JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Metaheuristics for Pattern Mining in Big Sequence Data

A thesis submitted for the degree of

Doktor der Naturwissenschaften

at the Department of Physics, Mathematics and Computer Science at the Johannes Gutenberg University in Mainz

Atif Raza
born in Rawalpindi, Pakistan

Mainz, February 20, 2021

Examination date: April 13, 2021 D77

Abstract

An ever-growing list of human endeavors in a variety of domains results in the generation of time-series data, i.e., data that are time-resolved and measured in equidistant time intervals. The continued developments in sensor and storage technology and the availability of database systems specifically designed for time-series data have also made it possible to record an exorbitant amount of such data. The vast yet readily available data places ever-increasing demands on data mining methods for fast and efficient knowledge discovery, which establishes the need for exceedingly fast algorithms.

The data mining research community has been actively investigating various avenues to develop algorithms for time series classification. Most research has focused on optimizing accuracy or error rate, although runtime performance and broad applicability are as important in practice. The result is a plethora of algorithms that have quadratic or higher computational complexities. Consequently, the algorithms have little to no use for deployment on a large scale.

This thesis addresses the complexity issue by introducing several time-series classification methods based on metaheuristics and randomized approaches to improve the state-of-the-art in time-series mining. We introduce three subsequence-based time series classification algorithms and an approximate distance measure for time series data. One subsequences-based time series classifier explicitly employs random sampling for subsequence discovery. The other two subsequences-based classifiers employ discretized time series data coupled with (i) a linear time and space string mining algorithm for extracting frequent patterns and (ii) a novel pattern sampling approach for discovering frequent patterns. The frequent patterns are translated back to subsequences for model induction. Both of these algorithms are up to two orders of magnitude faster than previous state-of-the-art algorithms. An extensive set of experiments establishes the effectiveness and classification accuracy of these methods against established and recently proposed methods.

Acknowledgments

Acknowledgments have been removed from the online version.

Publications and Manuscripts

The research work carried out for this thesis culminated in the following articles and manuscripts. Items marked with an asterisk (∗) indicate that the specific articles/manuscripts constitute this thesis.

∗ 1. **Atif Raza**, Jörg Wicker, and Stefan Kramer. "*Trading off Accuracy for Efficiency by Randomized Greedy Warping*". In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 883–890. ACM, 2016. https://doi.org/10.1145/2851613.2851651

∗ 2. **Atif Raza**, and Stefan Kramer. "*Ensembles of Randomized Time Series Shapelets Provide Improved Accuracy While Reducing Computational Costs*". ArXiv:1702.06712 [CS], February 22, 2017. http://arxiv.org/abs/1702.06712

∗ 3. **Atif Raza**, and Stefan Kramer. "*Accelerating Pattern-based Time Series Classification: A Linear Time and Space String Mining Approach*". In: Knowledge and Information Systems 62(3), March 2020: 1113–1141. https://doi.org/10.1007/s10115-019-01378-7

4. Nora Zannoni, Martin Wikelski, Anna Gagliardo, **Atif Raza**, Stefan Kramer, Chiara Seghetti, Nijing Wang, Achim Edtbauer, and Jonathan Williams. "*Identifying Volatile Organic Compounds Used for Olfactory Navigation by Homing Pigeons*". In: Scientific Reports 10(1), September 28, 2020: 15879. https://doi.org/10.1038/s41598-020-72525-2

∗ 5. **Atif Raza**, and Stefan Kramer. "*Pattern Sampling for Shapelet-Based Time Series Classification*". ArXiv:2102.08498 [CS], February 16, 2021. http://arxiv.org/abs/2102.08498

6. Julian Vexler, **Atif Raza**, and Stefan Kramer. “*Integrating LSTMs with Online Density Estimation for the Probabilistic Forecast of Energy Consumption*”. Under Review for Machine Learning Journal. (2021).

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

1

Nature has bestowed upon the human species the gift of intellectual superiority over all other life forms on this planet. The human intellect is the most important aspect which enables our species to discern and adapt to its surroundings, innovate, and progress above and beyond any other species on earth. One of the many exceptional abilities of the human species is the ability to identify patterns in complex scenarios. This ability, coupled with an innate curiosity about the environment helped in the development of our species in more ways than imaginable. The pursuit of discovering repeating patterns in the processes encountered in our lives has been a human endeavor since time immemorial. The earliest surviving examples of the various ways our ancestors kept track of environmental observations include devices and constructions used to record the relative positions of celestial objects, river or lake water levels, etc. Most of these observations would usually be recorded over several weeks, months, or years, therefore, they can be attributed as the earliest known time series observations. The ability to correlate repeated patterns in environmental observations with various aspects of human life led to further advancements and enabled the human civilization to flourish. An excellent example is that of the ancient Egyptians who used an ingenious structure to record the yearly inundation levels of the Nile flood plain to forecast the quality of the harvest based on the amount of silt sedimentation that resulted from the annual flood. Figure 1.1 shows a *Nilometer* that dates back to 861 AD and was based on ancient Egyptian designs [1]. The nilometers were so effective that their derived forms were still in use in the late 20th century.

Fast forwarding to the current day and age, we see that human development over the past few centuries has led to an exorbitant evolution in all walks of life. The developments in computing and storage technologies have made it

Fig. 1.1. A Nilometer dating back to 861 AD. Photo by B. Werner (CC-BY-SA).

possible for us to observe and record an enormous number of quantities that would have been unimaginable even a few decades ago. Almost every field of human endeavor is producing chronological data, and we are generating and amassing it at an exponential rate from fields as diverse as activity logs, business and finance, economics, medical records, natural processes, research, etc. [2–4]. The sheer scale of the time series data being generated today can be estimated by the amount of data produced in the medical domain alone. Every day electrocardiograms (ECG), electroencephalograms (EEG), and many other medical diagnostic observations are recorded as time series data. In most cases, the observations are recorded at a constant and equidistant interval over time. An ECG examination of a single patient can generate time series data with several million data points. Repeated medical diagnoses of a person can also be treated as time series data since they present the levels of the investigated value over the course of time, although, in this case the time series data is not equidistant over time. In addition to explicit time series data, sequential data without a temporal aspect, like DNA sequences and gene expression data, can also be treated as time series and the close resemblance of the two data types allows to employ the same algorithms for certain applications and tasks.

The pace of data generation is also increasing with large scale availability of better instruments. Thankfully, we have also transitioned from paper based archives kept in libraries to online digital archives, which has enabled us to extract the required observations at the click of a button. Weather observations provide a prime example of the scale of this transition: just over half a century ago, it was only feasible to keep track of the minimum and maximum temperatures of a city per day, whereas it is now possible to get the temperature of almost any location on earth at per minute granularity.

Time series mining and sequential pattern mining are two closely related subfields of data mining. Both these fields aim for repeated pattern discovery in the given chronological data so that the discovered patterns can subsequently be employed for exploration of the data by a human expert or various machine learning tasks [2–4]. The importance of both these fields stems from the fact that they have large scale applicability in diverse fields [5]. The main difference between the two fields is that sequential pattern mining deals with discrete/symbolic data, while time series mining is concerned with real-valued data. Sequential pattern mining can be applied to any data occurring in a sequence with or without a concrete notion of time, e.g., DNA or protein sequences, gene expression data, items sold, web links clicked, and so forth [6]. Similarly, time series mining can be applied to any data with some form of logical ordering irrespective of the fact that the ordering is temporal or otherwise, e.g., motion capture data, chemical composition data obtained from spectral analyses, images converted to series using shape outlines, and so forth. Discrete-valued sequences can have exact matches, whereas finding a perfect match for real-valued sequences is highly improbable due to the continuous nature of the data at hand [3].

The omnipresence of time series data has led to the initiation of several research efforts for the development of machine learning and data mining techniques specifically targeting this data type. Time series data comprises chronological real-valued sequences, which are inherently high-dimensional, and large in size. The temporal order of these sequences characterizes the varying behavior of the recorded process and allows to find patterns which can identify similar or anomalous behavior over time, therefore, it is vital to keep this order intact to infer information about time-dependent features, e.g., pattern discovery, visualization, summarization, etc. [2–4].

The various applications of time series mining include: dimensionality reduction [7–11], indexing [7, 12–15], classification [7, 12, 16–18], clustering [17, 19–21], segmentation [17, 22], anomaly detection [22–24], summarization [25, 26], etc. Indexing is necessary for efficient retrieval of time series instances in a database and plays a significant role in reducing the search space for similarity search. Dimensionality reduction can be employed for efficient storage, indexing, or matching time series instances. Subsequence matching has its applications in finding subsequences in a long time series that are similar to a given reference sequence. Segmenting is necessary to divide the data based on some domain-specific criterion to limit the number of data points per time series instance so that these instances can be processed with some algorithm. Summarization is often used to visually present the time series data such that the overall characteristics are preserved. Finally, anomaly detection is used to detect the occurrence of extraordinary events.

1.1 Motivation and Problem Statement

The availability of large quantities of highly accessible time series data also highlights new challenges regarding the extraction of useful information from these resources. The continuous nature of time series data dictates that the time series mining algorithms aim to find similar rather than exact matches of a given reference instance. Searching for similar instances is also more practical because it allows to query a time series database for:

- grouping experiments with similarly behaving result parameters,
- identifying products with similar sale trends,
- discovering stocks with similar price cadence, etc.

