variable over time, where $T = \langle x_1, \ldots, x_\ell \rangle$, with $x_i \in \mathsf{R}$. We use D to represent a training time series dataset and n to represent the number of time series in D.

We now present the main techniques used in TSC research. We also include a summary of training and test complexities of the methods present in this Section in Table 3.

2.1 Similarity-based techniques

These algorithms usually use 1-Nearest Neighbour (1-NN) with *elastic* similarity measures. Elastic measures are designed to compensate for local distortions, misalignments or warpings in time series that might be due to stretched or shrunken subsections within the time series.

The classic benchmark for TSC has been 1-NN using Dynamic Time Warping (DTW), with cross validated warping window size (Ding et al. 2008). The warping window is a parameter that controls the *elasticity* of the similarity measure. A zero window size is equivalent to the Euclidean distance, while a larger warping window size allows points from one series to match points from the other series over longer time frames.

Commonly used similarity measures include variations of DTW such as Derivative DTW (DDTW) (Keogh and Pazzani 2001; Górecki and Łuczak 2013), Weighted DTW (WDTW) (Jeong et al. 2011), Weighted DDTW (WDDTW) (Jeong et al. 2011), and measures based on edit distance such as Longest Common Subsequence (LCSS) (Hirschberg 1977), Move-Split-Merge (MSM) (Stefan et al. 2013), Edit Distance with Real Penalty (ERP)(Chen and Ng 2004) and Time Warp Edit distance TWE (Marteau 2009). Most of these measures have additional parameters that can be tuned. Details of these measures can be found in (Lines and Bagnall 2015; Bagnall et al. 2017).

Ensembles formed using multiple 1-NN classifiers with a diversity of similarity measures have proved to be significantly more accurate than 1-NN with any single measure (Lines and Bagnall 2015). Such ensembles help to reduce the variance of the model and thus help to improve the overall classification accuracy. For example, Elastic Ensemble (EE) combines 11 1-NN algorithms, each using one of the 11 elastic measures (Lines and Bagnall 2015). For each measure, the parameters are optimized with respect to accuracy using cross-validation (Lines and Bagnall 2015; Bagnall et al. 2017). Though EE is a relatively accurate classifier (Bagnall et al. 2017), it is slow to train due to high computational cost of the leave-one-out cross-validation used to tune its parameters—$O(n^2 \cdot \ell^2)$. Furthermore, since EE is an ensemble of 1-NN models, the classification time for each time series is also high—$O(n \cdot \ell^2)$.

Our recent contribution, Proximity Forest (PF), is more scalable and accurate than EE (Lucas et al. 2019). It builds an ensemble of classification trees, where data at each node are split based on similarity to a representative time series from each class. This contrasts with the standard attribute-value splitting methods used in decision trees. Degree of similarity is computed by selecting at random one measure among the 11 used in EE. The parameters of the measures are also selected at random. Proximity Forest is highly scalable owing to the use of a divide and conquer strategy, and stochastic parameter selection in place of computationally expensive parameter tuning.

2.2 Interval-based techniques

These algorithms select a set of intervals from the whole series and apply transformations to these intervals to generate a new feature vector. The new feature vector is then used to train a traditional machine learning algorithm, usually a forest of Random Trees, similar to Random Trees used in Random Forest (but without bagging). For instance, Time Series Forest (TSF) (Deng et al. 2013) applies three time domain transformations—mean, standard deviation and slope—to each of a set of randomly chosen intervals, and then trains a decision tree using this new data representation. The operation is repeated to learn an ensemble of decision trees, similar to Random Trees, on different randomly chosen intervals. Other notable interval-based algorithms are Time Series Bag of Features (TSBF) (Baydogan et al. 2013), Learned Pattern Similarity (LPS) (Baydogan and Runger 2016), and the recently introduced Random Interval Spectral Ensemble (RISE) (Lines et al. 2018).

RISE computes four different transformations for each random interval selected: Autocorrelation Function (ACF), Partial Autocorrelation Function (PACF), and Autoregressive model (AR) which extracts features in time domain, and Power Spec-

trum (PS) which extracts features in the frequency domain (Lines et al. 2018; Bagnall et al. 2015). Coefficients of these functions are used to form a new transformed feature vector. After these transformations have been computed for each interval, a Random Tree is trained on each of the transformed intervals. The training complexity of RISE is $O(k \cdot n \cdot \ell^2)$ (Lines et al. 2018), and the test complexity is $O(k \cdot log(n) \cdot \ell^2)$.

The algorithm presented in this paper has components inspired by RISE, therefore, further details are presented later (see Sect. 3.2.3).

2.3 Shapelet-based techniques

Rather than extracting intervals, where the location of sub-sequences are important, shapelet-based algorithms seek to identify sub-sequences that allow discrimination between classes irrespective of where they occur in a sequence (Ye and Keogh 2009). Ideally, a good shapelet candidate should be a sub-sequence similar to time series from the same class, and dissimilar to time series from other classes. Similarity is usually computed using the minimum Euclidean distance of a shapelet to all sub-sequences of the same length from another series.

The original version of the shapelet algorithm (Ye and Keogh 2009; Mueen et al. 2011), enumerates all possible sub-sequences among the training set to find the "best" possible shapelets. It uses Information Gain criteria to asses how well a given shapelet candidate can split the data. The "best" shapelet candidate and a distance threshold is used as a decision criterion at the node of a binary decision tree. The search for the "best" shapelet is then recursively repeated until obtaining pure leaves. Despite some optimizations proposed in the paper, it is still a very slow algorithm with training complexity of $O(n^2 \cdot \ell^4)$.

Much of the research about shapelets has focused on ways of speeding up the shapelet discovery phase. Instead of enumerating all possible shapelet candidates, researchers have tried to come up with ways of quickly identifying possible "good" shapelets. These include Fast Shapelets (FS) (Rakthanmanon and Keogh 2013) and Learned Shapelets (LS) (Grabocka et al. 2014). Fast Shapelet proposed to use an approximation technique called Symbolic Aggregate Approximation (SAX) (Lin et al. 2007) to shorten the time series during the shapelet discovery process in order to speed up by reducing the number of shapelet candidates. Learned Shapelets (LS) attempted to "learn" the shapelets rather than enumerate all possible candidates. Fast Shapelets algorithm is faster than LS, but it is less accurate (Bagnall et al. 2017).

Another notable shapelet algorithm is Shapelet Transform (ST) (Hills et al. 2014). In ST, the 'best' k shapelets are first extracted based on their ability to separate classes using a quality measure such as Information Gain, and then the distance of each of the "best" k shapelets to each of the samples in the training set is computed (Hills et al. 2014; Bostrom and Bagnall 2015; Large et al. 2017). The distance from k shapelets to each time series forms a matrix of distances which defines a new transformation of the dataset. This transformed dataset is finally used to train an ensemble of eight traditional classification algorithms including 1-Nearest Neighbour with Euclidean distance and DTW, C45 Decision Trees, BayesNet, NaiveBayes, SVM, Rotation Forest and Random

Forest. Although very accurate, ST also has a high training-time complexity of $O(n^2 \cdot \ell^4)$ (Hills et al. 2014; Lines et al. 2018).

One algorithm that speeds up the shapelet-based techniques is Generalized Random Shapelet Forest (GRSF) (Karlsson et al. 2016). GRSF selects a set of random shapelets at each node of a decision tree and performs the shapelet transformation at the node level of the decision tree. GRSF is fast because it is tree-based and uses random selection of shapelets instead of enumerating all shapelets. GRSF experiments were carried out on a subset of the 85 UCR datasets where the values of the hyperparameters—the number of randomly selected shapelets as well as the lower and upper shapelet lengths—are optimized by using a grid search.

2.4 Dictionary-based techniques

Dictionary-based algorithms transform time series data into bag of words (Senin and Malinchik 2013; Schäfer 2015; Large et al. 2018). Dictionary based algorithms are good at handling noisy data and finding discriminatory information in data with recurring patterns (Schäfer 2015). Usually, an approximation method is first applied to reduce the length of the series (Keogh et al. 2001; Lin et al. 2007; Schäfer and Högqvist 2012), and then a quantization method is used to discretize the values, and thus to form words (Schäfer 2015; Large et al. 2018). Each time series is then represented by a histogram that counts the word frequencies. 1-NN with a similarity measure, that compares the similarity between histograms, can then be used to train a classification model. Notable dictionary based algorithms are Bag of Patterns (BoP) (Lin et al. 2012), Symbolic Aggregate Approximation-Vector Space Model (SAX-VSM) (Senin and Malinchik 2013), Bag-of-SFA-Symbols (BOSS) (Schäfer 2015), BOSS in Vector Space (BOSS-VS) (Schäfer 2016) and Word eXtrAction for time SEries cLassification (WEASEL) (Schäfer and Leser 2017).

To compute an approximation of a series, BOP and SAX-VSM use a method called Symbolic Aggregate Approximation (SAX) (Lin et al. 2007). SAX uses Piecewise Aggregate Approximation (PAA) (Keogh et al. 2001) which concatenates the means of consecutive segments of the series and uses quantiles of the normal distribution as breakpoints to discretize or quantize the series to form a word representation. By contrast, BOSS, BOSS-VS, and WEASEL use a method called Symbolic Fourier Approximation (SFA) (Schäfer and Högqvist 2012) to compute the approximated series. SFA applies Discrete Fourier Transformation (DFT) on the series and uses the coefficients of DFT to form a short approximation, representing the frequencies in the series. This approximation is then discretized using a data-adaptive quantization method called Multiple Coefficient Binning (MCB) (Schäfer and Högqvist 2012; Schäfer 2015).

The most commonly used algorithm in this category is Bag-of-SFA-Symbols (BOSS), which is an ensemble of dictionary-based 1-NN models (Schäfer 2015). BOSS is a component of HIVE-COTE and our algorithm also has a component inspired by BOSS. Further details of the BOSS algorithm will be presented in Sect. 3. BOSS has a training time complexity of $O(n^2 \cdot \ell^2)$ and a testing time complexity of $O(n \cdot \ell)$ (Schäfer 2015). A variant of BOSS called BOSS-VS (Schäfer 2016) has a much faster train and test time while being less accurate. The more recent variant WEASEL

(Schäfer and Leser 2017) is more accurate but has a slower training time than BOSS and BOSS-VS, in addition to high space complexity (Schäfer and Leser 2017; Lucas et al. 2019; Middlehurst et al. 2019).

2.5 Combinations of transformations

Two leading algorithms that combine multiple transformations are Flat Collective of Transformation-Based Ensembles (FLAT-COTE) (Bagnall et al. 2015) and the more recent variant Hierarchical Vote COTE (HIVE-COTE) (Lines et al. 2018). FLAT-COTE is a meta-ensemble of 35 different classifiers that use different time series classification methods such as similarity-based, shapelet-based, and interval-based techniques. In particular, it includes other ensembles such as EE and ST. The label of a time series is determined by applying weighted majority voting, where the weighting of each constituent depends on the training leave-one-out cross-validation (LOO CV) accuracy. HIVE-COTE works similarly, but it includes new algorithms, BOSS and RISE, and changes the weighted majority voting to make it balance between each type of constituent module. These modifications result in a major gain in accuracy, and it is currently considered as the state of the art in TSC for accuracy. However, both variants of COTE have high training complexity, lower bounded by the slow cross-validation used by EE—$O(n^2 \cdot \ell^2)$—and exhaustive shapelet enumeration used by ST—$O(n^2 \cdot \ell^4)$.

2.6 Deep learning

Deep learning is interesting for time series both because of the structuring dimension offered by time (deep learning has been particularly good for images and videos) and for its linear scalability with training size. Most related research has focused on developing specific architectures based mainly on Convolutional Neural Networks (CNNs) (Wang et al. 2017; Fawaz et al. 2019), coupled with data augmentation, which is required to make it possible for them to reach high accuracy on the relatively small training set sizes present in the UCR archive (Le Guennec et al. 2016; Fawaz et al. 2019). While these approaches are computationally efficient, the two leading algorithms, Fully Connected Network (FCN) (Wang et al. 2017) and Residual Neural Network (ResNet) (Wang et al. 2017), are still less accurate than FLAT-COTE and HIVE-COTE (Fawaz et al. 2019).

3 TS-CHIEF

This section introduces our novel algorithm TS-CHIEF, which stands for **T**ime **S**eries **C**ombination of **H**eterogeneous and **I**ntegrated **E**mbeddings **F**orest. TS-CHIEF is an ensemble algorithm that makes the most of the scalability of tree classifiers coupled with the accuracy brought by decades of research into specialized techniques for time series classification. Traditional attribute-value decision trees form a tree by recursively splitting the data with respect to the value of a selected attribute. These techniques (and

ensembles thereof) do not in general perform well when applied directly to time series data (Bagnall et al. 2017). As they treat the value at each time step as a distinct attribute, they are unable to exploit the information in the series order. In contrast, TS-CHIEF utilizes splitting criteria that are specifically developed for time series classification.

Our starting point for TS-CHIEF is the Proximity Forest (PF) algorithm (Lucas et al. 2019), which builds an ensemble of classification trees with 'splits' using the proximity of a given time series T to a set of reference time series: if T is closer to the first reference time series, then it goes to the first branch, if it is closer to the second reference time series, then it goes to the second branch, and so on. Proximity Forest integrates 11 time series measures for evaluating similarity. At each node a set of reference series is selected, one per class, together with a similarity measure and its parameterization. These selections are made stochastically. Proximity Forest attains accuracies that are comparable to BOSS and ST (see Fig. 1). TS-CHIEF complements Proximity Forest's splitters with dictionary-based and interval-based splitters, which we describe below. Our algorithmic contributions are three-fold:

1. We take the ideas that underlie the best dictionary-based method, BOSS, and develop a tree splitter based thereon.
2. We take the ideas behind the best interval-based method, RISE, and develop a tree splitter based thereon.
3. We develop techniques to integrate these two novel splitters together with those introduced by Proximity Forest, such that any of the 3 types might be used at any node of the tree.

TS-CHIEF is an ensemble method: we thus paid particular attention to maximizing the diversity between the learners in its design. We do this by creating a very large space of possible splitting criteria. This diversity for diversity sake would be unreasonable if the objective was to create a single standalone classifier. By contrast, by ensembling, this diversity can be expected to reduce the covariance term of ensemble theory (Ueda and Nakano 1996). If ensemble member classifiers are too similar to one another, their collective decision will differ little from that of a single member.

3.1 General principles

During the training phase, TS-CHIEF builds a forest of k trees. The general principles of decision trees remain: tree construction starts from the root node and recursively builds the sub-trees, and at each node, the data is split into branches using a splitting function. Where TS-CHIEF differs is in the use of time-series-specific splitting functions. The details of these splitting functions will be discussed in Sect. 3.2. In short, we use different types of splitters either using time series similarity measures, dictionary-based or interval-based representations. At each node, we generate a set of *candidate* splits and select the best one using the weighted Gini index, *i.e.* the split that maximizes the purity of the created branches (similar to a classic decision tree). We describe the top-level algorithm in Algorithm 1; note that this algorithm is very typical of decision trees and that all the time-series-specific features are in the way we generate candidate splits, as shown in Algorithms 2, 3 and 4.

Algorithm 1: build_tree(D, C_e, C_b, C_r)

Input: D: a time series dataset
Input: C_e: no. of similarity-based candidates
Input: C_b: no. of dictionary-based candidates
Input: C_r: no. of interval-based candidates
Output: T: a TS-CHIEF Tree

if is_pure(D) **then**
  return create_leaf(D)
$T \leftarrow$ create_node() // Create tree represented by its root node
$S \leftarrow \emptyset$ // set of candidate splitters

$S_e \leftarrow$ generate_similarity_splitters(D, C_e)
Add all similarity-based splitters in S_e to S

$S_b \leftarrow$ generate_dictionary_splitters(D, C_b)
Add all dictionary-based splitters in S_b to S

$S_r \leftarrow$ generate_interval_splitters(D, C_r)
Add all interval-based splitters in S_r to S

$\delta^\star \leftarrow \underset{\delta \in S}{\arg\max}\ \text{Gini}(\delta)$ // select the best splitter using Gini

$T_\delta \leftarrow \delta^\star$ // store the best splitter in the new node T
$T_B \leftarrow \emptyset$ // store the set of branch nodes in T
// Partition the data using $\delta^\star$ and recurse
if $\delta^\star$ is similarity-based **then**
  foreach $e \in \delta^\star_E$ **do**
    // $\delta^\star_M$ is the distance measure of the best similarity-based splitter $\delta^\star$ selected by Gini
    $D^+ \leftarrow \{d \in D \mid \delta^\star_M(d, e) = \min_{x \in \delta^\star_E}(\delta^\star_M(d, x))\}$
    $t_e \leftarrow$ build_tree(D^+, C_e, C_b, C_r)
    Add new branch t_e to T_B
  end
else if $\delta^\star$ is dictionary-based **then**
  foreach $e \in \delta^\star_E$ **do**
    // For definition of BOSS_dist, see (Schäfer 2015, Definition 4)
    // $\delta^\star_T(d)$ is the BOSS transformation of d using the BOSS transform function $\delta^\star_T$ of the best dictionary-based splitter $\delta^\star$ selected by Gini
    $D^+ \leftarrow \{d \in D \mid \text{BOSS\_dist}(\delta^\star_T(d), e) = \min_{x \in \delta^\star_E}(\text{BOSS\_dist}(\delta^\star_T(d), x))\}$
    $t_e \leftarrow$ build_tree(D^+, C_e, C_b, C_r)
    Add new branch t_e to T_B
  end
else if $\delta^\star$ is interval-based **then**
  // $(\delta^\star_a, \delta^\star_v)$ is the best attribute-threshold tuple to split on when $\delta^\star_\lambda$ function is applied to the interval
  $D^{\leq} \leftarrow \{d \in D \mid \text{get\_att\_val}\big(\delta^\star_\lambda(\langle d_{\delta^\star_s}, \ldots, d_{\delta^\star_s + \delta^\star_m - 1}\rangle), \delta^\star_a\big) \leq \delta^\star_v\}$
  $t_{\text{left}} \leftarrow$ build_tree($D^{\leq}$, C_e, C_b, C_r)
  Add branch t_{left} to T_B
  $D^{>} \leftarrow \{d \in D \mid \text{get\_att\_val}\big(\delta^\star_\lambda(\langle d_{\delta^\star_s}, \ldots, d_{\delta^\star_s + \delta^\star_m - 1}\rangle), \delta^\star_a\big) > \delta^\star_v\}$
  $t_{\text{right}} \leftarrow$ build_tree($D^{>}$, C_e, C_b, C_r)
  Add branch t_{right} to T_B
return T

3.2 Splitting functions

As mentioned earlier, we choose splitting functions based on similarity measures, dictionary representations and interval-based transformations. This is motivated by the components of HIVE-COTE, namely EE (similarity-based), BOSS (dictionary-based) and RISE (interval-based). The number of *candidate* splits generated per node for each type of splitter type is denoted by C with a subscript as follows: C_e for the number of similarity-based splitters, C_b for the number of dictionary-based splitters and C_r for the number of interval-based splitters. We do not include ST (shapelets) because of its high training time computational complexity. We also omit TSF because

Algorithm 2: generate_similarity_splitters(D, C_e)

Input: D: a time series dataset.
Input: C_e: no. of similarity-based candidates
Output: S_e: a set of similarity-based splitting functions

```
// Note that this algorithm is reproduced from (Lucas et al. 2019, Algorithm 2)

S_e ← ∅                                   // set of candidate similarity splitters
for i = 1 to C_e do
    // sample a parameterized measure M uniformly at random from Δ
    M ←~ Δ   // Δ is the set of 11 similarity measures used in (Lucas et al. 2019)

    // Select one exemplar per class to constitute the set E
    E ← ∅
    foreach class c present in D do
        D_c ← {d ∈ D | class(d) = c}                    // D_c is the data for class c
        e ←~ D_c        // sample an exemplar e uniformly at random from D_c
        Add e to E
    end
    // Store measure M and exemplars E in the new splitter δ
    (δ_M, δ_E) ← (M, E)
    Add splitter δ to S_e
end
return S_e
```

Algorithm 3: generate_dictionary_splitters(D, C_b)

Input: D: a time series dataset
Input: C_b: no. of dictionary-based candidates
Output: S_b: a set of dictionary-based splitting functions

```
S_b ← ∅                                   // set of candidate dictionary splitters
for i = 1 to C_b do
    // See Section 3.2.2 for details of BOSS parameters
    T ← select_random_BOSS_transformation()

    // Select one BOSS histogram per class to constitute the set E
    E ← ∅
    foreach class c present in D do
        D_c ← {d ∈ D | class(d) = c}                    // D_c is the data for class c
        e ←~ D_c        // sample an exemplar e uniformly at random from D_c
        // Recall that we precomputed T(D) during initialization
        Add T(e) to E                          // T(e) is the BOSS histogram of e
    end
    // Store BOSS transform T and exemplar histograms E in the new splitter δ
    (δ_T, δ_E) ← (T, E)
    Add splitter δ to S_b
end
return S_b
```

Algorithm 4: generate_interval_splitters(D, C_r)

Input: D: a time series dataset
Input: C_r: no. of interval-based candidates
Output: S_r: a set of interval-based splitting functions

1 $S_r \leftarrow \emptyset$ // set of candidate interval splitters
2 $m_{\min} \leftarrow 16$ // minimum length of random intervals
3 $C_r^* \leftarrow \lfloor C_r/4 \rfloor$ // no. of attributes per transform
4 $R \leftarrow \lceil C_r^*/m_{\min} \rceil$ // no. of random intervals to compute
5 **for** $i = 1$ **to** R **do**
6 // Get random interval - length m ($m \in [m_{\min}, \ell]$), starting at index s
7 $(\delta_s, \delta_m) \leftarrow$ get_random_interval($m_{\min}, \ell$)
8 // Add splitters for each transformation
9 **foreach** δ_λ *in {ACF,PACF,AR,PS }* **do**
10 // Apply λ to each time series
11 $D_T \leftarrow \emptyset$
12 **foreach** d *in* D **do**
13 // Create d_T, a vector of m attribute-values obtained by applying δ_λ to the interval
14 $d_T \leftarrow \delta_\lambda(\langle d_{\delta_s}, \ldots d_{\delta_{s+m-1}} \rangle)$
15 Add d_T to D_T
16 **end**
17 // Calculate no. of attributes to select from ith random interval and transform function δ_λ
18 $A \leftarrow \lfloor C_r^*/R \rfloor$
19 // Select at random A attributes in D_T
20 $\tilde{P} \leftarrow$ get_random_attributes(D_T, A)
21 **foreach** attribute δ_a in $\tilde{P}$ **do**
22 $\delta_v \leftarrow$ find_best_threshold(δ_a)
23 Add $\big((\delta_s, \delta_m), \delta_\lambda, (\delta_a, \delta_v)\big)$ to S_r
24 **end**
25 **end**
26 **end**
27 **return** S_r

its accuracy is ranked lower than EE, ST and BOSS (Bagnall et al. 2017). We next describe how we generate each of these types of splitting function.

3.2.1 Similarity-based

This splitting function uses the method of Proximity Forest (Lucas et al. 2019), which splits the data based on the similarity of each time series to a set of reference time series (Lines 16 to 22 in Algorithm 1). At training time, for each candidate splitter, a random measure δ_M, that is randomly parameterized, is selected, as well as a set δ_E of random reference time series, one from each class (Algorithm 2). We use the same 11 similarity measures used in Proximity Forest (Lucas et al. 2019), and the parameters for these measures are also selected randomly from the same distributions used in Proximity Forest (Lucas et al. 2019). If TS-CHIEF is trained with only the

similarity-based splitter enabled (i.e. $C_b = C_r = 0$), then it is exactly Proximity Forest.

When designing our earlier work Proximity Forest (Lucas et al. 2019) we chose to select a single random reference per class instead of an aggregate representation because it is very fast and it introduces diversity to the ensemble. We found that using a single random reference per class was working very well in Proximity Forest, and so we used it in the equivalent similarity-based splitter, and also in the dictionary-based splitter presented in Sect. 3.2.2.

When splitting the data at training time and at classification time, the similarity of a query instance Q to each reference time series e in δ_E is evaluated using the selected measure δ_M. Q is passed down the branch corresponding to the e to which Q is closest.

3.2.2 Dictionary-based

This type of split functions also uses a similarity-based splitting mechanism, except that it works on a set of time series that have been transformed using the BOSS transformation (Schäfer 2015, Algorithm 1), and that it uses a variant of the Euclidean distance (Schäfer 2015, Definition 4) to measure similarity between transformed time series.

The BOSS transformation is used to convert the time series dataset into a bag-of-word model. We start by describing the BOSS transformation. To compute a BOSS transformation of a single time series, first, a window of fixed length w is slid over the time series, while converting each window to a Symbolic Fourier Approximation (SFA) word of length f (Schäfer and Högqvist 2012; Schäfer 2015). SFA is a two-step procedure: (1) it applies a low pass filter—using only the low frequency coefficients of the Discrete Fourier Transformation (DFT) –, (2) it converts each window (subseries) into a word using a data adaptive quantization method called Multiple Coefficient Binning (MCB). MCB defines a matrix of discretization levels for an alphabet size α (default is $\alpha = 4$) and a word length f. This leads to α^f possible words. There is also a parameter called *norm*. If it is equal to true, the first Fourier coefficient of the window is removed, which is equal to mean-centering the time series (i.e., subtracting the mean). SFA words are then counted to form a word frequency histogram that is used to compare two time series. BOSS uses a bespoke Euclidean distance, namely BOSS_dist, which measures the distance between sparse vectors (which here represent histograms) in a non-symmetric way, such that the distance is computed only on elements present in the first vector (Schäfer 2015).

We now turn to explaining how we use BOSS transformations to build our forest. Since BOSS has four different hyperparameters, many possible BOSS transformations of a time series can be generated. Before we start training the trees, t BOSS transformations (histograms for all time series) of the dataset are pre-computed based on t randomly selected sets of BOSS parameters. Similar to the values used in BOSS, the four parameters are selected uniformly at random from the following ranges: the window length $w \in \{10 \ldots \ell\}$, SFA word length $f \in \{6, 8, 10, 12, 14, 16\}$, the normalization parameter $norm \in \{true, false\}$, and $\alpha = 4$.

At training time (Algorithm 3), for each candidate splitter δ, a random BOSS transformation δ_T , with replacement, is chosen, as well as a set δ_E of random reference

time series from each class for which the transformation δ_T has been applied. Each training time series is then passed down the branch of the reference series for which the BOSS distance between histogram of the series and the reference time series is lowest. We then generate several such splitters and choose the best one according to the Gini index.

At classification time, when a query time series Q arrives at a node with a dictionary-based splitter, we start by calculating its transformation into a word histogram (the transformation δ_T selected at training). We then compare this histogram to each reference time series in δ_E, and Q is passed down the branch corresponding to the reference time series to which Q is closest.

3.2.3 Interval-based

This type of splitting function is designed to work in a similar fashion to the RISE component used in the HIVE-COTE. Recall that RISE is an interval-based algorithm that uses four transformations (ACF, PACF, AR—in time domain and PS—in frequency domain) to convert a set of random intervals to a feature vector. Once the feature vectors have been generated, RISE uses a classic attribute-value splitting mechanism to train a forest of binary decision trees (similar to Random Forest—but without bagging).

A notable difference between RISE, and our interval-based splitter is that the random intervals are selected per tree in RISE, whereas our interval-based splitter selects random intervals per candidate split at the node level. This choice is for two main reasons. Firstly, choosing intervals per candidate split at node level helps to explore a larger number of random intervals. Secondly, this also separates the hyperparameter k (number of trees) from the number of random intervals used by the interval-based splitters which depends on the hyperparameter C_r (number of interval-based splits per node). Separating these hyperparameters helps to change the effects of interval-based splitter on the overall ensemble, without changing the size of the whole ensemble. Consequently, this design decision also helps to increase the diversity of the ensemble.

Algorithm 4 describes the process of generating features using random intervals and the four transform functions to generate C_r interval-based candidates splits. Each candidate splitter δ is defined by a pair (δ_s, δ_m) that represent the interval start and its length respectively, a function δ_λ (one of ACF, PACF, AR or PS) which is applied to the interval and a pair (δ_a, δ_v) that indicates the attribute δ_a and threshold value δ_v on which to split. The values of (δ_s, δ_m) are randomly selected to get a random interval of length between minimum length $m_{\min} = 16$ and ℓ the length of the time series. We set $m_{\min}$, and other parameters required by the four transform functions to be exactly same as it was in RISE. The values of the pair (δ_a, δ_v) are optimized such that the Gini index is maximized when the data are split on the attribute δ_a for a threshold value δ_v.

When splitting the data at training time and at classification time, δ_λ is applied to the interval of query instance Q defined by δ_s and δ_m, obtaining the attribute vector Q_λ. If get_att_val$(Q_\lambda, \delta_a) \leq \delta_v$ (the value of attribute δ_a of Q_λ is less than the threshold value), Q is passed down the left branch. Otherwise it is passed down the right. Contrary to the similarity—and dictionary-based splitting functions, which used a distance based mechanism to partition the data (to produce a variable number of branches

depending on the number of classes present at the node), the "attribute-value" based splitting mechanism used by the interval-based splitting functions produce binary splits (Lines 32 to 38 in Algorithm 1).

3.3 Classification

For each tree, a query time series Q is passed down the hierarchy from the root to the leaves. The branch taken at each node depends on the splitting function selected at the node. Once Q reaches the leaf, it is labelled with the class with which the training instances that reached that leaf were classified. Recall that the tree is repeatedly split until pure, so all training instances that reach a leaf will have the same class. This process is presented in the Algorithm 5. Finally, a majority vote by the k trees is used to label Q.

Algorithm 5: classification(Q, T)

Input: Q: Query Time Series
Input: T: TS-CHIEF Tree
Output: a class label c

```
1  if is_leaf(T) then
2  |  return majority class of T
3
4  if T_δ is similarity-based then
5  |  (e, T*) ← arg min_{(e',T')∈T_B} δ_M(Q, e')
6  else if T_δ is dictionary-based then
7  |  (e, T*) ← arg min_{(e',T')∈T_B} BOSS_dist(δ_T(Q), e')
8  else if T_δ is interval-based then
9  |  Q_λ ← δ_λ(⟨Q_{δ_s}, ..., Q_{δ_s+δ_m−1}⟩)
10 |  // compare the δ_a^th attribute value from Q_λ to the split value
11 |  if get_att_val(Q_λ, δ_a) ≤ δ_v then
12 |  |  T* ← T_left
13 |  else
14 |  |  T* ← T_right
15 |
16
17 // recursive call on subtree T*
18 return classification(Q, T*)
```

3.4 Complexity

Training time complexity Proximity Forest, on which TS-CHIEF builds, has average training time complexity that is quasi-linear with the quantity of training data, $O(k \cdot n \log(n) \cdot C_e \cdot c \cdot \ell^2)$ for k trees, n training time series of length ℓ, C_e similarity-based candidate splits, and c classes (Lucas et al. 2019). The term k comes from the number of trees to train and $\log(n)$ from the average depth of the trees. In the worst case, tree depth may be n, however, on average, tree depth can be expected to be $\log(n)$. The term $n \cdot C_e \cdot c \cdot \ell^2$ represents the order of time required to select the best of C_e candidate

splits and partition the data thereon, based on the similarity of n training instances to c reference time series at the node using a random similarity measure. The slowest of the similarity measures used (WDTW) is bounded by $O(\ell^2)$.

The addition of the dictionary-based splitter adds a new initialization step and a new selection step to the Proximity Forest algorithm. The initialization part pre-computes t BOSS transformations for n time series. Since the cost of BOSS transforming one time series is $O(\ell)$ (Schäfer 2015, Section 6), the complexity of the initialization part is $O(t \cdot n \cdot \ell)$. The Euclidean-based BOSS distance has a complexity of $O(\ell)$ (Schäfer 2015, Definition 4) and must be applied to every example at the node for each of the C_b (dictionary-based candidate splits), resulting in order $O(C_b \cdot c \cdot n \cdot \ell)$ complexity for generating and evaluating dictionary splitters at each node of each tree.

The interval-based splitting functions are attribute-value splitters; we detail the complexity for training a node receiving n' time series. Each interval is transformed using 4 different functions (ACF, PACF, AR and PS), which takes at most $O(\ell^2)$ time (Lines et al. 2018, Table 4), leading to $O(r \cdot n' \cdot \ell^2)$ for r intervals taken where r is proportional to C_r. For each of the C_r candidate splits the data is then sorted and scanned through to find the best split—$O(C_r \cdot n \log(n))$. Put together, this adds $O(C_r \cdot n \cdot \ell^2 + C_r \cdot n \log(n))$ complexity to the split selection stage. Note that ℓ in this term represents an upper bound on the length of random intervals selected. The expected length of random interval is 1/3 of ℓ.

Overall, TS-CHIEF has quasi-linear average complexity with respect to the training size :

$$O\Big(\underbrace{t \cdot n \cdot \ell}_{\text{initialization}} + \underbrace{k \cdot \log(n)}_{\substack{\text{avg.depth} \\ \text{for } k \text{ trees}}} \cdot \Big[\underbrace{C_e \cdot c \cdot n \cdot \ell^2}_{\text{similarity}} + \underbrace{C_b \cdot c \cdot n \cdot \ell}_{\text{dictionary}} + \underbrace{C_r \cdot n \cdot \ell^2 + C_r \cdot n \log(n)}_{\text{interval}} \Big] \Big).$$

In Sect. 4.4, we have included an experiment to measure the fraction of training time taken by each splitter type over 85 UCR datasets (Chen et al. 2015). As expected, the dominant term in the training complexity is the term representing the similarity-based splitter. In practice, our experiments show that the similarity-based splitter takes about 80% of the training time (See Fig. 9, on page 21).

Classification time complexity Each time series is simply passed down k trees, traversing an average of $log(n)$ nodes. Moreover, the complexity at each node is dominated by the similarity-based splitters. Overall, this is thus a $O(k \cdot \log(n) \cdot c \cdot \ell^2)$ average case classification time complexity.

Memory complexity The memory complexity is linear with the quantity of data. We would need to store one copy of n time series of length ℓ—this is $O(n \cdot \ell)$. In the worst case there are as many nodes in each of the k trees as there are time series and at each node, and we store one exemplar time series for each of the c classes, $O(k \cdot n \cdot c)$. We pre-store all t dictionary-based transformations, $O(t \cdot n \cdot \ell)$. Overall, this is $O(n \cdot \ell + k \cdot n \cdot c + t \cdot n \cdot \ell)$.

4 Experiments

We start by evaluating the accuracy of TS-CHIEF on the UCR archive, and then assess its scalability on a large time series dataset. In essence, we show that TS-CHIEF can reach the same level of accuracy as HIVE-COTE but with much greater speed, thanks to TS-CHIEF's quasi-linear complexity with respect to the number of training instances. We then present a study on the variation of training accuracy against the ensemble size, followed by an assessment of the contribution of each type of splitter in TS-CHIEF. Finally, we finish this section by presenting a study of the memory requirements for TS-CHIEF.

We implemented a multi-threaded version of TS-CHIEF in Java, and have made it available via the Github repository https://github.com/dotnet54/TS-CHIEF. In these experiments, we used multiple threads when measuring the accuracy of TS-CHIEF under various configurations (Sects. 4.1, 4.3 and 4.4). For experiments with TS-CHIEF, we report the mean accuracy for 10 runs unless explicitly stated. However, we used a single thread (1 CPU) for both TS-CHIEF and HIVE-COTE when measuring the timings for scalability experiments in Sect. 4.2.

Throughout the experiments, unless mentioned otherwise, we use the following parameter values for TS-CHIEF: $t = 1000$ dictionary-based (BOSS) transformations, $k = 500$ trees in the forest. When training each node, we concurrently assess the following number of candidates: 5 similarity-based splitters, 100 dictionary-based splitters and 100 interval-based splitters. Ideally, we would also want to raise the number of candidates for the similarity-based splitter, but this has a significant impact on training time (since passing the instances down the branches measures in $O(\ell^2)$) with marginal improvement in accuracy (Lucas et al. 2019). Note that we have not done any tuning of these numbers of candidates of each type. For hyperparameters of the similarity-based splitters (e.g. parameters for distance measures), we used exactly the same values used in Proximity Forest (Lucas et al. 2019). Similarly, for dictionary—and interval-based splitters, we used the same hyperparameters used in BOSS and RISE components of HIVE-COTE (Lines et al. 2018).

4.1 Accuracy on the UCR Archive

We evaluate TS-CHIEF on the UCR archive (Chen et al. 2015), as is the de facto standard in TSC research (Bagnall et al. 2017). We use the 2015 version with 85 datasets, because the very recent update adding further datasets is still in beta (Dau et al. 2018a). All 85 datasets are fixed length univariate time series that have been z-normalized. We use the standard train/test split available at http://www.timeseriesclassification.com.

To compare multiple algorithms over the 85 datasets, we use critical difference diagrams, as it is the standard in machine learning research (Demšar 2006; Benavoli et al. 2016). We use the Friedman test to compare the ranks of multiple classifiers (Demšar 2006). In these statistical tests, the null hypothesis corresponds to no significant difference in the mean rankings of the multiple classifiers (at a statistical significant level $\alpha = 0.05$). In cases where null-hypothesis was rejected, we use the Wilcoxon signed

Table 1 p-Values for the pairwise comparison of classifiers

	BOSS	ST	PF	RN	FCT	HCT	TS-CHIEF
DTW	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**
BOSS		0.035	0.042	0.022	**< 0.001**	**< 0.001**	**< 0.001**
ST			0.684	0.112	**< 0.001**	**< 0.001**	**< 0.001**
PF				0.127	**0.002**	**< 0.001**	**< 0.001**
RN					0.330	0.005	0.017
FCT						**< 0.001**	0.045
HCT							0.687

Bold values indicate pairs of classifiers that are statistically different at the 0.05 level after applying a Holm correction. The algorithms are abbreviated as follows. RN: ResNet, DTW: 1-NN DTW, FCT: FLAT-COTE, and HCT: HIVE-COTE

rank test to compare the pair-wise difference in ranks between classifiers, while using Holm's correction to adjust for family-wise errors (Benavoli et al. 2016).

We compare TS-CHIEF to the 3 time series classifiers identified by (Bagnall et al. 2017) as the most accurate on the UCR archive (FLAT-COTE, ST and BOSS), as well as the de facto standard 1-NN DTW, deep learning method ResNet and the more recent HIVE-COTE (the current most accurate on the URC archive) and Proximity Forest (the inspiration for TS-CHIEF). We use results reported at the http://www.timeseriesclassification.com website for these algorithms, except for TS-CHIEF, Proximity Forest (our result (Lucas et al. 2019)) and the deep learning ResNet method for which we obtained the results from Fawaz et. al's (2019) review of Deep Learning methods for TSC.

Figure 1 displays mean ranks (on error) between the 8 algorithms; which is also the main result of this paper in terms of accuracy. TS-CHIEF obtains an average rank of 2.941, which rivals HIVE-COTE at 2.935 (statistically not different). FLAT-COTE comes next with an average rank of 3.818. Next, Residual Neural Network (ResNet) is ranked at 4.300.

Table 1 presents the results of a comparison between each pair of algorithms. We use Wilcoxon's signed rank test and judge significance at the 0.05 significance level using a Holm correction for multiple testing. The comparisons that are judged significant at the 0.05 level are displayed in bold type. TS-CHIEF, HIVE-COTE, FLAT-COTE and ResNet are all statistically indistinguishable from one another except that HIVE-COTE is significantly more accurate than FLAT-COTE. TS-CHIEF and the two COTEs are all significantly more accurate than all the other algorithms except ResNet.

To further examine the accuracy of TS-CHIEF against both COTE algorithms, Fig. 2 presents a scatter plot of pairwise accuracy. Each point represents a UCR dataset. TS-CHIEF wins above the diagonal line. TS-CHIEF wins 40 times against HIVE-COTE (green squares), loses 38 times and ties on 7 datasets. Compared to FLAT-COTE (red circles), TS-CHIEF wins 47 times, and loses 33 times, with 5 ties. It is interesting to see that TS-CHIEF gives results that are quite different to both COTE algorithms, with a few datasets for which the difference in accuracy is quite large.