Time series similarity is determined based on some suitable distance measure coupled with a similarity threshold. The distance measure provides a distance value for a pair of time series instances. Thereafter, similarity of the time series pair is determined based on a threshold that can either be user-specified, automatically determined, or relative to the distance values obtained for all the evaluated time series instances. If the distance between the pair of time series instances is below the threshold, then the two instances are treated as similar to each other. Indexing and clustering often make explicit use of a distance measure, while many approaches to classification, prediction, association detection, summarization, and anomaly detection make implicit use of a distance measure.

The process of finding similar time series instances involves a huge number of distance measure calculations. In the case of matching full length instances, similarity search involves exhaustively comparing all pairs of time series instances. On the other hand, subsequence search involves an exhaustive comparison of all possible subsequences of all lengths that can be generated for the time series instances in the database. Consequently, many time series mining algorithms are plagued with very high computational complexity, therefore, a number of research efforts have focused on countering this problem [27–39].

Metaheuristics is a major subfield of stochastic optimization. The algorithms which fall under this category are highly effective for problems with no obvious solutions and incomplete or imperfect information, or for problems with extremely large search spaces. These algorithms employ guiding heuristics coupled with some degree of randomness to steer the search process towards sufficiently good solutions to an optimization problem. The two main tactics of metaheuristics that make them so effective are *exploitation* and *exploration*. Exploitation carries out searches in the vicinity of existing solutions by making small variations to existing solutions. Exploration, on the other hand, randomly creates new solutions or modifies existing solutions such that the resulting solution is cast to a random location in the search space. Exploitive algorithms are faster and useful for problems with relatively stable search spaces, while explorative algorithms are relatively slower and useful in case the search space is riddled with local optima. Metaheuristics help in finding good enough solutions, therefore, the obtained solutions might be near-optimal given the constraints, but it is impossible to prove the optimality. The solutions provided by stochastic algorithms can vary based on the initial conditions, however, stable algorithms provide results with the same average accuracy levels. Metaheuristics also enable to search over a large set of solutions concurrently, which makes them even more appealing in multi-threaded environments. Examples of popular metaheuristic algorithms include hill climbing, simulated annealing (SA), tabu search (TS), variable neighborhood search (VNS), ant colony optimization (ACO), particle swarm optimization (PSO), genetic algorithms (GA), genetic programming (GP), and stochastic local search (SLS) [40–43].

A number of research efforts have employed metaheuristics to augment the process of data mining for sequential data [44–46]. Examples of metaheuristics based sequential pattern mining methods include: (i) genetic algorithms-based negative sequential pattern mining [44], (ii) genetic algorithm-based fuzzy pattern mining from gene expressions [45], (iii) genetic programming-based sequence mining from DNA microarray datasets [46], etc. However, metaheuristics have not been employed for time series mining extensively. Martínez-Ballesteros *et al.* proposed to extract quantitative association rules using evolutionary algorithms [47]. The successful application of metaheuristics for sequential data mining indicates that there is an opportunity to benefit from the pros of metaheuristics in the field of time series mining as well. Therefore, this research focused on the application of metaheuristics and randomization to the time series data mining problem.

1.2 Thesis Contributions and Structure

The principal contribution of this thesis is the development of time series classification algorithms based on metaheuristics and randomization. The proposed algorithms have been developed such that they require considerably less computational effort compared to the existing exact algorithms while providing on par classification accuracy. The efficiency and accuracy aspects are complementary to each other, since an algorithm that greatly reduces the required computational cost at the expense of the provided accuracy is of little use to the community. Therefore, designing these algorithms is a careful balancing act. The introduction of too much randomness can have detrimental effects on accuracy, while very little randomness might only shave off a fraction of the required computational effort.

Chapter 2 provides an overview of time series data mining in general, as well as providing the background and related work for time series classification in particular. The chapter presents the necessary details about the notation used in this text. Next, an introduction to the most common time series distance measures is provided. Subsequently, the chapter discusses previous research work to provide a context and groundwork for the following chapters, specifically the nearest neighbor based classification using whole matching, and the feature extraction based classification using subsequence matching.

The initial phase of this research focused on the development of a warping similarity measure for matching whole time series instances. Dynamic Time Warping (DTW) is the most widely used warping distance measure for time series data. It was introduced as a distance measure for time series data by Berndt and Clifford [16]. The computational complexity of the DTW distance measure is quadratic in the lengths of the two time series instances being compared [48, p. 72]. The quadratic computational cost of DTW coupled with increasing time series database sizes has warranted the design of algorithms which calculate an approximate version of the DTW distance measure to reduce its computational requirements. **Chapter 3** provides the details about our proposed algorithm, which is based on a look-ahead approach, augmented with a heuristic to steer the distance measure to be as close as possible to the exact warping path traversed by the DTW algorithm. This research work was published as:

> **Atif Raza**, Jörg Wicker, and Stefan Kramer. *"Trading off Accuracy for Efficiency by Randomized Greedy Warping"*. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 883–890. ACM, 2016.

Subsequently, we focused on feature-based time series classification, because it offers faster and more accurate classification of time series data. Subsequences which are representative of a class are referred to as *Shapelets*. Shapelet-based time series classification was proposed by Ye and Keogh [32]. Extracting shapelets is not trivially efficient, because the shapelet extraction process has to evaluate all possible subsequences of all lengths in a time series database to determine the best representative subsequences for each class of time series instances in the dataset. **Chapter 4** provides the details of our research work that reduces the computational complexity of the shapelet-based classification algorithm by randomly selecting shapelet candidates for evaluation instead of exhaustively evaluating all candidates. In addition, we created ensembles to provide diversity to the classification step of our algorithm. This chapter is based on the following manuscript:

Atif Raza, and Stefan Kramer. "*Ensembles of Randomized Time Series Shapelets Provide Improved Accuracy While Reducing Computational Costs*". ArXiv:1702.06712 [CS], February 22, 2017.

Our next proposed algorithm is discussed in **Chapter** 5 and is based on a linear time string mining algorithm for extraction of frequent patterns from discretized versions of the time series data [49–51]. The frequent patterns can be extracted in time linear to the cumulative length of the discretized time series data, which provides a drastic reduction in the overall computational cost of the algorithm, since the most time consuming process in a subsequence based time series mining algorithm is usually the subsequence extraction process. This chapter is based on the following journal article:

Atif Raza, and Stefan Kramer. "*Accelerating Pattern-based Time Series Classification: A Linear Time and Space String Mining Approach*". In: Knowledge and Information Systems 62(3), March 2020: 1113–1141.

Chapter 6 provides details about an algorithm based on random sampling to alleviate the exhaustive search of patterns in the discretized time series data. The premise is that the pattern explosion problem can be mitigated if a pattern sampler can be used to draw patterns instead of evaluating all possible patterns. This chapter is based on an article currently under review for an upcoming conference.

Atif Raza, and Stefan Kramer. "*Pattern Sampling for Shapelet-based Time Series Classification*". ArXiv:2102.08498 [CS], February 16, 2021.

Finally, **Chapter** 7 provides the concluding remarks.

In addition to the above mentioned research work that is included in this thesis, the following two articles were co-authored as well during this research work.

Nora Zannoni, Martin Wikelski, Anna Gagliardo, **Atif Raza**, Stefan Kramer, Chiara Seghetti, Nijing Wang, Achim Edtbauer, and Jonathan Williams. "*Identifying Volatile Organic Compounds Used for Olfactory Navigation by Homing Pigeons*". In: Scientific Reports 10(1), September 28, 2020: 15879.

Julian Vexler, **Atif Raza**, and Stefan Kramer. "*Integrating LSTMs with Online Density Estimation for the Probabilistic Forecast of Energy Consumption*". Under Review for Machine Learning Journal. (2021).

Related Work

2

2.1 Introduction

Time series data is so predominant that almost every human activity leads to some form of time series data being saved to the ever-growing databases. These huge amounts of time series data have a hidden wealth of useful information which, once extracted, can definitely lead to further advancements in the related fields which generate this data. Consequently, time series data mining research has seen an explosion of interest due to the substantial implications and applicability of the field. Time series data mining is generally concerned with, but not limited to, the following tasks:

- **Indexing**: Given a time series database, indexing allows to quickly and efficiently find the most similar time series instances in the database given a query time series [12–15].
- **Clustering**: Given an unlabeled set of time series instances, clustering allows to group similar instances into disjoint groups based on a given similarity measure [17, 19–21].
- **Classification**: Given a database with labeled time series instances and an unlabeled query time series instance, classification is the task of labeling the query instance as belonging to one or more classes in the database given a similarity measure [17, 18].
- **Forecasting**: Given a time series of length n, forecasting aims to predict the value for the $n + 1$th time point in the most basic case.
- **Summarization**: Given a time series instance, create an approximation (graphic or otherwise) which retains the general shape and characteristics of the original time series instance [25, 26].