Table 4 presents the accuracy of all 8 classifiers for the 85 datasets. TS-CHIEF is most accurate of all classifiers (rank 1) on 31 datasets, while HIVE-COTE is most

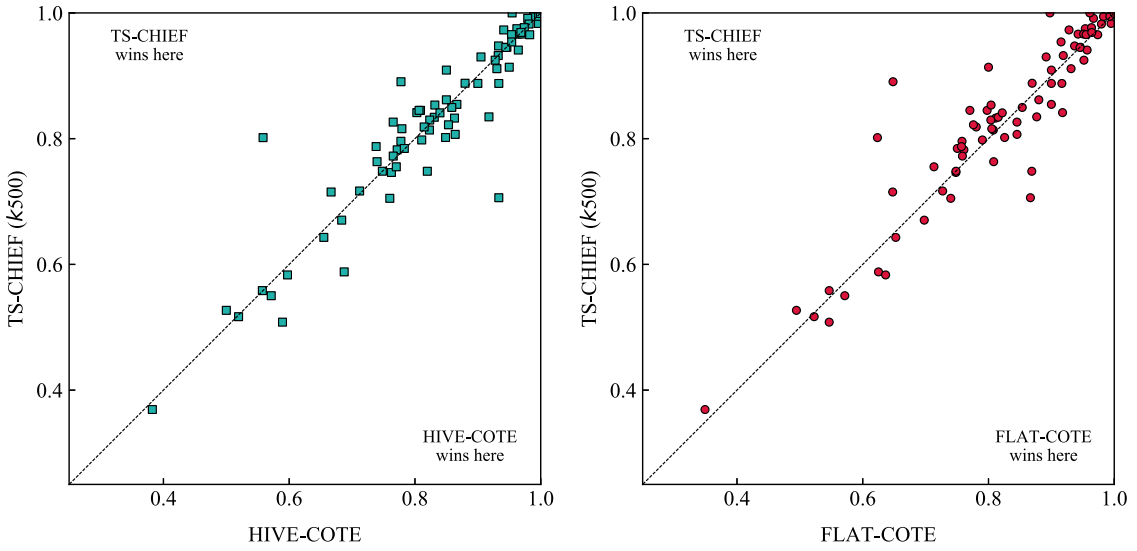


Fig. 2 Comparison of accuracy for TS-CHIEF *versus* HIVE-COTE (left) and TS-CHIEF *versus* FLAT-COTE (right) on 85 UCR datasets. TS-CHIEF's win/draw/loss against HIVE-COTE is 40/7/38 and against FLAT-COTE is 47/5/33

Table 2 Mean accuracy of TSC algorithms grouped by dataset types identified in the UCR archive (Chen et al. 2015)

Dataset type	DTW	BOSS	ST	PF	RN	FCT	HCT	CHIEF
DEVICE	59.54	66.81	70.58	64.40	72.94	69.47	**73.24**	69.26
ECG	87.14	91.69	94.43	92.34	92.87	**95.56**	95.20	94.88
IMAGE	74.87	81.27	79.71	82.30	79.79	82.67	84.05	**84.35**
MOTION	70.54	75.60	77.87	78.55	76.83	79.13	79.66	**81.40**
SENSOR	77.50	79.89	84.05	83.66	85.77	**86.01**	84.81	84.67
SIMULATED	87.25	92.61	92.20	89.26	93.13	93.96	94.49	**94.79**
SPECTRO	80.34	85.00	86.55	81.67	86.19	84.72	**88.31**	86.49

FCT and HCT indicates FLAT-COTE and HIVE-COTE respectively, and RN indicates ResNet
Bold values indicate classifiers with the highest mean accuracy for each dataset type

accurate on 23, despite their mean ranks being equal at 2.94. With respect to the benchmark UCR archive, TS-CHIEF rivals HIVE-COTE in accuracy (without being statistically different).

We also looked at the accuracy of TS-CHIEF and other TSC methods on different data domains as identified in the UCR achive (Chen et al. 2015). The results, in Table 2, shows that TS-CHIEF performed best in three data domains, although the mean accuracy in these cases are similar to HIVE-COTE.

Although we were not able to compare running time with either of the COTE algorithms because of their very high running time, even on the UCR archive, we give here a few indications of runtime for TS-CHIEF. The experiment was carried out using an AMD Opteron CPU (1.8 GHz) with 64 GB RAM, with 16 CPU threads. Note that this is the only timing experiment we ran with multiple threads, timing experiments in Sect. 4.2 were run using a single thread.

Average training and testing times were respectively of about 3 h and 27 min per dataset, but with quite a large difference between datasets. TS-CHIEF was trained on 69 datasets in less than 1 h each and less than one day was sufficient to train TS-CHIEF on all but 10 datasets. It however took about 10 days to complete training on all the datasets, mostly due to the `HandOutlines` dataset which took more than 4 days to complete. Our experiments confirmed our theoretical developments about complexity: TS-CHIEF was largely unaffected by dataset size with the largest dataset `ElectricDevices` trained in 2 h 24 min and tested in 9 min. `HandOutlines` is the dataset with the longest series and in the top-10 in terms of training size, which shows that the quadratic complexity with the length has still a non-negligible influence on training time. The next section details scalability with respect to length and size.

4.2 Scalability

TS-CHIEF is designed to be both accurate and highly scalable. Section 3.4 showed that the complexity of TS-CHIEF scales quasi-linearly with respect to number of training instances n and quadratically with respect to length of the time series ℓ. To assess how this plays out in practice, we carried out two experiments to evaluate the runtime of TS-CHIEF when (1) the number of training instances increases, and (2) the time series length increases. We compare TS-CHIEF to the HIVE-COTE algorithm which previously held the title of most accurate on the UCR archive. We performed these experiments with 100 trees (1 run). As the accuracy on the UCR archive has been evaluated for 500 trees (Sect. 4.1), we also estimated the timing for 500 trees (5 times slower). The experiments used a single run of each algorithm using 1 CPU (single thread) on a machine with an Intel(R) Xeon(R) CPU E5-2680 v3@2.50GHz processor with 200 GB of RAM.

4.2.1 Increasing training set size

First, we assessed the scalability of TS-CHIEF with respect to the training set size. We used a Satellite Image Time Series (SITS) dataset (Tan et al. 2017) composed of 1 million time series of length 46, with 24 classes. The training set was sampled using stratified random sampling method while making sure at least one time series from each class in the training data is present in the stratified samples. We also used a stratified random sample of 1000 test instances for evaluation. We evaluated the accuracy and the total runtime as a function of the number of training time series, starting from a subsample of 58, and logarithmically increasing up to 131,879 (a sufficient quantity to clearly define the trend).

Figures 3 and 4 show the training time and the accuracy, respectively, as a function of the training set size for TS-CHIEF (in olive) and HIVE-COTE (in red). Figure 3 shows that TS-CHIEF trains in time that is quasi-linear with respect to the number of training examples, rather than the quadratic time for HIVE-COTE. For about 1500 training time series, HIVE-COTE requires about 8 days to train, while TS-CHIEF was able to train in about 13 min. This is thus an 900x speed-up.

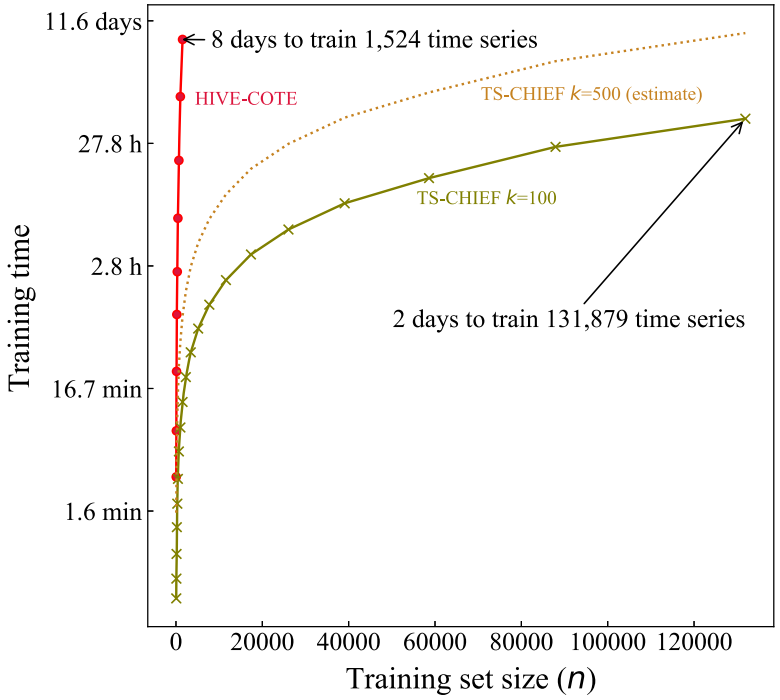


Fig. 3 Training time in logarithmic-scale for TS-CHIEF *versus* HIVE-COTE with increasing training size using the Satellite Image Time Series dataset (Tan et al. 2017). Even for 1500 time series, TS-CHIEF is more than 900 times faster than the current state of the art HIVE-COTE

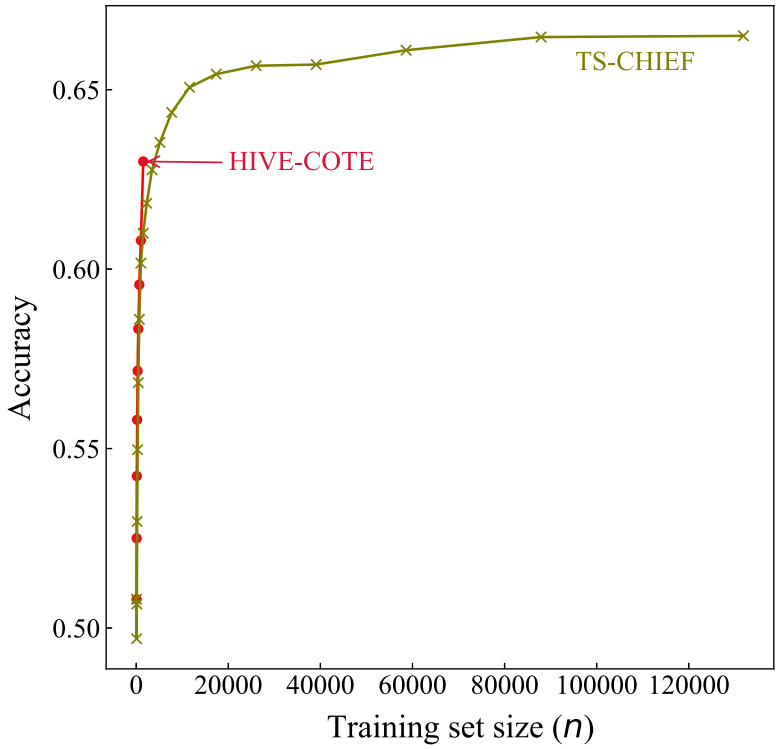


Fig. 4 Accuracy as a function of training set size for SITS dataset

Figure 4 shows that TS-CHIEF has similar accuracy to HIVE-COTE for any given number of training time series. However, TS-CHIEF achieves 67% accuracy within 2 days by learning from about 132 k time series. By fitting a quadratic curve through HIVE-COTE training time, we estimate that it will require 234 years for HIVE-COTE

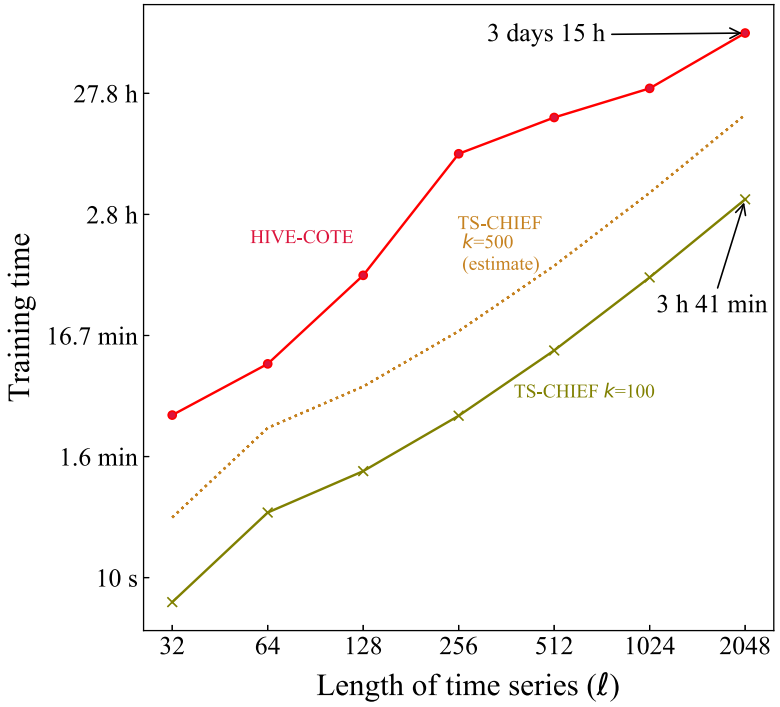


Fig. 5 Training time as a function of the series length ℓ for a one UCR dataset

to learn from 132 k time series. This is a speed-up of 46,000 times over HIVE-COTE. Furthermore, to train all one million time series in the SITS dataset, we estimated that it would take 13,550 years to train HIVE-COTE, while TS-CHIEF is estimated to take 44 days. This is a speed-up of 90,000 times over HIVE-COTE for 1M time series.

Moreover, Fig. 4 indicates that HIVE-COTE can only achieve 60% after 2 days of training, i.e. a decrease of 7.9% compared to TS-CHIEF. In practice, the execution time of TS-CHIEF thus scales very close to its theoretical average complexity (Sect. 3.4) by scaling quasi-linearly with the training set size.

4.2.2 Increasing length

Second, we assessed the scalability of TS-CHIEF with respect to the length ℓ of the time series. We use here `InlineSkate`, a UCR dataset composed of 100 time series and 550 test time series of original length 1882. We resampled the length from 32 to 2048 by using an exponential scale with base 2.

Figure 5 displays the training time for both TS-CHIEF (in olive) and HIVE-COTE (in red) as a function of the length of the time series. TS-CHIEF can learn from 100 time series of length 2048 in about 4 h, while HIVE-COTE requires more than 3 days. This is a 24x speed up. It also mirrors the theoretical training complexity of TS-CHIEF in $O(\ell^2)$, and HIVE-COTE in $O(\ell^4)$ (Lines et al. 2018) with respect to the length of the time series.

4.3 Ensemble size and variance of the results

We also conducted an experiment to study the accuracy (and variance) *versus* ensemble size k (see Fig. 6). It shows that the accuracy increases with k up to a point where it

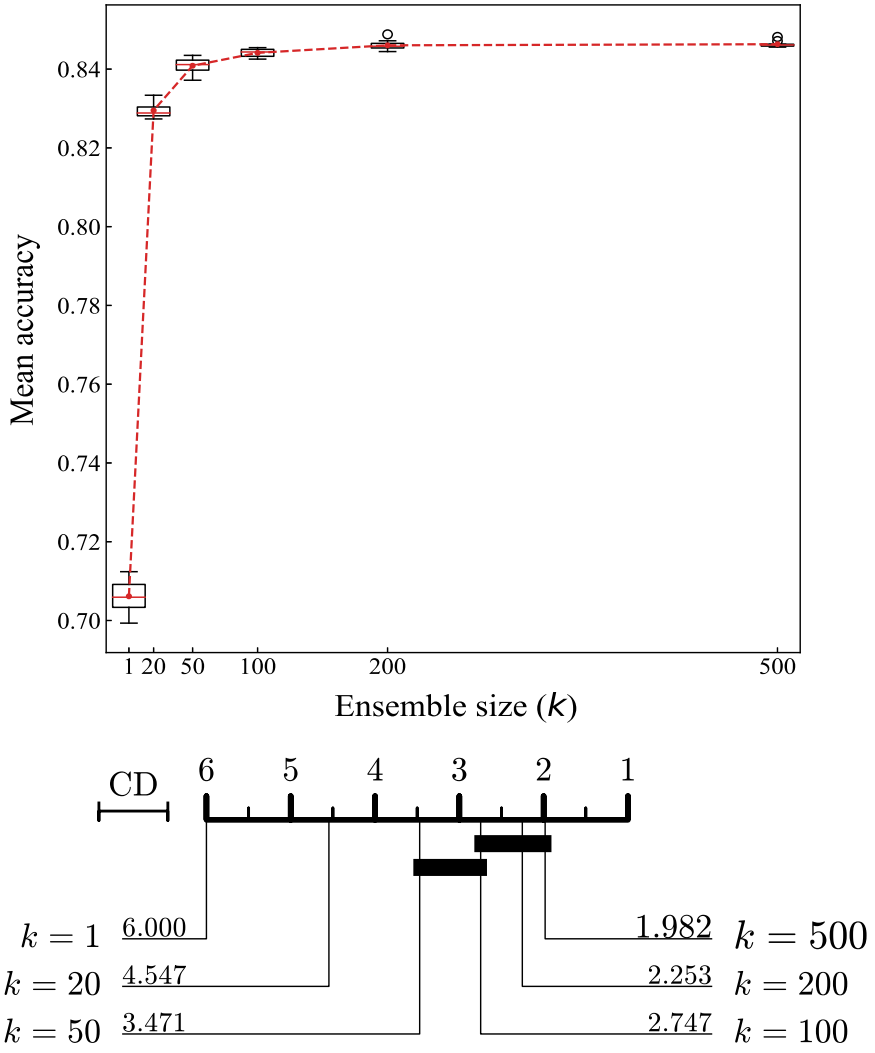


Fig. 6 Mean accuracy (and variance) *versus* ensemble size (top) and a critical difference diagram showing the mean rankings of different ensemble sizes (bottom). Mean accuracy is calculated over 85 datasets for 10 runs

plateaus. This follows ensemble theory which shows that increasing the size of the ensemble reduces the variance, but that at some point this variance is compensated by the covariance of the elements of the ensemble: when they all start resembling each other, no additional reduction of the variance of the error is obtained (Ueda and Nakano 1996; Breiman 2001). Our experiments show that using $k = 500$ is significantly better than using $k = 100$ (p-value is < 0.001 in a pairwise comparison after Holm's correction) but that the magnitude of the difference is very small. Importantly, however, it shows that, when going from 100 to 500, there is a substantial reduction in the variance in the accuracy between runs. In consequence, we make 500 trees the default, as it provides a good trade-off between accuracy and running times.

4.4 Contribution of splitting functions

We also conducted ablation experiments to assess the contribution of each type of splitting function: similarity-based, dictionary-based and interval-based. For this pur-

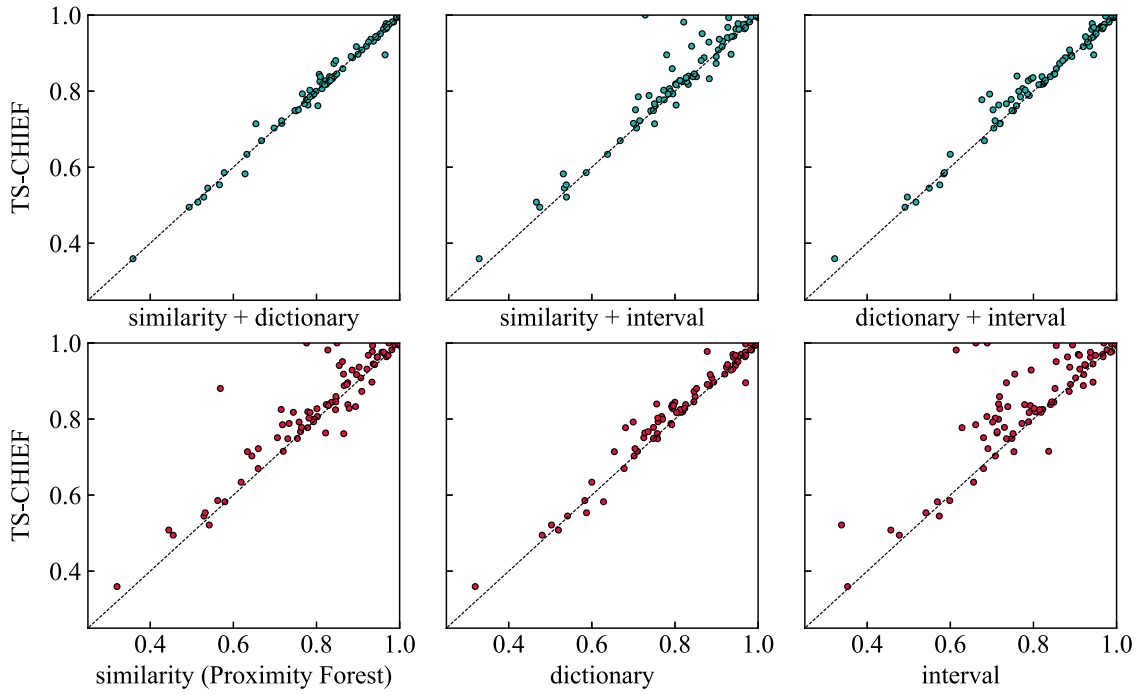


Fig. 7 Pairwise comparison of accuracy with one (bottom row) or two (top row) types of split functions *versus* TS-CHIEF (where all three types of split functions were used). Similarity *versus* TS-CHIEF (bottom-left) shows the pairwise comparison of Proximity Forest against TS-CHIEF

pose, we assess each variant of TS-CHIEF created by disabling one of the functions or a pair of the functions. We performed these experiments with 100 trees, and report the mean accuracy of 10 repetitions.

Figure 7 displays six scatter-plots comparing the accuracy of TS-CHIEF using all splitting functions to that of the six ablation configurations. The vertical axes indicate the accuracy of TS-CHIEF with all split functions enabled. The first row compares TS-CHIEF to variants with a single splitting function disabled (i.e with two types of split functions only). The second row compares TS-CHIEF to variants with only a single splitting function enabled. Please note that the use of only the similarity-based splitting function (first column, second row) corresponds to the Proximity Forest algorithm (Lucas et al. 2019). Each point indicates one of the 85 UCR datasets. Points above the diagonal dashed line indicate that TS-CHIEF with all three splitting functions has higher accuracy than the alternative.

The scatter plots on the bottom row indicate that, individually, the dictionary-based splitter contributes most to the accuracy with 18 wins, 59 losses and 8 ties relative to TS-CHIEF. We can also observe that the magnitudes of its losses tend to be smaller. Conversely, the interval-based splitter contributes least to the accuracy, with losses of the greatest magnitude relative to TS-CHIEF. However, it still achieves lower error on 17 datasets, demonstrating that there are some datasets for which the interval-based approach performs well.

When comparing similarity-based splitter (Proximity Forest) against TS-CHIEF ($k = 100$), the win/draw/loss is 67/2/16 in favor of TS-CHIEF. There are 5 datasets for which the wins are larger than 10%: Wine (31%), ShapeletSim (22%), OSULeaf

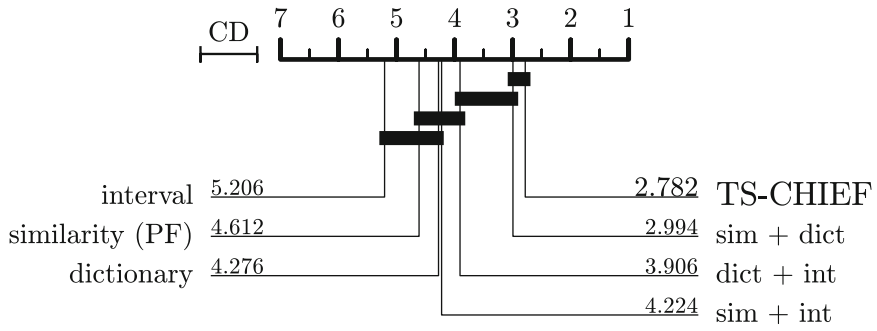


Fig. 8 Critical difference diagram showing the mean ranks of different combinations of split functions

(15%), ECGFiveDays (15%) and FordB (11%). When TS-CHIEF lost, the biggest three losses were for Lighting2 (10%) Lightning7 (6%) and FaceAll (5%).

In addition, the similarity-based splitter in conjunction with the dictionary-based splitter (that is, the variant with interval-based disabled) is closest to the accuracy of TS-CHIEF, with 26 wins against TS-CHIEF, 42 losses and 8 ties.

Figure 8 shows a critical difference diagram summarizing the the relative accuracy of all combinations of the splitting functions. This confirms our observations from the graphs in Fig. 7. The combination of all three types of splitters has the highest average rank. Next come the pairs of splitters, with all pairs outranking the single splitters, albeit marginally for the pair that excludes the dictionary splitter.

The contribution to accuracy from the interval-based splitter is small, and the sim+dict combination is not statistically different from TS-CHIEF (p-value is 0.777 in a pairwise comparison after Holm's correction) which uses the three splitters. There are three main reasons why we decided to keep the interval-based splitter in our method. (1) It ranks slightly higher than using only two. (2) It provides a different type of representation which we believe could be useful in real-world applications; in other words, we are conscious that there is a bias in the datasets of the UCR archive and want to prepare our method for unseen datasets as well. (3) Fig. 9, which displays the fraction of time used by each splitting function, shows that the interval-based splitter takes only a small fraction of the time in TS-CHIEF, so that the downsides of including it are small.

To analyze further, Fig. 10 displays the percentage of times each splitter type was selected at a node. We observe that the dictionary-based splitter ($C_b = 100$) is selected more often than the other two types of splitters, with an average of 60% of the time, across the 85 datasets. We used $C_e = 5$ for similarity-based splitters, but we also observe that similarity-based splitters were selected 30% of the time, whereas, an interval-based splitter ($C_r = 100$) was selected only 10% of the time. It is interesting that, despite that a dictionary-splitter was selected more often, it uses less time (15%) than the similarity-based splitter (80%)—this can be seen from Fig. 9.

4.5 Memory usage

In Sect. 3.4 we saw that the memory complexity of TS-CHIEF is $O(n \cdot \ell + k \cdot n \cdot c + t \cdot n \cdot \ell)$. Recall that t is the number of BOSS transformations precomputed at

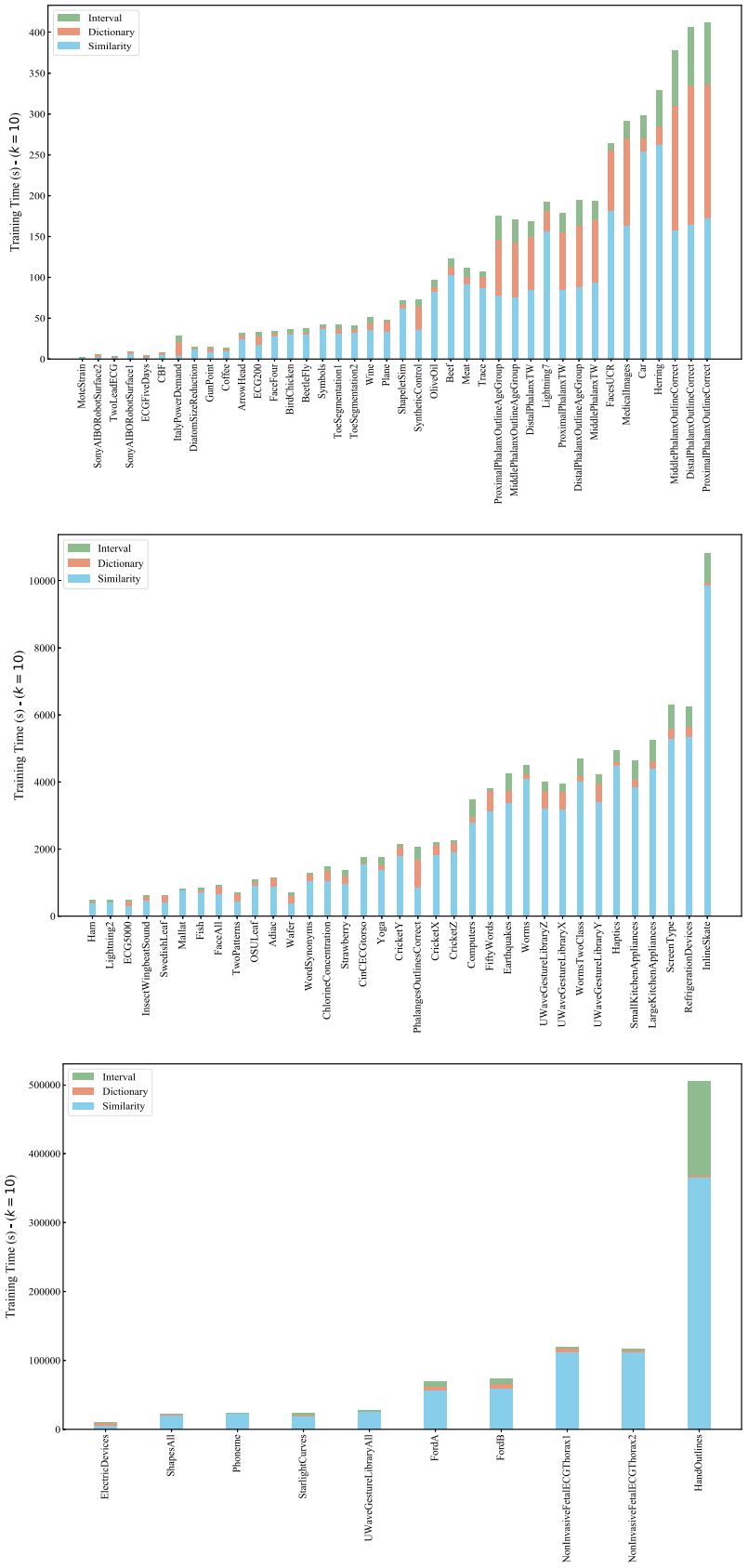


Fig. 9 Fraction of training time taken for each splitter type for 85 UCR datasets (Chen et al. 2015). In this experiment, we selected the hyperparameters as follows: number of similarity-based splitters $C_e = 5$, number of dictionary-based splitters $C_b = 100$, and the number of interval-based splitters $C_r = 100$. We ran this experiment with $k = 10$ trees to evaluate the fraction of training time used by each splitter type

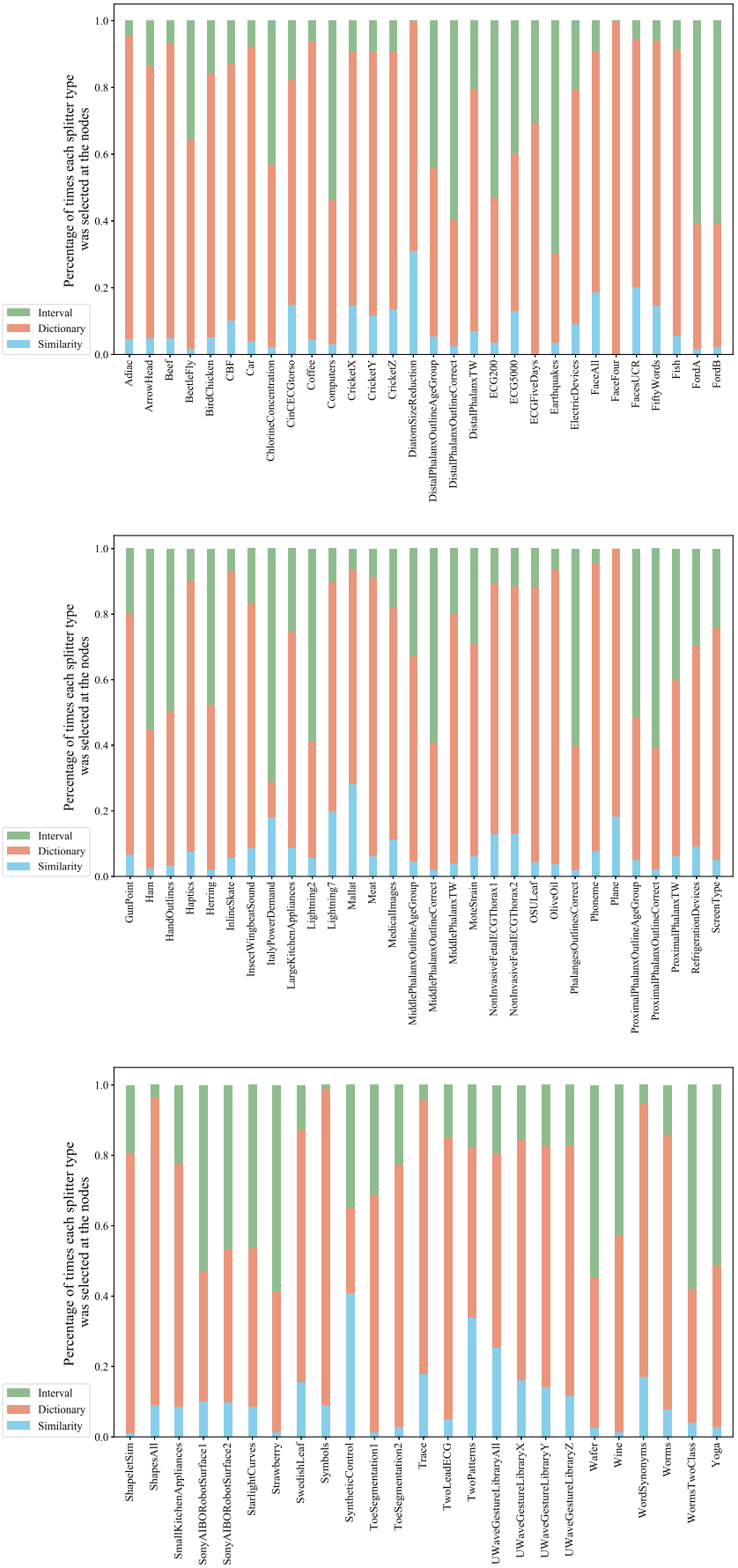


Fig. 10 Percentage of times each splitter type was selected at the nodes

the beginning of training. There is a memory *vs* computational time tradeoff between precomputing t BOSS transformations at the forest level and computing a random BOSS transformation at the tree or node level. To measure the actual memory usage due to the storage of BOSS transformations, we conducted an experiment using $k = 1$ on the longest UCR dataset *HandOutlines* ($n = 1000$, $\ell = 2700$) and on 131 k instances (same amount used in Fig. 3) of SITS dataset (see Sect. 4.2). We found that the *HandOutlines* uses 36.9 GB and SITS uses 49.8 GB of memory. Thus, our decision to precompute BOSS transformations at forest level is due to the following reasons: (1) memory usage is reasonable compared to the computational overhead of transforming at tree or node level, (2) a pool of transformations at the forest level will allow any tree to select any of the t transformations, which helps to improve diversity of the ensemble, whereas, if using, for example, one random BOSS transformation per tree, each tree is restricted to learn from one (or a less diverse pool, if using more than one) transformation.

5 Conclusions

We have introduced TS-CHIEF, which is a scalable and highly accurate algorithm for TSC. We have shown that TS-CHIEF makes the most of the quasi-linear scalability of trees relative to quantity of data, together with the last decade of research into deriving accurate representations of time series. Our experiments carried out on 85 datasets show that our algorithm reaches state-of-the-art accuracy that rivals HIVE-COTE, an algorithm which cannot be used in many applications because of its computational complexity.

We showed that on an application for land-cover mapping, TS-CHIEF is able to learn a model from 130,000 time series in 2 days, whereas it takes HIVE-COTE 8 days to learn from only 1500 time series—a quantity of data from which TS-CHIEF learns in 13 min. TS-CHIEF offers a general framework for time series classification. We believe that researchers will find it easy to integrate novel transformations and similarity measures and apply them at scale.

We conclude by highlighting possible improvements. This includes improving the tradeoff between computation time and memory footprint, incorporating information from different types of potential splitters, as well as finding an automatic way to balance the number of candidate splitters considered for each type (possibly in a manner that is adaptive to the dataset). Furthermore, future research on TS-CHIEF could extend it to multivariate time series and datasets with variable-length time series.

Supplementary material

To ensure reproducibility, a multi-threaded version of this algorithm implemented in Java and the experimental results have been made available in the github repository https://github.com/dotnet54/TS-CHIEF.

Acknowledgements This research was supported by the Australian Research Council under grant DE170100037. This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-17-1-4036. The authors would like to thank Prof. Eamonn Keogh and all the people who have contributed to the UCR time series classification archive. We also would like to acknowledge the use of source code freely available at http://www.timeseriesclassification.com and thank Prof. Anthony Bagnall and other contributors of the project. We also acknowledge the use of source code freely provided by the original author of BOSS algorithm, Dr. Patrick Schäfer. Finally, we acknowledge the use of two Java libraries (Osinski and Weiss 2015; Friedman and Eden 2013), which was used to optimize the implementation of our source code.

Appendix

See Tables 3 and 4.

Table 3 Complexities of the methods mentioned in Sect. 2

Method	Train complexity	Test complexity	Comments
2.1 Similarity-based			
1-NN DTW (CV)	$O(n^2 \cdot \ell^3)$	$O(n \cdot \ell \cdot w)$	Bagnall et al. (2017), CV: cross-validating all window sizes without using lower bounds
EE	$O(n^2 \cdot \ell^2)$	$O(n \cdot \ell^2)$	Lines and Bagnall (2015) and Bagnall et al. (2017, Table 1) (EE cross-validates 100 parameters)
PF	$O(k \cdot n \cdot log(n) \cdot C_e \cdot c \cdot \ell^2)$	$O(k \cdot log(n) \cdot c \cdot \ell^2)$	Lucas et al. (2019)
2.2 Interval-based			
RISE	$O(k \cdot n \cdot log(n) \cdot \ell^2)$	$O(k \cdot log(n) \cdot \ell^2)^{\#}$	Lines et al. (2018)
TSF	$O(k \cdot n \cdot log(n) \cdot \ell)$	$O(k \cdot log(n) \cdot \ell^2)^{\#}$	Bagnall et al. (2017, Table 1)
TSBF	$O(k \cdot n \cdot log(n) \cdot \ell \cdot R)$	*	Bagnall et al. (2017, Table 1)
LPS	$O(k \cdot n \cdot log(n) \cdot \ell \cdot R)$	*	Bagnall et al. (2017, Table 1)
2.3 Shapelet-based			
ST	$O(n^2 \cdot \ell^4)$	*	Hills et al. (2014). Uses a combination of 8 general purpose classifiers to classify
LS	$O(n^2 \cdot \ell^2 \cdot e \cdot \phi)$	*	Bagnall et al. (2017, Table 1)
FS	$O(n \cdot \ell^2)$	*	Rakthanmanon et al. (2013)
GRSF	$O(n^2 \cdot \ell^2 \cdot log(n\ell^2))$	*	Karlsson et al. (2016), amortized training time complexity

Table 3 continued

Method	Train complexity	Test complexity	Comments
2.4 Dictionary-based			
BOSS	$O(n^2 \cdot \ell^2)$	$O(n \cdot \ell)$	Schäfer (2015, Section 6)
BoP	$O(n \cdot \ell(n-w))$	*	Bagnall et al. (2017, Table 1)
SAX-VSM	$O(n \cdot \ell(n-w))$	*	Bagnall et al. (2017, Table 1)
BOSS-VS	$O(n \cdot \ell^{\frac{3}{2}})$	$O(n)$	Schäfer (2016, Table 1)
WEASEL	$O(min(n\ell^2, c^{(2f)} \cdot n))$	*	Schäfer and Leser (2017), high space complexity
2.5 Combinations of Ensembles			
FLAT-COTE	Bounded by ST	Bounded by EE	Bounded by the slowest algorithm
HIVE-COTE	Bounded by ST	Bounded by EE	Bounded by the slowest algorithm
2.6 Deep Learning			
FCN	*	*	
ResNet	*	*	

For tree-based methods, we present the average case complexity. Parameters used in this table are: n training size, ℓ series length, c no. classes, w window size, k number of trees, C_e no. candidate splits, e max. no. iterations, ϕ shapelet scale, f SFA word length, R no. of subseries
# Indicates that the information is not explicitly stated in the associated paper, but we derived the complexity based on our knowledge of the algorithm
*Indicates that the information is not explicitly stated in the associated paper

Table 4 Accuracy of leading TSC classifiers on 85 UCR datasets

Dataset	DTW	BOSS	ST	PF	RN	FCT	HCT	CHIEF
Adiac	60.87	76.47	78.26	73.40	**82.89**	79.03	81.07	79.80
ArrHead	80.00	83.43	73.71	**87.54**	84.46	81.14	86.29	83.27
Beef	66.67	80.00	90.00	72.00	75.33	86.67	**93.33**	70.61
BeetleFly	65.00	90.00	90.00	87.50	85.00	80.00	**95.00**	91.36
BirdChi	70.00	**95.00**	80.00	86.50	88.50	90.00	85.00	90.91
CBF	99.44	99.78	97.44	99.33	99.50	99.56	**99.89**	99.79
Car	76.67	83.33	91.67	84.67	**92.50**	90.00	86.67	85.45
ChConc	65.00	66.09	69.97	63.39	**84.36**	72.71	71.20	71.67
CinCECGT	93.04	88.70	95.43	93.43	82.61	99.49	**99.64**	98.32
Coffee	**100.0**	**100.0**	96.43	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Comp	62.40	75.60	73.60	64.44	**81.48**	74.00	76.00	70.51
CricketX	77.95	73.59	77.18	80.21	79.13	80.77	**82.31**	81.38
CricketY	75.64	75.38	77.95	79.38	80.33	82.56	**84.87**	80.19

Table 4 continued

Dataset	DTW	BOSS	ST	PF	RN	FCT	HCT	CHIEF
CricketZ	73.59	74.62	78.72	80.10	81.15	81.54	83.08	**83.40**
DiaSzRed	93.46	93.14	92.48	96.57	30.13	92.81	94.12	**97.30**
DiPhOAG	62.59	74.82	**76.98**	73.09	71.65	74.82	76.26	74.62
DiPhOC	72.46	72.83	77.54	**79.28**	77.10	76.09	77.17	78.23
DiPhTW	63.31	67.63	66.19	65.97	66.47	**69.78**	68.35	67.04
ECG200	88.00	87.00	83.00	**90.90**	87.40	88.00	85.00	86.18
ECG5000	92.51	94.13	94.38	93.65	93.42	94.60	**94.62**	94.54
ECG5D	79.67	**100.0**	98.37	84.92	97.48	99.88	**100.0**	**100.0**
Earthqua	72.66	74.82	74.10	**75.40**	71.15	74.82	74.82	74.82
ElectDev	63.08	**79.92**	74.70	70.60	72.91	71.33	77.03	75.53
FaceAll	80.77	78.17	77.87	89.38	83.88	**91.78**	80.30	84.14
FaceFour	89.77	**100.0**	85.23	97.39	95.45	89.77	95.45	**100.0**
FacesUCR	90.78	95.71	90.59	94.59	95.47	94.24	96.29	**96.63**
50Words	76.48	70.55	70.55	83.14	73.96	79.78	80.88	**84.50**
Fish	83.43	98.86	98.86	93.49	97.94	98.29	98.86	**99.43**
FordA	66.52	92.95	**97.12**	85.46	92.05	95.68	96.44	94.10
FordB	59.88	71.11	80.74	71.49	**91.31**	80.37	82.35	82.96
GunPoint	91.33	**100.0**	**100.0**	99.73	99.07	**100.0**	**100.0**	**100.0**
Ham	60.00	66.67	68.57	66.00	**75.71**	64.76	66.67	71.52
HandOut	87.84	90.27	**93.24**	92.14	91.11	91.89	**93.24**	93.22
Haptics	41.56	46.10	**52.27**	44.45	51.88	**52.27**	51.95	51.68
Herring	53.12	54.69	67.19	57.97	61.88	62.50	**68.75**	58.81
InlSkate	38.73	51.64	37.27	**54.18**	37.31	49.45	50.00	52.69
InWSnd	57.37	52.32	62.68	61.87	50.65	65.25	**65.51**	64.29
ItPwDem	95.53	90.86	94.75	96.71	96.30	96.11	96.31	**97.06**
LKitApp	79.47	76.53	85.87	78.19	**89.97**	84.53	86.40	80.68
Light2	**86.89**	83.61	73.77	86.56	77.05	**86.89**	81.97	74.81
Light7	71.23	68.49	72.60	82.19	**84.52**	80.82	73.97	76.34
Mallat	91.43	93.82	96.42	95.76	97.16	95.39	96.20	**97.50**
Meat	93.33	90.00	85.00	93.33	**96.83**	91.67	93.33	88.79
MdImg	74.74	71.84	66.97	75.82	77.03	75.79	77.76	**79.58**
MdPhOAG	51.95	54.55	**64.29**	56.23	56.88	63.64	59.74	58.32
MdPhOC	76.63	78.01	79.38	83.64	80.89	80.41	83.16	**85.35**
MdPhTW	50.65	54.55	51.95	52.92	48.44	**57.14**	**57.14**	55.02
MtStrain	86.58	87.86	89.70	90.24	92.76	93.69	93.29	**94.75**
NoECGT1	82.90	83.82	**94.96**	90.66	94.54	93.13	93.03	91.13
NoECGT2	87.02	90.08	**95.11**	93.99	94.61	94.55	94.45	94.50

Table 4 continued

Dataset	DTW	BOSS	ST	PF	RN	FCT	HCT	CHIEF
OSULeaf	59.92	95.45	96.69	82.73	97.85	96.69	97.93	**99.14**
OliveOil	86.67	86.67	**90.00**	86.67	83.00	**90.00**	**90.00**	88.79
PhalanOC	76.11	77.16	76.34	82.35	83.90	77.04	80.65	**84.50**
Phoneme	22.68	26.48	32.07	32.01	33.43	34.92	**38.24**	36.91
Plane	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
PrxPhOAG	78.54	83.41	84.39	84.63	85.32	85.37	**85.85**	84.97
PrxPhOC	79.04	84.88	88.32	87.32	**92.13**	86.94	87.97	88.82
PrxPhTW	76.10	80.00	80.49	77.90	78.05	78.05	81.46	**81.86**
RefDev	44.00	49.87	**58.13**	53.23	52.53	54.67	55.73	55.83
ScrType	41.07	46.40	52.00	45.52	**62.16**	54.67	58.93	50.81
ShpSim	69.44	**100.0**	95.56	77.61	77.94	96.11	**100.0**	**100.0**
ShpAll	80.17	90.83	84.17	88.58	92.13	89.17	90.50	**93.00**
SKitApp	67.20	72.53	79.20	74.43	78.61	77.60	**85.33**	82.21
SonyRS1	69.55	63.23	84.36	84.58	**95.81**	84.53	76.54	82.64
SonyRS2	85.94	85.94	93.39	89.63	**97.78**	95.17	92.76	92.48
StarCurv	89.83	97.78	97.85	98.13	97.18	97.96	98.15	**98.24**
Strwbe	94.59	97.57	96.22	96.84	**98.05**	95.14	97.03	96.63
SwdLeaf	84.64	92.16	92.80	94.66	95.63	95.52	95.36	**96.55**
Symbols	93.77	96.68	88.24	96.16	90.64	96.38	97.39	**97.66**
SynCtl	98.33	96.67	98.33	99.53	99.83	**100.0**	99.67	99.79
ToeSeg1	75.00	93.86	96.49	92.46	96.27	97.37	**98.25**	96.53
ToeSeg2	90.77	**96.15**	90.77	86.23	90.62	91.54	95.38	95.38
Trace	99.00	**100.0**	**100.00**	**100.0**	**100.00**	**100.0**	**100.0**	**100.0**
2LeadECG	86.83	98.07	99.74	98.86	**100.0**	99.30	99.65	99.46
2Pttrns	99.85	99.30	95.50	99.96	99.99	**100.0**	**100.0**	**100.0**
UWaAll	96.23	93.89	94.22	**97.23**	85.95	96.43	96.85	96.89
UWaX	77.44	76.21	80.29	82.86	78.05	82.19	83.98	**84.11**
UWaY	70.18	68.51	73.03	76.15	67.01	75.85	76.55	**77.23**
UWaZ	67.50	69.49	74.85	76.40	75.01	75.04	78.31	**78.44**
Wafer	99.59	99.48	**100.0**	99.55	99.86	99.98	99.94	99.91
Wine	61.11	74.07	79.63	56.85	74.44	64.81	77.78	**89.06**
WordSyn	74.92	63.79	57.05	77.87	62.24	75.71	73.82	**78.74**
Worms	53.25	55.84	74.03	71.82	79.09	62.34	55.84	**80.17**
Worms2C	58.44	**83.12**	**83.12**	78.44	74.68	80.52	77.92	81.58
Yoga	84.30	**91.83**	81.77	87.86	87.02	87.67	91.77	83.47
Avg.Rank	6.982	5.400	4.806	4.818	4.300	3.818	**2.941**	2.935
No. of times ranked 1	3	12	14	9	18	12	23	**31**

The classifiers are 1-Nearest Neighbour with DTW (labelled DTW), BOSS, PF (Proximity Forest), ST (Shapelet Transform), Residual Neural Network (RN), FLAT-COTE (FCT), HIVE-COTE (HCT), and TS-CHIEF (CHIEF). The last two rows show the number of wins (no. of times ranked at 1) and average ranking of accuracy (Refer to Fig. 1)
Bold values indicate classifiers with the highest mean accuracy for each dataset over 10 runs

References

Bagnall A, Davis L, Hills J, Lines J (2012) Transformation based ensembles for time series classification. In: Proceedings of the SIAM international conference on data mining, pp 307–318
Bagnall A, Lines J, Hills J, Bostrom A (2015) Time-series classification with COTE: the collective of transformation-based ensembles. IEEE Trans Knowl Data Eng 27(9):2522–2535
Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Discov 31(3):606–660
Baydogan MG, Runger G (2016) Time series representation and similarity based on local autopatterns. Data Min Knowl Discov 30(2):476–509
Baydogan MG, Runger G, Tuv E (2013) A bag-of-features framework to classify time series. IEEE Trans Pattern Anal Mach Intell 35(11):2796–2802
Benavoli A, Corani G, Mangili F (2016) Should we really use post-hoc tests based on mean-ranks? J Mach Learn Res 17(1):152–161
Bostrom A, Bagnall A (2015) Binary shapelet transform for multiclass time series classification. In: International conference on big data analytics and knowledge discovery, pp 257–269. Springer
Breiman L (2001) Random forests. Mach Learn 45(1):5–32 ISSN 08856125
Chen L, Ng R (2004) On The marriage of lp-norms and edit distance. In: Proceedings of the 13th international conference on very large data bases (VLDB), pp. 792–803
Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2015) The UCR time series classification archive. www.cs.ucr.edu/~eamonn/time_series_data/
Dau HA, Bagnall A, Kamgar K, Yeh C-CM, Zhu Y, Gharghabi S, Ratanamahatana CA, Keogh E (2018a) The UCR time series archive. arXiv:1810.07758. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
Dau HA, Keogh E, Kamgar K, Yeh C-CM, Zhu Y, Gharghabi S, Ratanamahatana CA, Yanping, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2018b) The UCR time series classification archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Inform Sci 239:142–153
Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh EJ (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. Proc VLDB Endow 1(2):1542–1552
Esling P, Agon C (2012) Time-series data mining. ACM Comput Surv (CSUR) 45(1):12
Fawaz HI, Forestier G, Weber J, Idoumghar L, Muller PA (2019) Deep learning for time series classification: a review. Data Min and Knowl Discov 33(4):917–963
Friedman E, Eden R (2013) GNU Trove: high-performance collections library for Java. https://bitbucket.org/trove4j/trove/src/master/. Accessed 25 June 2019
Górecki T, Łuczak M (2013) Using derivatives in time series classification. Data Min Knowl Discov 26(2):310–331 ISSN 13845810
Grabocka J, Schilling N, Wistuba M, Schmidt-Thieme L (2014) Learning time-series shapelets. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining—KDD '14, pp 392–401
Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014) Classification of time series by shapelet transformation. Data Min Knowl Discov 28(4):851–881 ISSN 13845810
Hirschberg DS (1977) Algorithms for the longest common subsequence problem. J ACM 24(4):664–675
Jeong YS, Jeong MK, Omitaomu OA (2011) Weighted dynamic time warping for time series classification. Pattern Recognit 44(9):2231–2240
Karlsson I, Papapetrou P, Boström H (2016) Generalized random shapelet forests. Data Min Knowl Discov 30(5):1053–1085
Keogh EJ, Pazzani MJ (2001) Derivative dynamic time warping. In: Proceedings of the 2001 SIAM international conference on data mining, pp 1–11
Keogh E, Kasetty S (2003) On the need for time series data mining benchmarks: a survey and empirical demonstration. Data Min Knowl Discov 7(4):349–371
Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001) Locally adaptive dimensionality reduction for indexing large time series databases. ACM Sigmod Record 30(2):151–162

Large J, Lines J, Bagnall A (2017) The heterogeneous ensembles of standard classification algorithms (HESCA): the whole is greater than the sum of its parts, pp 1–31. arXiv:1710.09220
Large J, Bagnall A, Malinowski S, Tavenard R (2018) From BOP to BOSS and beyond: time series classification with dictionary based classifiers, pp 1–22. arXiv:1809.06751
Le Guennec A, Malinowski S, Tavenard R (2016) Data augmentation for time series classification using convolutional neural networks. In: ECML/PKDD workshop on advanced analytics and learning on temporal data
Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing SAX: a novel symbolic representation of time series. Data Min Knowl Discov 15(2):107–144 ISSN 13845810
Lin J, Khade R, Li Y (2012) Rotation-invariant similarity in time series using bag-of-patterns representation. J Intell Inf Syst 39(2):287–315
Lines J, Bagnall A (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Discov 29(3):565–592 ISSN 13845810
Lines J, Taylor S, Bagnall A (2018) Time series classification with hive-cote: the hierarchical vote collective of transformation-based ensembles. ACM Trans Knowl Discov Data (TKDD) 12(5):52
Lucas B, Shifaz A, Pelletier C, O'Neill L, Zaidi N, Goethals B, Petitjean F, Webb GI (2019) Proximity Forest: an effective and scalable distance-based classifier for time series. Data Min Knowl Discov 33(3):607–635
Marteau P-F (2009) Time warp edit distance with stiffness adjustment for time series matching. IEEE Trans Pattern Anal Mach Intell 31(2):306–318
Middlehurst M, Vickers W, Bagnall A (2019) Scalable dictionary classifiers for time series classification. arXiv:1907.11815
Mueen A, Keogh E, Young N (2011) Logical-shapelets: an expressive primitive for time series classification. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining—KDD '11, p 1154
Nwe TL, Dat TH, Ma B (2017) Convolutional neural network with multi-task learning scheme for acoustic scene classification. In: 2017 Asia-pacific signal and information processing association annual summit and conference (APSIPA ASC), pp 1347–1350. IEEE
Nweke HF, Teh YW, Al-Garadi MA, Alo UR (2018) Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: state of the art and research challenges. Expert Syst Appl 105:233–261
Osinski S, Weiss D (2015) HPPC: High performance primitive collections for Java. https://labs.carrotsearch.com/hppc.html. Accessed 25 June 2019
Pelletier C, Webb GI, Petitjean F (2019) Temporal convolutional neural network for the classification of satellite image time series. Remote Sens 11(5):523
Rajkomar A, Oren E, Chen K, Dai AM, Hajaj N, Hardt M, Liu PJ, Liu X, Marcus J, Sun M et al (2018) Scalable and accurate deep learning with electronic health records. NPJ Digit Med 1(1):18
Rakthanmanon T, Keogh E (2013) Fast shapelets: a scalable algorithm for discovering time series shapelets. In: Proceedings of the 2013 SIAM international conference on data mining, pp 668–676
Rakthanmanon T, Campana B, Mueen A, Batista G, Westover B, Zhu Q, Zakaria J, Keogh E (2013) Addressing big data time series: mining trillions of time series subsequences under dynamic time warping. ACM Trans Knowl Discov Data (TKDD) 7(3):10
Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Min Knowl Discov 29(6):1505–1530
Schäfer P (2016) Scalable time series classification. Data Min Knowl Discov 30(5):1273–1298 ISSN 1573756X
Schäfer P, Högqvist M (2012) SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In: Proceedings of the 15th international conference on extending database technology, pp 516–527
Schäfer P, Leser U (2017) Fast and accurate time series classification with WEASEL. In: Proceedings of the 2017 ACM on conference on information and knowledge management (CIKM), pp 637–646. ISBN 9781450349185
Senin P, Malinchik S (2013) SAX-VSM: interpretable time series classification using SAX and vector space model. In: Proceedings of IEEE international conference on data mining, ICDM, pp 1175–1180, ISSN 15504786
Silva DF, Giusti R, Keogh E, Batista GE (2018) Speeding up similarity search under dynamic time warping by pruning unpromising alignments. Data Min Knowl Discov 32(4):988–1016

Stefan A, Athitsos V, Das G (2013) The move-split-merge metric for time series. IEEE Trans Knowl Data Eng 25(6):1425–1438 ISSN 10414347
Susto GA, Cenedese A, Terzi M (2018) Time-series classification methods: review and applications to power systems data. In: Big data application in power systems, pp 179–220 Elsevier
Tan CW, Webb GI, Petitjean F (2017) Indexing and classifying gigabytes of time series under time warping. In: Proceedings of the 2017 SIAM international conference on data mining, pp 282–290. SIAM
Ueda N, Nakano R (1996) Generalization error of ensemble estimators. In: IEEE international conference on neural networks, volume 1, pp 90–95. IEEE
Wang Z, Yan W, Oates T (2017) Time series classification from scratch with deep neural networks: A strong baseline. In: 2017 international joint conference on neural networks (IJCNN), pp 1578–1585. IEEE
Wang J, Liu P, She MF, Nahavandi S, Kouzani A (2013) Bag-of-words representation for biomedical time series classification. Biomed Signal Process Control 8(6):634–644
Wang J, Chen Y, Hao S, Peng X, Hu L (2019) Deep learning for sensor-based activity recognition: a survey. Pattern Recognit Lett 119:3–11
Yang Q, Wu X (2006) 10 challenging problems in data mining research. Int J Inf Technol Decis Mak 5(04):597–604
Ye L, Keogh E (2009) Time series shapelets. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining—KDD '09, p 947

Affiliations

Ahmed Shifaz[1] · Charlotte Pelletier[1,2] · François Petitjean[1] · Geoffrey I. Webb[1]

B Ahmed Shifaz
ahmed.shifaz@monash.edu
http://www.ahmedshifaz.com/

Charlotte Pelletier
charlotte.pelletier@univ-ubs.fr
https://sites.google.com/site/charpelletier

François Petitjean
francois.petitjean@monash.edu
http://www.francois-petitjean.com/

Geoffrey I. Webb
geoff.webb@monash.edu
http://i.giwebb.com/

1 Faculty of Information Technology, Monash University, 25 Exhibition Walk, 3800 Melbourne, VIC, Australia

2 IRISA, UMR CNRS 6074, University of Bretagne Sud, Campus de Tohannic, BP 573, 56 000 Vannes, France

4.2 Contributions

Contributions made in this chapter are:

1. TS-CHIEF: a novel forest algorithm for time series classification that leverages decades of research on developing various techniques for time series classification, and combines them in a unique way inside nodes of purpose built decision trees.

2. TS-CHIEF achieves state-of-the-art accuracy that rivals HIVE-COTE (alpha) within a fraction of its runtime. At the time of publication, TS-CHIEF and HIVE-COTE were considered as the two most accurate TSC algorithms in the field. Since publication, the main results of TS-CHIEF have been independently replicated [38], and TS-CHIEF is still independently assessed as one of the four state-of-the-art TSC algorithms [7, 94].

3. TS-CHIEF was the first algorithm that is able to learn a model from 130,000 SITS time series in 2 days with the demonstrated level of accuracy, whereas it takes HIVE-COTE (alpha) 8 days to learn from only 1500 SITS time series—a quantity of data from which TS-CHIEF learns in 13 minutes.

4. TS-CHIEF offers a general framework to combine multiple TSC techniques into one tree-based algorithm. It uses a simple, yet flexible, method to evaluate multiple candidate splits using Gini Index that was introduced in Proximity Forest in chapter 3. TS-CHIEF demonstrates that this technique can be used effectively to combine various transformations or representations of data at the node level of trees to select appropriate partitioning functions. As new techniques are developed for TSC, new types of splitters can be added to this general framework to extend TS-CHIEF further. This strategy to combine various data representation at the node level is very different and unique compared to the combination method used in HIVE-COTE, which simply combines independent algorithms using a voting technique.

Chapter 5

Multivariate Elastic Similarity Measures

5.1 Introduction

In this chapter I explore multivariate time series classification using elastic similarity measures. I present a paper that extends to multivariate time series the univariate similarity measures used in Elastic Ensemble and Proximity Forest. The presented paper is currently under review for a journal publication.

I include the submitted version of the paper without textual modifications except the addition of Section 4.3.2 and 4.3.5 for some further clarifications. I reiterate the contributions of this paper at the end of the chapter in Section 5.2.

- **Based on a submitted journal paper:** Shifaz, A., Pelletier, C., Petitjean, F. and Webb, G.I., 2021. Elastic Similarity Measures for Multivariate Time Series Classification. arXiv preprint arXiv:2102.10231.

Noname manuscript No.
(will be inserted by the editor)

Elastic Similarity Measures for Multivariate Time Series

Ahmed Shifaz[1] · Charlotte Pelletier[1,2] · François Petitjean[1] · Geoffrey I. Webb[1]

Received: date / Accepted: date

Abstract This paper contributes multivariate versions of seven commonly used elastic similarity measures for time series data analytics. Elastic similarity measures are a class of similarity measures that can compensate for misalignments in the time axis of time series data. We adapt two existing strategies used in a multivariate version of the well-known Dynamic Time Warping (DTW), namely, Independent and Dependent DTW, to these seven measures.

While these measures can be applied to various time series analysis tasks, we demonstrate their utility on multivariate time series classification using the nearest neighbor classifier. On 23 well-known datasets, we demonstrate that each measure achieves the highest accuracy relative to others on at least one dataset, establishing the importance of developing a suite of multivariate similarity measures. We also demonstrate that there are datasets for which either the dependent versions of all measures are more accurate than their independent counterparts or vice versa.

Keywords multivariate similarity measures, multivariate time series, multivariate time series classification, elastic similarity measures, dynamic time warping, independent measures, dependent measures,

This research has been supported by Australian Research Council grant DP210100072.

[1]Department of Data Science and Artifical Intelligence
Monash University, Melbourne
VIC 3800, Australia
[2] IRISA, UMR CNRS 6074
Univ. Bretagne Sud
Campus de Tohannic
BP 573, 56 000 Vannes, France
E-mail: {ahmed.shifaz,francois.petitjean,geo□.webb}@monash.edu,
charlotte.pelletier@univ-ubs.fr

1 Introduction

Elastic similarity measures such as the well known Dynamic Time Warping (DTW) [1] are a key tool in many forms of time series analytics. Elastic similarity measures can align temporal misalignments between two series while computing the similarity between them. Examples of their application include clustering [2, 3, 4], classification [5, 6], anomaly detection [7, 8], indexing [9], subsequence search [10] and segmentation [11].

While numerous elastic similarity measures have been developed [6, 12, 13], most of these measures have been defined only for univariate time series. Since many time series analysis tasks involve multivariate time series, it is crucial to extend these elastic measures to the multivariate case.

This paper extends seven univariate elastic similarity measures to the multivariate case. This extension is sometimes non-trivial. We demonstrate that each measure provides more accurate nearest neighbor classification than any alternative for at least one dataset. This demonstrates the importance of having a range of different elastic measures available for multivariate time series.

One elastic measure that has previously been extended to the multivariate case is DTW [14]. That work identified two key strategies for such extension. The *independent* strategy applies the univariate measure to each dimension and then sums the resulting distances. The *dependent* strategy treats the multivariate series as a single series in which each time step has a single multidimensional point. DTW is then applied using Euclidean distances between the multidimensional points of the two series. It was shown that each of these strategies outperformed the other on some tasks.

We develop methods for applying these two strategies to seven further key univariate similarity measures. One of the significant outcomes is that we demonstrate that there are some tasks for which the independent strategy is superior across all measures and others for which the dependent strategy is better. This establishes a fundamental relationship between the two strategies and different tasks, countering the possibility that differing performance for the two strategies when applied to DTW might have been coincidental.

To give an example of when these two strategies could be useful, consider two real-world example datasets from the UEA multivariate time series archive [15]: *Handwriting* records movements of a hand writing letters by recording X, Y and Z coordinates from an accelerometer. *PEMS-SF* records traffic congestion for highways using multiple sensors at different locations. In the first task it is necessary to consider movement in each dimension simultaneously. For example, letter 'N' is indicated by negligible X coupled with positive Y, followed by positive X coupled with negative Y, followed by negligible X coupled with positive Y. In the second dataset, each sensor might be expected to contribute to the global assessment of congestion without strong interdependencies. This is exactly what we observe in our empirical study presented in Section 5.

We choose these seven specific measures in this study because our previous research is mainly focused on the classification task and these measures have

been used in many well known univariate similarity-based classifiers such as Elastic Ensemble [5], Proximity Forest [16], and two state-of-the-art univariate time series classifier HIVE-COTEv1 [17] and TS-CHIEF [18].

The major contributions of this paper can be summarized as follows:

- We extend seven commonly used univariate elastic similarity measures to the multivariate case by adopting two strategies — using multiple dimensions independently and dependently. Univariate versions of these seven measures have been widely used in time series analysis tasks such as classification, clustering, indexing, segmentation, anomaly detection and subsequence search. Therefore, we hope that our multivariate versions would be of great benefit to the time series analysis community. This extension is straightforward in several cases, but non-trivial for some cases.
- We demonstrate the utility of these measures by focusing on the multivariate nearest neighbor classification problem. We compare the classification performance of these measures on 23 datasets from the University of East Anglia (UEA) multivariate archive. We show that each measure outperforms all others on at least one task (or ties with the highest performing measures in two cases), demonstrating the value of having a suite of alternatives multivariate elastic measures.
- We demonstrate that there are some classification tasks for which the independent strategy is superior across all measures and others for which the dependent strategy is better. It was previously demonstrated for DTW that each approach sometimes substantially outperformed the other [14]. However, it has not previously been established whether this is a fundamental property of the tasks per se. In this paper we provide evidence that in at least some cases there is a fundamental connection between classification tasks and whether they are best addressed by considering all variables at each time step in conjunction or by considering each variable independently of the others.

We organize the rest of the paper as follows. Section 2 presents key definitions. Section 3 provides a brief review of existing methods. Section 4 describes our new multivariate similarity measures. Section 5 presents multivariate time series classification experiments on the UEA multivariate time series archive, and includes discussion of the implications of the results. Finally, we draw conclusions in Section 6, with suggestions for future work.

2 Definitions

We here present key notation and definitions.

A *time series* T of length L is an ordered sequence of L time-value pairs $T = (t_1, \mathbf{x}_1), \cdots, (t_L, \mathbf{x}_L)$, where t_i is the timestamp at sequence index i, $i \in \{1, \cdots, L\}$, and $\mathbf{x}_i$ is a D-dimensional point representing observations of D real-valued variables or features at timestamp t_i. Each time point $\mathbf{x}_i \in \mathbb{R}^D$ is defined by $\{x_i^1, \cdots, x_i^d, \cdots, x_i^D\}$. Usually, timestamps t_i are assumed to

be equidistant, and thus omitted, which results in a simpler representation where $T = x_1, \cdots, x_L$.

A *univariate* (or single-dimensional) time series is a special case where a single variable is observed ($D = 1$). Therefore, x_i is a scalar, and consequently, $T = x_1, \cdots, x_L$.

A *labeled time series dataset* S consists of N labeled time series indexed by n, where $n \in \{1, \cdots, N\}$. Each time series T_n in S is associated with a label $y_n \in \{1, \cdots, c\}$, where c is the number of classes.

A *similarity measure* computes a real value that quantifies similarity between two sets of values. For time series Q and C, similarity measure M is defined as

$$M(Q, C) \rightarrow \mathbb{R} \tag{1}$$

A similarity measure M is a *metric* if it has the following properties:

1. Non-negativity: $M(Q, C) \geq 0$,
2. Identity: $M(Q, C) = 0$, if and only if $Q = C$,
3. Symmetry: $D(Q, C) = D(C, Q)$,
4. Triangle Inequality: $D(Q, C) \leq D(Q, T) + D(T, C)$ for any time series Q, C and T.

All similarity measures considered in this work satisfy 1, 2 and 3.

In a Time Series Classification (TSC) task, a time series classifier is trained on a labeled time series dataset, and then used to predict labels of unlabeled time series. The classifier is a predictive mapping function that maps from the space of input variables to discrete class labels.

In this paper, to perform TSC tasks, we use 1-nearest neighbor (1-NN) classifiers, which use time series specific similarity measures to compute the nearest neighbors between each time series.

3 Time Series Classification

3.1 Univariate TSC

A comprehensive review of the most common univariate TSC methods developed prior to 2017 can be found in [6]. Here we summarize key univariate TSC methods using a traditionally used method of categorization as follows:

– *Similarity-based* methods compare whole time series using similarity measures, usually in conjunction with 1-NN classifiers. Particularly, 1-NN with DTW [1, 19] was long considered as the de facto standard for univariate TSC. More accurate similarity-based methods combine multiple measures, including 1-NN-based ensemble Elastic Ensemble (EE) [5], and tree-based ensemble Proximity Forest (PF) [16]. In Section 4 we will explore more details of several similarity measures used in TSC.

- *Interval-based* methods use summary statistics relating to subseries in conjunction with location information as discriminatory features. Examples include Time Series Forest (TSF) [20], Random Interval Spectral Ensemble (RISE) [17], and Canonical Interval Forest (CIF) [21].
- *Shapelet-based* methods extract or learn a set of discriminative subseries for each class which are then used as search keys for the particular classes. The presence, absence or distance of a shapelet is used as discriminative information for classification. Examples include Shapelet Transform (ST) [22] and Generalized Random Shapelet Forest (gRSF) [23].
- *Dictionary-based* methods transform time series into a bag-of-word model. The series is either discretized in time domain such as in Bag of Patterns (BoP) [24] or it is transformed into the frequency domain such as in Bag-of-SFA-Symbols (BOSS) [25], and Word eXtrAction for time SEries cLassification (WEASEL) [26]. Multiple Representation Sequence Learner (MrSEQL) [27] is another recent dictionary-based classifier, which is more accurate than WEASEL but uses less computational resources.
- *Other Transformation-based* methods transform the time series using a transformation function and then use a general purpose classifier. A notable example is RandOm Convolutional KErnel Transform (ROCKET) [28] which uses random convolutions to transform the data, and then uses logistic regression for classification.
- *Deep-learning* methods can be divided into two main types of architectures: (1) based on recurrent neural networks [29], or (2) based on temporal convolutions, such as Residual Neural Network (ResNet) [30] and InceptionTime [31]. A recent review of deep learning methods shows that architectures that use temporal convolutions show higher accuracy [32].
- *Combinations of Methods* combine multiple methods to form ensembles. Examples include HIVE-COTE (Hierarchical Vote Collective of Transformation-based Ensembles) [17], which ensembles EE, ST, RISE and BOSS, and TS-CHIEF (Time Series Combination of Heterogeneous and Integrated Embeddings Forest) [18], which is a tree-based ensemble where the tree nodes use similarity, dictionary or interval-based splitters.

Currently, HIVE-COTE, TS-CHIEF and ROCKET are considered to be state-of-the-art classifiers for TSC [33]. While all three methods are competitive in accuracy (*i.e.* statistically indistinguishable), TS-CHIEF leads in terms of accuracy and ROCKET leads in terms of speed.

We also note that the latest version of HIVE-COTE, which does not include EE, called HIVE-COTE 1.0, has significantly improved its speed, but is still behind on accuracy with respect to ROCKET and TS-CHIEF [33].

3.2 Multivariate TSC

Research into multivariate TSC has lagged behind univariate research. A recent paper [34] reviews several methods used for multivariate TSC and com-

pares their performance on the UEA multivariate time series archive [15]. Here, we present a short summary of multivariate methods:

– *Similarity-based* Multivariate similarity measures can be used with 1-nearest neighbor for classification. DTW has previously been extended to the multivariate case using two key strategies [14]. The *independent* strategy applies the univariate measure to each dimension and then sums the resulting distances. The *dependent* strategy treats each time step as a multi-dimensional point. DTW is then applied on the Euclidean distances between these multidimensional points. Figure 1 illustrates these approaches and we present definitions in Section 4.

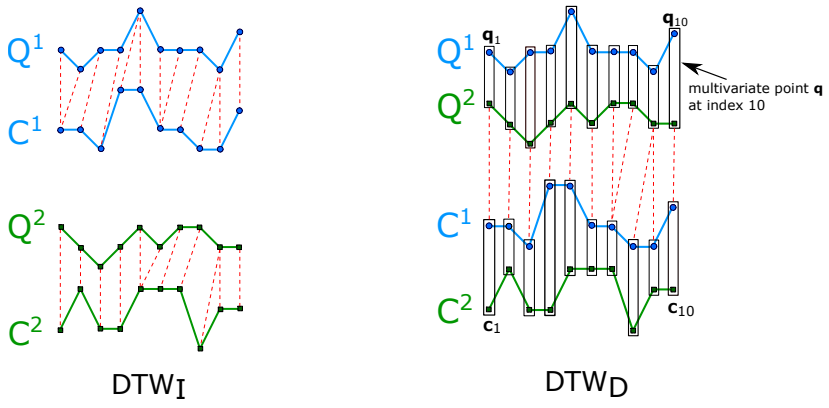


Fig. 1 Independent DTW (DTW_I, left) and dependent DTW (DTW_D, right). Dimension 1 in series Q and C is shown in blue, and the dimension 2 is shown in green.

– *Interval-based* methods include RISE [17], TSF [20], and the recently introduced CIF [21]. They extract intervals from each dimension separately. CIF has shown promising results for multivariate classification [34].
– *Shapelet-based* methods include gRSF [23] and time contracted Shapelet Transform (STC) [33]. According to a the review [34], STC is the current most accurate multivariate method that uses shapelets (which is ranked below ResNet) [34].
– *Dictionary-based* methods include WEASEL with a Multivariate Unsupervised Symbols and dErivatives (MUSE) (a.k.a WEASEL+MUSE, or simply MUSE) [35] and time contracted Bag-of-SFA-Symbols (CBOSS) [36].
– *Other Transformation-based* methods include an extension of ROCKET to the multivariate case, implemented in the *sktime* library [37]. It combines information from multiple dimensions using small subsets of dimensions.
– *Combinations of Methods* include a multivariate version of HIVE-COTE (v1.0) which combines STC, TSF, CBOSS, and RISE [33], applying each constituent algorithm to each dimension separately.
– *Deep-learning-based* methods that directly support multivariate series include Time Series Attentional Prototype Network (TapNet) [38], ResNet and InceptionTime.

To give an overview of the review, Ruiz et. al. compared 12 classifiers on 20 UEA multivariate datasets with equal length that completed in a reasonable time. They found that the most accurate multivariate TSC algorithms are ROCKET, InceptionTime, MUSE, CIF, HIVE-COTE and MrSEQL in that order [34, Figure 12a].

4 Similarity Measures

In this section we present the proposed similarity measures. For this study, we extend to the multivariate case the set of univariate similarity measures used in EE and PF (and thus TS-CHIEF and some versions of HIVE-COTE).

The independent strategy proposed by [14] simply sums over the results of applying DTW separately to each dimension. For completeness we propose to extend this idea to allowing any p-norm. In this case, the previous approach extends directly to any univariate measure as follows.

Definition 1 Independent Measures

For any univariate measure $\phi(Q^1, C^1) \rightarrow \mathsf{R}$ and multivariate series Q and C, an independent multivariate extension of ϕ is defined by,

$$Ind(\phi, Q, C, p) = \left(\sum_{d=1}^{D} \left| \phi(Q^d, C^d) \right|^p \right)^{1/p} \tag{2}$$

We compute the distance between Q and C separately for each dimension, and then take the p-norm of the results. Here, Q^d (or C^d) represents the univariate time series of dimension d such that $Q^d =< q_1^d, \cdots, q_L^d >$ (or $C^d =< c_1^d, \cdots, c_L^d >$). The parameter p is set to 1 in [14].

For consistency with previous work, we assume a 1-norm unless otherwise specified. For ease of comprehension, we indicate an independent extension of a univariate measure by adding the subscript I. Hence, $DTW_I(Q, C) = Ind(DTW, Q, C, 1)$, $WDTW_I(Q, C) = Ind(WDTW, Q, C, 1)$ and so forth.

However, in most cases it requires more than such a simple formulation to derive a dependent extension, and hence we below introduce each of the univariate measures together with our proposed dependent variant.

4.1 Lp Distance (Lp)

4.1.1 Univariate Lp Distance

The simplest way to calculate similarity between two time series is to use Lp distance, also known as the Minkowski distance.

Let us denote by Q and C two univariate ($D = 1$) time series of length L where q_i and c_i are scalar values at time point i from the two time series. Equation 3 formulates the Lp distance between Q and C.

$$Lp(Q,C) = \left(\sum_{i}^{L} |q_i - c_i|^p \right)^{1/p} \tag{3}$$

The parameter p is the order of the distance. The L_1 (Manhattan distance) and L_2 (Euclidean distance) distances are widely used.

In the context of TSC, Lp distances are of limited use because they cannot align two series that are misaligned in the time dimension, since they compute one-to-one differences between corresponding points only.

For example, in an electrocardiogram (ECG) signal, two measurements from a patient at different times may produce slightly different time series which belong to the same class (e.g certain heart condition). Ideally, if they belong to the same class, an effective similarity measure should account for such "misalignments" in the time axis, while capturing the similarity.

Elastic similarity measures such as DTW tackle this issue. Elastic similarity measures are designed to compensate for temporal misalignments in time series that might be due to stretched, shrunken or misaligned subsequences. From Section 4.2 to 4.6 we will present various *elastic* similarity measures.

4.1.2 Multivariate Lp Distance

We here show that Independent Lp distance (Lp_I) and Dependent Lp distance (Lp_D) are identical.

Definition 2 Independent Lp Distance (Lp_I)

In this case, we simply compute the Lp distance between Q and C separately for each dimension, and then take the p-norm of the results.

$$\begin{aligned} Lp_I(Q,C) &= \left(\sum_{d=1}^{D} \left| Lp(Q^d, C^d) \right|^p \right)^{1/p} \\ &= \left(\sum_{d=1}^{D} \sum_{i=1}^{L} \left| q_i^d - c_i^d \right|^p \right)^{1/p} \end{aligned} \tag{4}$$

Definition 3 Dependent Lp Distance (Lp_D)

In this case, we compute the Lp distance between each multidimensional point ($\mathbf{q_i}$ $\mathbb{R}^D$ and $\mathbf{c_i}$ $\mathbb{R}^D$, and take the p-norm of the results.

$$\begin{aligned} Lp_D(Q,C) &= \left(\sum_{i=1}^{L} |Lp(\mathbf{q_i} - \mathbf{c_i})|^p \right)^{1/p} \\ &= \left(\sum_{i=1}^{L} \sum_{d=1}^{D} \left| q_i^d - c_i^d \right|^p \right)^{1/p} \end{aligned} \tag{5}$$

Consequently both the independent and the dependent versions of the ***non-elastic*** Lp distance will produce the same result. However, as we shall see in the following sections, for ***elastic*** similarity measures the two strategies are substantially different.

4.2 Dynamic Time Warping (DTW) and Related Measures

4.2.1 Univariate DTW

The most widely used ***elastic*** similarity measure is DTW [1]. By contrast to measures such as the Lp distance, DTW is an elastic similarity measure, which allows one-to-many alignment ("warping") of points between two time series. For many years, 1-NN with DTW was considered as the traditional benchmark algorithm for TSC [39].