- **Anomaly detection**: Given a database of "normal" time series instances and an unlabeled time series instance, find all sections of the unlabeled time series instance which have unexpected values given the values for the preceding time points [22–24].

- **Segmentation**: (a) Given a time series Q containing n data points, construct a model Q', from K piecewise segments where $K \ll n$, such that Q' closely approximates Q [17]. (b) Given a time series Q, partition it into K internally homogeneous sections (also known as change detection [22].

This chapter provides an overview of the related work in the time series classification domain. Section 2.2 introduces the notation used for representing real-valued time series data along with an introduction to the time series datasets used by the time series mining community for benchmarking their proposed research work. Section 2.3 introduces the most commonly employed time series distance measures. Section 2.4 provides details about previous research regarding time series classification based on whole matching and subsequence matching. Specifically, seminal or widely used research work is described in detail, while other research work is described briefly to provide a reasonable introduction of the subject. Any research work that is specifically relevant to our own research is introduced in the respective chapter. Finally, Section 2.5 provides a brief introduction to the recent trends in time series mining research.

2.2 Time Series Data

A *time series* is an ordered, real-valued sequence of observations which can be denoted as $T = (t_1, t_2, \ldots, t_n)$, where each t_i is an observation data point and n is the length of the time series instance. When a time series belongs to a specific class of instances, it is assigned the corresponding label $y \in C$, where C is the set of all possible class labels. A set of N labeled time series instances forms a time series dataset $D = \{(T_1, y_1), (T_2, y_2), \ldots, (T_N, y_N)\}$. An implicit time stamp corresponding to each data point $t_i \in T$ is assumed since time series observations are usually recorded at a fixed interval. Similarly, for time series derived from observations with a non-temporal ordering,

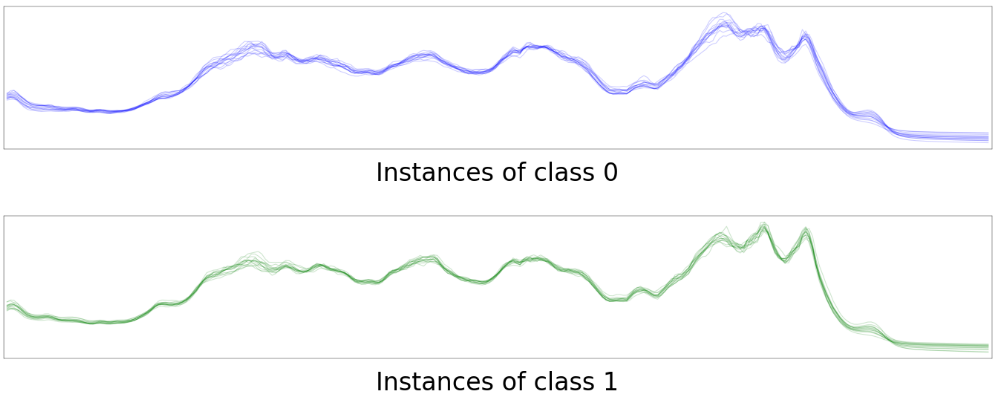


Fig. 2.1. Illustration of a real-valued binary class time series dataset obtained from the Diffuse Reflection Infrared Fourier Transform (DRIFT) spectrographs of freeze-dried samples of the coffee species Arabica (Class 0) and Robusta (Class 1). Each class has 14 time series instances and each instance is 286 time points long.

the data points are usually separated by discrete steps, therefore, explicitly identifying the ordering variable is usually not necessary. Figure 2.1 illustrates a popular time series dataset that is made up of the Diffuse Reflection Infrared Fourier Transform (DRIFT) spectrographs of freeze-dried samples of the coffee species *Arabica* and *Robusta*, respectively [52].

The field of time series mining took off in the 1990s, especially after the publication of the research work by Agrawal, Faloutsos, and Swami on similarity search for sequence databases [7]. Early research efforts in time series mining evaluated the proposed algorithms on either a private dataset or a few datasets from a specific application domain only [53]. The lack of generally available benchmark datasets also made it difficult for other researchers to systematically ascertain the efficacy of the proposed algorithms. In order to address the above stated concerns and inspired by the *UCI Machine Learning Repository* [54], the *UCR Time Series Classification Archive* was established as a central repository for time series datasets from various application domains, including instrumentation data (medical, environmental, industrial, etc.), spectral analyses, motion capture, astronomy, shape contours, traffic, power consumption profiles, and simulated events [56].[1] Many researchers and groups have contributed datasets to the *UCR Archive*. Some noteworthy contributors include the *UCR Archive* team itself, and the team behind the *UEA Time Series Repository*.[2] Over the years, the *UCR Archive* has seen a

[1]*UCR Archive* webpage: https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
[2]*UEA TSML Repository*: https://www.timeseriesclassification.com

number of expansions: by 2014 the repository had reached 47 datasets [58], the 2015 refresh increased the number of datasets to 85 [59], while the latest refresh in 2018 has increased the number of datasets to 128 [55]. The 2018 update also introduced variable length time series datasets as well as datasets that consist of time series with missing data.

The *UCR Archive* only consists of univariate time series datasets, although a number of datasets are actually made up of individual dimensions of inherently multivariate time series datasets, e.g., the *UWaveGestureLibrary* or *Cricket* datasets are actually 3-D motion capture datasets, but the *UCR Archive* has split the X, Y, and Z dimensions of these datasets to create individual datasets for each dimension. Recently, a multivariate time series dataset archive was made available by the *UEA TSML Repository*.

The *UEA TSML Repository* also provides a Java-based time series classification, clustering, and transformation framework, that is also Weka-compatible.[3] This framework provides implementations of many time series mining algorithms. In addition to the availability of cross-domain datasets, the provision of open source implementations of most time series mining algorithms has enabled the research community to effectively determine the efficacy of the existing and newly proposed time series mining algorithms.

2.3 Time Series Similarity Measures

Time series mining tasks rely on determining the time series similarity either directly or indirectly, therefore, a number of approaches have been proposed for finding time series similarity. These approaches can generally be divided into four main families: lock-step, elastic, feature-based, and model-based. This section aims to introduce the most commonly used similarity measures for time series data without an extensive discussion or comparison of the similarity measures. For a detailed analysis of the different similarity measures, the reader is encouraged to refer to the extensive reviews carried out about the field [3, 61–63].

[3]Weka is an open source machine learning framework that can be accessed through a graphical user interface, standard terminal applications, or a Java API. It contains a plethora of built-in tools for standard machine learning tasks [60].

2.3.1 Lock-step Measures

A lock-step distance measure calculates the distance between two equi-sized time series instances by aggregating the distance between the data points that coincide with each other. The most commonly used lock-step distance measure for time series data is the Euclidean distance (ED) that belongs to the L_p family of norms. Mathematically, the L_p norm of a vector $X = (x_1, \ldots, x_n)$ is defined as:

$$\|X\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}},$$

where n is the length of the vector and $p \geq 1$.[4] Two well-known cases of the L_p norm are the L_1 and the L_2 norms, which are also referred to as the Manhattan norm and the Euclidean norm, respectively. The L_∞ norm is a special case of the L_p norm and is referred to as the Maximum norm or the Chebyshev distance that is mathematically defined as:

$$\|X\|_\infty = \mathbf{max}\left(|x_1|, \ldots, |x_n|\right).$$

Figure 2.2 shows the unit circles for different L_p norms given an R^2 space and $p \geq 1$.

The L_p norm can also be used to calculate the distance between two equi-sized vectors by aggregating the distances between the coinciding points. Mathematically, the L_p norm between vectors S and T is defined as:

$$\|S - T\|_p = \left(\sum_{i=1}^{n} |s_i - t_i|^p\right)^{\frac{1}{p}},$$

where n is the length of the two vectors. In practice, the pth root can be safely ignored because it is a concave and monotonous function. This allows to incorporate early abandoning into the distance calculation procedure, i.e., when the running distance between two instances exceeds the current minimum distance, the computation can be aborted. This allows to move on to comparing the next instance without wasting computational power.

[4]For $0 < p < 1$, the L_p norm does not satisfy the properties of the "length function", therefore, it is not used for distance calculations.