DTW is efficiently solved using a dynamic programming technique. Let Δ_{DTW} be an L-by-L dynamic programming cost matrix in which the element (i, j) of the matrix is defined as the squared Euclidean distance between two corresponding points q_i and c_j – ***i.e.*** $\Delta_{DTW}(i, j) = (q_i - c_j)^2$ and the minimum of the cumulative distances of the previous points. Equation 6 defines element (i, j) of the cost matrix as follows:

$$\Delta_{DTW}(i,j) = (q_i - c_j)^2 + min \begin{cases} \Delta_{DTW}(i-1, j-1) & \textbf{if } i,j > 1 \\ \Delta_{DTW}(i, j-1) & \textbf{if } j > 1 \\ \Delta_{DTW}(i-1, j) & \textbf{if } i > 1. \end{cases} \quad (6)$$

The cost matrix represents the alignment of the two series as according to the DTW algorithm. DTW between two series Q and C is the accumulated cost in the last element of the cost matrix (***i.e.*** $i, j = L$) as defined in Equation 7:

$$DTW(Q, C) = \Delta_{DTW}(L, L). \quad (7)$$

DTW has a parameter called "window size" (w), which helps to prevent pathological warpings by constraining the maximum allowed warping distance. For example, when $w = 0$, DTW produces a one-to-one alignment, which is equivalent to the Euclidean distance. A larger warping window allows one-to-many alignments where points from one series can match points from the other series over longer time frames. Therefore, w controls the ***elasticity*** of the similarity measure. In this paper, we use DTW to refer to DTW a cross-validated window parameter and DTWF to refer to DTW with full window.

Different methods have been used to select the parameter w. In some methods, such as EE, and HIVE-COTE, w is selected using leave-one-out cross-validation. Some algorithms select the window size randomly (***e.g.*** PF and TS-CHIEF select window sizes from the uniform distribution $U(0, L/4)$).

Parameter w also improves the computational efficiency, since in most cases, the ideal w is much less than the length of the series [40]. When w

is small, DTW runs relatively fast, especially with lower bounding, and early abandoning techniques [41, 42, 43, 44]. Time complexity to calculate DTW with a warping window is $O(L \cdot w)$, instead of $O(L^2)$ for the full DTW.

4.2.2 Dependent Multivariate DTW

Definition 4 Dependent DTW (DTW_D)

Dependent DTW (DTW_D) uses all dimensions together when computing the point-wise distance between each point in the two time series. In this method, for each point in the series, DTW is allowed to warp across the dimensions.

In this case, the squared Euclidean distance between two univariate points – $(q_i - c_j)^2$ – in Equation 6 is replaced with the L_2-norm computed between the two multivariate points $\mathbf{q_i}$ and $\mathbf{c_j}$ as in Equation 8.

$$L_2(\mathbf{q_i}, \mathbf{c_j})^2 = \sum_{d=1}^{D} (q_i^d - c_j^d)^2 \tag{8}$$

4.2.3 Derivative DTW (DDTW)

Derivative DTW (DDTW) is a variation of DTW, which computes DTW on the first derivatives of time series. Keogh et. al. [45] developed this version to mitigate some pathological warpings, particularly, cases where DTW tries to explain variability in the time series values by warping the time-axis, and cases where DTW misaligns features in one series which are higher or lower than its corresponding features in the other series. The derivative transformation of a univariate time point q_i is defined as:

$$q_i = \frac{(q_i - q_{i-1} + (q_{i+1} - q_{i-1})/2)}{2} \tag{9}$$

Note that q_i is not defined for the first and last element of the time series. Once the two series have been transformed, DTW is computed as in Equation 7.

Multivariate versions of DDTW are very straightforward to implement. We calculate the derivatives separately for each dimension, and then use Equations 2 and 8 to compute from the derivatives independent DDTW ($DDTW_I$) and dependent DDTW ($DDTW_D$), respectively.

4.2.4 Weighted DTW (WDTW)

Weighted DTW (WDTW) is another variation of DTW, proposed by [46], which uses a "soft warping window" in contrast to the fixed warping window sized used in classic DTW. WDTW penalises large warpings by assigning a non-linear multiplicative weight w to the warpings using the modified logistic function in Equation 10:

$$weight_{|i-j|} = \frac{weight_{max}}{1 + e^{-g \cdot ((|i-j|-L)/2)}}, \tag{10}$$

where $weight_{max}$ is the upper bound on the weight (set to 1), L is the series length and g is the parameter that controls the penalty level for large warpings. Larger values of g increases the penalty for warping.

When creating the dynamic programming distance matrix Δ_{WDTW}, the weight penalty $weight_{|i-j|}$ for a warping distance of $|i-j|$ is applied, so that the (i,j)-th entry in the matrix is $\Delta_{WDTW}(i,j) = weight_{|i-j|} \cdot (q_i - c_i)^2$. Therefore, the new equation for WDTW is defined as

$$\Delta_{WDTW}(i,j) = weight_{|i-j|} \cdot (q_i - c_j)^2 + min \begin{cases} \Delta_{WDTW}(i-1, j-1) & \textbf{if } i,j > 1 \\ \Delta_{WDTW}(i, j-1) & \textbf{if } j > 1 \\ \Delta_{WDTW}(i-1, j) & \textbf{if } i > 1. \end{cases} \tag{11}$$

$$WDTW(Q,C) = \Delta_{WDTW(L,L)}. \tag{12}$$

Parameter g may be selected using leave-one-out cross-validation as in EE and HIVE-COTE, or selected randomly as in PF and TS-CHIEF ($g \quad U(0,1)$).

Since WDTW does not use a constrained warping window (*i.e.* the maximum warping distance $|i-j|$ may be as large as L), its time complexity is $O(L^2)$, which is higher than DTW.

4.2.5 Dependent Multivariate WDTW

Definition 5 Dependent WDTW

The dependent version of WDTW simply inserts the weight into DTW_D. We define Dependent WDTW ($WDTW_D$) as,

$$\Delta_{WDTW_D}(i,j) = weight_{|i-j|} \cdot L_2(\mathbf{q_i}, \mathbf{c_j})^2 + min \begin{cases} \Delta_{WDTW_D}(i-1, j-1) & \textbf{if } i,j > 1 \\ \Delta_{WDTW_D}(i-1, j) & \textbf{if } j > 1 \\ \Delta_{WDTW_D}(i, j-1) & \textbf{if } i > 1, \end{cases} \tag{13}$$

$$WDTW_D(Q,C) = \Delta_{WDTW_D}(L,L). \tag{14}$$

4.2.6 Weighted Derivative DTW (WDDTW)

The ideas behind DDTW and WDTW may be combined to implement another measure called Weighted Derivative DTW (WDDTW). This method has also been traditionally used in some ensemble algorithms [6].

Multivariate versions of WDDTW are also straightforward to implement. We calculate the derivatives separately for each dimension, and then use Equations 2 and 14 with them to compute independent WDDTW ($WDTW_I$) and dependent WDDTW ($WDTW_D$), respectively.

4.3 Longest Common Subsequence (LCSS)

4.3.1 Univariate LCSS

Longest Common Subsequence (LCSS) distance is based on the edit distance algorithm, which is used for string matching [47]. In TSC, the LCSS algorithm is modified to work with real-valued data by adding a threshold ϵ for real-value comparisons. Two real-values are considered a match if the difference between them is less than the threshold ϵ. A warping window can also be used in conjunction with the threshold to constrain the degree of local warping.

The unnormalized LCSS distance ($LCSS_{UN}$) between Q and C is

$$\Delta_{LCSS}(i,j) = \begin{cases} (q_i - c_j)^2 & \textbf{if } i = 1, j = 1 \\ 1 + \Delta_{LCSS}(i-1, j-1) & \textbf{if } |q_i - c_j| \leq \epsilon \\ max \begin{cases} \Delta_{LCSS}(i-1, j) \\ \Delta_{LCSS}(i, j-1) \end{cases} & \textbf{otherwise}, \end{cases} \tag{15}$$

$$LCSS_{UN}(Q,C) = \Delta_{LCSS}(L,L), \tag{16}$$

In practice, $LCSS_{UN}$ is then normalized based on the series length L.

$$LCSS(Q,C) = 1 - \frac{LCSS_{UN}(Q,C)}{L}, \tag{17}$$

LCSS can be used with a window parameter w similar to DTW. With a window parameter, LCSS has a time complexity of $O(L \cdot w)$. In EE and PF, the parameter ϵ is selected from $[\frac{\sigma}{5}, \sigma]$, where σ being the standard deviation of the whole dataset.

4.3.2 Independent Multivariate LCSS

Independent multivariate LCSS uses the Equation 2 and computes the LCSS for dimensions separately. However, in this case we use a separate ϵ parameter for each dimension. We computer the standard deviation σ per dimension when sampling ϵ. Sampling e for each dimension can be useful if the data is not normalized.

4.3.3 Dependent Multivariate LCSS

Definition 6 Dependent LCSS

Dependent LCSS ($LCSS_D$) is similar to Equation 15, except that to compute distance between two multivariate points we use Equation 8 and an adjustment is made to the parameter ϵ (see Section 4.3.4). In this case, parameter ϵ is selected using the standard deviation of the whole dataset. This is similar to the way it was selected in the univariate LCSS, in EE.

$$\Delta_{LCSS_D}(i,j) = \begin{cases} L_2(\mathbf{q_i},\mathbf{c_j})^2 & \textbf{if } i=1, j=1 \\ 1+\Delta_{LCSS_D}(i-1,j-1) & \textbf{if } L_2(\mathbf{q_i},\mathbf{c_j})^2 \leq 2\cdot D\cdot\epsilon \\ max\begin{cases}\Delta_{LCSS_D}(i-1,j)\\ \Delta_{LCSS_D}(i,j-1)\end{cases} & \textbf{otherwise}, \end{cases} \quad (18)$$

$$LCSS_{UND}(Q,C) = \Delta_{LCSS_D}(L,L), \quad (19)$$

Similar to the univariate case, $LCSS_{UND}$ is then normalized based on the series length L.

$$LCSS_D(Q,C) = 1 - \frac{LCSS_{UND}(Q,C)}{L}. \quad (20)$$

4.3.4 Adjusting ϵ parameter in LCSS

In Equation 18 we multiply ϵ by 2 times the number of dimensions D, because the term $L_2(\mathbf{q_i},\mathbf{c_j})^2$ increases with the number of dimensions and the parameter ϵ (floating point comparison threshold) is independent of the number of dimensions (compare with the univariate definition in Equation 15).

The adjustment factor $2\cdot D$ can be used with a number of assumptions. First, data should follow a normal distribution. Second, the squared Euclidean distance ($L_2(\mathbf{q_i},\mathbf{c_j})^2$) is required rather than the Manhattan distance (L_1).

To compensate for the increase in the magnitude of squared Euclidean distance with respect to the increasing number of dimensions, the adjustment factor needs to normalize this value to make Equation 18 work similarly to Equation 15.

Consider two vectors X and Y in D dimension X_i, Y_i $N(0,1)$ where i $[1,D]$. Assume that X and Y are independent and each dimension is independent as well.

Let another random variable $Z = X - Y$. By property of the normal distribution, we have Z $N(0,2)$ (as X and Y are independent we can add both mean and variance, hence $N(0+0, 1+1)$).

We are interested in $E[Z^2]$. This follows chi-square distribution, which is the sum of square of independent normally distributed variables.

However, the chi-square distribution is only if the variance is 1 and here we have variance of 2 for Z. Let V be another random variable of variance 1. To do so, we divide Z^2 by squared standard deviation (*i.e.* variance 2), so $V = (Z^2/2)$ $\chi^2(D)$. Then we have

$E[Z^2/2] = E[V]$ and we know $E[V] = D$ by property of the chi square distribution

$E[Z^2]/2 = D$

$E[Z^2] = 2\cdot D$

Therefore, if data are normally distributed, $(L_2)^2$ distance between points, with any number of dimensions may be scaled by a factor of $2\cdot D$ to make the values comparable to average magnitude of values in one dimension. For

this reason we multiply the right hand side (ϵ) by this factor. Once the left hand side has been adjusted, it may be compared with ϵ similarly as in the univariate LCSS definition.

We found that this adjustment works for the datasets we tested. In practice, if the data distribution differs substantially from the normal distribution, this adjustment factor may need to be revised, hence further investigation of methods to adjust the parameter ϵ is a potential direction for future work.

4.3.5 Other $LCSS_D$ Formulations

An early work by Vlachos et al. [48, 49] also presented a way to extend measures to the multivariate case. They proposed a dependent DTW and LCSS with the claim that their method can be extended to work with any measure and any number of dimensions. However our work shows that such extensions can be non-trivial in some cases, e.g. LCSS and MSM.

Their proposed dependent DTW is similar to Shokoohi et al.'s DTW_D formulation in Equation 8, but their LCSS's formulation is slightly different to our LCSS formulation present in Equation 18. Equation 21 defines this version of LCSS named $Vlachos\_LCSS_D$.

$$\Delta_{Vlachos\_LCSS_D}(i,j) = \begin{cases} L_2(\mathbf{q_i}, \mathbf{c_j})^2 & \textbf{if } i = 1, j = 1 \\ 1 + \Delta_{LCSS_D}(i-1, j-1) & \textbf{if } \quad d \quad D, |q_i^d - c_j^d| < \epsilon \\ & \textbf{and } |i - j| \leq \delta \\ max \begin{cases} \Delta_{LCSS_D}(i-1, j) \\ \Delta_{LCSS_D}(i, j-1) \end{cases} & \textbf{otherwise}, \end{cases} \tag{21}$$

Our method treats $\mathbf{q_i}$ and $\mathbf{c_i}$ as each being multi-dimensional points, placing a single constraint on the distance between them, whereas Vlachos et al. treat the dimensions independently, with separate constraints on each. We believe that this is more consistent with the spirit of dependent measures.

$Vlachos\_LCSS_D$ allows matching values within both data dimensions and the time dimension, while our $LCSS_D$ matches values only in data dimensions. They have an additional parameter δ that controls matching in the time dimension.

Our method also requires an adjustment to LCSS's ϵ parameter as described in Section 4.3.4. This may help to cater for unnormalized data, since $Vlachos\_LCSS_D$ use the one ϵ across all dimensions.

In addition, Vlachos et al. [48, 49] tested their algorithm on 8 trajectory datasets with only 2 dimensions. Whereas, our work is evaluated on 23 datasets from the UEA Multivariate Time Series Archive [15] which has a diverse range of domains and dimensionality.

4.4 Move-Split-Merge (MSM)

4.4.1 Univariate MSM

Move-Split-Merge (MSM) is introduced by [50]. The goal is to propose a similarity measure that is a metric invariant to translations and robust to temporal misalignments. Measures such as DTW and LCSS are not metrics because they fail to satisfy the triangle inequality.

MSM is an edit distance-based similarity measure. Similarity between two series is computed based on the number and type of edit operations required to transform one series to the other.

MSM defines three types of edit operations: move, merge and split. The move operation substitutes one value into another value. The split operation inserts a copy of the value immediately after itself, and the merge operation is used to delete a value if it directly follows an identical value.

The cost for a move operation is the pairwise distance between two points, and the cost of split or merge operation depends on the parameter c.

Formally, MSM is defined as,

$$\Delta_{MSM}(i,j) = min \begin{cases} \Delta_{MSM}(i-1,j-1) + |q_i - c_j| \\ \Delta_{MSM}(i-1,j) + \mathit{cost}(q_i, q_{i-1}, c_j) \\ \Delta_{MSM}(i,j-1) + \mathit{cost}(c_j, q_i, c_{i-1}), \end{cases} \tag{22}$$

$$MSM(Q,C) = \Delta_{MSM}(L,L). \tag{23}$$

The costs of split and merge operations are defined by Equation 24. In the univariate case, the algorithm either merges two values or splits a value if the the value of a point q_i *is_between* two adjacent values (q_{i-1} and c_j).

$$\mathit{cost}(q_{i-1}, q_i, c_j) \quad = \quad min \begin{cases} c \textbf{ if } q_{i-1} \leq q_i \leq c_j \\ c \textbf{ if } q_{i-1} \geq q_i \geq c_j \\ c + min \begin{cases} |q_i - q_{i-1}| \\ |q_i - c_j| \end{cases} \textbf{otherwise}. \end{cases} \tag{24}$$

In most algorithms (*e.g.* EE, PF, HIVE-COTE and TS-CHIEF), the cost parameter c for MSM is selected from an exponential sequence $\{10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, \cdots, 1\}$ as proposed in [50].

4.4.2 Dependent Multivariate MSM

Definition 7 Dependent MSM

Here we combine Equation 22 and Equation 8. The *cost_multiv* function is explained in Section 4.4.3, and presented in Algorithm 1.

$$\Delta_{MSM_D}(i,j) \quad = \quad min \begin{cases} \Delta_{MSM_D}(i-1,j-1) + L_2(\mathbf{q_i}, \mathbf{c_j})^2 \\ \Delta_{MSM_D}(i-1,j) + \mathit{cost\_multiv}(\mathbf{q_i}, \mathbf{q_{i-1}}, \mathbf{c_j}) \\ \Delta_{MSM_D}(i,j-1) + \mathit{cost\_multiv}(\mathbf{c_j}, \mathbf{q_i}, \mathbf{c_{j-1}}) \end{cases} \tag{25}$$

$$MSM_D(Q,C) = \Delta_{MSM_D}(L,L) \tag{26}$$

4.4.3 Cost function for dependent MSM

A nontrivial issue when deriving a dependent variant of MSM is how to translate the concept of one point being between two others.

A naive approach would test whether a point x *is_between* points y and z in multidimensional space by projecting x onto the hyperplane defined by y and z. However, this has serious limitations. For an intuitive example, let us use cities to represent points on a 2-D plane. Assume that we have two query cities Chicago and Santiago wish to determine which is between New York and San Francisco. If we use vector projections, and project the position of Santiago on to the line between New York and San Francisco we will find that it is between them. Similarly, we will also find that Chicago is between New York and San Francisco using this method. However, orthogonally Santiago is extremely far away from both New York and San Francisco, so it would seem more intuitive to define this function in a way that Chicago is in between New York and San Francisco, but Santiago is not. Using this intuition we define the cost function in such a way that a point is considered to be in between two points only if the point is "inside the hypersphere" defined by the other two points. Figure 2 illustrates this concept using three points.

We implement this idea in Algorithm 1. First we find the diameter of the hypersphere in line 1 by computing $||\mathbf{q_{i-1}} - \mathbf{c_j}||$. In line 2 we find the mid point **mid** along the line $\mathbf{q_{i-1}}$ and $\mathbf{c_j}$. Then we calculate distance to the mid point using $||\mathbf{mid} - \mathbf{q_i}||$ (line 3). Once we have the *distance_to_mid*, we check if this distance is larger than half the diameter. If its larger, then the point $\mathbf{q_i}$ is outside the hypersphere, and so we return c (line 5). If *distance_to_mid* is less than half the diameter, then $\mathbf{q_i}$ is inside the hypersphere, so we check to which point (either $\mathbf{q_{i-1}}$ or $\mathbf{c_j}$ it is closest). Then we return c plus the distance to the closest point as the cost of the edit operation (line 9 to 12).

4.5 Edit Distance with Real Penalty (ERP)

4.5.1 Univariate ERP

Edit Distance with Real Penalty (ERP) [51, 52] is also based on string matching algorithms. In a typical string matching algorithm, two strings, possibly of different lengths, may be aligned by doing the least number of add, delete or change operations on the symbols. When aligning two series of symbols, the authors proposed that the delete operations in one series can be thought of as adding a special symbol to the other series. Chen et. al. [51] refers to these added symbols as a "gap" element.

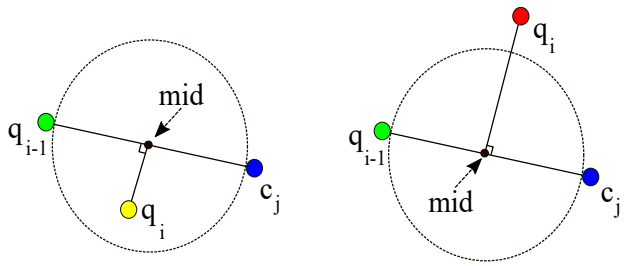


Fig. 2 Example of checking whether a point is between two other points in 2 dimensions using a circle. In the first case (left side), the yellow point is considered "in between" blue and green points. In the second case (right side), the red point is considered to be "not in between" the green and blue points even though its projection falls between red and blue points because orthogonally it is outside the circle defined by theses points. This is one way to adapt the idea of checking if a point is between two other points in the univariate case (1-dimension) as defined in Equation 24. In Algorithm 1 we use a generalization of this idea and check if a point is inside a hypersphere in D-dimensions.

Algorithm 1: Cost of checking if a mid point is inside the hypersphere defined by the other two points

Input: $cost\_multiv(\mathbf{q_i}, \mathbf{q_{i-1}}, \mathbf{c_j}, c)$: three points, and cost parameter c for MSM
Output: cost of operation

```
1  diameter = ||q_{i-1} − c_j||;
2  mid = (q_{i-1} + c_j)/2;
3  distance_to_mid = ||mid − q_i||;
4  if distance_to_mid ≤ (diameter/2) then
5  |  return c;
6  else
7  |  dist_to_q_prev = ||q_{i-1} − q_i||;
8  |  dist_to_c = ||c_j − q_i||;
9  |  if dist_to_q_prev < dist_to_c then
10 |  |  return c + dist_to_q_prev;
11 |  else
12 |  |  return c + dist_to_c ;
```

ERP uses the Euclidean distance between elements when there is no gap, and a constant penalty when there is a gap. This penalty parameter for a gap is denoted as g (see Equation 27).

For time series, with real values, similar to the parameter ϵ in LCSS, a floating point comparison threshold may be used to determine a match between two values. This idea was used in a measure called Edit Distance on Real sequences (EDR) [51]. However, using a threshold breaks the triangle inequality. Therefore, the same authors proposed a variant, namely ERP, which is a measure that follows the triangle inequality.

ERP can also be used with a window parameter w similar to DTW. With the window parameter, ERP has the same time complexity as DTW. The parameter g is selected from $[\frac{\sigma}{5}, \sigma]$, with σ being the standard deviation of the training data. Formally, ERP is defined as,

$$\Delta_{ERP}(i,j) = min \begin{cases} \Delta_{ERP}(i-1,j-1) + (q_i - c_j)^2 \\ \Delta_{ERP}(i-1,j) + (q_i - g)^2 \\ \Delta_{ERP}(i,j-1) + (c_j - g)^2 \end{cases} \tag{27}$$

$$ERP(Q,C) = \Delta_{ERP}(L,L). \tag{28}$$

4.5.2 Dependent ERP

Definition 8 Dependent ERP
We define Dependent ERP (ERP_D) as,

$$\Delta_{ERP_D}(i,j) = min \begin{cases} \Delta_{ERP_D}(i-1,j-1) + L_2(\mathbf{q_i},\mathbf{c_j})^2 \\ \Delta_{ERP_D}(i-1,j) + L_2(\mathbf{q_i},\mathbf{g}) \\ \Delta_{ERP_D}(i,j-1) + L_2(\mathbf{c_j},\mathbf{g}), \end{cases} \tag{29}$$

$$ERP_D(Q,C) = \Delta_{ERP_D}(L,L). \tag{30}$$

In Equation 29, note that the parameter $\mathbf{g}$ is a vector which represents the standard deviation of each dimension separately. This is in contrast to the univariate case in Equation 27 which uses the standard deviation of the whole training dataset (parameter g).

In this case, all terms increases proportionally with respect to the increase in the number of dimensions. So we do not need to adjust for the parameter $\mathbf{g}$ as we adjusted for ϵ in LCSS in Section 4.3.4.

4.6 Time Warp Edit (TWE)

4.6.1 Univariate TWE

Time Warp Edit (TWE) [53] is a further edit-distance based algorithm adapted to the time series domain. The goal is to combine an Lp distance based technique with an edit-distance based algorithm that supports warping in the time axis, *i.e.* has some sort of *elasticity* like DTW, while also being a distance metric (*i.e.* it respects the triangle inequality). Being a metric helps in time series indexing, since it speeds up time series retrieval process.

TWE uses three operations named *match*, $delete_A$, and $delete_B$. If there is a *match*, Lp distance is used, and if not, a constant penalty λ is added. $delete_A$ (or $delete_B$) is used to remove an element from the first (or second) series to match the second (or first) series. Equations 31, 32 and 33 define TWE and these three operations, respectively.

$$\Delta_{TWE}(i,j) = min \begin{cases} \Delta_{TWE}(i-1,j-1) + \gamma_M & match \\ \Delta_{TWE}(i-1,j) + \gamma_A & delete_A \\ \Delta_{TWE}(i,j-1) + \gamma_B & delete_B \end{cases} \tag{31}$$

$$TWE(Q,C) = \Delta_{TWE}(L,L) \tag{32}$$

$$\begin{array}{ll} \gamma_M = (q_i - c_j)^2 + (q_{i-1} - c_{j-1})^2 + 2 \cdot \nu & match \\ \gamma_A = (q_i - q_{i-1})^2 + \nu + \lambda & delete_A \\ \gamma_B = (c_j - c_{j-1})^2 + \nu + \lambda & delete_B \end{array} \tag{33}$$

The multiplicative penalty ν[1] is called the *stiffness* parameter. When $\nu = 0$, TWE becomes more stiff like the Lp distance, and when $\nu = \infty$, TWE becomes less stiff and more elastic like DTW. The second parameter λ is the cost of performing either a $delete_A$ or $delete_B$ operation.

Following [5, 16, 53], λ is selected from $\{\frac{i}{9}\}_{i=0}^{9}$ and ν from the exponentially growing sequence $\{10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, \cdots, 1\}$, resulting in 100 possible parameterizations.

4.6.2 Dependent TWE

Definition 9 Dependent TWE

Dependent version of TWE follows a similar pattern. We define Dependent TWE (TWE_D) as,

$$\Delta_{TWE_D}(i,j) = min \begin{cases} \Delta_{TWE_D}(i-1, j-1) + \gamma_M & match \\ \Delta_{TWE_D}(i-1, j) + \gamma_A & delete_A \\ \Delta_{TWE_D}(i, j-1) + \gamma_B & delete_B \end{cases} \tag{34}$$

$$TWE_D(Q, C) = \Delta_{TWE_D}(L, L). \tag{35}$$

$$\begin{array}{ll} \gamma_M = L_2(\mathbf{q_i}, \mathbf{c_j})^2 + L_2(\mathbf{q_{i-1}}, \mathbf{c_{j-1}})^2 & \\ \quad + (2 \cdot \nu) \cdot 2 \cdot D & match \\ \gamma_A = L_2(\mathbf{q_i}, \mathbf{q_{i-1}})^2 + (\nu + \lambda) \cdot 2 \cdot D & delete_A \\ \gamma_B = L_2(\mathbf{c_j}, \mathbf{c_{j-1}})^2 + (\nu + \lambda) \cdot 2 \cdot D & delete_B \end{array} \tag{36}$$

Similar to LCSS, in Dependent TWE, we need to make an adjustment to the parameters. Once again, we multiply the terms that do not grow with $2 \cdot D$.

5 Experiments

We conduct experiments to investigate two hypotheses. The first is that there are different datasets to which each of the new multivariate distance measures is best suited. The second arises from the observation that there are datasets for which either the independent or dependent version of DTW are consistently more accurate than the alternative [14]. However, it is not clear whether this is

[1] In the published definition of TWE [53], ν is multiplied with the time difference in the timestamps of two consecutive time points. We simplified this equation, for clarity, by assuming that this time difference is always 1 (UEA datasets do not contain the actual timestamps).

a result of there being an advantage in treating multivariate series as either a single series of multivariate points or multiple independent series of univariate points; or rather due to some other property of the measures.

It is credible that there should be some time series data for which it is beneficial to treat multiple variables as multivariate points in a single series. Suppose, for example, that the variables each represent the throughput of independent parts of a process and the quantity relevant to classification is aggregate throughput. In this case, the sum of the values at each point is the relevant quantity. In contrast, if classification relates to a failure in any of those parts, it seems clear that independent consideration of each is the better approach.

We seek to assess whether there are multivariate datasets for which each of dependent and independent analysis are best suited, or whether there are other reasons, such as their mathematical properties, that underlie the systematic advantage on specific datasets of either DTW_I or DTW_D.

We start by describing our experimental setup and the datasets we used. We then conduct an analysis of similarity measures in the context of TSC by comparing accuracy measures of independent and dependent versions. We then conduct a statistical test to determine if there is a difference between independent and dependent versions of the measures.

5.1 Experimental Setup

We implemented a multi-threaded version of the multivariate similarity measures in Java. We also release the full source code in the github repository: https://github.com/dotnet54/multivariate-measures.

In these experiments, for parameterization of the measures, we use leave-one-out cross-validation of 100 parameters for each similarity measure. We follow the same setting proposed in [5]. This parameterization is also used in HIVE-COTE, PF, and TS-CHIEF.

In this study, we use multivariate datasets obtained from https://www.timeseriesclassification.com. We also use the standard train/test splits provided in the repository. Out of the available 30 datasets, we use 23 datasets in this study. Since we focus only on fixed-length datasets, the four variable length datasets (*CharacterTrajectories*, *InsectWingbeat*, *JapaneseVowels*, and *SpokenArabicDigits*) are excluded from this study. We also we omit *EigenWorms*, *MotorImagery* and, *FaceDetection*, which take too long to run the leave-one-out cross-validation for 100 parameters in a practical time frame. Table 2 (on page 30) summarizes the characteristics of the 23 fixed length datasets. Further descriptions of each dataset can be found in [15].

We ran experiments for all measures for both z-normalized and unnormalized datasets. We z-normalized each dataset on a per series, per dimension basis. We found that the accuracy is higher without further normalization (note that 4 datasets - *ArticularyWordRecognition*, *Cricket*, *HandMovementDirection*, and *UWaveGestureLibrary* - are already normalized). This agrees

with a recent paper which conducted a similar experiment using DTW_I and DTW_D [34].

However, we note that this does not indicate that not normalizing is always the optimal solution for all datasets. Sometimes normalization can be useful when using similarity measures. For example, consider a scenario with two dimensions temperature (e.g a scale from 0 to 100 degree Celsius) and relative humidity as a proportion (between 0 and 1). In such a case, temperature will dominate the result of the similarity calculation, and normalization will help to compute the similarity with similar scales across the dimensions.

We ran the experiments on a cluster of Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50 GHz CPUs, using 32-threads. Total time to train 23 datasets with leave-one-out-cross-validation was about 1648 hours. The two slowest datasets were *PEMS-SF* (291 hours) and *PhonemeSpectra* (1153 hours). The slowest measure to train was MSM_D, which took a total of 801 hours across all datasets.

5.2 Accuracy of Independent *Vs* Dependent Measures

First we look at the accuracy of each measure used with a 1-NN classifier. Tables 3 and 4 (on page 31) present the accuracy for independent measures and dependent measures, respectively. For each dataset, the highest accuracy is typeset in bold. Of the values reported in Tables 3 and 4, accuracy for measures other than Euclidean distance (labeled "L_2" in the table) and DTW are newly published results in this paper.

Our first observation is that for every similarity measure there is at least one dataset for which that measure obtains the highest accuracy. This supports our first hypothesis, that each measure will have datasets for which it is well suited.

To compare multiple algorithms over the multiple datasets, first a Friedman test is performed to reject the null hypothesis. The null hypothesis is that there is no significant difference in the mean ranks of the multiple algorithms (at a statistical significance level $\alpha = 0.05$). In cases where the null-hypothesis is rejected, we use the Wilcoxon signed rank test to compare the pair-wise difference in ranks between algorithms, and then use Holm–Bonferroni's method to adjust for family-wise errors [54, 55].

Figure 3 displays mean ranks (on error) between the 20 similarity measures. Measures on the right side indicate higher rank in accuracy (lower error). We do not include $L2$ distance here because the accuracy for independent and dependent $L2$ is the same. Since we use Holm–Bonferroni's correction, there is not a single "critical difference value" that applies to all pairwise comparisons. Hence, we refer to these visualisations as "average accuracy ranking diagrams".

In Figure 3, DTW_I, which is to the further right, is the most accurate measure on the evaluated datasets. DTW_I obtained a ranking of 7.326 from the Wilcoxon test. By contrast, $DDTWF_D$ (ranked 14.783) is the least accurate measure on these datasets (suffix "F" is added to the measure name to denote the use of full window). After Holm–Bonferroni's correction, computed

p-values indicate that only the pairs $DDTWF_D$ and DTW_I, and $DDTWF_D$ and $WDTW_D$, are statistically different from each other. We referred to the table of p-values because this is difficult to see from the figure since lines for DTW_D, DTW_I and $WDTW_D$ are very close together. This small number of significant differences could be because the number of datasets in the study is small relative to the number of measures compared.

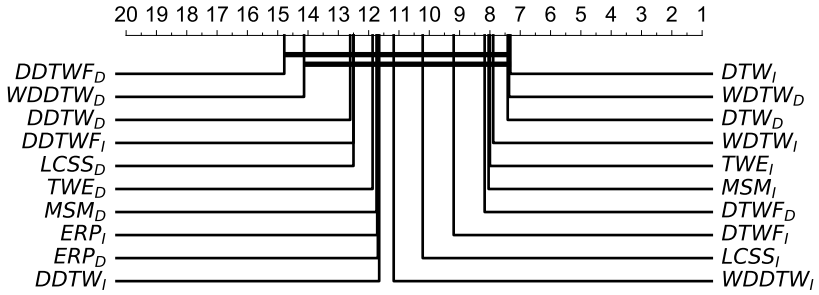


Fig. 3 Average accuracy ranking diagram showing the ranks of measures on the error rates (thus more accurate measures are to the right side).

5.3 Are Independent and Dependent Measures Significantly Different?

In this section, we test if there are datasets for which independent or dependent version is always more accurate. We also test if there is a statistically significant difference between independent and dependent similarity measures. Answering these questions will help us determine the usefulness of developing these two variations of the multivariate similarity measures. It will also help us to construct ensembles of similarity measures with more diversity, that is expected to perform well in terms of accuracy over a wide variety of datasets.

Figure 4 shows the difference in accuracy between independent and dependent versions of the measures - deeper reddish colours indicate cases where independent is more accurate (positive on the scale), and deeper bluish colours indicate cases where dependent is more accurate (negative on the scale). The datasets are sorted based on average colour values to show contrasting colours on the two ends. (Dimensions D / length L / number of classes c are given in the bracket after the dataset name.)

From Figure 4 we observe that there are datasets for which either independent or dependent is always more accurate. For example, the independent versions of all measures are consistently more accurate for datasets *PEMS-SF* and Basic Motions (indicated by red colour rows in the heatmap). On the other hand, we see that *Handwriting* always wins for the dependent versions (indicated by the blue colour row).

Recall the example of two datasets described in Section 1: *Handwriting* dataset records "X", "Y" and "Z" coordinates of hand writing letters, and *PEMS-SF* dataset records traffic congestion rate of highways using multiple

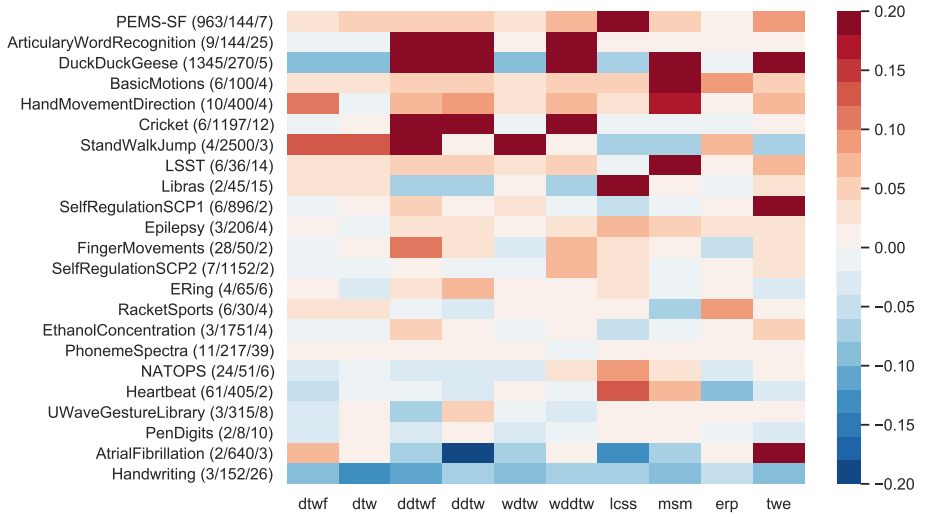


Fig. 4 Heatmap showing the difference in accuracy between independent and dependent versions of the measures - deeper reddish colours indicate cases where independent is more accurate (positive on the scale), and deeper bluish colours indicate cases where dependent is more accurate (negative on the scale). The datasets are sorted based on average colour values to show contrasting colours on the two ends. (Dimensions / length / number of classes are shown in the bracket after the dataset name.)

sensors at different locations. From Figure 4 we observe that the dimensions in *Handwriting* are highly coupled and works best when all dimensions are considered together and *PEMS-SF* works best when all dimensions are considered independently. Since each sensor is a dimension in this dataset, there are 963 dimensions. It is highly unlikely that all of them will be coupled together.

In general, We observe that the dependent strategy tends to perform poorly when there are a large number of dependent dimensions in the dataset.

It is interesting to note that there appears to be considerable correlation between the relative desirability of the independent and dependent approaches across all the measures that are applied to the derivative of the original series, $DDTWF$, $DDTW$ and $WDDTW$. It particularly stands out that there seems to be a strong advantage to independent variants of these measures with respect to *ArticularyWordRecognition*, *DuckDuckGeese* and *Cricket*. Possible connections between transformations and the relative efficacy of independent or dependent approaches may be a productive topic for future research.

Next we investigate the hypothesis that there are some multivariate TSC tasks that are inherently best suited to either treating the multivariate series as a single series of multvariate points or as multiple independent series of univariate points. To this end we present the results of a Wilcoxon signed-rank test on each of the 10 pairs of measures (without L_2), to test whether the difference between accuracy of independent and dependent versions across 23 datasets are statistically significant. We conduct this test with the null hypothesis that the mean of the difference between the accuracy of the independent and dependent versions will be zero. We reject the null hypothesis with statistical significance value $\alpha = 0.05$, and accept that there is a significant statistical

difference in accuracy if the $p \leq \alpha$. Table 1 shows the p-value for each dataset and the bold values mark the p-values for which there is a significant difference. We also report the adjusted α value after Holm–Bonferroni corrections, α_{HB}. Before Holm–Bonferroni correction, out of the 23 datasets, we observe that for 7 datasets there is a statistically significant difference in accuracy between independent and dependent measures. After Holm–Bonferroni correction, we still find 3 statistically significant differences (where $p \leq \alpha_{HB}$) and so conclude there are indeed datasets that are inherently best suited to either independent or dependent treatment.

Table 1 p-values for two-sided Signed Rank Wilcoxon test with $\alpha = 0.05$ (significant values are in bold face) and α values adjusted for multiple testing using the Holm-Bonferonni correction.

dataset	p-value	α_{HB}
BasicMotions	**0.0020**	**0.0021**
PEMS-SF	**0.0020**	**0.0022**
Handwriting	**0.0020**	**0.0023**
HandMovementDirection	**0.0059**	0.0024
LSST	**0.0059**	0.0025
Epilepsy	**0.0103**	0.0026
ArticularyWordRecognition	**0.0282**	0.0028
PhonemeSpectra	0.1794	0.0029
PenDigits	0.1934	0.0031
StandWalkJump	0.1988	0.0033
DuckDuckGeese	0.2324	0.0036
AtrialFibrillation	0.2820	0.0038
FingerMovements	0.3071	0.0042
Heartbeat	0.4316	0.0045
Cricket	0.5043	0.0050
SelfRegulationSCP2	0.5566	0.0056
SelfRegulationSCP1	0.5748	0.0063
RacketSports	0.6094	0.0071
ERing	0.6101	0.0083
NATOPS	0.7695	0.0100
UWaveGestureLibrary	0.7695	0.0125
Libras	0.8457	0.0167
EthanolConcentration	1.0000	0.0250

6 Conclusion

In this paper, we present multivariate versions of seven commonly used elastic similarity measures. Our approach is inspired by independent and dependent DTW measures, which have proven very successful as strategies for extending univariate DTW to the multivariate case.

These measures can be used in a wide range of time series analysis tasks including classification, clustering, anomaly detection, indexing, subsequence

search and segmentation. This study demonstrates their utility for time series classification. Our experiments show that each of the univariate similarity measures excels at nearest neighbor classification on different datasets, highlighting the importance of having a range of such measures in our analytic toolkits.

It has been shown that there are datasets for which the independent version of DTW is more accurate than the dependent version and vice versa. Until now there was no way to determine whether this is a result of a fundamental difference between treating each dimension independently or not, or whether it arises from other properties of the algorithms. Our results showing that there are some datasets for which dependent or independent treatments are consistently superior across all distance measures provides strong support for the conclusion that it is a fundamental property of the datasets, that either the variables are best considered as a single multivariate point at each time step or are not.

We observe that the dependent strategy tends to perform poorly when there are a large number of dependent dimensions in the dataset. Addressing this limitation may be a productive direction for future research.

Another potential direction for future research is to ensemble these measures in a nearest-neighbor based classifier such as EE [5] or a tree-based classifier (similar to Proximity Forest [16]) or TS-CHIEF [18]).

Acknowledgment

This research was supported by the Australian Research Council under grant DP210100072. This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-18-1-4030.

The authors would like to thank Prof. Anthony Bagnall and everyone who contributed to the UEA multivariate time series classification archive.

References

1. H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.
2. D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16. Seattle, WA, USA:, 1994, pp. 359–370.
3. S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.
4. T. W. Liao, "Clustering of time series data—a survey," *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

5. J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 565–592, 2015.
6. A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.
7. H. Izakian and W. Pedrycz, "Anomaly detection and characterization in spatial time series data: A cluster-centric approach," *IEEE Transactions on Fuzzy Systems*, vol. 22, no. 6, pp. 1612–1624, 2014.
8. M. Steiger, J. Bernard, S. Mittelstädt, H. Lücke-Tieke, D. Keim, T. May, and J. Kohlhammer, "Visual analysis of time-series similarities for anomaly detection in sensor networks," in *Computer graphics forum*, vol. 33, no. 3. Wiley Online Library, 2014, pp. 401–410.
9. D. Gunopulos and G. Das, "Time series similarity measures and time series indexing," *Acm Sigmod Record*, vol. 30, no. 2, p. 624, 2001.
10. S. Park, S.-W. Kim, and W. W. Chu, "Segment-based approach for subsequence searches in sequence databases," in *Proceedings of the 2001 ACM symposium on Applied computing*, 2001, pp. 248–252.
11. C. Cassisi, P. Montalto, M. Aliotta, A. Cannata, and A. Pulvirenti, "Similarity measures and dimensionality reduction techniques for time series data mining," *Advances in data mining knowledge discovery and applications'(InTech, Rijeka, Croatia, 2012,*, pp. 71–96, 2012.
12. H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proc. of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.
13. E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks," *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02*, p. 102, 2002.
14. M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, and E. Keogh, "Generalizing DTW to the multi-dimensional case requires an adaptive approach," *Data Mining and Knowledge Discovery*, vol. 31, no. 1, pp. 1–31, 2017.
15. A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh, "The UEA multivariate time series classification archive, 2018," *arXiv preprint arXiv:1811.00075*, 2018.
16. B. Lucas, A. Shifaz, C. Pelletier, L. O'Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. I. Webb, "Proximity Forest: an effective and scalable distance-based classifier for time series," *Data Mining and Knowledge Discovery*, vol. 33, no. 3, pp. 607–635, May 2019.
17. J. Lines, S. Taylor, and A. Bagnall, "Time series classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 12, no. 5, p. 52, 2018.
18. A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, "TS-CHIEF: a scalable and accurate forest algorithm for time series classification," *Data*

Mining and Knowledge Discovery, vol. 34, no. 3, pp. 742–775, 2020.
19. F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.
20. H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Information Sciences*, vol. 239, pp. 142–153, 2013.
21. M. Middlehurst, J. Large, and A. Bagnall, "The canonical interval forest (CIF) classifier for time series classification," *arXiv preprint arXiv:2008.09172*, 2020.
22. J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, "Classification of time series by shapelet transformation," *Data Mining and Knowledge Discovery*, vol. 28, no. 4, pp. 851–881, 2014.
23. I. Karlsson, P. Papapetrou, and H. Boström, "Generalized Random Shapelet Forests," *Data Mining and Knowledge Discovery*, vol. 30, no. 5, pp. 1053–1085, 2016.
24. J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012.
25. P. Schäfer, "The BOSS is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.
26. P. Schäfer and U. Leser, "Fast and accurate time series classification with WEASEL," in *Proceedings of the 2017 ACM on Conf. on Information and Knowledge Management (CIKM)*, 2017, pp. 637–646.
27. T. Le Nguyen, S. Gsponer, I. Ilie, M. O'Reilly, and G. Ifrim, "Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 1183–1222, 2019.
28. A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, 2020.
29. C. Gallicchio and A. Micheli, "Deep Echo State Network (DeepESN): a brief survey," *arXiv preprint arXiv:1712.04323*, 2017.
30. Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 1578–1585.
31. H. I. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "InceptionTime: Finding AlexNet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, 2020.
32. H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

33. A. Bagnall, M. Flynn, J. Large, J. Lines, and M. Middlehurst, "On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (HIVE-COTE v1. 0)," in *International Workshop on Advanced Analytics and Learning on Temporal Data*. Springer, 2020, pp. 3–18.
34. A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, "The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, pp. 1–49, 2020.
35. P. Schäfer and U. Leser, "Multivariate time series classification with WEASEL+MUSE," *arXiv preprint arXiv:1711.11343*, 2017.
36. M. Middlehurst, W. Vickers, and A. Bagnall, "Scalable dictionary classifiers for time series classification," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2019, pp. 11–19.
37. M. Löning, A. Bagnall, S. Ganesh, V. Kazakov, J. Lines, and F. J. Király, "sktime: A Unified Interface for Machine Learning with Time Series," in *Workshop on Systems for ML at NeurIPS 2019*.
38. X. Zhang, Y. Gao, J. Lin, and C.-T. Lu, "Tapnet: Multivariate time series classification with attentional prototypical network." in *AAAI*, 2020, pp. 6845–6852.
39. H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.
40. C. W. Tan, M. Herrmann, G. Forestier, G. I. Webb, and F. Petitjean, "Efficient search of the best warping window for dynamic time warping," in *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018, pp. 225–233.
41. E. Keogh, L. Wei, X. Xi, M. Vlachos, S. H. Lee, and P. Protopapas, "Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures," *VLDB Journal*, vol. 18, no. 3, pp. 611–630, 2009.
42. D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern Recognition*, vol. 42, no. 9, pp. 2169–2180, 2009.
43. C. W. Tan, G. I. Webb, and F. ç. o. Petitjean, "Indexing and classifying gigabytes of time series under time warping," in *Proceedings of the 2017 SIAM Int. Conf. on Data Mining*. SIAM, 2017, pp. 282–290.
44. M. Herrmann and G. I. Webb, "Early abandoning PrunedDTW and its application to similarity search," *arXiv preprint arXiv:2010.05371*, 2020.
45. E. J. Keogh and M. J. Pazzani, "Derivative Dynamic Time Warping," *Proceedings of the 2001 SIAM Int. Conf. on Data Mining*, pp. 1–11, 2001.
46. Y. S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted Dynamic Time Warping for time series classification," *Pattern Recognition*, vol. 44, no. 9, pp. 2231–2240, 2011.

47. D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM*, vol. 24, no. 4, pp. 664–675, 1977.
48. M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh, "Indexing multi-dimensional time-series with support for multiple distance measures," in *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 216–225.
49. ——, "Indexing multidimensional time-series," *The VLDB Journal*, vol. 15, no. 1, pp. 1–20, 2006.
50. A. Stefan, V. Athitsos, and G. Das, "The move-split-merge metric for time series," *IEEE Trans. on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1425–1438, 2013.
51. L. Chen and R. Ng, "On The Marriage of Lp-norms and Edit Distance," in *Proceedings of the 13th Int. Conf. on Very Large Data Bases (VLDB)*, 2004, pp. 792–803.
52. L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05*, pp. 491–502, 2005.
53. P.-F. Marteau, "Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 306–318, 2009.
54. J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.
55. A. Benavoli, G. Corani, and F. Mangili, "Should we really use post-hoc tests based on mean-ranks?" *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 152–161, 2016.

A Summary of the Datasets

Table 2 Summary of the 23 fixed-length multivariate datasets we used from the UAE repository.

#	dataset	code	trainsize	testsize	dims	length	classes
1	ArticularyWordRecognition	AWR	275	300	9	144	25
2	AtrialFibrillation	AF	15	15	2	640	3
3	BasicMotions	BM	40	40	6	100	4
4	Cricket	CR	108	72	6	1197	12
5	DuckDuckGeese	DDG	50	50	1345	270	5
6	Epilepsy	EP	137	138	3	206	4
7	EthanolConcentration	EC	261	263	3	1751	4
8	ERing	ER	30	270	4	65	6
9	FingerMovements	FM	316	100	28	50	2
10	HandMovementDirection	HMD	160	74	10	400	4
11	Handwriting	HW	150	850	3	152	26
12	Heartbeat	HB	204	205	61	405	2
13	Libras	LIB	180	180	2	45	15
14	LSST	LSST	2459	2466	6	36	14
15	NATOPS	NATO	180	180	24	51	6
16	PenDigits	PD	7494	3498	2	8	10
17	PEMS-SF	PEMS	267	173	963	144	7
18	Phoneme	PS	3315	3353	11	217	39
19	RacketSports	RS	151	152	6	30	4
20	SelfRegulationSCP1	SRS1	268	293	6	896	2
21	SelfRegulationSCP2	SRS2	200	180	7	1152	2
22	StandWalkJump	SWJ	12	15	4	2500	3
23	UWaveGestureLibrary	UW	120	320	3	315	8

B Accuracy of Dependent and Independent Measures

Table 3 Accuracy of independent similarity measures. Note that dtwf and ddtwf refers to measures that use full window.

dataset	L_2	dtwf	dtw	ddtwf	ddtw	wdtw	wddtw	lcss	msm	erp	twe
AWR	0.97	0.98	0.98	0.60	0.69	**0.99**	0.70	**0.99**	**0.99**	**0.99**	0.98
AF	0.27	0.27	0.27	0.07	0.07	0.13	0.27	0.20	0.20	0.27	**0.33**
BM	0.60	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.85	**1.00**	0.85	**1.00**
CR	0.92	0.99	**1.00**	0.96	0.96	0.99	0.96	0.97	0.99	0.96	0.99
DDG	0.34	0.48	0.48	0.54	0.54	0.48	0.54	0.34	**0.60**	0.34	**0.60**
EP	0.67	0.98	0.95	0.96	0.96	0.96	0.96	**0.99**	**0.99**	0.90	0.98
ER	**0.95**	0.92	0.92	0.81	0.91	0.93	0.84	**0.95**	0.89	0.94	0.91
EC	0.32	0.30	0.31	0.29	0.27	0.29	0.27	0.27	**0.33**	**0.33**	0.30
FM	0.56	0.52	0.54	**0.62**	0.54	0.51	0.59	0.52	0.51	0.51	0.52
HMD	0.28	0.30	0.23	0.34	0.34	0.26	0.34	0.26	0.32	0.23	**0.42**
HW	0.34	**0.51**	0.48	0.31	0.34	**0.51**	0.33	0.46	0.49	0.41	0.37
HB	0.66	0.66	0.69	0.69	0.69	0.69	0.69	**0.75**	**0.75**	0.65	0.72
LIB	0.82	0.89	0.89	0.90	0.90	0.89	**0.92**	0.84	0.86	0.82	0.89
LSST	0.45	**0.58**	**0.58**	0.49	0.48	**0.58**	0.49	0.34	0.55	0.46	0.53
NATO	0.82	0.85	**0.87**	0.82	0.83	0.86	0.82	0.81	0.83	0.82	0.83
PEMS	0.77	0.73	0.77	0.62	0.61	0.76	0.64	**0.83**	0.77	0.73	0.79
PD	**0.98**	0.94	**0.98**	0.95	0.97	0.96	0.96	0.96	0.96	0.97	0.94
PS	0.10	0.15	0.15	0.16	0.17	0.16	0.17	0.15	**0.18**	0.10	0.17
RS	0.80	0.84	0.84	0.77	0.76	0.86	0.78	**0.89**	0.82	0.83	0.79
SRS1	**0.78**	0.76	**0.78**	0.63	0.58	**0.78**	0.56	0.75	0.77	**0.78**	**0.78**
SRS2	0.48	0.53	0.54	0.49	0.51	0.53	0.56	0.51	0.51	0.50	**0.57**
SWJ	0.27	0.33	0.33	0.40	0.33	**0.53**	0.20	0.27	0.27	0.33	0.20
UW	0.88	0.87	**0.91**	0.72	0.84	0.88	0.80	**0.91**	0.88	0.90	0.88
Wins	3	3	7	2	1	6	2	7	7	3	6

Table 4 Accuracy of dependent similarity measures. Note that dtwf and ddtwf refers to measures that use full window.

dataset	L_2	dtwf	dtw	ddtwf	ddtw	wdtw	wddtw	lcss	msm	erp	twe
AWR	0.97	**0.99**	0.98	0.35	0.34	**0.99**	0.36	0.98	0.98	0.98	0.97
AF	0.27	0.20	0.27	0.13	**0.33**	0.20	0.27	**0.33**	0.27	0.27	0.13
BM	0.60	**0.97**	**0.97**	0.95	0.95	**0.97**	0.95	0.80	0.68	0.75	0.95
CR	0.94	**1.00**	**1.00**	0.75	0.78	**1.00**	0.75	0.99	**1.00**	0.97	0.97
DDG	0.50	**0.58**	**0.58**	0.32	0.32	**0.58**	0.32	0.42	0.26	0.36	0.26
EP	0.63	**0.96**	**0.96**	0.93	0.93	**0.96**	0.93	0.92	0.94	0.87	0.94
ER	0.94	0.91	0.94	0.79	0.84	0.93	0.83	0.93	0.90	0.94	**0.95**
EC	0.32	0.32	0.32	0.24	0.27	0.30	0.25	0.32	**0.35**	0.31	0.25
FM	0.55	0.53	0.53	0.51	0.51	0.54	0.51	0.50	0.51	**0.56**	0.50
HMD	0.26	0.19	0.24	0.27	0.24	0.23	0.27	0.23	0.16	0.22	**0.35**
HW	0.33	**0.61**	**0.61**	0.42	0.42	**0.61**	0.41	0.54	0.57	0.47	0.45
HB	0.62	0.72	0.70	0.71	0.71	0.68	0.71	0.62	0.68	0.74	**0.75**
LIB	0.83	0.87	0.87	**0.98**	**0.98**	0.88	**0.98**	0.33	0.85	0.83	0.86
LSST	0.45	**0.55**	**0.55**	0.43	0.43	**0.55**	0.43	0.36	0.36	0.45	0.45
NATO	0.84	**0.88**	**0.88**	0.85	0.85	**0.88**	0.79	0.72	0.81	0.84	0.82
PEMS	**0.73**	0.71	**0.73**	0.57	0.57	**0.73**	0.57	0.12	0.71	0.72	0.71
PD	**0.98**	**0.98**	**0.98**	0.97	0.97	**0.98**	0.97	0.95	0.94	**0.98**	**0.98**
PS	0.10	0.15	0.15	0.16	**0.17**	0.16	**0.17**	0.15	0.16	0.10	**0.17**
RS	0.82	0.80	0.82	0.78	0.80	0.85	0.78	**0.89**	**0.89**	0.74	0.78
SRS1	0.78	0.77	0.78	0.58	0.57	0.76	0.57	**0.80**	0.78	0.78	0.42
SRS2	0.48	**0.54**	**0.54**	0.48	0.52	**0.54**	0.49	0.47	0.52	0.49	**0.54**
SWJ	0.20	0.20	0.20	0.20	**0.33**	**0.33**	0.20	**0.33**	**0.33**	0.27	0.27
UW	0.88	**0.90**	**0.90**	0.79	0.80	**0.90**	0.83	0.89	0.88	0.89	0.88
Wins	2	11	11	1	4	13	2	4	4	2	6

5.2 Contributions

Contributions made in this chapter are:

- We extend seven commonly used univariate elastic similarity measures to the multivariate case by adopting two strategies — using multiple dimensions independently and dependently. Univariate versions of these seven measures have been widely used in time series analysis tasks such as classification, clustering, indexing, segmentation, anomaly detection and subsequence search. Therefore, we hope that our multivariate versions would be of great benefit to the time series analysis community. This extension is straightforward in several cases, but non-trivial for others.
- We demonstrate the utility of these measures by focusing on the multivariate nearest neighbor classification problem. We compare the classification performance of these measures on 23 datasets from the University of East Anglia (UEA) multivariate archive. We show that each measure outperforms all others on at least one task (or ties with the highest performing measures in two cases), demonstrating the value of having a suite of alternatives multivariate elastic measures.
- We demonstrate that there are some classification tasks for which the independent strategy is superior across all measures and others for which the dependent strategy is better. It was previously demonstrated for DTW that each approach sometimes substantially outperformed the other [42]. However, it has not previously been established whether this is a fundamental property of the tasks per se. In this paper we provide evidence that in at least some cases there is a fundamental connection between classification tasks and whether they are best addressed by considering all variables at each time step in conjunction or by considering each variable independently of the others.

Chapter 6

Multivariate Elastic Ensembles

6.1 Introduction

In Chapter 5, I presented seven similarity measures for multivariate time series analysis. They use two strategies to combine the dimensions when computing the similarity between multivariate time series (*i.e.* independent and dependent measures). In this chapter, I present ensembles formed from these measures using strategies similar to univariate Elastic Ensemble and Proximity Forest.

This chapter addresses the following objectives:

- Develop and evaluate three versions of a multivariate Elastic Ensemble employing, independent, dependent and both types of multivariate similarity measures. Since Elastic Ensemble is not a scalable classifier, my goal of presenting a multivariate Elastic Ensemble is to establish a baseline for comparison for similarity-based multivariate methods in terms of accuracy.
- Create three similar versions of multivariate Proximity Forest and compare their performance against each other and state-of-the-art multivariate classifiers.
- Investigate the impact on performance of using all dimensions or a subset of dimensions when training. Investigate and compare three strategies to select between these subsets.

Unlike the previous three chapters, the works presented in this chapter have not yet been submitted for publication. I plan to submit this work for publication post thesis submission.

6.2 Background

From Section 2.5, recall that a variety of methods have been developed for multivariate TSC. When learning, some of these methods only use independent or dependent strategy, and some consider all dimensions or select a random subset of dimensions. For example, both DTW_I and DTW_D use all dimensions together [42], meanwhile, ROCKET and HIVE-COTEv2 can select a subset of dimensions to use. HIVE-COTEv2 components such as STC and DrCIF, which selects a random dimension at each node of a tree, only supports the independent strategy. Meanwhile, some methods such as ROCKET, only use a dependent strategy (Note that ROCKET's dependent strategy is different from the strategy used in DTW_D – see Section 2.5.6).

When using the dependent strategy (see Section 2.5.1) it is important to choose how many dimensions to combine together. Particularly when the dimensionality is large, there are various issues we must consider. These include, 1) it is unlikely that all dimensions are coupled together, 2) or contain useful information for discrimination, 3) and theoretical limitations due to the "curse of dimensionality" problem [110, 111].

Intuitively, it makes sense to think that the optimal set of dimensions will be a subset of dimensions. For example, consider a dataset with 6 dimensions of accelerometer data from left hand and right hand. Each subsets of 3 dimensions correspond to X,Y, and Z axis of either left or right hand. It is likely that there could be some coupling between X,Y, and Z of left or right hand. There could also be another coupling between X of left hand and X of right hand, and Y of left hand and Y of right hand, and so on. Therefore, we could hypothesize that a strategy that selects a subset size and then samples a set of dimensions to be included in this subset could be expected to perform better than using all dimensions together. In practice, exploring all such combinations of subsets is computationally infeasible because it grows factorially large. Therefore, in this study, we explore selecting random subsets of dimensions as described in Section 6.3.3.

If the dimensionality is reasonably small, which might be the case for many datasets, it could be worth while to explore systematic strategies for selection of dimensions, such as forward sequential selection or backward sequential elimination [112].However, we leave such an endeavor to future work.

Its also important to highlight that studying multivariate TSC classifiers that use all dimensions using the UEA Multivariate TS Archive [5] is a particularly problematic issue because some of the datasets have extremely high dimensionality. Eight out of the 30 datasets have 50 or more dimensions. One dataset, *PEMS-SF*, has 963 dimensions, while another dataset, *DuckDuckGeese*, has 1,345 dimensions. Its very likely that many

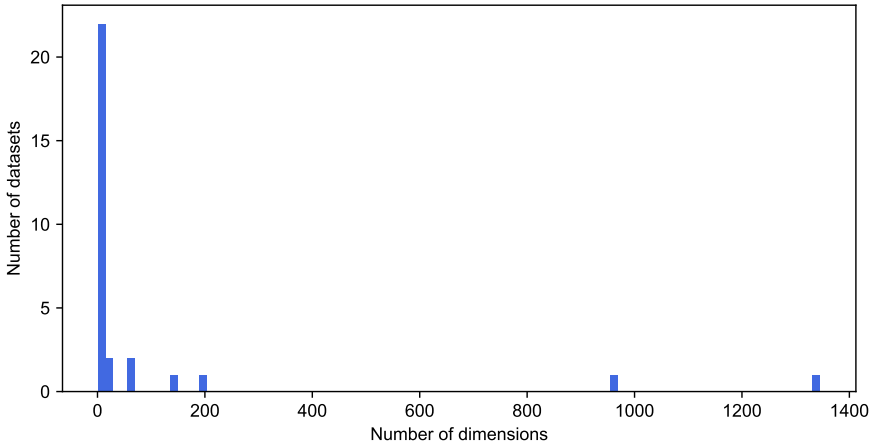


FIGURE 6.1: Distribution of the dimensions of the 30 datasets available in UEA Multivariate TS Archive [5].

of these dimensions contain little or redundant information. Figure 6.1 shows a plot of the distribution of dimensions of the datasets.

6.3 Similarity-based Multivariate TSC Ensembles

In this section I present the two ensembles created using the multivariate similarity measures presented in Chapter 5.

6.3.1 Multivariate Elastic Ensemble

The first ensemble is constructed to be similar to the Elastic Ensemble [15]. We call this classifier Multivariate Elastic Ensemble (MEE). From Section 2.4.1, recall that EE is a 1-NN based ensemble of 11 similarity measures. We keep the design of the ensemble similar to the univariate EE, except that MEE uses our multivariate similarity measures. Similar to EE, MEE also uses leave-one-out cross validation of 100 parameters when choosing the parameters for similarity measures. Both EE and MEE predict the final label by using majority voting. The voting is weighted by the leave-one-out cross validation accuracy of each measure on the training set. Any ties are broken using a uniform random choice. All measures used in MEE use all dimensions in the dataset, similarly to DTW_I and DTW_D [42].

As we discussed in Chapter 1, EE is not designed for scalability. Similarly, the goal of MEE is not to tackle the scalability issue but to create a multivariate similarity based-classifier as a baseline for accuracy comparison with other similarity-based multivariate classifiers. I also note that measures such as DTW can be scaled to millions of time

series when used in conjunction with lower bounding and early abandoning [3, 33, 34]. Therefore, if various research on lower bounding and early abandoning of other similarity measures [51–54] are combined and extended to multivariate measures, then it might be possible to create a more scalable version of MEE.

We explore three variations of MEE, which are constructed as follows.

- MEE_I : An ensemble of 1-NN classifiers formed using 10 independent multivariate similarity measures and multivariate Euclidean distance[1].
- MEE_D : An ensemble of 1-NN classifiers formed using 10 dependent multivariate similarity measures and multivariate Euclidean distance[1].
- MEE_{ID} : An ensemble of 1-NN classifiers formed using 10 independent measures, 10 dependent measures and multivariate Euclidean distance.

6.3.2 Multivariate Proximity Forest

The second ensemble is constructed to be similar to Proximity Forest (Chapter 3, hence named Multivariate Proximity Forest (MPF). Everything is kept similar to Proximity Forest, except the following changes:

- Univariate similarity measures are replaced with multivariate versions presented in Chapter 5.
- Similar to MEE, three variations MPF_I, MPF_D and MPF_{ID} are implemented as follows:
  - MPF_I : which selects a measure uniformly at random, out of 11 measures (10 independent measures and Euclidean) per candidate split at the node .
  - MPF_D : which selects a measure uniformly at random, out of 11 (10 dependent measures and Euclidean) measures per candidate split at the node.
  - MPF_{ID} : which selects a measure uniformly at random, out of 21 measures (10 independent, 10 dependent and Euclidean) per candidate split at the node.
- An additional parameter, *dims*, is added to choose a strategy to select the size of subsets of dimensions to use at a node. Once the size of subset is selected, dimensions to be included in the subset are sampled uniformly at random (without

[1]Recall that Euclidean distance does not vary between dependent and independent forms (see Section 5.4.1.2).

replacement) from all available dimensions. For example, in a dataset with 9 dimensions, and if the selected subset size is 3, then three dimensions from 1 to 9 are selected uniformly at random. The three strategies we explored are:

- $dims = all$: Uses all dimensions.
- $dims = uniform$: Selects the subset of dimensions based on the random value from a uniform distribution, *i.e.* number of dimensions, $dims \quad U(1, D)$, where D is the total number of dimensions.
- $dims = beta13sqrt$: Selects the subset size randomly using a modified version of beta distribution as explained in Section 6.3.3.

6.3.3 On the choice of Beta Distribution

The beta distribution is chosen because random values sampled from the beta distribution are bounded to the interval $[0, 1]$. This allows its upper bound to be easily adjusted by multiplying with D to $[0, D]$. The beta distribution has two parameters, α and β, which controls the shape of the curve. By setting $\alpha = 1$ and $\beta = 1$, we can obtain a flat distribution. By increasing β, we can obtain a distribution with lower probabilities of obtaining a larger value as shown in Figure 6.2. This is very useful because we want the probability of selecting a very high subset size to be extremely low. By default we choose $\beta = 3$ because it gives the desired shape without making large subsets too improbable. This parameter is worthy of further investigation, but there was no opportunity to do so within the scope of this thesis.

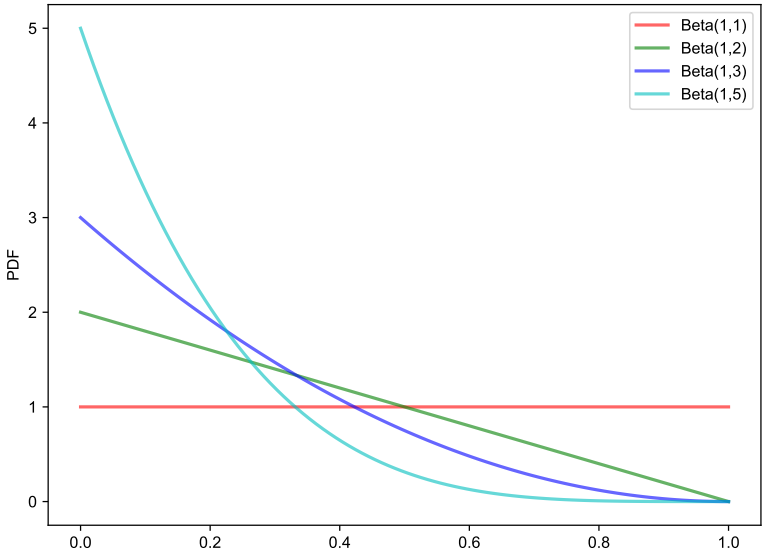


FIGURE 6.2: Shape of beta distribution as its parameter β increases.

However, direct implementation of this scheme would mean that the highest probability subset size was one. To avoid this we modify the distribution to make the left side of it flat. We introduced a threshold such that if the sampled value from beta distribution is less than $\frac{p}{D}$ then we fall back to sampling from the uniform distribution, as described in Algorithm 4.

Algorithm 4: Modified_Beta(α, β, D)

Input: α: parameter of Beta distribution
Input: β: parameter of Beta distribution
Input: D: Number of dimensions
Output: *dims*: Random subset size

```
// Sample from beta distribution
dims  Beta(α, β)  D
if dims < p/D then
    // Sample from uniform distribution
    return dims  U(1, p/D)
return dims
```

6.4 Experiments

Experimental results for MEE are obtained by synthesis from the results of the single multivariate measure experiments described in Section 5.1. While conducting those experiments, for all of the measures, I saved the classification output of every time series in each dataset for both train and test sets. Similar to the methodology used in EE, I calculated the predictions for test instances by majority voting and then weighting the vote by the training accuracy of each measure. Experiments were conducted on the standard train/test splits of UEA Multivariate TS Archive [5], provided in timeseriesclassification.com. Note that in Section 5 we reported the results of 23 datasets that completed the training within the time limits we had.

MPF experiments were run on a cluster of mixed hardware with Intel Xeon-E5-2680-v3 and Intel Xeon-E5-2680-v4 CPUs using 32-threads. Once again, I use the standard train/test splits available in UEA Multivariate TS Archive [5].

For benchmarking of classifiers I follow the methodology described in Section 2.6.

6.4.1 Multivariate Measures *vs* MEE

EE showed that 1-NN ensembles formed using a diverse set of similarity measures are more accurate than any of the single measures on the 85 univariate UCR datasets [9, 15].

In this experiment we replicate this evaluation in the multivariate context. We compare single independent and dependent measures with MEE ensembles to assess whether each of the three versions of MEE are significantly different to the individual measures.

Figures 6.3 and 6.4 shows accuracy ranking of independent measures and MEE_I; and dependent measures and MEE_D, respectively. Comparison in Figure 6.3 indicates that the following pairs are significantly different: $DDTW_I$ and MEE_I, ERP_I and MEE_I, $DTWF_I$ and MEE_I, DTW_I and ERP_I, $DDTWF_I$ and MEE_I. Figure 6.4 indicates that the following pairs are significantly different: $DDTWF_D$ and $WDTW_D$, $DDTWF_D$ and MEE_D, MEE_D and $WDDTW_D$, $DDTWF_D$ and DTW_D, $DDTWF_D$ and $DTWF_D$. Both comparisons indicate that MEE_I and MEE_D are more accurate than individual measures with 1-NN.

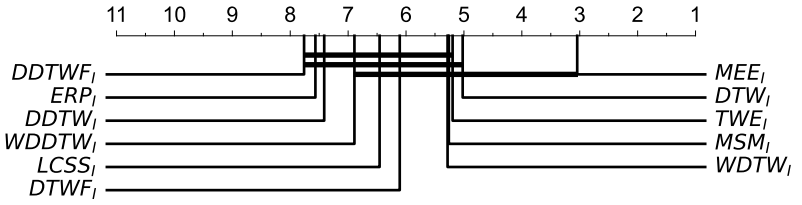


FIGURE 6.3: Average accuracy ranking diagram showing the ranks on the error rate of the independent similarity measures and MEE_I.

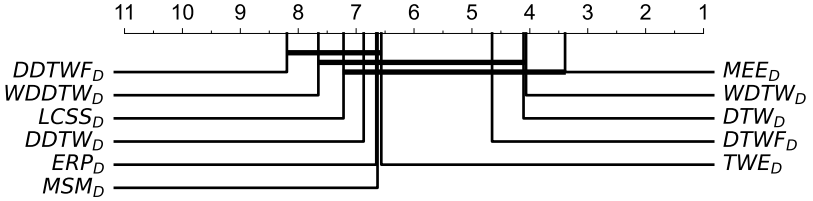


FIGURE 6.4: Average accuracy ranking diagram showing the ranks on the error rate of the dependent similarity measures and MEE_D.

Figure 6.5 shows accuracy ranking of our three ensembles *vs* the top seven (out of twenty-one) individual similarity measures with 1-NN (selected based the rankings shown in Figure 5.3). We can observe that all ensembles are more accurate than the classifiers using a single measure. We found that the EE_{ID} (avg. rank 3.2609) performs better than both EE_I (avg. rank 3.9565) and EE_D (avg. rank 5.5435). Based on the p-values, we found that all pairs of classifiers are statistically indistinguishable except the following pairs: $DTWF_D$ and MEE_{ID} (p-value=0.0003), $DTWF_I$ and MEE_{ID} (p-value=0.0005), $DTWF_I$ and MEE_I (p-value=0.0006), MEE_D and MEE_{ID} (p-value=0.0006) and MEE_{ID} and $WDTW_D$ (p-value=0.0007). All other pairs had a

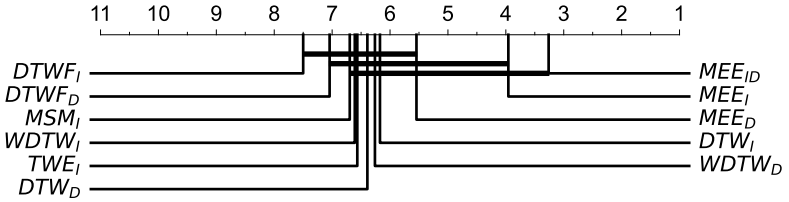


FIGURE 6.5: Average accuracy ranking diagram showing the ranks on the error rate of the top seven similarity measures and three variants of MEE.

p-value above 0.0012 (after the Holm-Bonferroni correction) indicating that they are not significantly different. One reason why so few of the pairs are assessed as significantly different could be because we are limited to just 23 datasets.

6.4.2 Comparing the Variations of MPF

Experiments for Multivariate Proximity Forest were conducted with ensemble size $k = 500$ and number of candidate splits $C_e = 5$. As explained in Section 6.3.2, we developed three variants of MPF, namely, MPF_I, MPF_D and MPF_{ID}. We also use an additional parameter called *dims*, which defines the number of dimensions to use for each candidate split. In this section I present comparisons of accuracy and total training and test times between different variations of MPF and its settings.

6.4.2.1 Comparing the Three Ensembles

When comparing the three ensembles without changing the *dims* parameter, generally MPF_{ID} performs better than MPF_I and MPF_D. The magnitude of these wins and losses are usually marginal, especially between MPF_I and MPF_D, so for brevity, I have omitted including the scatter plots of all combinations. Instead, these results are summarized in Table 6.1.

6.4.2.2 Comparing the Strategies to Select Dimensions

We then compared the three settings $dim = all$, $dim = uniform$ and $dim = beta13sqrt$ for each of the three MPF ensembles. For example, MPF_I $(dim = all)$ *vs* MPF_I $(dim = uniform)$. Comparisons between $dim = all$ *vs* $dim = uniform$ is very similar across the three ensembles, while total training time is approximately reduced to half (see the timings in Table 6.1).

Setting(X)	Setting(Y)	Win/Draw/Loss	Total Time (hr)
MPF_I (all)	MPF_D (all)	7/6/10	42.4*vs* 39.8
MPF_I (all)	MPF_{ID} (all)	7/2/14	42.4 *vs* 46.6
MPF_D (all)	MPF_{ID} (all)	6/3/14	39.8 *vs* 46.6
MPF_I $(uniform)$	MPF_D $(uniform)$	13/4/6	21.9 *vs* 25.3
MPF_I $(uniform)$	MPF_{ID} $(uniform)$	9/3/11	21.9 *vs* 26.6
MPF_D $(uniform)$	MPF_{ID} $(uniform)$	9/3/11	25.3 *vs* 26.6
MPF_I $(beta13sqrt)$	MPF_D $(beta13sqrt)$	8/5/10	13.0 *vs* 13.3
MPF_I $(beta13sqrt)$	MPF_{ID} $(beta13sqrt)$	10/3/10	13.0 *vs* 16.9
MPF_D $(beta13sqrt)$	MPF_{ID} $(beta13sqrt)$	10/2/11	13.3 *vs* 16.9

TABLE 6.1: A comparison of accuracy and total training and test time of variants of MPF on 23 UEA multivariate datasets. In this table, for win/draw/loss, wins are counted for Setting(X) *vs* Setting(Y).

Comparison between using all dimensions $dim = all$ and selecting a subset using a modified beta distribution $dim = beta13sqrt$ is shown in Figure 6.6. It shows that total time is reduced approximately 2 to 3 times with similar accuracy when using a subset of dimensions (see also Table 6.1). The fastest configuration took approximately 13 hours in total. Recall from Chapter 5.5.1 that training all measures used in MEE took 1648 hours on 32 CPU threads.

We did not observe a substantive difference in accuracy when comparing $dim = all$ and $dim = beta13sqrt$ across the three ensembles. While this was somewhat unexpected, maintaining a competitive accuracy with much lower training and test time makes $dim = beta13sqrt$ the preferred configuration. Based on our intuition, we had expected that using all dimensions would reduce the accuracy in most cases. The reason for this counter-intuitive observation could be due to the characteristics of the datasets we used.

It would be an important future work to investigate this further to understand if these difference arise in other multivariate classifiers that use all dimensions *vs* a subset of the dimensions.

Overall, we conclude that MPF_{ID} with $dim = beta13sqrt$, $k = 500$ and $C_e = 5$ produced the best performance results for tested 23 datasets from the UEA Multivariate TS Archive.

6.4.3 MEE and MPF *vs* SOTA Multivariate TSC Algorithms

Next, we compare the most accurate versions of MEE and MPF with state-of-the-art multivariate TSC algorithms [6, 7] (see Chapter 2.5). Figure 6.7 shows the accuracy

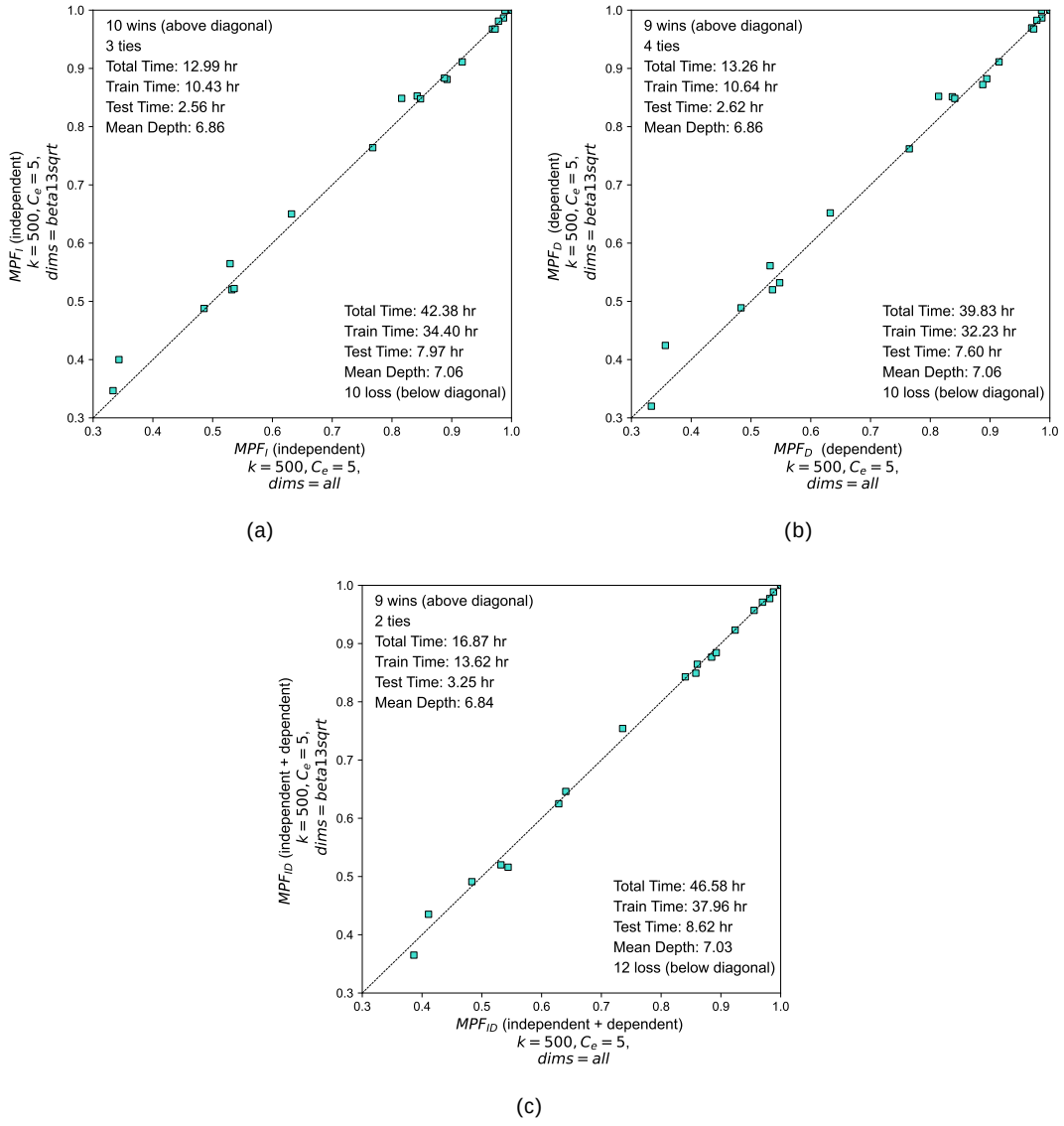


FIGURE 6.6: Scatter plots showing the accuracy of Multivariate Proximity Forest ($k = 500, C_e = 5$) with three variations that combine different types of measures and two strategies to select dimensions. a) MPF_I, $dims = all$ *vs* MPF_I, $dims = beta13sqrt$ b) MPF_D, $dims = all$ *vs* MPF_D, $dims = beta13sqrt$ c) MPF_{ID}, $dims = all$ *vs* MPF_{ID}, $dims = beta13sqrt$.

ranking of our ensembles *vs* 8 of these algorithms. I have kept the two classic measures DTW_I and DTW_D [2] to show the relative performance of more complex SOTA methods.