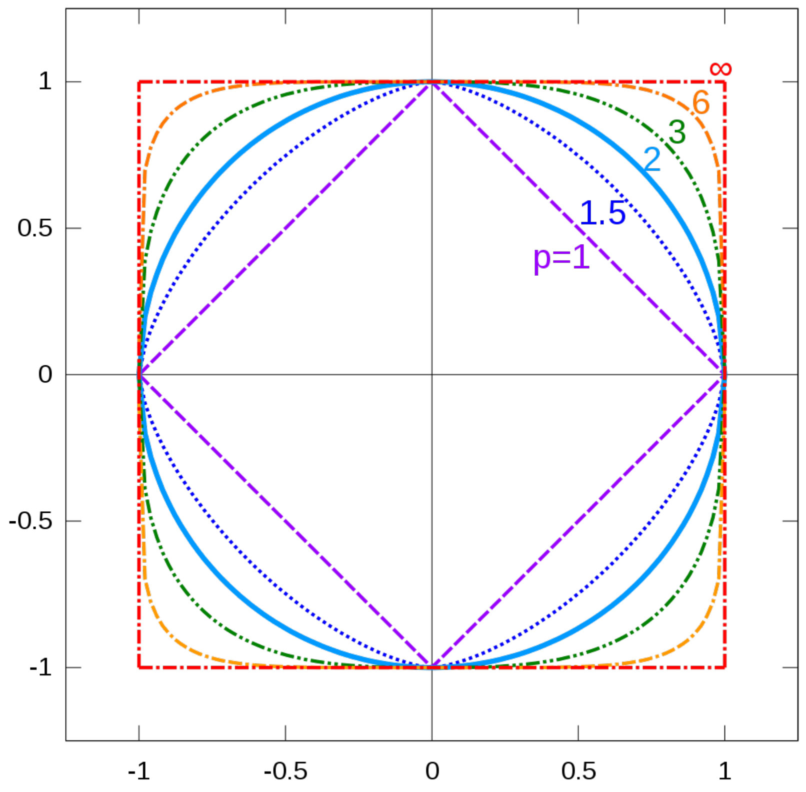


Fig. 2.2. Unit circles in different p-norms for a two-dimensional vector. Every vector from the origin to the unit circle has a length of one, the length being calculated with the length-formula of the corresponding p. Illustration by cmglee (CC-BY-SA).

The L_2 norm or the Euclidean distance (ED) between two time series instances is easy to understand, implement, and compute, which has ensured that it is the most widely used time series distance measure. The simplicity of ED, however, also leads to a couple of disadvantages. First, ED is a lock-step measure, therefore, it loses its efficacy for time series instances with many time shifted data points, or many segments which are stretched or compressed in the time domain. Second, ED is susceptible to amplitude scaling, i.e., if the amplitude of a time series fluctuates between -1 and 1, while the amplitude of the other time series fluctuates between 0 and 10, the distance obtained will be much larger even if the two time series have a very similar shape. In most cases, amplitude variation is irrelevant for similarity search, therefore this problem can be alleviated by normalizing the time series instances before calculating the distance [64]. There are two main normalization approaches: (i) normalize the time series instances to a fixed range [65], or (ii) normalize the time series instances to a zero mean and

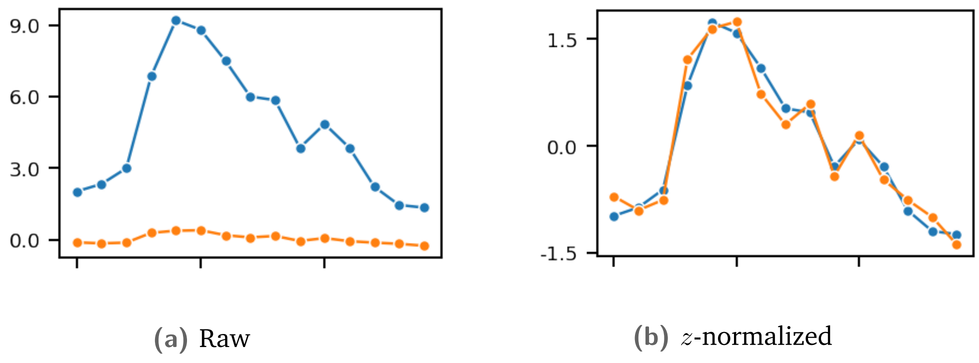


Fig. 2.3. Two time series instances in their (a) raw and (b) z-normalized forms. The overall shape of the two time series instances is remarkably similar, however, the distance obtained for both the versions is drastically different due to the large difference in the magnitudes of the raw time series instances.

unit variance (also referred to as z-normalization) [14]. For a time series T, the z-normalized time series T' is given as:

$$T' = \frac{T - \mu(T)}{\sigma(T)},$$

where $\mu(T)$ and $\sigma(T)$ give the mean and standard deviation of the data points in the time series, respectively, and the subtraction and division operations are performed element-wise. Figure 2.3 shows the raw and z-normalized versions of two time series instances, which illustrates the magnitude of the difference between the time series in the raw and z-normalized forms.

2.3.2 Feature-based Measures

The high dimensionality of time series data implies that similarity search based on raw time series instances can become daunting even if the distance measure being used is as simple as the Euclidean distance. This problem presents itself when the number of time series instances in the database becomes exceedingly large. One way to address this problem is to create and store such representations of the time series instances that allow efficient calculation or approximation of the distance between different instances. In the context of databases, indexing is the primary method for tackling this problem. It involves extracting salient features from the data, which can be used to divide the database into small subgroups of similar instances. If the time series instances can be indexed based on a low cost and low storage overhead method, then the querying process can be optimized using indexing. When a query is made, the same features are extracted for the query time series instance and the most similar subgroup is fetched from the database, which can then be sequentially scanned for exact matches.

The most widely used transformation method, which allows to extract features that can be used for calculating approximate distance values between time series instances, is the Discrete Fourier Transform (DFT). It maps a sequence from time domain to the frequency domain. A frequency domain mapping allows to represent the time series much more concisely, since only a few frequency components are enough to represent (and reconstruct) the time series such that the overall shape of the time series is kept intact while higher frequency components may be filtered out. Another benefit of the DFT is due to the Parseval's theorem, which states that the Fourier transform preserves the Euclidean distance in time and frequency domains. This property allows efficient calculation of a lower bounded distance between time series instances. A DFT based frequency domain transformation of the time series instances S and T of length n results in a sequence of n complex-value pairs $\acute{S}$ and $\acute{T}$. Mathematically, the Euclidean distance between the frequency domain transformations and the time domain instances is defined as:

$$\mathbf{ED}\left(\acute{S},\acute{T}\right) = \mathbf{ED}\left(S,T\right)$$
$$\sqrt{\sum_{f=0}^{n-1}\left|\acute{S}_f - \acute{T}_f\right|^2} = \sqrt{\sum_{i=1}^{n}\left|S_i - T_i\right|^2}.$$

Limiting the number of complex-value pairs for the frequency components gives the following:

$$\sqrt{\sum_{f=0}^{\theta}\left|\acute{S}_f - \acute{T}_f\right|^2} \leq \sqrt{\sum_{i=1}^{n}\left|S_i - T_i\right|^2},$$

where θ is the number of first few frequency components used to approximate the distance between the instances such that $\theta \ll n$. Fast Fourier Transform (FFT) is the most well-known algorithm for DFT and requires $O(n\ log\ n)$ time for transforming a time domain sequence into a frequency domain sequence. The time series instances in a very large database are saved along with their DFT transformations. The storage overhead associated with this strategy is negligible as compared to the size of the real-valued time series instances. When a query time series is presented to the database, the DFT transformation of the query instance is calculated, and the ED based similarity search can be carried out with significantly less computational cost, since the number of points required to determine the ED in the frequency domain is much less than the number of points in the time domain. A number of time series mining methods have been based on DFT transformation of time series data [7, 12]. In addition to employing DFT, the research community has also used other feature-based measures for time series similarity and indexing, including Discrete Wavelet Transform (DWT) [13, 15, 23, 66–68] and Chebyshev polynomials [69].

2.3.3 Elastic Measures

The Euclidean distance is calculated using a one-to-one lock-step mapping of the data points, which makes ED an inappropriate distance measure for problem domains that involve similarly shaped, but misaligned, stretched, and/or compressed time series instances. Since ED cannot "align" the sequences, this results in a superficial distance contribution from the deformities and misalignments. In such cases, an elastic distance measure is a better option for determining time series similarity. Elastic distance measures allow to warp the time axis such that there is a proper alignment of the overall shape of the data points between the time series instances even if it requires considering some data points more than once for calculating the distance. Some noteworthy elastic distance measures include Dynamic Time Warping [16], Time Warp Edit Distance [70], Longest Common Subsequence Similarity [71], Minimum jump cost dissimilarity [72], Edit Distance on real sequences [62], Lucky Time Warping [73], and Gliding Elastic Match [74].

Dynamic Time Warping (DTW) is the most widely used elastic distance measure for time series data. DTW was already being used by the speech recognition research community when it was introduced to the time series data mining community by Berndt and Clifford [16]. For time series instances $C = (c_1, \ldots, c_m)$ and $Q = (q_1, \ldots, q_n)$ of length m and n, respectively, the

DTW distance measure is calculated as follows. First, a "local" cost matrix $d^{m \times n}$ is calculated, where each cell (i, j) corresponds to the squared distance between the data points q_i and c_j, i.e., $d_{i,j} = (c_i - q_j)^2$. Next, the local cost matrix is used to compute an accumulated (global) cost matrix $D^{m \times n}$ using dynamic programming. The first element of the local cost matrix makes up the first element of the global cost matrix, i.e., $D_{1,1} = d_{1,1}$. The remaining elements of the global cost matrix $D^{m \times n}$ are calculated as follows: $D_{i,j} = d_{i,j} + \mathbf{min}\{(D_{i-1,j}), (D_{i-1,j-1}), (D_{i,j-1})\}, \forall\, i \in [1\,..\,m], j \in [1\,..\,n]$. Elements outside the allowed range of the matrix indices i and j are assumed to have a value of ∞. Once the entire global cost matrix $D^{m,n}$ is calculated, the element $D_{m,n}$ is returned as the distance between the two time series instances.