The most accurate algorithm, HIVE-COTEv2, obtained an average rank of 3.3043. It is followed by ROCKET (ranked 3.9565), DrCIF (ranked 4.3478), InceptionTime (4.6957) and then our two ensembles MEE_{ID} (5.4565) and MPF_{ID} (5.6957), which are in turn ahead of the remaining single strategy learners.

[2] This result for DTW_I and DTW_D obtained from Ruiz et al. [6], which use 20% of series length as window size. Note that its slightly di erent than the results reported in Chapter 5 which uses leave-one-out-cross validation from 1 to 25% of length of the series to select the window size.

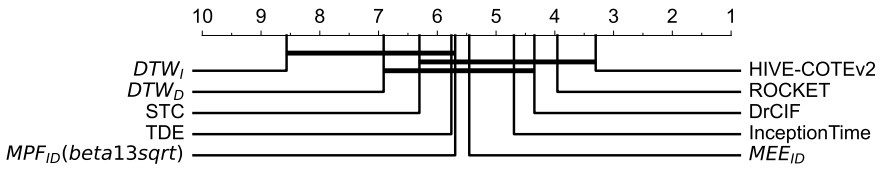


FIGURE 6.7: Average accuracy ranking diagram showing the ranks on the error rate of 8 classifiers from Ruiz et al. [6] and Middlehurst et al. [7] and the our best performing ensembles, MEE_{ID} and MPF_{ID}.

Based on the p-values, 9 pairs of classifiers are significantly different. They are: DTW_I *vs* HIVE-COTEv2, ROCKET, DrCIF, TDE, STC, InceptionTime, MEE_{ID}; and then DTW_D *vs* HIVE-COTEv2 and ROCKET. Therefore, our two ensembles, MEE_{ID} and MPF_{ID}, are statistically not different from the leading algorithm HIVE-COTEv2. Note that there are limitations of this method of comparison. Since the number of datasets is only 23, and some classifiers have correlation to others (*i.e.* STC, TDE, DrCIF are all components of HIVE-COTEv2), it might affect the groupings of statistically different pairs.

To understand the differences more clearly, Figure 6.8 compares the accuracy of MPF_{ID} and HIVE-COTEv1 (which was benchmarked with highest accuracy circa 2020) and HIVE-COTEv2 (which is currently highest rank on accuracy across the UEA multivariate archive). Figure 6.8(a) with MPF_{ID} *vs* HIVE-COTEv1, indicates a 11/1/11 win/draw/loss for MPF_{ID}. In this figure, datasets for which the two algorithms differ in accuracy by more than 5% are labeled. We observe that there are 5 datasets where the accuracy is larger than 5% for HIVE-COTE, 4 datasets where it is larger than 5% for MPF_{ID}. The magnitude of wins for HIVE-COTEv1 is larger on average. Figure 6.8(b) with MPF_{ID} *vs* HIVE-COTEv2, shows that HIVE-COTEv2 is much more accurate than MPF_{ID} with a large difference in win/draw/loss.

The two HIVE-COTEs are much more complex algorithms than MPF_{ID} that use the best interval, shapelet and dictionary-based classifiers. Our ensemble, MPF_{ID}, achieves similar accuracy to HIVE-COTEv1 while using only similarity-based methods. Recall that HOVE-COTEv1 removed EE from the ensemble because of its slow speed. However, it did not replace EE with the much more scalable Proximity Forest. Perhaps the other reason for that could be that EE and PF lacked support for multivariate time series. While further work can be done to improve MPF_{ID} in terms of both accuracy and speed, these contributions may assist in incorporating a similarity-based classifier to a future version of HIVE-COTE while supporting multivariate time series.

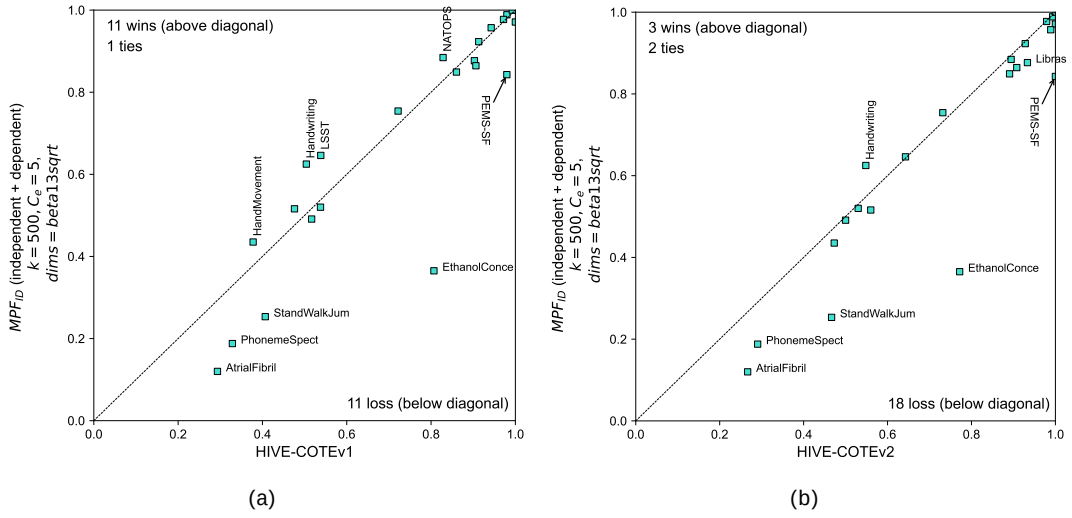


FIGURE 6.8: Scatters plots showing the accuracy of a) HIVE-COTEv1 *vs* MPF_{ID} b) HIVE-COTEv2 *vs* MPF_{ID}, on 23 datasets from UEA multivariate TS Archive.

Finally, Table 6.2 shows the accuracy of leading algorithms (selected from Figure 6.7) and our most accurate ensembles MEE_{ID} and MPF_{ID}. Across the 8 algorithms, MEE_{ID} achieves the highest accuracy for 2 datasets, and MPF_{ID} for 3 datasets. By comparison, shapelet-based STC wins on 1 dataset and dictionary-based TDE on 2 datasets. Interestingly interval-based DrCIF wins 8 times which is more than the HIVE-COTEv2 and InceptionTime.

TABLE 6.2: Accuracy of our most accurate ensembles MEE_{1D} and MPF_{1D} compared against top multivariate TSC algorithms on 23 datasets from UEA Multivariate TS Archive. Column names are shortened as follows: HC1 for for HIVE-COTEv1, IT for InceptionTime, RT for ROCKET, HC2 for HIVE-COTEv2. Wins indicate the number of time each classifier achieved the highest accuracy for each dataset.

dataset	MEE	MPF	STC	TDE	IT	DrCIF	RT	HC2
ArticularyWordRecognition	0.99	0.99	0.98	0.99	0.99	0.98	**1.00**	0.99
AtrialFibrillation	0.27	0.12	0.32	0.27	0.22	**0.33**	0.25	0.27
BasicMotions	0.97	**1.00**	0.98	**1.00**	**1.00**	**1.00**	0.99	**1.00**
Cricket	**1.00**	**1.00**	0.99	0.99	0.99	0.99	**1.00**	**1.00**
DuckDuckGeese	0.60	0.52	0.43	0.34	**0.63**	0.54	0.46	0.56
Epilepsy	0.98	0.97	0.99	0.99	0.99	0.98	0.99	**1.00**
EthanolConcentration	0.35	0.37	**0.82**	0.56	0.28	0.69	0.45	0.77
ERing	0.97	0.96	0.84	0.96	0.92	**0.99**	0.98	0.99
FingerMovements	0.57	0.52	0.53	0.56	0.56	**0.60**	0.55	0.53
HandMovementDirection	0.28	0.44	0.35	0.38	0.42	**0.53**	0.45	0.47
Handwriting	0.58	0.62	0.29	0.56	**0.66**	0.35	0.57	0.55
Heartbeat	0.75	0.75	0.72	0.75	0.73	**0.79**	0.72	0.73
Libras	0.91	0.88	0.84	0.85	0.89	0.89	0.91	**0.93**
LSST	0.60	**0.65**	0.58	0.57	0.34	0.56	0.63	0.64
NATOPS	0.88	0.88	0.84	0.84	**0.97**	0.84	0.89	0.89
PenDigits	0.98	0.98	0.98	0.94	**1.00**	0.98	1.00	0.98
PEMS-SF	0.73	0.84	0.98	**1.00**	0.83	**1.00**	0.86	**1.00**
PhonemeSpectra	0.20	0.19	0.31	0.25	**0.37**	0.31	0.28	0.29
RacketSports	0.87	0.86	0.88	0.84	0.92	0.90	**0.93**	0.91
SelfRegulationSCP1	0.77	0.85	0.85	0.81	0.85	0.88	0.87	**0.89**
SelfRegulationSCP2	**0.54**	0.49	0.52	0.50	0.52	0.49	0.51	0.50
StandWalkJump	0.33	0.25	0.44	0.33	0.42	**0.53**	0.46	0.47
UWaveGestureLibrary	0.92	0.92	0.87	0.93	0.91	0.91	**0.94**	0.93
Wins	2	3	1	2	6	8	4	6

6.5 Conclusions

This chapter introduced two similarity-based multivariate TSC algorithms: Multivariate Elastic Ensemble and Multivariate Proximity Forest.

MEE is designed to be similar to EE while using a combination of both independent and dependent multivariate similarity measures introduced in Chapter 5. Rather than focusing on scalability, the goal of MEE is to present a baseline for accuracy comparison using similarity-based classifiers. However, by using recent advances in lower bounding and early abandoning there is potential to improve its scalability.

MPF is designed to be similar to Proximity Forest and uses a combination of independent and dependent multivariate similarity measures. Unlike DTW_I and DTW_D, which use all dimensions, MPF uses a random subset of dimensions. When compared with using all dimensions, we showed that by using a subset of dimensions, similar accuracy can be achieved 2 to 3 times faster.

Our classifiers MEE and MPF are both single domain (similarity) classifiers. While they do not achieve the same level of accuracy as the multi-domain ensemble HIVE-COTEv2, they are competitive with the current state of the art in single domain classifiers STC (shapelet-based), TDE (dictionary-based) and DrCIF (interval-based).

6.6 Contributions

Contributions made in this chapter include:

1. Multivariate Elastic Ensemble: A similarity-based baseline for accuracy comparison against other multivariate TSC classifiers.
2. Multivariate Proximity Forest: A scalable, tree-based, and similarity-based multivariate TS classifier.
3. A comparative study that examine the use of all dimensions against using random subsets of dimensions for multivariate similarity measures. I note that this thread of research is less studied in the literature but is an important topic to explore when studying multivariate TS classifiers. I hope that this contribution will stimulate further research on this topic.
4. Two similarity-based, multivariate TS classifiers that is competitive with the current state-of-the-art single domain classifiers STC (shapelet-based), TDE (dictionary-based) and DrCIF (interval-based).

Chapter 7

Conclusions

7.1 Summary of Research

Time series classification is an important area of machine learning research that has been receiving increasing attention in recent years. Nonetheless, at the beginning of my PhD in 2018, many algorithms faced important challenges as I outlined in Chapter 1. These challenges included lack of algorithms that were both highly accurate and scalable, as well as limited support for multivariate time series.

In this research, I addressed these challenges by contributing to the development of several novel TSC algorithms. In Chapter 3 I presented Proximity Forest which was the first algorithm to demonstrate state-of-the-art accuracy while being able to scale to large quantities of data. In Chapter 4 I presented an improved classifier, TS-CHIEF, that is still among one of the four main state-of-the-art TSC methods.

TS-CHIEF demonstrated that it was possible to attain state-of-the-art TSC accuracy while only requiring relatively modest computation. It is credible that my demonstration of this possibility has been a significant driver of recent research in the field. There have been dramatic gains in computational efficiency in recent state-of-the-art TSCs [7, 98]. The report that Elastic Ensemble was not included in recent iterations of HIVE-COTE due to its excessive compute despite its improving accuracy illustrates how computational efficiency has become front of mind in recent work [7].

While newer algorithms such as ROCKET and HIVE-COTEv2 have surpassed the speed of TS-CHIEF, there is considerable potential to benefit from recent developments in order to speed-up TS-CHIEF while maintaining or even improving its high accuracy.

Some types of time series data, such as the Earth observation satellite data, are inherently short but have large numbers of series. For example 1 year of satellite time

series data has only series with length 52 if data is collected once a week. But there can be very large numbers of series (1.5 trillion if one considers the entire land area of the planet at 10 meter scale as provided by the Sentinel II satellites). Proximity Forest and TS-CHIEF are very useful for such data because of their quasi-linear time complexity with respect to the number of series.

My research on multivariate time series extended the independent and dependent methods originally developed for DTW to ten further distance measures. It helped to understand the differences between these two strategies for combining multiple dimensions and whether differences in their performance arose from fundamental properties of different multivariate time series data. It also showed that simple strategies to select subsets of dimensions can provide effective results and opened new doors for further exploration.

7.2 Contributions to Knowledge

At the end of Chapters 3 to 6, I highlighted my contributions in general and specifically to the work presented in the chapters. Here, I distill the most important contributions made I to the advancement knowledge during my PhD work.

1. *Proximity Forest (co-author)*: I contributed to the development of the novel TSC algorithm Proximity Forest [8]. Proximity Forest is the first algorithm with accuracy that competes with state-of-the-art TSC classifiers of the time such as EE, ST, BOSS or FLAT-COTE, while impressively more scalable than any of its other competitors. My contribution, a Gini-based evaluation criteria to select between multiple candidate splits at the nodes of the trees, boosted its accuracy significantly.

2. TS-CHIEF (primary author): One of the most important contributions of this thesis is the novel TSC algorithm TS-CHIEF [35]. We demonstrated that TS-CHIEF achieves state-of-the-art accuracy that rivals HIVE-COTE (alpha) within a fraction of its runtime. At the time, TS-CHIEF and HIVE-COTE were considered as the two most accurate TSC algorithms in the field. We showed that TS-CHIEF's theoretical train time complexity is quasi-linear with respect to the quantity of data. While most TSC algorithms of the time do not scale beyond a few thousand series, on one dataset, we empirically showed TS-CHIEF is scalable to more than 150k time series within a time frame that is far beyond the reach of any other competitor of the time. Since publication, the main results of TS-CHIEF have been independently replicated [38], and TS-CHIEF is still accepted as one of the four state-of-the-art TSC algorithms in the field [7, 94, 99].

3. The contribution from TS-CHIEF is not only an advancement of the state-of-the-art accuracy and scalability. It is also a contribution to tree-based algorithm design. TS-CHIEF provides a flexible, yet simple, framework to embed various TSC techniques inside the nodes of decision trees and then use the Gini index to evaluate and select the best splitting function among them. It builds a heterogeneous tree with different splitting techniques (e.g. nearest exemplar based method using similarity measures and an attribute-value based method similar to classical decision trees). Such a framework leverages the benefits of the divide-and-conquer strategy of trees with time complexity for tree building that is quasi-linear with respect to training set size. This is extremely useful, since many of the prior algorithms are limited by the use of slow 1-nearest neighbor as base classifiers in the ensembles, or use of simple decision trees such as Random Trees that work on only one such transformation of data at a time. The flexible framework in TS-CHIEF allows new "splitters" to be developed and integrated into nodes of trees. This strategy is very different and unique compared to the method used in HIVE-COTE, which ensembles independent algorithms using a voting technique.

4. Multivariate Similarity Measures (primary author) [40]: This work extended seven commonly used univariate elastic similarity measures to the multivariate case by adopting two strategies — using multiple dimensions independently and dependently. Univariate versions of these seven measures have been widely used in time series analysis tasks such as classification, clustering, indexing, segmentation, anomaly detection and subsequence search. Therefore, this work is expected to be of great benefit to the time series analysis community.

5. Chapter 5 demonstrates the utility of these measures by focusing on the multivariate nearest neighbor classification problem. We show that each measure outperforms all others on at least one task (or ties with the highest performing measures in two cases), demonstrating the value of having a suite of alternative multivariate elastic measures.

6. Chapter 5 demonstrates that there are some classification tasks for which the independent strategy is superior across all measures and others for which the dependent strategy is better. It was previously demonstrated for DTW that each approach sometimes substantially outperformed the other [42]. However, it has not previously been established whether this is a fundamental property of the tasks per se. Chapter 5 provides evidence that in at least some cases there is a fundamental connection between classification tasks and whether they are best addressed by considering all variables at each time step in conjunction or by considering each variable independently of the others.

7. Multivariate Elastic Ensemble: A multivariate TS classifier that has accuracy competitive with the current state-of-the-art single domain classifiers STC (shapelet-based), TDE (dictionary-based) and DrCIF (interval-based). This provides a similarity-based baseline for comparison against other multivariate TSC classifiers.

8. Multivariate Proximity Forest: A scalable, tree-based, and similarity-based multivariate TS classifier that has accuracy competitive with the current state-of-the-art single domain classifiers STC (shapelet-based), TDE (dictionary-based) and DrCIF (interval-based).

9. A comparative study that examines the use of all dimensions against using random subsets of dimensions for multivariate similarity measures. I note that this thread of research is less studied in the literature but is an important topic to explore when studying multivariate TS classifiers. I hope that this contribution will stimulate further research on this topic.

7.3 Limitations

Some of the limitations of my works include:

1. Experiments carried out in this thesis used the SITS dataset [3], univariate UCR 2015 datasets [1], and multivariate UCR 2018 datasets [5]. Therefore, all experimental techniques have the limitations associated with these datasets. These include, limitations arising from normalization techniques, distribution of data, any issues with data labeling, and limitations of the provided train/test splits. Due to time constraints and resource constraints, my experiments did not use a large number of resamples or do cross-validation to reduce variance of the models.

2. One of the ways we can improve TS-CHIEF is by improving its memory usage. Currently most of the memory is being used by dictionary-based splitters so if we could improve the current implementation or perhaps replace it with a variant more similar to TDE than BOSS, we may be able to improve its memory footprint as well as accuracy.

3. Multivariate similarity measures presented in Chapter 5 can be improved further. For example, the method to adjust the parameter of multivariate LCSS (and TWE) explained in Chapter 5, Section 3.3.3, assumes that the data are normally distributed. If this assumption doesn't hold, the accuracy of these measures may be affected. Also, when using multivariate MSM with 1-NN, out of all measures

MSM_D, used almost 50% of the runtime. This slows down Multivariate Proximity Forest.

7.4 Future Work

1. Both Proximity Forest and TS-CHIEF have quadratic worst-case runtime $O(L^2)$ with respect to the length of the series. This is due to the $O(L^2)$ complexity of similarity measures such as WDTW, which does not use a warping window. For example, on UCR datasets, most of the training time is used by a handful of datasets such as *Handoutline* which has long time series. Strategies that can be explored further to reduce the average runtime complexity include 1) replacing slower measures such as WDTW with faster measures with similar or better accuracy, 2) investigating the effect of limiting the maximum warping window size to smaller values for measures such as DTW (currently it is 25% of the series length), 3) using early abandoning techniques [54] to "shortcut" similarity calculations.

2. One of the ways that HIVE-COTEv2 improved its accuracy is by replacing slow and older algorithms with newer components. Since TS-CHIEF's first version, many new algorithms have been introduced which can be used to create new splitters at the node level. For example, the current interval-based splitter inspired by RISE could be replaced with a new splitter that works on the "catch-22" features used by the DrCIF component of HIVE-COTEv2 Similarly, the latest dictionary-based splitter TDE which is more accurate and memory efficient can be used to create a new splitter to replace the current BOSS inspired dictionary-based splitter. Furthermore, splitters that use other techniques, such as deep-learning based methods, could be integrated into the TS-CHIEF framework.

3. One of the current goals I have for TS-CHIEF is to extend it to support multivariate TSC. Multivariate Proximity Forest is one step of the process. Adding two new DrCIF and TDE based splitters can also transform TS-CHIEF to support multivariate classification. This is becoming increasingly important since multivariate versions of ROCKET, InceptionTime and HIVE-COTE have been introduced in recent years.

4. Multivariate Proximity Forest currently explores simple random strategies to select subsets of dimensions. There is potential future work to investigate more complex strategies to select subsets of dimensions.

Appendix A

Proximity Forest

In this Appendix, I have included a copy of the Proximity Forest paper [8] as it was published in Data Mining and Knowledge Discovery journal.

Data Mining and Knowledge Discovery (2019) 33:607–635
https://doi.org/10.1007/s10618-019-00617-3

Proximity Forest: an effective and scalable distance-based classifier for time series

Benjamin Lucas[1] · Ahmed Shifaz[1] · Charlotte Pelletier[1] · Lachlan O'Neill[1] · Nayyar Zaidi[1] · Bart Goethals[1] · François Petitjean[1] · Geoffrey I. Webb[1]

Received: 24 March 2018 / Accepted: 21 January 2019 / Published online: 6 February 2019

Abstract

Research into the classification of time series has made enormous progress in the last decade. The UCR time series archive has played a significant role in challenging and guiding the development of new learners for time series classification. The largest dataset in the UCR archive holds 10,000 time series only; which may explain why the primary research focus has been on creating algorithms that have high accuracy on relatively small datasets. This paper introduces Proximity Forest, an algorithm that learns accurate models from datasets with millions of time series, and classifies a time series in milliseconds. The models are ensembles of highly randomized Proximity Trees. Whereas conventional decision trees branch on attribute values (and usually perform poorly on time series), Proximity Trees branch on the proximity of time series to one exemplar time series or another; allowing us to leverage the decades of work into developing relevant measures for time series. Proximity Forest gains both efficiency and accuracy by stochastic selection of both exemplars and similarity measures. Our work is motivated by recent time series applications that provide orders of magnitude more time series than the UCR benchmarks. Our experiments demonstrate that Proximity Forest is highly competitive on the UCR archive: it ranks among the most accurate classifiers while being significantly faster. We demonstrate on a 1M time series Earth observation dataset that Proximity Forest retains this accuracy on datasets that are many orders of magnitude greater than those in the UCR repository, while learning its models at least 100,000 times faster than current state-of-the-art models Elastic Ensemble and COTE.

Keywords Time series classification · Scalable classification · Time-warp similarity measures · Ensemble

Responsible editor: Panagiotis Papapetrou.

B Benjamin Lucas
benjamin.lucas@monash.edu

Extended author information available on the last page of the article

1 Introduction

A growing number of time series applications address training from orders of magnitude more series than the largest in the benchmark UCR repository—the 8926 training series ElectricDevices data set. In contrast, the phoneme dataset (Hamooni and Mueen 2014) contains 370,000 series. The satellite dataset (Tan et al. 2017) contains 1,000,000 series. The prior state-of-the-art in time series classification does not scale to such quantities. In 2017, a meticulous study was conducted to compare the behaviour of the state of the art (Bagnall et al. 2017). The authors draw the following conclusions:

1. The state of the art is led by four classifiers that are: Collective of transformation-based ensembles (COTE) (Bagnall et al. 2015), Elastic ensemble (EE) (Lines and Bagnall 2015), Shapelet transform (ST) (Hills et al. 2014) and Bag of SFA symbols (BOSS) (Schäfer 2015).
2. COTE is a special case in that it subsumes two of the other classifiers: it is a large ensemble classifier that includes EE and ST as sub-classifiers; COTE is on average, "clearly superior to other published techniques."
3. COTE's runtime complexity is bounded by (a) Shapelet transform, which is $O(n^2 \cdot l^4)$ (Bagnall et al. 2015) for n time series of length l, and (b) the parameter searches for EE, some of which are $O(n^2 \cdot l^3)$. The authors conclude "An algorithm that is faster than COTE but not significantly less accurate would be a genuine advance in the field."

This is the challenge we tackle in this paper: developing an algorithm that is competitive with the accuracy of the state of the art, but can learn from datasets with millions of time series. We call our algorithm *Proximity Forest*. It is a tree-based ensemble that makes the most of the decades of research into developing consistent similarity measures for time series.

Typical decision trees branch on the value of an attribute. Treating the values at each time stamp as belonging to a single attribute does not work well on time series because the relevant signals are not necessarily aligned by time stamp. Instead, Proximity Forest branches on the *proximity* of a query time series to a set of reference series. 'Proximity' is defined by a given (time series) similarity measure and a set of parameters (most time series measures have parameters that are critical to their proper function). Our trees define separating hyperplanes for which the position is supported by time series themselves (whereas a traditional tree would split using a threshold on the value of an attribute). Proximity Forest, as opposed to nearest neighbour approaches, truly abstracts a model from data, which makes it possible to classify with time that is logarithmic with respect to training set size, as opposed to linear time for EE and COTE. Moreover, we will show that we specifically designed its training to scale linearly with the quantity of data, as opposed to at least quadratically for EE, COTE and ST.

Figure 1 shows the accuracy and the training time required by Proximity Forest and EE on our satellite dataset with increasing training set size. Two important elements are illustrated: (1) Proximity Forest scales linearly with training set size while EE scales quadratically; and, (2) Proximity Forest's classification accuracy on this dataset is substantially better than EE, even when they train on the same quantity of data. For our application, Proximity Forest can learn from 1M time series in 17 h (on 1 CPU)

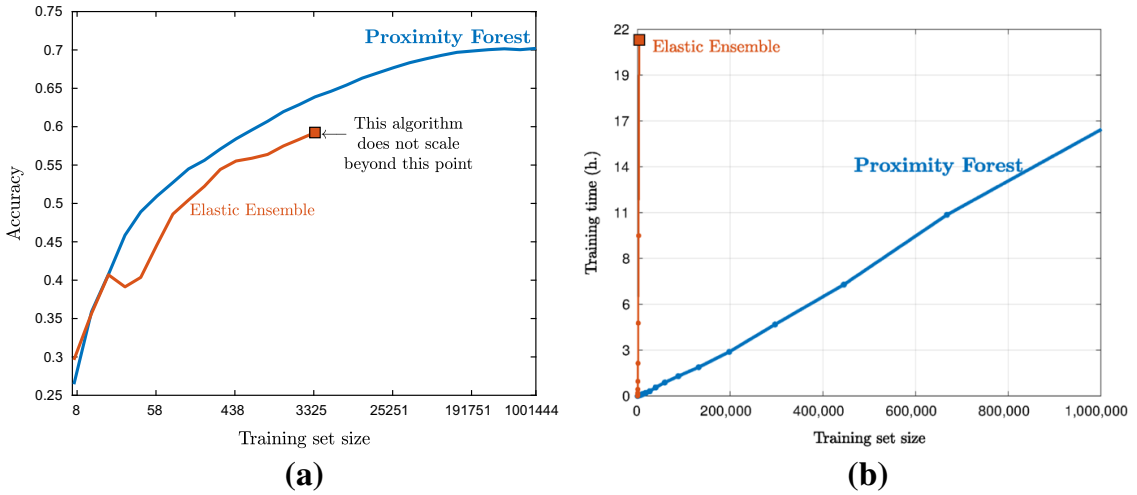


Fig. 1 Comparison of Proximity Forest (in blue) with Elastic Ensemble (in red). **a** Classification accuracy as a function of the size of the training data. **b** Training time as a function of the size of the training data. Note that we take Elastic Ensemble because it is the classifier that prevents scalability of the state-of-the-art COTE ensemble, which includes EE as one of its classifiers (Color figure online)

while it would take over 200 years for EE—a 103,000x speedup. Furthermore, ST and COTE learn slower than EE and thus will have *even larger* training times. Note that EE is a component of COTE and hence sets a lower bound on COTE's training time. Our discussion will often focus on EE because EE is very similar to our algorithm in that it is trying to leverage existing time series similarity measures.

The virtues of Proximity Forest are not limited to large datasets, however. Our experiments show that it also outperforms EE in classification accuracy on the majority of the datasets of the UCR Archive.

The remainder of this paper is structured as follows: in Sect. 2, we review the state of the art in time series classification, with a particular focus on scalability. We then introduce Proximity Forest in Sect. 3. Our experiments (Sect. 4) show that Proximity Forest (1) outperforms all other scalable algorithms on our case study in both accuracy and training time; and (2) is competitive with the state of the art on UCR data in terms of accuracy. The section ends with a study of Proximity Forest's parameters and a discussion of how their values vary the results.

2 Time series classification: related work

We present here a non-exhaustive review of the state of the art in time series classification and similar decision tree-based algorithms. We focus on our particular interest in this paper: scalable training and classification.

2.1 Distance-based classification

2.1.1 Distances

Time series have particular properties that have led to the development of specific similarity measures: they are often auto-correlated (the value of the time series at a

timestamp is likely to be close to the ones just before and after), and often include non-linear distortions in the time axis (for example, because the start of the phenomenon of interest is delayed, or because sections of the phenomena are faster or slower). This has rendered typical similarity measures severely flawed and led to the development of specific similarities, of which most have an ability to re-align the series along a common intrinsic time-line. Important measures include Dynamic time warping (DTW) (Sakoe and Chiba 1971, 1978), Derivative DTW (DDTW) (Keogh et al. 2001; Górecki and Łuczak 2013), Weighted DTW (WDTW) (Jeong et al. 2011), Longest common subsequence (LCSS) (Vlachos et al. 2006), Edit distance with real penalty (ERP) (Chen and Ng 2004; Chen et al. 2005), Time warp edit distance (TWE) (Marteau 2009) and Move-Split-Merge (MSM) (Stefan et al. 2013). A more complete description and comparison of these distance measures can be found in Bagnall et al. (2017); Lines and Bagnall (2015); Wang et al. (2013). Note also that most of these distances have parameters of which tuning is critical to their functioning.

2.1.2 Nearest-neighbor approaches

It is most common to classify time series data using Nearest Neighbour classification based on a relative distance (such as those returned from the measures above) (Haghiri et al. 2017). In fact, for more than a decade, the NN algorithm combined with the DTW measure was extremely difficult to beat (Wang et al. 2013).

It is important to note here that when researchers mention the use of NN with a time series measure, the measure is not directly applied with a default parameterization, but rather its parameters are first learned on the data, usually by cross-validation.

There are two main issues with NN approaches: (1) the tuning of the measures' parameters is usually quadratic with the size of the training data, and (2) the classification is at least linear with the size of the training data. Both of these issues are further compounded by the fact that most measures have a computational complexity that sits between linear and quadratic with the length of the series.

To alleviate the second issue of scalable classification, targeted techniques have been developed. Data reduction techniques aim at simplifying the training database without penalizing the classification quality; either by directly removing objects from the original database (Pękalska et al. 2006; Ueno et al. 2006) or by summarizing the database and replacing sets of time series with representatives using average time series (Petitjean and Gançarski 2012; Petitjean et al. 2014, 2016; Marteau 2016). Indexing is more difficult on time series than it is on traditional data, mostly because time series measures do not obey the triangular equality (at most obeying a relaxed p-triangular inequality Lemire 2009), which makes exact pruning very inefficient (however, a general approach to improving the efficiency of NN searches in non-Euclidean space is available in Lifshits 2010). To perform exact indexing, the main research effort has been put onto developing lower bounds (and mostly for DTW) (Keogh et al. 2006; Lemire 2009). Recently, impelled by the motivating application of Earth observation data analytics, we have developed an algorithm for approximate and efficient NN search under DTW (Tan et al. 2017), an algorithm using the idea of a hierarchical k-means clustering.

As mentioned previously, Elastic ensemble (EE) (Lines and Bagnall 2015) is a recent state-of-the-art time series classifier. It is an ensemble of 11 NN classifiers, each learned with a different time series measure (with their parameters tuned accordingly). The EE algorithm has played a significant role in the design of Proximity Forest and will be discussed at greater depth in Sects. 3.1 and 3.2.

2.2 Approaches that learn features

The following approaches construct an abstraction of the training dataset by learning features that represent the classes in the time series.

2.2.1 Shapelets

The aim of shapelet algorithms is to find subseries (or consecutive subsets of time series) that can help discriminate between the different classes. To classify a time series, the learned shapelet is placed at the best position in the time series (usually under Euclidean distance), and the 'matching' of the shapelet to the time series correspond to its distance at this best position. The original shapelet-classifier (Ye and Keogh 2011) inserted this algorithm at the node of a decision tree as a splitting criterion. This algorithm has a high training complexity ($O(n^4)$) due to the large number of candidate shapelets and the repeated scanning of the data. Subsequent research has focused on optimising the original algorithm to address both classification accuracy and scalability, notably Fast Shapelets (Rakthanmanon and Keogh 2013), Learning Time Series Shapelets (Grabocka et al. 2016) and Shapelet Transforms (Hills et al. 2014).

Shapelet transforms (ST) is a current state-of-the-art classifier that identifies the best k shapelets in a single scan of the data (the number of shapelets can be reduced afterwards). The data is then transformed by defining an attribute to represent each shapelet with the value being the (usually Euclidean) distance between the shapelet and the best position in the time series. The transformed dataset can now be used with any classifier or ensemble of classifiers (such as in Bagnall et al. 2015). While ST is considered a state-of-the-art classifier, it has little potential to scale to large datasets given its training complexity of $O(n^2 \cdot l^4)$.

2.2.2 Bag of words approaches

Bag of words are similar to shapelets in that they start by identifying exemplar subseries in the data to discriminate between classes. However rather than finding the similarity to the relative best positions in a time series, bag of words approaches differentiate classes by the relative frequency of the subseries. To calculate these frequencies, the algorithms discretise the values into a series of symbols, assigning letters to each subseries, and thus representing the original time series as 'words'. Notable approaches are the Bag of patterns (Lin et al. 2012); the Symbolic aggregate approximation vector space model (SAX-VSM) (Senin and Malinchik 2013); and the Bag of SFA symbols (BOSS) (Schäfer 2015), which is currently considered state of the art.

The BOSS algorithm transforms the time series into a word using a symbolic Fourier approximation (SFA) (Schäfer and Högqvist 2012) thus making it robust to noise and delivering a high classification accuracy. It is however of limited use on large datasets as it has a high training complexity $O(n^2)$ (Bagnall et al. 2017). The authors identified this as a weakness and subsequently produced similar approaches with improved scalability, the Bag of SFA symbols in Vector space (BOSS-VS) (Schäfer 2015). The same authors recently proposed WEASEL (Schäfer and Leser 2017), which improves on the computation time of BOSS and on the accuracy of BOSS-VS, but has a very high memory complexity (our experiments will show that it doesn't scale beyond 10,000 time series). In this way, WEASEL is more optimised for speed on small datasets than for scalability.

2.3 Ensemble approaches

Ensemble approaches are combinations of multiple classifiers. Each contributing algorithm can be weighted to maximize classification accuracy, while the time complexity is that of the slowest constituent. Some of these approaches have been discussed above as they are based around one main type of classifier, for example EE and ST.

The Collective of transformation-based ensembles (COTE) (Bagnall et al. 2015) is an ensemble comprising 35 classifiers across four time series domains: time, frequency, change and shapelet transformation. For the time domain, COTE uses the 11 distance measures of EE, while in the other three domains, classifiers are recruited from outside time series classification—k-nearest neighbours, naive Bayes, decision trees, random forest, rotation forest, support vector machines (two models) and a Bayesian network approach. On the benchmark UCR datasets, COTE has the highest average classification accuracy of all current approaches. However, its time complexity is bound by that of the Shapelet transform, which is $O(n^2 \cdot l^4)$ and the parameter searches for the elastic distance measures (EE), some of which are $O(n^2 \cdot l^3)$.

2.4 Decision tree approaches

A number of decision tree approaches have been developed for time series classification.

Time series forest (TSF) (Deng et al. 2013) first derives summary features for all time series by dividing them into intervals and summarising each interval by its mean, standard deviation and gradient. Then a Random Forest-like strategy is employed to select between a random subset of these features at each node in each of an ensemble of trees. A novel selection criterion is used that considers both entropy gain and the margin by which a feature separates the classes. This continues until the entropy gain ceases to improve, at which stage the node is defined as a leaf. TSF has been shown to be a reasonably accurate classifier: its accuracy ranks behind EE and DTW without being significantly worse (Bagnall et al. 2017). However, its main virtue is computational efficiency. TSF learns in $O(n \log(n) \cdot l \cdot k)$ for a forest of k trees built from n series of length l, which is a much lower complexity than the current state of the art.

Generalized random shapelet forest (gRSF) (Karlsson et al. 2016) extracts a shapelet from a randomly chosen time series and finds the distance between this time series and each other time series. The data is then split according to whether it is above or below a threshold distance to the representative shapelet. This is applied recursively until the node is either pure on the number of instances remaining at a node is less than 3. As mentioned in Sect. 2.2.1, the main pitfall of shapelet-based methods is the high computational cost of finding candidate shapelets and comparing shapelets to other time series. The gRSF minimises this issue by randomising many of the model choices—a candidate shapelet is generated from a randomly chosen time series by choosing a random starting point and random length, this is repeated r times and the best candidate is chosen for a given split. The resulting algorithm has accuracy competitive with Learning Time Series Shapelets and better than DTW.

A number of approaches have been developed that form decision trees where splits are based on similarity to chosen exemplars (Balakrishnan and Madigan 2006; Douzal-Chouakria and Amblard 2012; Yamada et al. 2003). One strategy is to select a single exemplar and then choose a cut point on a similarity measure with respect to that exemplar. Series with similarity scores lower than the cut point follow one branch and the remaining examples follow the other. The other strategy is to select multiple exemplars, one associated with each branch. Series follow the branch with whose exemplar they are most similar. These approaches are hampered by the high computational complexity of their search for exemplars at each node. Similarity Forests (Sathe and Aggarwal 2017) and Comparison-based Random Forests (Haghiri et al. 2018) generalise this idea to attribute-value data with random selection of exemplars and developed forests of such trees. Similarity Forests add a cutoff value on the difference in the distance between the two exemplars, and optimizes that cutoff value based on weighted Gini. The idea of using similarity as the splitting criterion in tree structures has also been successfully used for *indexing* of regular tabular data (www.cs.ubc.ca/research/flann/, Bernhardsson 2013; Sathe and Aggarwal 2017) and of time series with DTW (Tan et al. 2017).

3 Proximity forest

In this section, we present our novel algorithm for time series classification: Proximity Forest. We start by highlighting why there is a need for a new time series classifier. We then present our model and the two key algorithms (1) how to learn a Proximity Forest and (2) classifying with a Proximity Forest. We conclude this section with some comments about its complexity.

3.1 Why do we need a novel time series classifier?

The previous section highlighted that the last decade has seen numerous classifiers and distance measures specifically designed for time series classification. Based on this, one could wonder why there is a need for a novel algorithm; the answer is simple: most state-of-the-art algorithms do not scale to large time series datasets. We have

seen that some do not scale in the learning phase (ST, EE, COTE). Others require a scan of the training database to perform each classification (EE, COTE). Those that do scale to medium-size datasets, such as BOSS-VS, compromise accuracy in order to do so (as we will show for both our case study and for UCR datasets). Throughout the development and advancement of much of the current state of the art, scalability has usually been secondary to accuracy. This is because time is not a significant issue when considering data with only few time series. However, a growing number of modern applications consist of hundreds of thousands to millions of time series. These applications require a classifier that is both accurate *and* scalable in both learning and classification.