The DTW warping path is calculated using the global cost matrix $D^{m,n}$ and is governed by three constraints. The **boundary** condition requires the warping path to start at cell $(1, 1)$ and end at cell (m, n) of the global cost matrix. The **monotonicity** condition requires all elements of a warping path to have the property $i_1 \leq i_2 \leq \ldots \leq i_{m-1} \leq i_m$ and $j_1 \leq j_2 \leq \ldots \leq j_{n-1} \leq j_n$. Finally, the **continuity** condition restricts the number of cells that can be traversed in one move, i.e., $W_l - W_{l-1} \in \{(1,0), (1,1), (0,1)\}$, where W_l and W_{l-1} are consecutive elements in the warping path and each element is a vertex pair corresponding to a cell of the global cost matrix. The warping path is determined in reverse order, i.e., from cell (m, n) to cell $(1, 1)$, therefore the global cost matrix has to be calculated beforehand. Starting from the last cell in the global cost matrix $D_{m,n}$, the warping path W is determined as follows.

$$W_k = (m, n)$$

$$W_{k-1} = \begin{cases} (1, n-1), & \text{if } m = 1 \\ (m-1, 1), & \text{if } n = 1 \\ \mathbf{argmin}\{D_{m,n-1}, D_{m-1,n-1}, D_{m-1,n}\}, & \text{otherwise} \end{cases}$$

A naïve implementation of DTW has two main drawbacks, (i) the straightforward DTW implementation requires memory for two matrices of size $m \times n$, and (ii) the computational complexity of DTW is $O(mn)$. The memory requirements can be halved by implementing the DTW distance calculation in such a way that the local and global distance values for each cell are calculated in a single step instead of the two-step approach listed above. The first element of the global cost matrix is calculated using q_1 and c_1 data points, i.e., $D_{1,1} = (c_1 - q_1)^2$. Next, each $D_{i,j}$ element is calculated as: $D_{i,j} = (c_i - q_j)^2 + \mathbf{min}\{(D_{i-1,j}), (D_{i-1,j-1}), (D_{i,j-1})\}, \forall\, i \in [1\,..\,m], j \in [1\,..\,n]$, alleviating the first problem. The second problem has been researched extensively, and a number of approaches have been proposed. The computation of the global cost matrix can be restricted to a warping window that covers a reasonable space on both sides of the main diagonal. The two main warping window approaches are the Sakoe-Chiba Band [27] and the Itakura Parallelogram [28]. The DTW distance can also be lower bound to enable early abandoning [29–31, 75]. In addition to the vanilla DTW algorithm, a number of variations of the DTW distance measure have also been proposed for various application areas, e.g., Derivative DTW [76], Weighted DTW [77], Sparse DTW [78], etc. Finally, and to a lesser degree of effectiveness, FastDTW calculates the DTW distance with variable resolution [79].

Lucky Time Warping (LTW) is a greedy algorithm for finding warping time series distances [73]. It is governed by the same three constraints as DTW, i.e., boundary, continuity, and monotonicity conditions. DTW is a dynamic programming based approach which relies on the calculation of a global cost matrix to determine the distance between the given time series instances. On the other hand, LTW relies on a look-ahead approach based on local squared distances for finding a good enough warping path between the time series instances and, in doing so, also estimates the distance between the time series instances.

Algorithm 2.1 provides the details about distance calculation using LTW. First, the algorithm initializes the starting (i and j) and ending (m and n) indices, and calculates the distance d for the data points corresponding to the starting indices (Line 2.1-1). Next, the algorithm enters the loop which determines the LTW distance (Lines 2.1-2 to 2.1-16). The current cell is indicated by the (i, j) index pair, while its immediate neighbors have the index pairs $(i, j + 1)$, $(i + 1, j + 1)$, and $(i + 1, j)$, and are referred to as the $right$, $diag$, and $down$ cells, respectively. At each iteration, the three distance values d_{right}, d_{diag} and d_{down} are initialized with large values to mimic an infinite distance if the respective cells are outside the valid range (Line 2.1-3). Next, the algorithm calculates the d_{right}, d_{diag} and d_{down} distance values for the respective valid

Algorithm 2.1 LTW (C, Q, w)

```
1:  $i, \leftarrow 1$, $j \leftarrow 1$, $m \leftarrow |C|$, $n \leftarrow |Q|$, $d \leftarrow (c_i - q_j)^2$
2:  while $(i \leq m) \wedge (j \leq n)$ do
3:      $d_{right} \leftarrow \infty,\ d_{diag} \leftarrow \infty,\ d_{down} \leftarrow \infty$
4:      if $(j+1) \leq \mathbf{min}(w+i, n)$ then
5:          $d_{right} \leftarrow\ (c_i - q_{j+1})^2$
6:      end if
7:      if $(i+1 \leq m) \wedge (j+1 \leq n)$ then
8:          $d_{diag} \leftarrow\ (c_{i+1} - q_{j+1})^2$
9:      end if
10:     if $(i+1) \leq \mathbf{min}(w+j, m)$ then
11:         $d_{down} \leftarrow\ (c_{i+1} - q_j)^2$
12:     end if
13:     $d_{min} = \mathbf{min}(d_{right}, d_{diag}, d_{down})$
14:     $i, j \leftarrow \mathbf{index}(d_{min})$
15:     $d \leftarrow d + d_{min}$
16: end while
17: return $d$
```

neighboring cells. Now, the accumulated distance value d is incremented with the minimum of d_{right}, d_{diag} and d_{down} distance values, while the indices i and j are updated to point to the cell which has the minimum distance from the current cell (Lines 2.1-13 to 2.1-15). The loop is terminated when the (i, j) index pair equals (m, n) and the accumulated distance d is reported as the distance between the two time series instances.

Gliding Elastic Match (GEM) is another elastic distance measure for time series data [74]. GEM is designed to be an application specific distance measure, but it can also perform equally well in the general time warping case. In addition to being translation invariant and scale invariant in the time domain for a given lower and upper bound of a given scaling factor, GEM is specifically designed to handle amplitude variations along the measurement axis. Amplitude variations can often lead the DTW algorithm astray, which results in an incorrect warping path, and consequently the wrong distance between the time series instances. z-normalization is also of little use in the presence of amplitude variations due to the relative differences between the peaks and/or troughs in the measurement axis. Many real-world time series exhibit subtle artifacts, e.g., a sudden change in the sampling rate, local drift, or sensor failure, that can lead to the introduction of aspects that can not be handled effectively by DTW. Such artifacts are rarely observed in the heavily preprocessed datasets of the *UCR Archive*. The empirical evaluations carried out by Hundt *et al.* show that, GEM is a significantly better warping distance measure for time series that exhibit amplitude variations. The overall classification accuracy results also show that the GEM distance measure provides on par classification accuracy compared to DTW. GEM is also a highly parallelizable algorithm unlike DTW. In this regard, the authors showed that parallelizing GEM provides a significant speedup against the UCR Suite as the length of time series instances increases.

2.4 Time Series Classification

Time series classification can be divided into two categories depending on the methodology. The classification can be carried out after matching the entire length of the time series instances or based on the presence of small subsequences in the time series instances.

2.4.1 Classification based on Whole Matching

Time series classification based on whole-matching is suitable for problems where each class of instances has a basic underlying shape spanning the entire length of the time series. The shape can be time shifted, stretched and/or compressed, or distorted due to noise. For these problems, the best known approach is to use the Nearest Neighbor (1-NN) algorithm coupled with a suitable distance measure. The ED (Section 2.3.1) and DTW (Section 2.3.3) distance measures are the two most widely used time series distance measures. For datasets having a small number of instances, the 1-NN algorithm coupled with an elastic distance measure, e.g., Dynamic Time

Warping (DTW), performs best, however, for datasets with a large number of instances, the accuracy of 1-NN with an elastic distance measure converges to that of 1-NN with Euclidean distance (ED) [80].

In order to use the 1-NN algorithm, the query time series instance is compared against all the time series instances in the database to determine the instance that has the shortest distance from the query instance. Once this is done, the query instance is classified as belonging to the same class as the instance that has been identified in the previous step. The computational cost of a DTW based 1-NN approach is $O(Nn^2)$, where N is the total number of time series instances in the database and n is the length of the time series instances.

2.4.2 Classification based on Subsequence Matching

Many time series classification problems are concerned with data for which small subsequences are indicative of the class of a time series instance, while the overall shape provides little to no information about the classification. These particular subsequences are (i) much smaller than the overall length of the time series, (ii) phase independent, and (iii) can occur at any point in the time series. For such problems, determining the presence or absence of the particular subsequences in the time series instances is a better suited classification approach. Ye and Keogh proposed an algorithm to handle problems of this nature and referred to the discriminative subsequences as *Shapelets* [32].