BOSS-VS is a classifier that appears to have developed with a focus on scalability. However, as we will see in Sect. 4, its accuracy ranks some 30% points lower in our case study, and therefore is not competitive with the accuracy of the state of the art.

While COTE is currently *the* state of the art in terms of accuracy, its learning phase is bound by the runtime complexity of both ST and EE. On our 1M dataset—and as depicted in Fig. 1—the sole learning phase of COTE associated to training EE would require 73,000 days, or 200 years. This is even more startling knowing that the series in this dataset are very short with only 46 timestamps.

The large runtime complexity of COTE is largely due to the fact that EE does not abstract much information during the learning phase, and therefore has a significantly greater number of processes to complete during testing. A corollary of this is that a distance-based classifier that learns faster than EE for the same level of accuracy would also present an improvement to COTE. It is for this reason that our design of algorithm incorporates many elements of EE—11 distance measures and similar parametrisation—and why our experiments provide a direct comparison of Proximity Forest against EE.

We therefore argue that the need for a scalable and accurate classifier has not yet been met. We incorporate three critical elements into the design of our novel approach:

1. We make the most of over 30 years of research into designing consistent measures for time series.
2. We specifically design our ensemble to have a high variability between the different individual classifiers. This results in an improved overall classification accuracy over a single classifier, based on the principle of ensemble methods. In general, averaging the predictions of multiple models each having high variance and low bias results in an ensemble classifier with a lower total error than any single classifier. This is analogous to how a Random Forest model, another ensemble of decision trees, will only learn from a fraction of the available features for each individual node in order to introduce variability between the trees (Breiman 2001; Ho 1995). This observation is important, because we did not design the learning of an individual tree to maximize its accuracy; if we had wanted to design a single tree model, we would have made different design choices. We designed the learning of individual trees so that the overall classification performance is maximised.
3. We design Proximity Forest to be extremely scalable with an average-case learning complexity of $O(n \log(n) \cdot l^2)$ and a classification complexity of $O(\log(n) \cdot l^2)$ per tree for n training time series of length l. This contrasts with the state-of-the-art

learning in $O(n^2 \cdot l^3)$ (Elastic ensemble) or $O(n^2 \cdot l^4)$ (Shapelet transform, COTE). Again here, we might have made different choices if scalability wasn't a design objective.

To achieve scalability we employ tree-based classifiers. These are scalable due to their use of a divide-and-conquer strategy. At each level the data are divided into multiple subsets, as result of which the trees are on average of depth $O(\log n)$, hence increasing sublinearly in depth relative to training set size.

Our use of decision trees for time series classification is not novel in itself. Trees are attractive due to their divide and conquer methodology and resulting potential for efficient learning and classification. Previous implementations, however, have lacked competitiveness in accuracy (Deng et al. 2013; Douzal-Chouakria and Amblard 2012; Yamada et al. 2003) or time (Balakrishnan and Madigan 2006; Douzal-Chouakria and Amblard 2012; Yamada et al. 2003).

To achieve scalability we merge the strategy of learning decision trees where splits are based on similarity to chosen time series exemplars (Balakrishnan and Madigan 2006; Douzal-Chouakria and Amblard 2012; Yamada et al. 2003) with the strategy of forming forests of such trees in which the exemplars are chosen at random (Sathe and Aggarwal 2017). To this amalgam we add the critical ingredient of stochastic selection between a large range of similarity measures, which both reduces bias and provides a beneficial increase in variance between ensemble members.

3.2 How to learn a Proximity Forest?

We seek to learn a Proximity Forest from a training set comprising n labeled time series, each of which is of length l, where the labels are integers from 1 to c.

A Proximity Forest is an ensemble of k Proximity Trees. A Proximity Tree is similar to a regular decision tree, but differs in the tests applied at internal nodes. Whereas a regular decision tree applies a test based on the value of an attribute (e.g. if height > 160 cm, follow the left branch, otherwise follow the right branch), each *branch* of an internal node of a Proximity Tree has an associated exemplar and an object follows the branch corresponding to the exemplar to which it is *closest* according to a parameterized similarity measure. We will see later how the exemplars and measures are chosen. A tree is either a leaf or an internal node.

An internal node has two fields, `measure`, a function $object \times object \rightarrow \mathsf{R}$, and `branches`, a vector of branches. Each branch has two fields, a time series (`exemplar`) and a tree to which an object is passed if it is nearer to the branch's exemplar than any other (`subtree`).

If all data reaching a node has the same class, i.e. is pure, the $create\_leaf$ function creates a new leaf node and assigns this class label to its field `class`. This label is then assigned to any query time series reaching this leaf during the testing phase.

How do we choose the splitting criteria? A Proximity Tree creates, at each node, one branch for each class that exists in the data it receives from its parent. These exemplars are chosen uniformly at random among each class. The parameterized similarity measures are also chosen uniformly at random among a pool that will be

described below after we have given the main overview of the algorithm. We will detail, after the main algorithm, how it is possible to *learn* with randomized trees.

Algorithm 1 presents the algorithm for learning a single tree. Each node is constructed recursively from the root node down to the leaves. If the data at the node is pure—i.e., all data belongs to the same class—then the node becomes a leaf and the recursion finishes.

At each node, a pool of r candidate splits are evaluated (Algorithm 2). For each candidate, a parameterised measure is chosen uniformly at random among a pool of such measures. We then select an exemplar for each class represented at the node and pass the data down the branches by finding the closest exemplar (one per class) for each time series in the data using the split's distance measure.

Once each candidate split has been created, we then select the candidate that maximizes the difference between the Gini impurity of the parent node and the weighted sum of Gini impurity of the child nodes. We then call the construction of the tree recursively on each branch for the successful candidate; this constructs all subtrees. When this is done, the tree is constructed.

Increasing the number of candidate splits per node will lead to an improvement of the quality of each split. However, it will also lead to an increase of the training time. The choice for the value of r will be discussed later in Sect. 4.3.2.

The case of $R = 1$: is selecting at random still 'learning'? One might wonder what the tree is actually learning when one only considers a single candidate ($R = 1$). In that case, no selection of 'the best possible split' is performed. It is interesting to note that choosing splitting criteria independently of the output value has been studied before, a key example being Extremely randomized trees (Geurts et al. 2006). In that work, they showed that splitting completely at random still ensures consistency (tending to Bayes optimal error as the data tends to infinity). The main reason is that the exemplars are not random points in the input space. They are sampled from the data distribution of each class. In consequence, the trees *are* still learning an abstraction of the data, using the trees as a density estimator (Ting et al. 2016).

We depict in Fig. 2 a graphical representation of a simple split obtained on the `Trace` dataset. It is interesting to see that in Euclidean space, the splitting criterion is actually forming a hyperplane that is equidistant to the exemplars. Note that this intuition is more complex for time series measures, because most of them do not have properties of a metric (Lemire 2009). The scatter plot depicts each time series as a dot in this space, with the x-axis representing distance to the first exemplar and the y-axis distance to the second.

How to choose the parametrised measure on which to split? The parametrised distance measure gives a measure of the similarity between the exemplar time series. For each candidate split at each node, the algorithm chooses a distance measure at random from the following 11 distance measures used by the Elastic ensemble (EE) learner that we described above: Euclidean distance (ED); Dynamic time warping using the full window (DTW); Dynamic time warping with a restricted warping window (DTW-R); Weighted dynamic time warping (WDTW); Derivative dynamic time warping using the full window (DDTW); Derivative dynamic time warping with a restricted warping window (DDTW-R); Weighted derivative dynamic time warping (WDDTW); Longest

Algorithm 1: build_tree(D, Δ, R)

Input: D: a time series dataset
Input: Δ: a set of parameterized distance measures
Input: R: number of candidate splits to consider at each node
Output: T: a Proximity Tree

if is_pure(D) **then**
  return create_leaf(D)

// create tree, to be returned, represented as its root node

$T \leftarrow$ create_node()

// Creating R candidate splitters
$R \leftarrow \emptyset$
for $i = 1$ **to** R **do**
  $r \leftarrow gen\_candidate\_splitter\,(D, \Delta)$ // generate random splitter
  Add splitter r to R

// select best splitter using measure $\delta^\star$ and exemplars $E^\star$
$(\delta^\star, E^\star) \leftarrow \arg\max_{r \in R}$ Gini (r)

$T_\delta \leftarrow \delta^\star$ // retain measure for root node of T
$T_B \leftarrow \emptyset$ // T_B will store the branches under root node of T
foreach exemplar $e \in E^\star$ **do**
  // $D_e^\star$ is the subset of D that are the closest to e based on $\delta^\star$
  $D_e^\star \leftarrow \left\{ d \in D \;\middle|\; \arg\min_{e' \in E^*} \delta^\star(d, e') = e \right\}$
  $t \leftarrow$ build_tree($D_e^\star$, Δ, R) // build subtree for that branch
  Add branch (e, t) to T_B // a branch is a pair (exemplar, sub-tree)

return T

Algorithm 2: gen_candidate_splitter(D, Δ)

Input: D: a time series dataset
Input: Δ: a set of parameterized distance measures to sample from
Output: (δ, E): a parameterized distance measure and a set of exemplars

// sample a parameterized measure δ uniformly at random from Δ
$\delta \overset{\sim}{\leftarrow} \Delta$

// select one exemplar per class to constitute the set E
$E \leftarrow \emptyset$
foreach class c present in D **do**
  $D_c \leftarrow \{d \in D \mid class(d) = c\}$ // D_c is the data for class c
  $e \overset{\sim}{\leftarrow} D_c$ // sample an exemplar e uniformly at random from D_c
  Add e to E

return (δ, E)

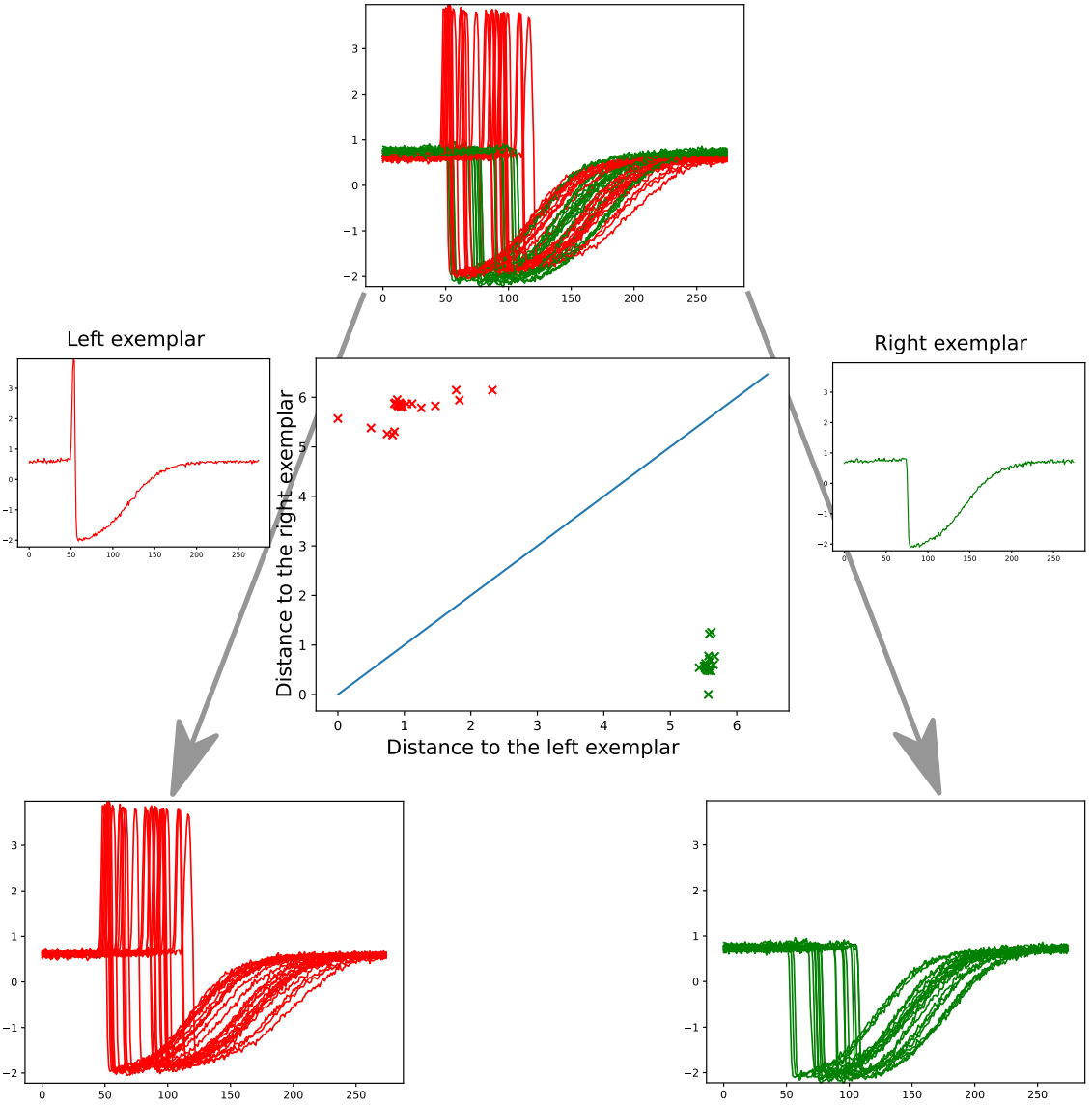


Fig. 2 Visual depiction of the root node for the 'Trace' dataset (simplified to 2 classes). The top chart represents the data at the root node (one colour per class) while the data at the bottom left and right represent the data once split by the tree. The two time series in the middle left and right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

common subsequence (LCSS); Edit distance with real penalty (ERP); Time warp edit distance (TWE); and, Move-Split-Merge (MSM). Randomising the choice of distance measure is a deliberate decision to introduce variability between each tree, for the reasons stated earlier.

Once a distance measure is chosen at random, it is then parametrised. The parametrisations are addressed in turn. They are deliberately chosen to mimic as closely as possible the EE algorithm. Even though better values might be chosen here, we mimic EE's parameterization to allow direct comparison. Euclidean distance, full DTW, and full DDTW distances have no parameters to select. DTW-R and DDTW-R only require a warping window parameter that is chosen uniformly at random in $[0, \lfloor \frac{l+1}{4} \rfloor]$ (thus allowing a warping of elements at most $\frac{l}{2}$ apart). WDTW and WDDTW requires

also one parameter to select that it is used into the weighted value g to control the level of penalization between two different time stamps—we use $g \sim U(0, 1)$. The parametrisation of ERP is a distance threshold that controls for how close elements have to be to be considered similar; we sample it uniformly at random in $[\frac{\sigma}{5}, \sigma]$, with σ being the standard deviation of the data. LCSS has as first parameter the same distance threshold value (which is sampled in the same way), and has a second parameter—the warping window size—which is chosen in the same way as for DTW-R. TWE has two parameters γ and λ which respectively control for the stiffness and penalty value in the alignment. Following (Marteau 2009), λ is sampled at random from $\cup_{i=0}^{9} \frac{i}{9}$ and γ following at random from the exponentially growing sequence $\{10^{-5}, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, \cdots, 1\}$, resulting in 100 possible parameterizations. The final measure, MSM, has a single parameter which is sampled from an exponential sequence similar to the one for γ in TWE with 100 values ranging from 10^{-2} to 10^{2}, as recommended in Stefan et al. (2013).

Choosing the parameter at random has a twofold effect: (1) it skips the cross-validation step which has a quadratic complexity; and (2) it introduces variability between trees, which provides superior learning through lower-biased trees and ensembling. In the following experiments we will show that Proximity Forest is not only orders of magnitude faster than EE, but that its accuracy also ranks higher than EE.

3.3 Classifying with a Proximity Forest

The process of classification for a single Proximity Tree is detailed in Algorithm 3: a query time series begins at the root node and the distance from the query to each of the exemplar time series is calculated, by using the node's distance measure and exemplars selected when constructing the tree. The query time series is then passed down the branch of the exemplar to which it is nearest. The query time series then traverses down the tree by repeating this process until it reaches a leaf, where it is assigned the class represented by that leaf. This process is repeated for each tree constructed as part of the forest. A Proximity Forest then uses majority voting between its constituent Proximity Trees.

Algorithm 3: classification(Q, T)

Input: Q: Query Time Series
Input: T: Proximity Tree

if is_leaf(T) **then**
  return majority class of T

`// find the branch with exemplar closest to` Q `using measure` T_δ

$(e, T^\star) \leftarrow \arg\min_{(e', T') \in T_B} T_\delta(Q, e')$

return classification$\left(Q, T^\star\right)$ `// recursive call on subtree` $T^\star$

3.4 Comparative complexity analysis

During the training phase, at each node, let us assume that n' data points are present at the node. We first scan it once taking $O(n')$ time to split the data into c groups, one for each class c present at the node.

We then generate r candidate splits, i.e., r sets of exemplar time series. For each such candidate set, we sample $c' \leqslant c$ exemplar time series, i.e., one time series for class represented among the n' time series available at the node—this is done in $O(1)$ given that the data is already organised by class. For the candidate split to be operational, we also require a parameterized measure to use to compare against these c' exemplars. Most of the parametrized measures can be chosen in $O(1)$, except for LCSS and ERP which calculate the standard deviation in $O(n' \cdot l)$ for data at the node while selecting the parameter.[1]

We now have r candidate splits that are ready to be evaluated. We now wash the n' time series down the branches for all candidate splits. This is done by comparing each time series to the c' exemplars, each comparison taking from $O(c \cdot l)$ to $O(c \cdot l^2)$. Overall, this takes $O(n' \cdot c' \cdot l^2)$. If $r = 1$, the training process at this node is finished and we call the training function recursively for each of the c' children nodes. If $r > 1$, we calculate the Gini coefficient for each of the r candidate splits in $O(c^2)$, keep the best one, and delete other candidate splits.

As the total number of examples that reach any of the nodes at a single given level cannot be greater than the total number of examples, n, the total computation per level of the tree is thus $O(n \cdot r \cdot c \cdot l^2)$. In the worst case, the majority of the training data at each level will pass down a single branch and the depth of the tree will be $O(n)$, resulting in a worst training time complexity of $O(n^2 \cdot r \cdot c \cdot l^2)$. However, as the exemplars are following the class distribution, unless the data are in some way degenerate (for example if one class comprises only outliers), the average tree depth can be expected to be $O(\log n)$. In practice it will often be much smaller, because, unlike typical divide-and-conquer approaches, the tree terminates as soon as a node is pure rather than having to separate each individual object. Thus, for non-degenerate data we can expect average case training time complexity of $O\big(n \log(n) \cdot r \cdot c \cdot l^2\big)$ for a single tree and thus $O\big(k \cdot n \log(n) \cdot r \cdot c \cdot l^2\big)$ for a full Proximity Forest comprising k Proximity Trees.

The experiments presented in the next section will include runtimes and comparison to current state-of-the-art algorithms. These confirm that this expected average case quasi-linear complexity with respect to data quantity is borne out in practice.

During classification, a time series of length l will pass through an average of $\log n$ nodes on each of the k trees. At each node, the distance to at most c exemplars must be computed. For each of these distance computations, the complexity will again depend upon the chosen distance measure; the fastest being $O(l)$ and the slowest $O(l^2)$. Thus, the resulting average case complexity is $O(k \cdot \log n \cdot c \cdot l^2)$.

[1] Note that these parametrisations can be performed in constant time also if the data are z-normalized, which is the case for all UCR datasets.

4 Experiments

This section describes the experiments that evaluate our Proximity Forest. We start with the satellite image time series (SITS) dataset, a (very) large time series dataset describing the evolution of the Earth as pictured every five days by a high-resolution satellite. This dataset is an example of the large time series datasets that motivate the need for a new time series classification algorithm, as no current state-of-the-art approach scales to this magnitude. Conversely, there are classifiers designed for scalability, namely BOSS-VS, that compromise classification accuracy to do so. The first experiments presented in this section use the SITS dataset to demonstrate the ability of Proximity Forest to be both scalable *and* accurate. The second section assesses the Proximity Forest on the datasets of the UCR time series classification repository (Chen et al. 2015), the benchmark in the field. It demonstrates that the classification accuracy of Proximity Forest is competitive with the current state of the art. The final section discusses other considerations surrounding Proximity Forest, such as the effect of varying the number of trees, and the standard deviation of the results.

It should be mentioned that throughout the following experiments we have emphasized a comparison with EE. This is because it is viewed as the closest relative to Proximity Forest amongst the current state of the art, given that neither method includes data transforms or shapelets. It is also the constituent of COTE that bounds its learning time and therefore any improvement over the runtime of EE, for the same classification accuracy, would also equate to an improvement on COTE, the current leader in the field.

To facilitate others to build on our work, as well as to ensure reproducibility, we have made our code and the full raw results available at https://github.com/fpetitjean/ProximityForest/.

4.1 Case study: satellite image time series dataset

The SITS dataset contains approximately 1 million time series with a train-test split of approximately 90–10%.[2] Each time series has a length of 46 and is labeled as one of 24 possible land-use classes (e.g. 'wheat', 'corn', 'plantation', 'urban'). Here the labeled data has been extracted from three sources: (1) ground field campaigns for most of the vegetation classes, (2) farmer's declaration to complete the data for some crop classes, and (3) existing map for the urban areas.

The experiments presented in this section were performed on this dataset, comparing the performance of Proximity Forest against three competitors: BOSS-VS (designed for scalability), WEASEL (designed for speed and quality), and EE (designed for quality). We use 5 runs for each experiment of Proximity Forest and 1 run of each of the competitors—as their results are deterministic. Throughout this experiment, we use 100 trees; we will see in Sect. 4.3 that this gives a good tradeoff between accuracy and computational time/memory. Although we are mainly assessing the scalability, we will also have a quick look to the accuracy.

[2] The split ensures that no 2 times series come from the same plot of land.

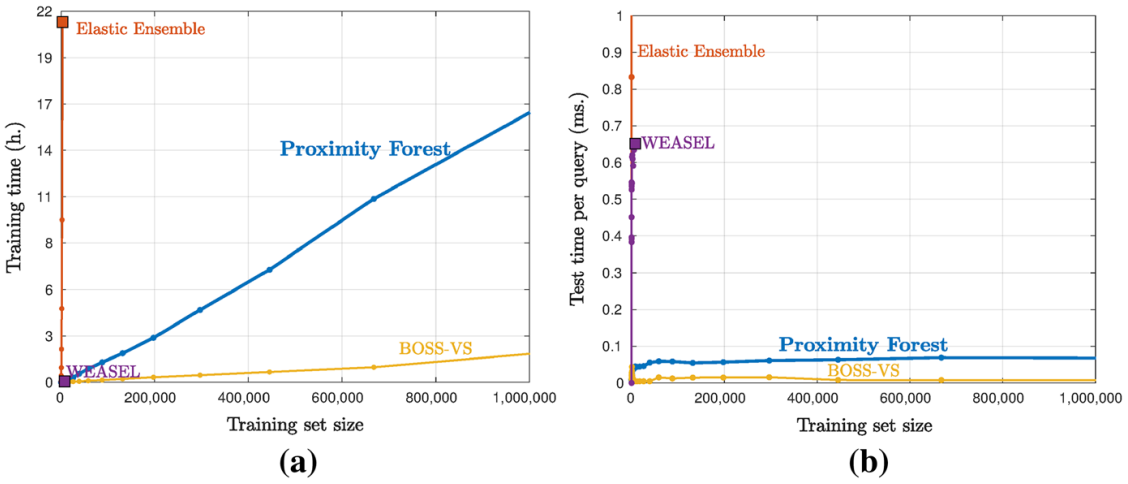


Fig. 3 Training time (**a**) and testing time per query (**b**) as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS

4.1.1 Training scalability

To assess scalability, we train and test each algorithm on subsample data with increasing training set size, allowing training time, testing time and accuracy to be measured as a function of training size. Figure 3a shows training time against training size for each of the 4 algorithms.

Versus EE. First, it is evident that Proximity Forest presents a notable saving in training time over EE, confirming that it trains in linear time rather than the quadratic time for EE. Even for a small training set of about 2000 time series, learning an EE model took about 10 h, compared to Proximity Forest's 79 sec. Fitting a quadratic curve through both EE and Proximity Forest is quite informative: EE returns a quadratic component of 6.3 while Proximity Forest only -8.10×10^{-6}, clearly highlighting both the quadratic complexity of EE, and also that Proximity Forest is in practice very close to its theoretical average complexity and scales quasi-linearly with n.

Versus WEASEL. WEASEL is very fast but its memory footprint did not allow it to scale beyond 8000 time series even when given 64 GB of RAM. This clearly highlights the difference with BOSS-VS: we can see that WEASEL was not developed for scalability, but rather for speed on small datasets.

Versus BOSS-VS Proximity Forest trains slower than BOSS-VS for a given training size, however this is counteracted by the low accuracy of BOSS-VS discussed below.

4.1.2 Testing scalability

Figure 3b shows testing time against training size for each of the 4 algorithms. The story here is very similar to that of training: it confirms the way Proximity Forest scales logarithmically with training set size, while EE must scan the full database many times. Here again, WEASEL becomes infeasible to apply with relatively small quantities of training data. Proximity Forest and BOSS-VS require respectively 0.0679 ms and 0.0077 ms to classify a time series with a model trained on 1M time series.

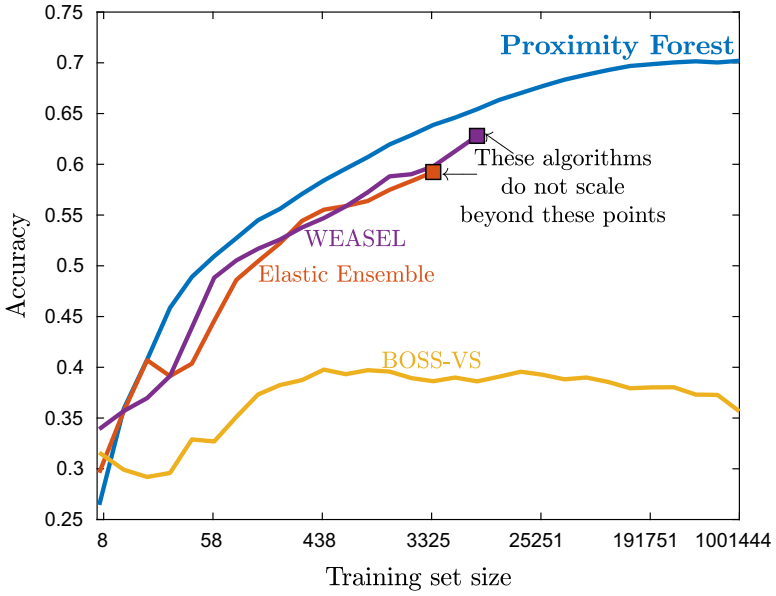


Fig. 4 Accuracy as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS

4.1.3 Is proximity forest accurate and scalable?

We have now seen that Proximity Forest is highly scalable and only beaten by BOSS-VS in terms of training time. We will now study how its accuracy scales with training set size. The main results are presented in Fig. 4 which plots the accuracy as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS.

The first element to observe is that Proximity Forest obtains greater accuracy than the competitors for large training sets. WEASEL and EE become infeasible to apply at relatively small data quantities and BOSS-VS—which is faster than Proximity Forest—does not learn effective classifiers on this dataset. With 63.8% accuracy at 3400 training set size, this is 26.3 percentage points more accurate than BOSS-VS, and 4.6 and 4.7 percentage points more accurate than WEASEL and EE, respectively. Such differences are substantial in a problem comprising 24 classes.

Moreover, Proximity Forest is more accurate than the other algorithms from 500 training instances upwards. This is not surprising, as trees usually have a better control over variance than NN algorithms, because of their higher bias and abstraction capabilities. Proximity Forest thus appears to be both accurate and highly scalable. We will show in the next subsection that this result holds also on the benchmark UCR archive.

4.2 Experiments on the UCR archive

In this section, we study the behavior of Proximity Forest on the 85 datasets of the traditional UCR archive (Chen et al. 2015). It is useful to remember here that our aim is not to show that Proximity Forest is more accurate than the state of the art, but only that it is competitive while being highly scalable. We compare the mean error-rate of Proximity Forest to the error-rates on the standard train/test split for the state of the art, as tested in Bagnall et al. (2017). We average Proximity Forest results over 10

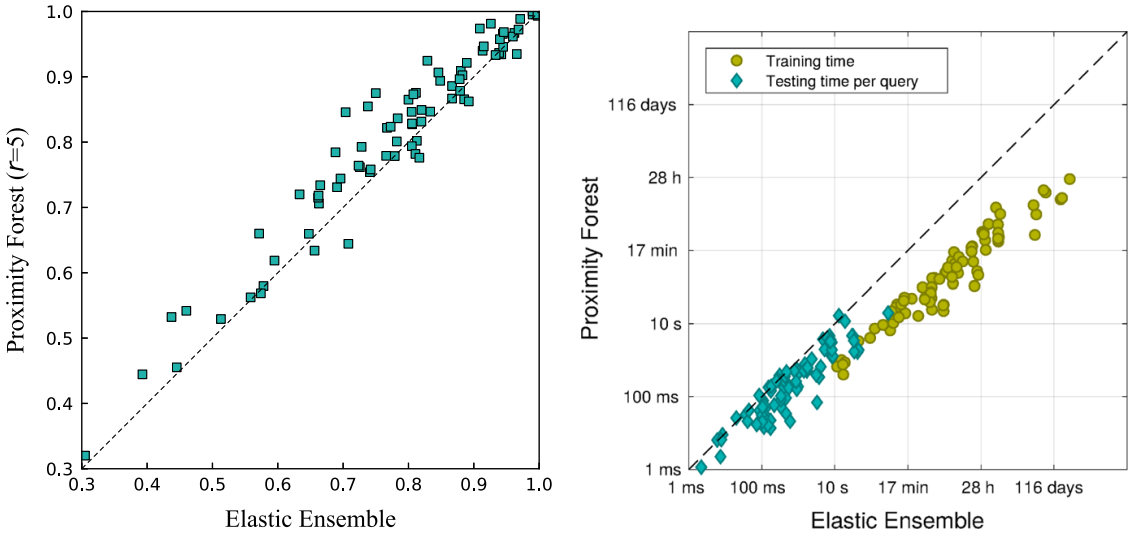


Fig. 5 Comparison of Proximity Forest and Elastic Ensemble classifiers on UCR datasets in terms of **a** accuracy and **b** training and testing times in log scale

runs for each experiment. We compare Proximity Forest to five classifiers currently representing the state-of-the-art —DTW-R, COTE, EE, ST and BOSS[3]. The Proximity Forest results are obtained for 100 trees with selection between 5 candidates per node. A detailed discussion about the Proximity Forest parameters will be performed in Sect. 4.3.

We first show the comparison with Proximity Forest's closest relative, EE. Figure 5 provides scatter plots of the relative accuracy, total training time and total testing time of each of these classifiers. Each point represents a different UCR dataset. Figure 5a shows that Proximity Forest is more accurate on 60 datasets and less accurate on only 11 datasets, with 14 ties. Moreover, for many datasets Proximity Forest is substantially more accurate than EE.

Figure 5b demonstrates that Proximity Forest has several orders of magnitude advantage in training time. When considering testing time, Proximity Forest has greater test time per query than EE for 12 datasets, the majority of which are small datasets (i.e. less than 50 training instances). The largest such difference is observed for the Phonemes dataset for which Proximity Forest takes about 17 sec per query compared to 13 sec per query for EE. In contrast, the test time for Proximity Forest is much smaller than EE for the biggest datasets (i.e. more than 800 training instances). For example, the biggest test time difference is for the HandOutlines dataset for which Proximity Forest takes about 19 sec per query compared to 286 sec per query for EE.

The commonly accepted method to compare multiple classifiers over multiple datasets is by average ranks. For each dataset, we rank the classifiers and then calculate the average of each classifier's ranks across all datasets. When comparing 6 algorithms over 85 datasets, Demšar (2006) shows that for the rankings to be significantly differ-

[3] It should be highlighted that the results presented here are for the original BOSS algorithm, and not the BOSS-VS discussed above in the SITS experiments. BOSS-VS is a scalable variation of BOSS, where concessions are made to accuracy in favor of training time. The original BOSS is therefore more competitive in this section.

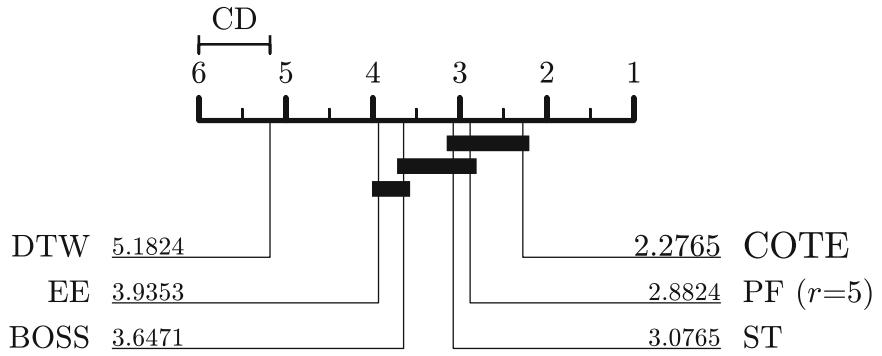


Fig. 6 Critical difference diagram for five state-of-the-art classifiers and Proximity Forest (PF) with 5 candidates

ent at level $\alpha = 0.05$, the critical difference (CD) between the average ranks has to be greater than:

$$CD = q_{0.05}(A) \cdot \sqrt{\frac{A(A+1)}{6 \cdot N_d}} = 2.850 \cdot \sqrt{\frac{42}{510}} \approx 0.82 \quad (1)$$

The average ranks and critical difference are presented in Fig. 6; the critical difference of 0.82 is displayed by the black line. It can be seen that COTE ranks highest (average rank of 2.28), which is to be expected considering it incorporates the other state-of-the-art algorithms. However, COTE is not ranked significantly higher than Proximity Forest (average rank of 2.88) or ST (average rank of 3.08). Proximity Forest is ranked second. Its rank is not significantly different to COTE, ST or BOSS, but it is ranked significantly higher than both EE and DTW. This affirms Proximity Forest as a classifier with accuracy competitive with the state of the art.

Proximity Forest is the most accurate classifier for 22 of the 85 datasets. However, there is no obvious commonality between these datasets to suggest conditions under which the algorithm is likely to excel. The detailed accuracy results for Proximity Forest and the five state-of-the-art algorithms are shown in "Appendix A".

4.3 Parameters of Proximity Forest

Proximity Forest has two main parameters that merit further investigation. We first explore the sensitivity of accuracy to the number of trees in each ensemble. Then, we discuss the influence of the number of candidates assessed at each node. A third design choice, random selection of similarity measure per tree as opposed to per node, is explored in "Appendix B".

4.3.1 On the choice of the number of trees

The number of trees is the first parameter of the Proximity Forest algorithm with the optimal value being large enough to provide competitive accuracy, yet small enough

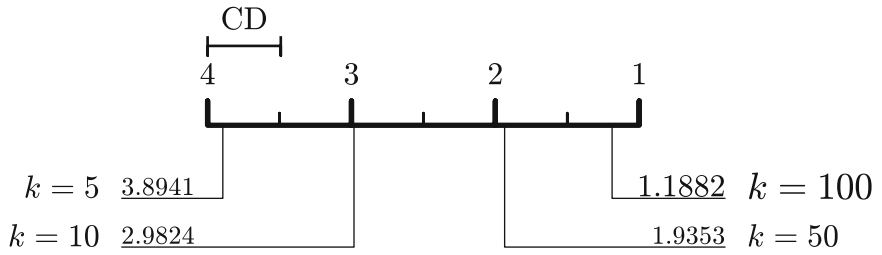


Fig. 7 Critical difference diagram for Proximity Forest with 5, 10, 50 or 100 trees

not to create excessive computational expense. The UCR datasets experiments outlined above were repeated with values of 5, 10, 50, and 100 trees to analyse how many trees were required to meet our needs. Here, the number of candidates r has been fixed to 1. The Proximity Forest results are averaged over 50 runs. Figure 7 presents the critical difference diagram for accuracy and different number of trees. As expected, the more trees the higher the average accuracy: models with 100 trees had an average rank of 1.19 compared to 1.93, 2.98 and 3.89 for models with 50, 10, and 5 trees respectively. The difference between the highest ranked models are large enough to say that models with 100 trees are significantly better than models with 50 trees at the level of alpha equals 0.05.

Figure 8 compares the classification accuracy for 100 trees against 10 and 50 trees by representing them as a ratio of their error rates. Each point represents a single dataset. This shows that having 100 trees is better on most datasets. Moreover, the fact that the data is gathered close to the line with equation $x = 1$ shows that it is unlikely that more trees would provide a very significant improvement, because the ratio of error-rates between 100 and 50 is already close to 1 (i.e., the errors are only slightly reduced). We have not experimented with forests comprising more than 100 trees as we felt the computational demands outweighed the expected benefits for our large set of experiments. Memory, training time and testing time all scale linearly with the number of trees, which means that doubling the number of trees doubles the required memory and time. However, where computational resources are not an issue, the take home message is that the more trees the better.

As a randomized algorithm, it is finally interesting to study the standard deviation of the errors for Proximity Forest and how it varies with the number of trees. This is what we present in Fig. 9 where the y-axis represents the standard deviation on error-rate for 100 trees as a function of the standard deviation on k equals to 5, 10, and 50 trees. Each point represents a single dataset. One can see that the standard deviation reduces as we increase the number of trees, and that the magnitude of this improvement reduces when increasing k. Results for 50 trees are starting to be relatively close to the $y = x$ line, showing that only marginal improvements could be expected when going to $k > 100$.

4.3.2 Split selection using the Gini index

This section explores the influence of the number of candidates r that are randomly selected at each node. As a reminder, a set of r candidates—exemplars and parametrized distance measures—is evaluated at each node based on the Gini index. The one maximizing the Gini index is retained. To evaluate the influence of r, the

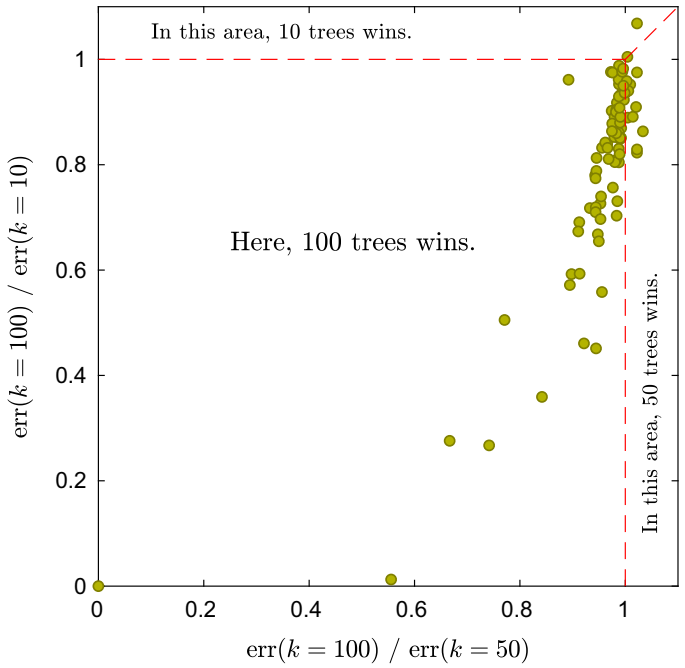


Fig. 8 Ratio of the error rates of Proximity Forest models: 100 trees over 10 trees (x-axis) against 100 trees over 50 trees (y-axis). A value of less than 1 on either axis indicates that the model with 100 trees has higher accuracy

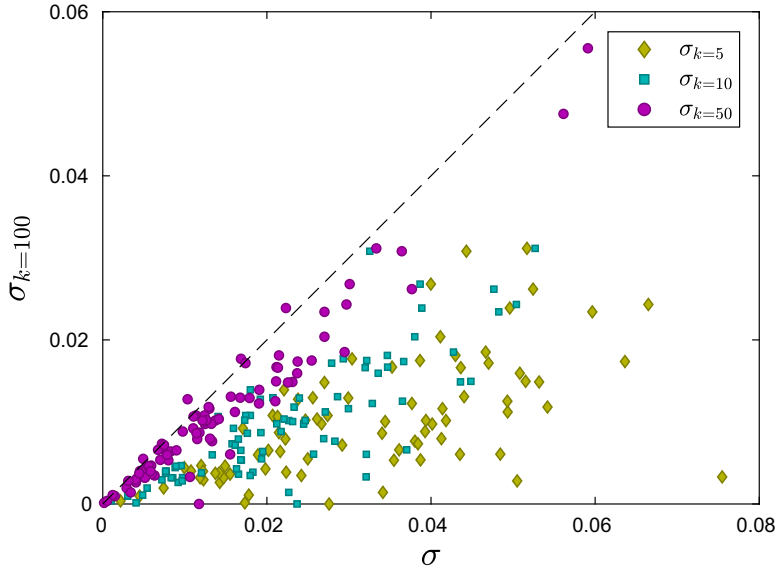


Fig. 9 Standard deviations σ of error rates on the 85 datasets of the UCR archive for Proximity Forest models: 100 trees against 50, 10 and 5 trees

UCR experiments were repeated for 1, 2, and 5 candidates on 100 trees. The results are averaged over 10 runs.