Effectively, shapelets are subsequences occurring frequently in a specific class of time series instances while being absent or infrequent in the instances of other classes. A *subsequence* S is a contiguous chunk of data points of a time series T, such that $S = (t_p, t_{p+1}, \ldots, t_{p+l-1})$, where n is the length of the time series, l is the length of the subsequence, and p is the starting point of the subsequence in T such that $1 \leq p \leq n-l+1$. A more specific notation for the subsequence S can be S_p^l, where the starting point and length are explicitly mentioned. The set of all subsequences of length l for a time series of length n can be denoted as $S^l = \{S_1^l, S_2^l, \ldots, S_{n-l+1}^l\}$, where the number of possible subsequences of length l in a time series of length n is $|S^l| = n - l + 1$. For a dataset with N time series instances, each having length n, the total number of subsequences of length $l \in [min \,..\, max]$ is equal to:

$$N \times \sum_{l=min}^{max} (n - l + 1).$$

The presence of a shapelet in a time series is determined based on whether the distance between the shapelet and the time series is less than a threshold value. The distance between a shapelet S and a time series T is defined as the minimum observed distance between the shapelet and all shapelet-length subsequences of the time series instance, i.e.,

$$dist(S, T) = \mathbf{min}(sdist(S, R)), \quad \forall R \in S^l,$$

where $sdist(S, R)$ denotes the distance between two subsequences of equal length. Mathematically, the distance between two subsequences S and R is defined as:

$$sdist(S, R) = \sum_{i=1}^{l} (s_i - r_i)^2,$$

which is simply the squared Euclidean distance. The square root can be safely ignored because it is a concave and monotonous function, which also allows early abandoning the distance calculations.

The original shapelets based algorithm [32], referred to as *YK-shapelets* in this text, creates a classification model by embedding shapelets and their corresponding distance thresholds in the internal nodes of a decision tree. Each node of the decision tree splits the incoming time series dataset split into "purer" subsets to be passed on to the subsequent nodes until the purity criterion is satisfied and the node can be marked as a leaf node. The YK-shapelets algorithm determines the purity of the dataset splits using the Information Gain (IG), although other approaches can also be used [33]. For each internal node, the shapelet and its corresponding distance threshold are selected such that splitting the dataset split on the basis of the selected shapelet maximizes the IG. For a given shapelet and distance threshold, the incoming dataset split D can be split into two subsets D_L and D_G, such that all instances whose distance compared to the shapelet is less than the distance threshold are grouped together as D_L, while the instances whose distance compared to the shapelet is greater than the distance threshold form another group D_G. The IG for the dataset D and the respective splits D_L and D_G is given as:

$$IG = H(D) - \left(\frac{|D_L|}{|D|} \times H(D_L) + \frac{|D_G|}{|D|} \times H(D_G) \right),$$

where H denotes the entropy of a given dataset split, and is defined as:

$$H(split) = -\sum_{c \in C} (p_c \times log(p_c)),$$

where C is the set of all class labels in the dataset $split$ and p_c is the fraction of instances belonging to class c.

Algorithm 2.2 lists the steps involved in creating an internal node of the shapelet based classification model. The recursive calls to the procedure result in the creation of a decision tree. If the incoming dataset split satisfies the purity criterion (Line 2.2-1), the procedure returns a leaf node with the majority class of the instances reaching that node (Line 2.2-2). Otherwise, the procedure searches for a shapelet S and a corresponding split distance δ, which maximize the IG when used to split the dataset (Line 2.2-4). Next, D is split into D_L and D_G on the basis of the split distance δ and the order line d_{map} (Line 2.2-5). Next, calls are initiated for the creation of the subsequent decision tree nodes (Lines 2.2-6 and 2.2-7). Finally, an internal node is returned that consists of the shapelet S, distance threshold δ, and the two subsequent nodes $node_L$ and $node_G$ (Line 2.2-8). The shapelet S and split distance δ constitute the decision criterion of the node while the nodes $node_L$ and $node_G$ form the child nodes of the created node.

The process of classifying a given time series instance starts at the root node and involves (i) calculating the distance between the time series and the shapelet specific to the current node, (ii) moving to a subsequent node depending on whether the distance value is less than or greater than the threshold value, and (iii) repeating the previous steps until a leaf node is reached, at which point the class label specific to the leaf node is reported as the classification label for the given time series instance. The induction of a shapelet based classification model may lead to the identification of a

Algorithm 2.2 CreateNode (D, l_{min}, l_{max})

```
1: if IsPure($D$) then
2:     return LeafNode($D$)
3: else
4:     $(\mathcal{S}, \delta, d_{map}) \leftarrow$ FindShapelet($D, l_{min}, l_{max}$)
5:     $(D_L, D_G) \leftarrow$ SplitData($D, \delta, d_{map}$)
6:     $node_L \leftarrow$ CreateNode($D_L, l_{min}, l_{max}$)
7:     $node_G \leftarrow$ CreateNode($D_G, l_{min}, l_{max}$)
8:     return InternalNode($\mathcal{S}, \delta, node_L, node_G$)
9: end if
```

number of shapelets, whose presence ultimately determines a specific class of instances. This also indicates that a class of instances can have multiple representative shapelets and the order of the discovered shapelets depends on the initial conditions used to initiate the shapelet discovery process.

Shapelet Discovery

Algorithm 2.3 lists the brute force shapelet discovery procedure that takes as input a time series dataset split D and the minimum and maximum allowed shapelet lengths l_{min} and l_{max}, respectively. The first loop iterates over the range of shapelet candidate lengths (Lines 2.3-2 to 2.3-20), the second loop iterates over the time series instances in the dataset (Lines 2.3-3 to 2.3-19), the third loop iterates over all possible subsequence start positions in a time series instance (Lines 2.3-4 to 2.3-18), while the fourth loop iterates over the time series instances again in order to determine the distance between the shapelet candidate and the time series instances (Lines 2.3-7 to 2.3-10). A shapelet candidate S of length l is extracted from the ith time series instance starting at time point p (Line 2.3-5). Next, the procedure calculates the distance between the shapelet candidate S and each time series instance in the dataset in order to populate the order line (Lines 2.3-7 to 2.3-10). The order line is a priority queue with time series instances inserted on the basis of their distance from the shapelet candidate. It is used to determine the best distance threshold to be used for the current shapelet and the corresponding IG value (Line 2.3-11). The process of determining the best split distance involves an exhaustive evaluation of the possible splitting points of the dataset based on the obtained order line. For a given order line with r entries, where $r \leq |D|$ and each entry contains either one or more time series instances, $r - 1$ distance values are evaluated to determine the best distance threshold for the current shapelet candidate, i.e., for each pair of consecutive distance values in the order line, the procedure calculates a mean distance value and determines the IG using this value as the split distance for splitting the dataset. The procedure keeps track of the best IG and the corresponding distance threshold values, which are returned once all the distance value pairs have been evaluated. If the new IG is greater than the "best so far" IG, the values for "best so far" IG IG_{best}, split distance δ_{best}, shapelet S_{best} and order line d_{best} are updated (Lines 2.3-12 to 2.3-17). Once all candidates have been evaluated, the best-found shapelet S_{best}, the corresponding split distance δ_{best}, and the best order line d_{best} are returned.

Computational Challenges and Speed-up Strategies

Despite its many advantages, the huge computational cost of shapelets based classification is a major drawback of shapelet based time series classification. For a dataset with N instances of length n, the number of all possible shapelet candidates of all lengths is $\frac{1}{2}Nn(n+1)$, which is on the order of $O(Nn^2)$. Evaluating each candidate requires its comparison with all the candidates in the dataset, that is, on average, on the order of $O(Nn)$. Each comparison (using squared Euclidean Distance) on average requires $O(n)$ operations. Aggregating the above yields the computational complexity of a single call to the brute force shapelet discovery procedure, which is on the order of $O(N^2n^4)$. The nested loops in Algorithm 2.3 also confirm the estimated computational cost. Since the shapelet discovery procedure is called at each node of the decision tree, the overall computational cost is a multiple of the estimated computational cost, where the multiple depends on the depth of the decision tree. Therefore, the computational complexity of training a shapelets based classifier can become untenable even for relatively small datasets. Consequently, a number of techniques have been proposed to reduce the computational complexity of the shapelet discovery process. The authors of the YK-shapelets algorithm proposed early abandoning distance

Algorithm 2.3 FindShapelet (D, l_{min}, l_{max})

```
1: $IG_{best} \leftarrow 0$
2: for $l \in [l_{min} \mathinner{..} l_{max}]$ do
3:     for $i \in [1 \mathinner{..} |D|]$ do
4:         for $p \in [1 \mathinner{..} (|D[i]| - l + 1)]$ do
5:             $S \leftarrow D[i]_p^l$
6:             $order\_line \leftarrow \varnothing$
7:             for $j \in [1 \mathinner{..} |D|]$ do
8:                 $dist_j \leftarrow dist(S, D[j])$
9:                 place $D[j]$ on the $order\_line$ at position $dist_j$
10:            end for
11:            $IG, \delta \leftarrow$CheckCandidate($order\_line$)
12:            if $IG > IG_{best}$ then
13:                $IG_{best} \leftarrow IG$
14:                $\delta_{best} \leftarrow \delta$
15:                $S_{best} \leftarrow S$
16:                $d_{best} \leftarrow order\_line$
17:            end if
18:        end for
19:    end for
20: end for
21: return $S_{best}$, $\delta_{best}$, $d_{best}$
```

calculations and early candidate pruning using an upper-bound on the IG in their seminal paper on shapelets and reported a speed-up of three orders of magnitude compared to the brute force approach [32].