Figure 10 compares the classification accuracy for 5 candidates against 1 and 2 candidates. Each point represents the ratio of the error for 5 candidates to that for the alternative on an UCR dataset. Choosing between 5 candidates results in higher accuracy for most datasets. More precisely, selecting between 5 candidates results in greater accuracy than either 1 or 2 candidates on 61 datasets. Increasing the number of candidates lead to a reduction of the randomness on each node by discarding the worse splitters. Accordingly, the overall Proximity Forest accuracies are improved.

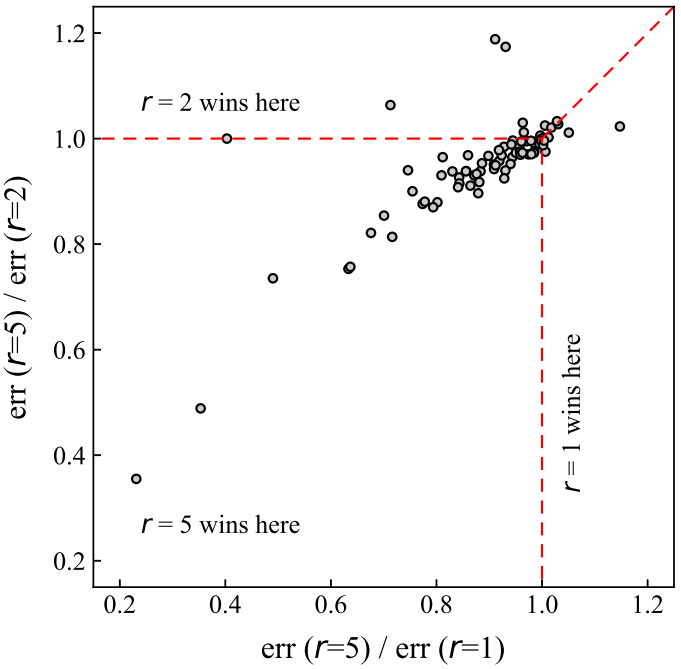


Fig. 10 Ratio of the error rates of Proximity Forest models: 5 candidates over 1 candidate (x-axis) against 5 candidates over 2 candidates (y-axis). A value of less than 1 on either axis suggests that the model with 5 candidates has superior accuracy

Increasing the number of candidates to more than 5 may further improve the classification accuracy. However, increasing the number of candidates per node has substantial impact on training time. Indeed, the analysis of the Proximity Forest's computational complexity in Sect. 3.4 shows that the training time scales linearly with the number of candidates. To verify this analysis, we compare both training and testing time of Proximity Forest for 1 and 5 candidates in Fig. 11. The testing time is displayed per query. Each point represents a dataset. The results show a mean increase of 4.6 times in training time between 1 and 5 candidates, and a mean decrease of 0.93 times in testing time.

It is notable that selection between multiple alternatives both reduces testing time and increases the training time by slightly less than the expected multiple of 5 times. This is because it results in slightly shallower trees. Selection of better splits better separates the classes, requiring fewer splits to obtain pure nodes that are made into leaves.

The tuning of the number of candidates is therefore driven by a trade-off between accuracy and time.

5 Conclusion

We introduced Proximity Forest: a novel, scalable algorithm for accurate time series classification. Motivated by a need for an algorithm that could learn from millions of time series, Proximity Forest is an ensemble of trees with a novel splitting criterion that makes it possible to make the most of decades of work in designing time series measures. In our case study, we demonstrated that Proximity Forest scales quasi-linearly with the quantity of training data, whereas most state of the art algorithms scale quadratically. Our experiments on the UCR datasets show that Proximity Forest

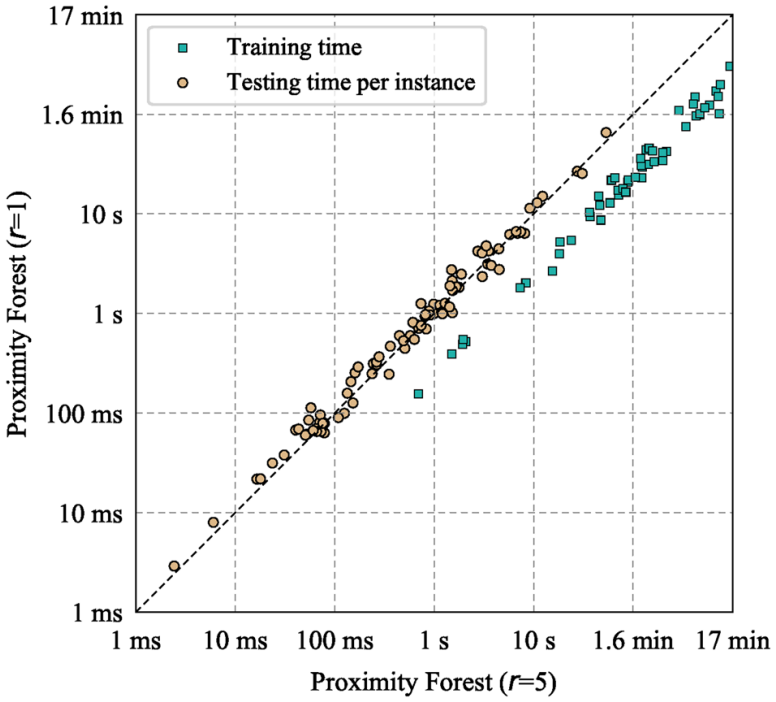


Fig. 11 Training and testing time of Proximity Forest for 1 and 5 candidates on UCR datasets

is not only very fast. It also has highly competitive accuracy relative to the current state-of-the-art, and is significantly more accurate than EE.

We believe that there are a number of improvements that can be explored to increase the accuracy of Proximity Forest while maintaining its quasi-linear complexity, such as improving the randomized selection of parameters for the distance measures—the current strategy was designed primarily to emulate EE as directly as possible. We would also like to investigate to what extent this novel algorithm might shed new light on the task of time series indexing.

Supplementary material

To ensure reproducibility, we make available the results of the experiments as well as our source code at https://github.com/fpetitjean/ProximityForest/.

Acknowledgements This research was supported by the Australian Research Council under Grant DE170100037. This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-17-1-4036. We are grateful to the editor and anonymous reviewers whose suggestions and comments have greatly strengthened the paper. The authors would also like to thank Prof Eamonn Keogh and all of the people who have contributed to the UCR time series classification archive.

Appendix A: Detailed UCR results

See Table 1.

Table 1 Detailed UCR results for five state-of-the-art algorithms and Proximity Forest

	Train	*Test*	*l*	*c*	DTW	BOSS	ST	EE	COTE	PF
Adiac	390	391	176	37	60.87	76.47	78.26	66.5	**79.03**	73.4
ArrHead	36	175	251	3	80.0	83.43	73.71	81.14	81.14	**87.54**
Beef	30	30	470	5	66.67	80.0	**90.0**	63.33	86.67	72.0
BeetFly	20	20	512	2	65.0	**90.0**	**90.0**	75.0	80.0	87.5
BirdChi	20	20	512	2	70.0	**95.0**	80.0	80.0	90.0	86.5
Car	60	60	577	4	76.67	83.33	**91.67**	83.33	90.0	84.67
CBF	30	900	128	3	99.44	**99.78**	97.44	**99.78**	99.56	99.33
ChloCon	467	3840	166	3	65.0	66.09	69.97	65.62	**72.71**	63.39
CinCECG	40	1380	1639	4	93.04	88.7	95.43	94.2	**99.49**	93.43
Coffee	28	28	286	2	**100.0**	**100.0**	96.43	**100.0**	**100.0**	**100.0**
Comput	250	250	720	2	62.4	**75.6**	73.6	70.8	74.0	64.44
CricketX	390	390	300	12	77.95	73.59	77.18	**81.28**	80.77	80.21
CricketY	390	390	300	12	75.64	75.38	77.95	80.51	**82.56**	79.38
CricketZ	390	390	300	12	73.59	74.62	78.72	78.21	**81.54**	80.1
DiaSizRed	16	306	345	4	93.46	93.14	92.48	94.44	92.81	**96.57**
DisPhAG	400	139	80	3	62.59	74.82	**76.98**	69.06	74.82	73.09
DisPhOC	600	276	80	2	72.46	72.83	77.54	72.83	76.09	**79.28**
DisPhTW	400	139	80	6	63.31	67.63	66.19	64.75	**69.78**	65.97
Earthqua	322	139	512	2	72.66	74.82	74.1	74.1	74.82	**75.4**
ECG200	100	100	96	2	88.0	87.0	83.0	88.0	88.0	**90.9**
ECG5000	500	4500	140	5	92.51	94.13	94.38	93.87	**94.6**	93.65
ECG5days	23	861	136	2	79.67	**100.0**	98.37	82.0	99.88	84.92
ElecDev	8926	7711	96	7	63.08	**79.92**	74.7	66.29	71.33	70.6
FaceAll	560	1690	131	14	80.77	78.17	77.87	84.85	**91.78**	89.38
FaceFour	24	88	350	4	89.77	**100.0**	85.23	90.91	89.77	97.39
FacesUCR	200	2050	131	14	90.78	**95.71**	90.59	94.49	94.24	94.59
50Words	450	455	270	50	76.48	70.55	70.55	81.98	79.78	**83.14**
Fish	175	175	463	7	83.43	**98.86**	**98.86**	96.57	98.29	93.49
FordA	3601	1320	500	2	66.52	92.95	**97.12**	73.79	95.68	85.46
FordB	3636	810	500	2	59.88	71.11	**80.74**	66.17	80.37	71.49
GunPoint	50	150	150	2	91.33	**100.0**	**100.0**	99.33	**100.0**	99.73
Ham	109	105	431	2	60.0	66.67	**68.57**	57.14	64.76	66.0
HandOut	1000	370	2709	2	87.84	90.27	**93.24**	88.92	91.89	92.14
Haptics	155	308	1092	5	41.56	46.1	**52.27**	39.29	**52.27**	44.45
Herring	64	64	512	2	53.12	54.69	**67.19**	57.81	62.5	57.97
InlSkate	100	550	1882	7	38.73	51.64	37.27	46.0	49.45	**54.18**
InsWinSou	220	1980	256	11	57.37	52.32	62.68	59.49	**65.25**	61.87
ItPowDem	67	1029	24	2	95.53	90.86	94.75	96.21	96.11	**96.71**
LaKitAp	375	375	720	3	79.47	76.53	**85.87**	81.07	84.53	78.19

Table 1 continued

	Train	*Test*	*l*	*c*	DTW	BOSS	ST	EE	COTE	PF
Light2	60	61	637	2	86.89	83.61	73.77	**88.52**	86.89	86.56
Light7	70	73	319	7	71.23	68.49	72.6	76.71	80.82	**82.19**
Mallat	55	2345	1024	8	91.43	93.82	**96.42**	93.99	95.39	95.76
Meat	60	60	448	3	**93.33**	90.0	85.0	**93.33**	91.67	**93.33**
MedImg	381	760	99	10	74.74	71.84	66.97	74.21	75.79	**75.82**
MidPhAG	400	154	80	3	51.95	54.55	**64.29**	55.84	63.64	56.23
MidPhOC	600	291	80	2	76.63	78.01	79.38	78.35	80.41	**83.64**
MidPhTW	399	154	80	6	50.65	54.55	51.95	51.3	**57.14**	52.92
MotStr	20	1252	84	2	86.58	87.86	89.7	88.26	**93.69**	90.24
NoECGT1	1800	1965	750	42	82.9	83.82	**94.96**	84.58	93.13	90.66
NoECGT2	1800	1965	750	42	87.02	90.08	**95.11**	91.35	94.55	93.99
OliveOil	30	30	570	4	86.67	86.67	**90.0**	86.67	**90.0**	86.67
OSULeaf	200	242	427	6	59.92	95.45	**96.69**	80.58	**96.69**	82.73
PhalOC	1800	858	80	2	76.11	77.16	76.34	77.27	77.04	**82.35**
Phoneme	214	1896	1024	39	22.68	26.48	32.07	30.49	**34.92**	32.01
Plane	105	105	144	7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
ProPhAG	400	205	80	3	78.54	83.41	84.39	80.49	**85.37**	84.63
ProPhOC	600	291	80	2	79.04	84.88	**88.32**	80.76	86.94	87.32
ProPhTW	400	205	80	6	76.1	80.0	**80.49**	76.59	78.05	77.9
RefrigDev	375	375	720	3	44.0	49.87	**58.13**	43.73	54.67	53.23
ScrType	375	375	720	3	41.07	46.4	52.0	44.53	**54.67**	45.52
ShapSim	20	180	500	2	69.44	**100.0**	95.56	81.67	96.11	77.61
ShapAll	600	600	512	60	80.17	**90.83**	84.17	86.67	89.17	88.58
SmKitAp	375	375	720	3	67.2	72.53	**79.2**	69.6	77.6	74.43
SonyAIR1	20	601	70	2	69.55	63.23	84.36	70.38	84.53	**84.58**
SonyAIR2	27	953	65	2	85.94	85.94	93.39	87.83	**95.17**	89.63
StarCur	1000	8236	1024	3	89.83	97.78	97.85	92.61	97.96	**98.13**
Strawber	613	370	235	2	94.59	**97.57**	96.22	94.59	95.14	96.84
SwedLeaf	500	625	128	15	84.64	92.16	92.8	91.52	**95.52**	94.66
Symbols	25	995	398	6	93.77	**96.68**	88.24	95.98	96.38	96.16
SynCon	300	300	60	6	98.33	96.67	98.33	99.0	**100.0**	99.53
ToeSeg1	40	228	277	2	75.0	93.86	96.49	82.89	**97.37**	92.46
ToeSeg2	36	130	343	2	90.77	**96.15**	90.77	89.23	91.54	86.23
Trace	100	100	275	4	99.0	**100.0**	**100.0**	99.0	**100.0**	**100.0**
2LeECG	23	1139	82	2	86.83	98.07	**99.74**	97.1	99.3	98.86
2Patterns	1000	4000	128	4	99.85	99.3	95.5	**100.0**	**100.0**	99.96
UWaAll	1000	6164	152	8	96.23	93.89	94.22	96.85	96.43	**97.23**
UWaX	896	3582	315	8	77.44	76.21	80.29	80.54	82.19	**82.86**
UWaY	896	3582	315	8	70.18	68.51	73.03	72.56	75.85	**76.15**

Table 1 continued

	Train	*Test*	*l*	*c*	DTW	BOSS	ST	EE	COTE	PF
UWaZ	896	3582	945	8	67.5	69.49	74.85	72.36	75.04	**76.4**
Wafer	896	3582	315	2	99.59	99.48	**100.0**	99.74	99.98	99.55
Wine	57	54	234	2	61.11	74.07	**79.63**	57.41	64.81	56.85
WordSyn	267	638	270	25	74.92	63.79	57.05	**77.9**	75.71	77.87
Worms	181	77	900	5	53.25	55.84	**74.03**	66.23	62.34	71.82
Worms2	181	77	900	2	58.44	**83.12**	**83.12**	68.83	80.52	78.44
Yoga	300	3000	426	2	84.3	**91.83**	81.77	87.9	87.67	87.86
Av. rank					5.18	3.65	3.08	3.95	**2.28**	2.88
Wins					3	20	**30**	8	26	22

Bold values indicate the best accuracy scores. Proximity Forest results are obtained for 100 trees and 5 candidates, and averaged over 10 runs

Appendix B: On a variation of the proximity forest

We decided to explore another variant of the Proximity Forest algorithm by randomly selecting a distance measure for each tree, rather than for each node. In this new variant, only the exemplars and the parameters of the distance-metric are randomly chosen at each node. The UCR experiments were repeated for 100 trees and 1 candidate for this new 'on tree' variant. Each Proximity Forest result is averaged over 50 runs.

Figure 12 compares classification accuracy for the original version 'on node', presented in Sect. 3.2, and the proposed variant 'on tree'. Each point represents a single dataset of the UCR dataset. The number of trees has been fixed to 100.

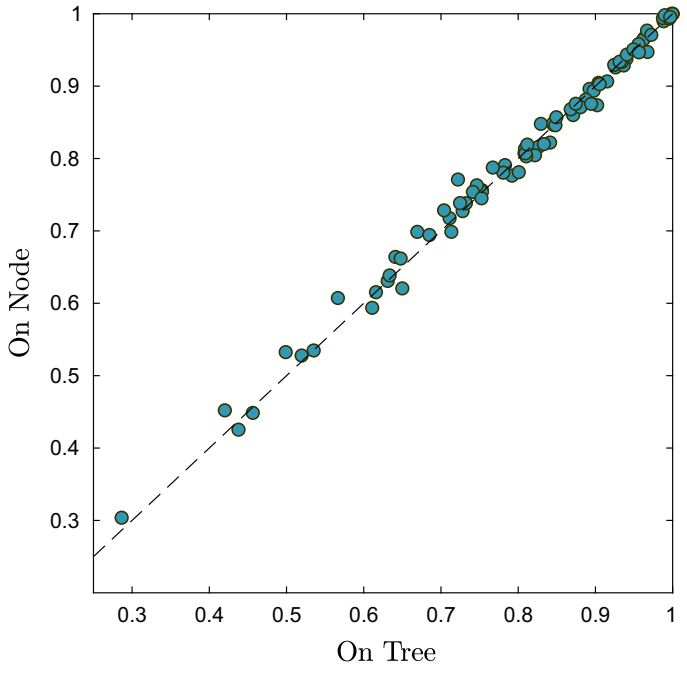


Fig. 12 Accuracy of Proximity Forest when randomly selecting the distance measure 'on node' and 'on tree'

The results show a slight advantage for the 'on node' approach with 44 wins, 39 losses and 2 ties. Where the 'on tree' variant uses a single distance measure per tree, the 'on node' variant allows multiple combinations of measures in a single tree, thus making it more robust to inefficient metrics.

References

Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Discov 31(3):606–660
Bagnall A, Lines J, Hills J, Bostrom A (2015) Time-series classification with COTE: the collective of transformation-based ensembles. IEEE Trans Knowl Data Eng 27(9):2522–2535
Balakrishnan S, Madigan D (2006) Decision trees for functional variables. In: IEEE international conference on data mining (ICDM-06), pp 798–802
Bernhardsson E (2013) Indexing with annoy. https://github.com/spotify/annoy. Accessed 23 March 2018
Breiman L (2001) Random forests. Mach Learn 45(1):5–32
Chen L, Ng R (2004) On the marriage of lp-norms and edit distance. In: Proceedings of the thirtieth international conference on very large data bases, vol 30, pp 792–803. VLDB Endowment
Chen L, Özsu M T, Oria V (2005) Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, pp 491–502. ACM
Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2015) The UCR time series classification archive . www.cs.ucr.edu/~eamonn/time_series_data/. Accessed 23 March 2018
Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Inf Sci 239:142–153
Douzal-Chouakria A, Amblard C (2012) Classification trees for time series. Pattern Recognit 45(3):1076–1091
Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42
Górecki T, Łuczak M (2013) Using derivatives in time series classification. Data Min Knowl Discov 26(2):310–331
Grabocka J, Wistuba M, Schmidt-Thieme L (2016) Fast classification of univariate and multivariate time series through shapelet discovery. Knowl Inf Syst 49(2):429–454
Haghiri S, Ghoshdastidar D, von Luxburg U (2017) Comparison-based nearest neighbor search. arXiv e-prints, arXiv:1704.01460
Haghiri S, Garreau D, von Luxburg U (2018) Comparison-based random forests. arXiv e-prints, arXiv:1806.06616
Hamooni H, Mueen A (2014) Dual-domain hierarchical classification of phonetic time series. In: 2014 IEEE international conference on data mining, pp 160–169. IEEE
Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014) Classification of time series by shapelet transformation. Data Min Knowl Discov 28(4):851–881
Ho TK (1995) Random decision forests. In: Proceedings of the third international conference on document analysis and recognition, 1995, vol 1, pp 278–282. IEEE
Jeong YS, Jeong MK, Omitaomu OA (2011) Weighted dynamic time warping for time series classification. Pattern Recognit 44(9):2231–2240
Karlsson I, Papapetrou P, Boström H (2016) Generalized random shapelet forests. Data Min Knowl Discov 30(5):1053–1085
Keogh E, Wei L, Xi X, Lee S H, Vlachos M (2006) LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In: Proceedings of the 32nd international conference on very large data bases, pp 882–893. VLDB Endowment
Keogh EJ, Pazzani MJ (2001) Derivative dynamic time warping. In: Proceedings of the 2001 SIAM international conference on data mining, pp 1–11. SIAM
Lemire D (2009) Faster retrieval with a two-pass dynamic-time-warping lower bound. Pattern Recognit 42(9):2169–2180
Lifshits Y (2010) Nearest neighbor search: algorithmic perspective. SIGSPATIAL Spec 2(2):12–15

Lin J, Khade R, Li Y (2012) Rotation-invariant similarity in time series using bag-of-patterns representation. J Intell Inf Syst 39(2):287–315
Lines J, Bagnall A (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Discov 29(3):565
Marteau PF (2009) Time warp edit distance with stiffness adjustment for time series matching. IEEE Trans Pattern Anal Mach Intell 31(2):306–318
Marteau PF (2016) Times series averaging and denoising from a probabilistic perspective on time-elastic kernels. arXiv preprint, arXiv:1611.09194
Muja M. FLANN-Fast library for approximate nearest neighbors. www.cs.ubc.ca/research/flann/. Accessed 23 March 2018
Pękalska E, Duin RP, Paclík P (2006) Prototype selection for dissimilarity-based classifiers. Pattern Recognit 39(2):189–208
Petitjean F, Forestier G, Webb GI, Nicholson AE, Chen Y, Keogh E (2014) Dynamic time warping averaging of time series allows faster and more accurate classification. In: 2014 IEEE international conference on data mining, pp 470–479. IEEE
Petitjean F, Forestier G, Webb GI, Nicholson AE, Chen Y, Keogh E (2016) Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. Knowl Inf Syst 47(1):1–26
Petitjean F, Gançarski P (2012) Summarizing a set of time series by averaging: from Steiner sequence to compact multiple alignment. Theor Comput Sci 414(1):76–91
Rakthanmanon T, Keogh E (2013) Fast shapelets: a scalable algorithm for discovering time series shapelets. In: Proceedings of the 13th SIAM international conference on data mining, pp 668–676. SIAM
Sakoe H, Chiba S (1971) A dynamic programming approach to continuous speech recognition. In: Proceedings of the seventh international congress on acoustics, vol 3, pp 65–69. Budapest, Hungary
Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust Speech Signal Process 26(1):43–49
Sathe S, Aggarwal CC (2017) Similarity forests. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, KDD '17, pp 395–403. ACM. https://doi.org/10.1145/3097983.3098046
Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Min Knowl Discov 29(6):1505–1530
Schäfer P (2015) Scalable time series classification. Data Min Knowl Discov 2:1–26
Schäfer P, Högqvist M (2012) SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In: Proceedings of the 15th international conference on extending database technology, EDBT '12, pp 516–527. ACM. https://doi.org/10.1145/2247596.2247656
Schäfer P, Leser U (2017) Fast and accurate time series classification with WEASEL. In: Proceedings of the 2017 ACM on conference on information and knowledge management, pp 637–646. ACM
Senin P, Malinchik S (2013) SAX-VSM: Interpretable time series classification using SAX and vector space model. In: 2013 IEEE 13th international conference on data mining, pp 1175–1180. IEEE
Stefan A, Athitsos V, Das G (2013) The move-split-merge metric for time series. IEEE Trans Knowl Data Eng 25(6):1425–1438
Tan CW, Webb GI, Petitjean F (2017) Indexing and classifying gigabytes of time series under time warping. In: Proceedings of the 2017 SIAM international conference on data mining, pp 282–290. SIAM
Ting K.M, Zhu Y, Carman M, Zhu Y, Zhou Z.H (2016) Overcoming key weaknesses of distance-based neighbourhood methods using a data dependent dissimilarity measure. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1205–1214. ACM
Ueno K, Xi X, Keogh E, Lee DJ (2006) Anytime classification using the nearest neighbor algorithm with applications to stream mining. In: 6th international conference on data mining, 2006. ICDM'06, pp 623–632. IEEE
Vlachos M, Hadjieleftheriou M, Gunopulos D, Keogh E (2006) Indexing multidimensional time-series. Int J Very Large Data Bases 15(1):1–20
Wang X, Mueen A, Ding H, Trajcevski G, Scheuermann P, Keogh E (2013) Experimental comparison of representation methods and distance measures for time series data. Data Min Knowl Discov 26(2):275–309
Yamada Y, Suzuki E, Yokoi H, Takabayashi K (2003) Decision-tree induction from time-series data based on a standard-example split test. In: Proceedings of the twentieth international conference on international

conference on machine learning, ICML'03, pp 840–847. AAAI Press. http://dl.acm.org/citation.cfm?id=3041838.3041944
Ye L, Keogh E (2011) Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min Knowl Discov 22(1):149–182

Affiliations

Benjamin Lucas[1] · Ahmed Shifaz[1] · Charlotte Pelletier[1] · Lachlan O'Neill[1] · Nayyar Zaidi[1] · Bart Goethals[1] · François Petitjean[1] · Geoffrey I. Webb[1]

Ahmed Shifaz
ahmed.shifaz@monash.edu

Charlotte Pelletier
charlotte.pelletier@monash.edu

Lachlan O'Neill
lsone1@monash.edu

Nayyar Zaidi
nayaar.zaidi@monash.edu

Bart Goethals
bart.goethals@monash.edu

François Petitjean
francois.petitjean@monash.edu

Geoffrey I. Webb
geoff.webb@monash.edu

[1] Faculty of Information Technology, Monash University, 25 Exhibition Walk, Melbourne, VIC 3800, Australia

Bibliography

[1] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The UCR Time Series Classification Archive, July 2015. www.cs.ucr.edu/~eamonn/time_series_data/, Last accessed on 2021-08-01.

[2] Robert Thomas Olszewski. *Generalized feature extraction for structural pattern recognition in time-series data*. PhD thesis, Carnegie Mellon University, 2001.

[3] Chang Wei Tan, Geoffrey I Webb, and François Petitjean. Indexing and classifying gigabytes of time series under time warping. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 282–290. SIAM, 2017.

[4] Charlotte Pelletier, Geoffrey I Webb, and François Petitjean. Temporal Convolutional Neural Network for the classification of satellite image time series. *Remote Sensing*, 11(5):523, 2019.

[5] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The UEA Multivariate Time Series Classification Archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

[6] 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 Mining and Knowledge Discovery*, 35(2):401–449, 2021.

[7] 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. *arXiv preprint arXiv:2104.07551*, 2021.

[8] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey I Webb. Proximity Forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, 33(3):607–635, 2019.

[9] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3): 606–660, 2017.

[10] Philippe Esling and Carlos Agon. Multiobjective time series matching for audio classification and retrieval. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(10):2057–2072, 2013.

[11] Sadi Evren Seker, Cihan Mert, Khaled Al-Naami, Ugur Ayan, and Nuri Ozalp. Ensemble classification over stock market time series and economy news. In *2013 IEEE International Conference on Intelligence and Security Informatics*, pages 272–273. IEEE, 2013.

[12] Xiao-Ye Wang and Zheng-Ou Wang. Stock market time series data mining based on regularized neural network and rough set. In *Proceedings. International Conference on Machine Learning and Cybernetics*, volume 1, pages 315–318. IEEE, 2002.

[13] Alina Delia Calin. Gesture recognition on Kinect time series data using Dynamic Time Warping and Hidden Markov Models. In *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scienti c Computing (SYNASC)*, pages 264–271. IEEE, 2016.

[14] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26 (2):275–309, 2013.

[15] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592, 2015.

[16] Donald J Berndt and James Clifford. Using Dynamic Time Warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[17] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information Systems*, 53:16–38, 2015.

[18] T Warren Liao. Clustering of time series data—a survey. *Pattern Recognition*, 38 (11):1857–1874, 2005.

[19] Chang Wei Tan, Christoph Bergmeir, Francois Petitjean, and Geoffrey I Webb. Monash University, UEA, UCR Time Series Regression Archive. *arXiv preprint arXiv:2006.10996*, 2020.

[20] Tao Hong, Pierre Pinson, Shu Fan, Hamidreza Zareipour, Alberto Troccoli, and Rob J Hyndman. Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond, 2016.

[21] Ahmed Tealab. Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Computing and Informatics Journal*, 3(2): 334–340, 2018.

[22] Hesam Izakian and Witold Pedrycz. Anomaly detection and characterization in spatial time series data: A cluster-centric approach. *IEEE Transactions on Fuzzy Systems*, 22(6):1612–1624, 2014.

[23] Martin Steiger, Jürgen Bernard, Sebastian Mittelstädt, Hendrik Lücke-Tieke, Daniel Keim, Thorsten May, and Jörn Kohlhammer. Visual analysis of time-series similarities for anomaly detection in sensor networks. In *Computer Graphics Forum*, volume 33, pages 401–410. Wiley Online Library, 2014.

[24] Dimitrios Gunopulos and Gautam Das. Time series similarity measures and time series indexing. *ACM Sigmod Record*, 30(2):624, 2001.

[25] Sanghyun Park, Sang-Wook Kim, and Wesley W Chu. Segment-based approach for subsequence searches in sequence databases. In *Proceedings of the 2001 ACM symposium on Applied computing*, pages 248–252, 2001.

[26] Carmelo Cassisi, Placido Montalto, Marco Aliotta, Andrea Cannata, and Alfredo Pulvirenti. Similarity measures and dimensionality reduction techniques for time series data mining. *Advances in Data Mining Knowledge Discovery and Applications'(InTech, Rijeka, Croatia, 2012,*, pages 71–96, 2012.

[27] Hossein Hamooni and Abdullah Mueen. Dual-domain hierarchical classification of phonetic time series. In *2014 IEEE International Conference on Data Mining*, pages 160–169. IEEE, 2014.

[28] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with COTE: the Collective of Transformation-based Ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[29] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2014.

[30] Patrick Schäfer. *Scalable time series similarity search for data analytics*. PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2015.

[31] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal processing*, 26(1):43–49, 1978.

[32] Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal processing*, 23(1): 67–72, 1975.

[33] Eamonn Keogh, Li Wei, Xiaopeng Xi, Sang-Hee Lee, and Michail Vlachos. LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *Proceedings of the 32nd International Conference on Very Large Databases*, pages 882–893. Citeseer, 2006.

[34] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1 (2):1542–1552, 2008.

[35] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I Webb. TS-CHIEF: a scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, 34(3):742–775, 2020.

[36] Jason Lines, Sarah Taylor, and Anthony Bagnall. HIVE-COTE: The Hierarchical Vote Collective of Transformation-based Ensembles for time series classification. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1041–1046. IEEE, 2016.

[37] Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-based Ensembles. *ACM Transactions on Knowledge Discovery from Data*, 12(5), 2018.

[38] Anthony Bagnall, Jason Lines, William Vickers, and Eammon Keogh. The UEA and UCR Time Series Classification Repository, 2018. URL http://timeseriesclassification.com. Last accessed on 2021-08-01.

[39] Anthony Bagnall, Michael Flynn, James Large, Jason Lines, and Matthew Middlehurst. On the usage and performance of the Hierarchical Vote Collective of Transformation-based Ensembles version 1.0 (HIVE-COTE v1. 0). In *International Workshop on Advanced Analytics and Learning on Temporal Data*, pages 3–18. Springer, 2020.

[40] Ahmed Shifaz, Charlotte Pelletier, Francois Petitjean, and Geoffrey I Webb. Elastic similarity measures for multivariate time series classification. *arXiv preprint arXiv:2102.10231*, 2021.

[41] Patrick Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[42] Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery*, 31(1):1–31, 2017.

[43] 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.

[44] Charlotte Pelletier, Zehui Ji, Olivier Hagolle, Elizabeth Morse-McNabb, Kathryn Sheffield, Geoffrey I Webb, and François Petitjean. Using sentinel-2 image time series to map the state of victoria, australia. In *2019 10th International Workshop on the Analysis of Multitemporal Remote Sensing Images (MultiTemp)*, pages 1–4. IEEE, 2019.

[45] Jordi Inglada, Arthur Vincent, Marcela Arias, Benjamin Tardy, David Morin, and Isabel Rodes. Operational high resolution land cover map production at the country scale using satellite image time series. *Remote Sensing*, 9(1):95, 2017.

[46] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[47] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[48] Toni Giorgino. Computing and visualizing Dynamic Time Warping alignments in r: the dtw package. *Journal of Statistical Software*, 31(1):1–24, 2009.

[49] Chang Wei Tan, Matthieu Herrmann, Germain Forestier, Geoffrey I Webb, and Francois Petitjean. Efficient search of the best warping window for Dynamic Time Warping. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 225–233. SIAM, 2018.

[50] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of Dynamic Time Warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[51] Eamonn Keogh, Li Wei, Xiaopeng Xi, Michail Vlachos, Sang-Hee Lee, and Pavlos Protopapas. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures. *The VLDB journal*, 18(3):611–630, 2009.

[52] Daniel Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition*, 42(9):2169–2180, 2009.

[53] Chang Wei Tan, François Petitjean, and Geoffrey I Webb. Elastic bands across the path: A new framework and method to lower bound dtw. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 522–530. SIAM, 2019.

[54] Matthieu Herrmann and Geoffrey I Webb. Early abandoning PrunedDTW and its application to similarity search. *arXiv preprint arXiv:2010.05371*, 2020.

[55] Eamonn J Keogh and Michael J Pazzani. Derivative Dynamic Time Warping. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–11. SIAM, 2001.

[56] Tomasz Górecki and Maciej Łuczak. Using derivatives in time series classification. *Data Mining and Knowledge Discovery*, 26(2):310–331, 2013.

[57] Young-Seon Jeong, Myong K Jeong, and Olufemi A Omitaomu. Weighted Dynamic Time Warping for time series classification. *Pattern Recognition*, 44(9):2231–2240, 2011.

[58] Daniel S Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4):664–675, 1977.

[59] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. The move-split-merge metric for time series. *IEEE transactions on Knowledge and Data Engineering*, 25(6): 1425–1438, 2012.

[60] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 491–502, 2005.

[61] Pierre-François Marteau. Time Warp Edit Distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):306–318, 2008.

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

[63] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. A bag-of-features framework to classify time series. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2796–2802, 2013.

[64] Mustafa Gokce Baydogan and George Runger. Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30 (2):476–509, 2016.

[65] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[66] Nestor Cabello, Elham Naghizade, Jianzhong Qi, and Lars Kulik. Fast and accurate time series classification through supervised interval search. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 948–953. IEEE, 2020.

[67] Nestor Cabello, Elham Naghizade, Jianzhong Qi, and Lars Kulik. Fast, accurate and interpretable time series classification through randomization. *arXiv preprint arXiv:2105.14876*, 2021.

[68] Matthew Middlehurst, James Large, and Anthony Bagnall. The canonical interval forest (CIF) classifier for time series classification. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 188–195. IEEE, 2020.

[69] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[70] Carl H Lubba, Sarab S Sethi, Philip Knaute, Simon R Schultz, Ben D Fulcher, and Nick S Jones. catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery*, 33(6):1821–1852, 2019.

[71] Ben D Fulcher and Nick S Jones. hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell Systems*, 5(5):527–531, 2017.

[72] 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*, pages 947–956, 2009.

[73] Abdullah Mueen, Eamonn Keogh, and Neal Young. Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1154–1162, 2011.

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

[75] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–401, 2014.

[76] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[77] Isak Karlsson, Panagiotis Papapetrou, and Henrik Boström. Generalized random shapelet forests. *Data Mining and Knowledge Discovery*, 30(5):1053–1085, 2016.

[78] Aaron Bostrom and Anthony Bagnall. Binary shapelet transform for multiclass time series classification. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 257–269. Springer, 2015.

[79] James Large, Jason Lines, and Anthony Bagnall. The Heterogeneous Ensembles of Standard Classification Algorithms (HESCA): the whole is greater than the sum of its parts. *arXiv preprint arXiv:1710.09220*, 2017.

[80] Aaron Bostrom and Anthony Bagnall. A shapelet transform for multivariate time series classification. *arXiv preprint arXiv:1712.06428*, 2017.

[81] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation Forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, 2006.

[82] Pavel Senin and Sergey Malinchik. SAX-VSM: Interpretable time series classification using SAX and vector space model. In *2013 IEEE 13th International Conference on Data Mining*, pages 1175–1180. IEEE, 2013.

[83] James Large, Anthony Bagnall, Simon Malinowski, and Romain Tavenard. From BOP to BOSS and beyond: Time series classification with dictionary based classifiers. *arXiv preprint arXiv:1809.06751*, 2018.

[84] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 151–162, 2001.

[85] Patrick Schäfer and Mikael Högqvist. SFA: a Symbolic Fourier Approximation and index for similarity search in high dimensional datasets. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 516–527, 2012.

[86] Jessica Lin, Rohan Khade, and Yuan Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315, 2012.

[87] Patrick Schäfer. Scalable time series classification. *Data Mining and Knowledge Discovery*, 30(5):1273–1298, 2016.

[88] 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*, pages 637–646, 2017.

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

[90] James Large, Anthony Bagnall, Simon Malinowski, and Romain Tavenard. On time series classification with dictionary-based classifiers. *Intelligent Data Analysis*, 23(5):1073–1089, 2019.

[91] Matthew Middlehurst, James Large, Gavin Cawley, and Anthony Bagnall. The Temporal Dictionary Ensemble (TDE) classifier for time series classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 660–676. Springer, 2020.

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

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

[94] 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*, 34(6):1936–1962, 2020.

[95] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[97] Angus Dempster, François Petitjean, and Geoffrey I Webb. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[98] 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 SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 248–257, 2021.

[99] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I Webb. MultiRocket: Effective summary statistics for convolutional outputs in time series classification. *arXiv preprint arXiv:2102.00457*, 2021.

[100] Eammon Keogh. The UCR Time Series Data Mining Archive. *http://www.cs.ucr.edu/eamonn/TSDMA/index.html*, 2002.

[101] Mustafa Gokce Baydogan and George Runger. Learning a symbolic representation for multivariate time series classification. *Data Mining and Knowledge Discovery*, 29(2):400–422, 2015.

[102] Martin Wistuba, Josif Grabocka, and Lars Schmidt-Thieme. Ultra-fast shapelets for time series classification. *arXiv preprint arXiv:1503.05018*, 2015.

[103] Thach Le Nguyen, Severin Gsponer, Iulia Ilie, Martin O'Reilly, and Georgiana Ifrim. Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. *Data Mining and Knowledge Discovery*, 33(4):1183–1222, 2019.

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

[105] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, 2016.

[106] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, pages 65–70, 1979.

[107] Salvador Garcia and Francisco Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9(12), 2008.

[108] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[109] L Breiman, JH Friedman, R Olshen, and CJ Stone. *Classi cation and Regression Trees*. Wadsworth, 1984.

[110] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *International Conference on Database Theory*, pages 217–235. Springer, 1999.

[111] Alexander Hinneburg, Charu C Aggarwal, and Daniel A Keim. What is the nearest neighbor in high dimensional spaces? In *26th Internat. Conference on Very Large Databases*, pages 506–515, 2000.

[112] Kezhi Z Mao. Orthogonal forward selection and backward elimination algorithms for feature subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):629–634, 2004.