The Euclidean Distance calculations can be abandoned as soon as the distance between the shapelet candidate and the current subsequence exceeds the running "best so far" distance value. A "best so far" value is maintained while calculating the distance for a shapelet candidate and each window in the time series. Whenever the distance between the shapelet and the current window exceeds this "best so far" value, the computation is abandoned for

Algorithm 2.4 dist (S, T)

```
1: $d_{bsf} \leftarrow \infty$
2: for $i \in [1 \,..\, (|T| - |S| + 1)]$ do
3:     $isPruned \leftarrow False$
4:     $d \leftarrow 0$
5:     for $j \in [1 \,..\, |S|]$ do
6:         $d \leftarrow d + (S_j - T_{i+j})^2$
7:         if $d > d_{bsf}$ then
8:             $isPruned \leftarrow True$
9:             $BREAK$
10:        end if
11:    end for
12:    if $isPruned = False$ then
13:        $d_{bsf} \leftarrow d$
14:    end if
15: end for
16: return $d_{bsf}$
```

the window. Algorithm 2.4 provides the details of distance calculation with early abandoning.

The shapelet candidates can also be pruned based on an inexpensive IG computation. The idea is to compute an optimistic IG value after placing each time series instance on the order line to estimate whether such a placement of the remaining instances will provide a better IG than the "best so far" value. For a C class problem, we need to calculate at most $2 \times C$ optimistic IG values. For each class c in the dataset, there are two possible scenarios to be checked. One, we place the remaining class c instances in the dataset on the order line at distance position 0 while placing all remaining instances on the current maximum distance position on the order line and calculate the IG to check if it is greater than the "best so far" value, and two, we place the c class instances on the current maximum distance position on the order line and all other instances on distance position 0 to check if we get a better IG value than the "best so far". If either of the above tests gives a greater IG than the “best so far” IG value, we terminate the optimistic IG calculation and evaluate the next time series instance in the dataset. If placing the instances of each class on the extremes of the current order line does not provide an improvement for the current “best so far” IG value, then it will not be possible to achieve a better value with the current shapelet candidate, because we have already tried all of the “most optimistic” cases of time series instance placement resulting in no improvement. Therefore, the current shapelet candidate can be safely discarded. Computing an IG is much less computation intensive than calculating the distance between the time series instances and the current shapelet, therefore, this procedure provides an effective speedup.

A number of subsequent research efforts have focused on reducing the computational complexity of the shapelet discovery algorithm. The *Logical-Shapelets* algorithm reduces computational costs by reusing previously calculated distances and pruning candidates using the triangular inequality [81]. It can reduce the computational complexity to $O(N^2n^3)$, however, caching the computations imposes a memory requirement on the order of $O(Nn^2)$ which limits the use of this algorithm for large datasets in memory constrained environments. The *Fast-Shapelets* algorithm reduces the dimensionality of the data using Symbolic Aggregate Approximation (SAX) [10] and then performs a random projection based shapelet discovery using this lower dimensional data [34]. It uses a heuristic approach and provides a huge reduction in computational costs, but requires tuning a number of parameters according to the dataset characteristics or performance requirements. The *Random-Shapelets* algorithm performs a uniform random sampling of candidates to reduce the number of evaluated candidates, however, the asymptotic complexity of the algorithm is still $O(N^2n^4)$ [82].

2.5 Recent Trends in Time Series Mining

Time Series Classification using Discretized Data

Over the last decade, the time series community has focused on employing text mining algorithms to tackle the time series classification problem by discretizing the time series instances and treating the symbolic data as strings. The **Bag of Patterns (BoP)** approach builds a histogram of SAX words for each time series [35]. A new sample is classified by comparing the histograms to find the nearest neighbor in the training set. **Symbolic Aggregate Approximation - Vector Space Model (SAX-VSM)** is another SAX-based classifier which also uses the BoP framework [36]. It first builds a dictionary of distinct SAX words from training data (a vector space representation) and for efficiency reasons, instead of building histograms, it computes a single vector of tf-idf weights for each class. In addition, it employs an optimization algorithm to search for the optimal parameters for discretizing the time series data using SAX. However, the tuning cost is substantial due to the need for cross-validation. The SAX-VSM authors also analysed the interpretability of SAX-VSM models by mapping SAX words having high tf-idf scores back to the original time series. A DFT based symbolic representation was introduced by Schäfer and Högqvist for indexing time series, that is aptly called Symbolic Fourier Approximation (SFA) [11]. Several time series classification algorithms have been proposed on the basis of SFA, including **Bag of SFA Sequences (BOSS)** [37], **Bag of SFA Sequences in Vector Space (BOSS VS)** [38], and **Word Extraction for Time Series Classification (WEASEL)** [39]. BoP, SAX-VSM, and BOSS are ensemble methods based on text mining algorithms. BOSS uses ensembles of histograms of SFA-words and 1-NN classifiers, while BOSS VS uses tf-idf class centroids and 1-NN for classification. In the SFA-based family of TSC algorithms, WEASEL is the most recent work on univariate time series. WEASEL is more accurate than the BOSS and BOSS VS algorithms, but suffers from memory efficiency issues since it does not include effective methods for pruning features early, and hence also needs to carefully restrict the feature space (e.g., by restricting the SFA parameters and the type of features). The authors of WEASEL do not discuss the interpretability of these methods, arguably because of the nonlinear characteristics of the SFA transformation. Nguyen, Gsponer, and Ifrim also proposed a SAX-based time series classification algorithm [83] that combines a fixed symbolic representation (SAX) and two adaptations of a sequence classifier (SEQL by Ifrim and Wiuf (2011)). An improved variation of the above algorithm was proposed by Nguyen *et al.*, which is called **Mr-SEQL** [84]. This algorithm is also an ensemble method and creates multi-resolution SAX-based features to be used for classification with a sequence classifier.

Time Series Classification using Deep Learning

Deep learning based strategies have also started gaining traction for time series classification. The use of text mining approaches or shapelets-based classification algorithms was aimed at interpretable knowledge discovery from time series data, however, the dominance of deep learning in other application domains has also lured in the time series mining community. In 2019, Ismail Fawaz *et al.* published a review of various deep learning architectures for time series mining [85]. The following year, two independently developed deep learning-based time series classification approaches were proposed by Dempster, Petitjean, and Webb (**ROCKET** [86]) and Ismail Fawaz *et al.* (**InceptionTime** [87]). Both these approaches currently mark the state-of-the-art in deep learning-based time series classification. Overall, the deep learning-based classification schemes have caught up with the previous approaches, and it is safe to say that a lot of future research regarding time series mining will employ the ever evolving deep learning architectures.

Time Series Mining using Matrix Profile

Apart from time series classification, the *all-pairs-similarity-search* problem for time series has also been extensively researched in the last few years. The basic problem statement for the all-pairs-similarity-search problem can be stated as follows: "provided a collection of objects, retrieve the nearest neighbor for each object in the collection". A scalable algorithm for performing an all-pairs-similarity-search for time series data can enable a vast array of time series data mining tasks, e.g., motif discovery, discord discovery, density estimation, etc. Given a time series instance, a subsequence that occurs multiple times in the time series (possibly with slight variations) is called a "motif", while a subsequence that occurs rarely in the time series is called a "discord". The high dimensionality of time series data makes the all-pairs-similarity-search problem a daunting task. The usual speed-up techniques of indexing, lower-bounding, triangular inequality-based pruning, and early abandoning, also provide a one to two orders of magnitude speedup at best, and that too when all these techniques are being used in conjunction with each other.

The state-of-the-art in time series all-pairs-similarity-search is the **Matrix Profile (MP)** algorithm [88]. The MP algorithm is "simple, parameter free, exact, space efficient, anytime, incrementally maintainable, scalable, invariant to missing data, and embarrassingly parallelizable" [88]. MP is the name of the algorithm as well as the data structure that is created when performing the all-pairs-similarity-search. MP is, in essence, a data structure that annotates the original time series such that it is possible to determine the nearest neighbor of any subsequence using a single lookup of the Matrix Profile data structure. A fascinating aspect of the MP algorithm is that, once the MP data structure is calculated for a time series, the different data mining tasks become trivial, e.g., tasks like extracting top k motifs or discords, density estimation, rule discovery, clustering, etc. become extremely easy and efficient. The Matrix Profile algorithm has been refined and adapted for a number of different scenarios and applications. In this regard, we refer the interested reader to the corresponding article for a detailed analysis of the Matrix Profile algorithm and its various applications [89].

3 Randomized Time Warping for Full-length Similarity Search

Many time series classification problems are concerned with data which has a specific underlying shape for each class of instances, however, the individual instances might be time shifted, stretched/compressed, or distorted due to noise. The best known approach for such problems is the Nearest Neighbor (NN) algorithm coupled with a suitable distance measure [90]. Specifically, for datasets with a relatively small number of training instances, the NN algorithm coupled with an elastic distance measure performs best compared to the NN algorithm coupled with a lock-step distance measure [80].

Unconstrained Dynamic Time Warping (DTW) has a quadratic computational complexity, therefore, classifying a time series instance using NN-DTW can lead to a large computational cost because any given test instance has to be compared against all the training instances based on a computationally expensive DTW distance measure. Many techniques have been developed for reducing the computational complexity of DTW, which include:

- limiting the required computations using windowing [27, 28, 91],
- lower bounding techniques [29, 30, 75],
- numerosity reduction [92, 93],
- calculating DTW for abstract representations of the data [79], and
- calculating an approximate warping distance [73].

3.1 Randomized Time Warping

A number of application areas can accept a tentative, quick, and reasonably accurate classification even if there is a small chance that the exact classification may turn out to be different afterwards. Examples of such application areas may include real-time or early classification systems. Early classification systems aim to predict a classification as early as possible, while being accurate enough that the decisions based on the early classification are not drastically different than those taken on the basis of an exact classification [94]. In this context, a couple of research efforts have focused on a greedy approach in order to approximate the warping time series distance values [73, 95]. LTW is a greedy warping time series distance measure that was introduced in Section 2.3.3. The main advantage of LTW compared to DTW is the reduced computational cost. In terms of classification accuracy, LTW fares much better against ED than against DTW. The LTW authors also noted that it was faster than DTW for a majority of datasets, however, the number of datasets where LTW was faster and more accurate was less than a quarter of the *UCR Archive (2014)* datasets.

Randomized Time Warping (RTW) is a time series distance approximation algorithm based on a randomized greedy approach. It complies with the three constraints governing DTW and allows to warp the time series instances. RTW relies on a look-ahead approach based on local squared distances for finding an approximate warping path between the time series instances. However, unlike the purely greedy approach of LTW, the distances to the $right$, $diag$, and $down$ neighboring cells are used to make a probabilistic decision about choosing the next cell. The cell with the shortest distance from the current cell has the highest chance of getting selected as the next cell, however, a distant cell may also be chosen which provides an exploratory aspect to an exploitive (greedy) algorithm.

3.1.1 Methodology

DTW calculates the entire global cost matrix before finding the warping path. For constrained DTW, a windowed portion of the global cost matrix has to be calculated before determining the warping path. RTW can determine the warping path during the process of the RTW distance calculation itself. The warping path W is a sequence of (i, j) vertex pairs starting at $(1, 1)$ and ending at (m, n), such that $W = \{(1, 1), \dots, (i, j), \dots, (m, n)\}$, where m and n are the lengths of the given time series instances, respectively. The RTW distance between a time series pair C and Q is defined as:

$$\begin{aligned} RTW(C,Q) &= \sum_{k=1}^{K} d(W_k) \\ &= d(W_1) \qquad + \cdots + d(W_k) \qquad + \cdots + d(W_K) \\ &= d(1,1) \qquad + \cdots + d(i,j) \qquad + \cdots + d(m,n) \\ &= (c_1 - q_1)^2 \qquad + \cdots + (c_i - q_j)^2 \qquad + \cdots + (c_m - q_n)^2 \end{aligned}$$

where $d(W_k)$ is the distance between the time series data points represented by the kth element of the warping path and K is the warping path length.

Algorithm 3.1 lists the steps involved in the calculation of the RTW distance between two time series instances. The inputs to the algorithm include the two time series instances C and Q, along with a window size parameter w, a random number generator RNG, and the required scaling type $rankType$. First, the algorithm initializes the starting (i and j) and ending (m and n) indices, and calculates the distance d for the data points corresponding to the starting indices (Line 3.1-1). Now the procedure enters the loop which determines the RTW distance (Lines 3.1-2 to 3.1-25). The current cell is indicated by the (i, j) index pair, while its immediate neighbors have the index pairs $(i, j + 1)$, $(i + 1, j + 1)$, and $(i + 1, j)$, and are referred to as the $right$, $diag$, and $down$ cells, respectively. At the start of each iteration, the three neighboring distance values d_{right}, d_{diag} and d_{down}, are initialized with large values to mimic an infinite distance if the respective cells are out of bounds (Line 3.1-3). Next, the algorithm calculates the d_{right}, d_{diag}, and d_{down} distance values for the respective valid neighboring cells (Lines 3.1-4 to 3.1-12). Next, the distance values are scaled and normalized so that their sum equals one (Line 3.1-13). Next, a random number is drawn in the range $[0, 1)$ (Line 3.1-14). In the case of a Gaussian random number generator (or a similarly behaving random number generator), the drawn random numbers are restricted to the $[0, 1)$ range by discarding any random numbers outside the allowed range to handle the edge case that can cause repeated

Algorithm 3.1 RTW (C, Q, w, RNG, $rankType$)

```
1:  $i, \leftarrow 1$, $j \leftarrow 1$, $m \leftarrow |C|$, $n \leftarrow |Q|$, $d \leftarrow (c_i - q_j)^2$
2:  while $(i \leq m \wedge j \leq n)$ do
3:      $d_{right} \leftarrow \infty,\ d_{diag} \leftarrow \infty,\ d_{down} \leftarrow \infty$
4:      if $(j+1) \leq \min(w+i, n)$ then
5:          $d_{right} \leftarrow\ (c_i - q_{j+1})^2$
6:      end if
7:      if $(i+1) \leq m \wedge (j+1) \leq n$ then
8:          $d_{diag} \leftarrow\ (c_{i+1} - q_{j+1})^2$
9:      end if
10:     if $(i+1) \leq \min(w+j, m)$ then
11:         $d_{down} \leftarrow\ (c_{i+1} - q_j)^2$
12:     end if
13:     $\{r_{right}, r_{diag}, r_{down}\} \leftarrow RankDistances(d_{right}, d_{diag}, d_{down}, rankType)$
14:     $x \sim RNG$                                   ▷ RNG range is restricted to [0,1)
15:     if $(j+1) \leq \min(w+i, n) \wedge x < r_{right}$ then
16:         $d \leftarrow d + d_{right}$
17:         $j \leftarrow j + 1$
18:     else if $(i+1) \leq m \wedge (j+1) \leq n \wedge x < r_{diag}$ then
19:         $d \leftarrow d + d_{diag}$
20:         $i \leftarrow i + 1,\ j \leftarrow j + 1$
21:     else if $(i+1) \leq \min(w+j, m) \wedge x < r_{down}$ then
22:         $d \leftarrow d + d_{down}$
23:         $i \leftarrow i + 1$
24:     end if
25: end while
26: return $d$
```

iterations of the main loop without any updates to the (i, j) index pair. The exact procedure and possible design options for scaling the distance values as well as for the random number generators are discussed in Section 3.1.2.

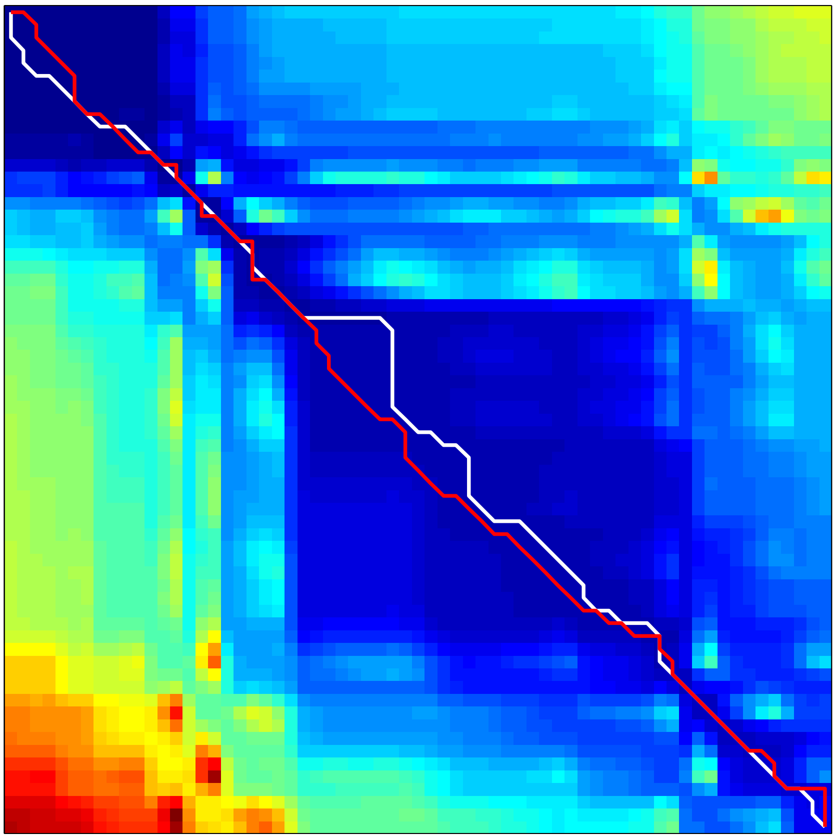


Fig. 3.1. Visual comparison of the DTW (white) and RTW (red) warping paths for two time series instances plotted on a heatmap of the global distance matrix. The global distance matrix is calculated using DTW. The RTW warping path approximates the DTW warping path and avoids any high difference points from the two time series.

The next cell as well as the update value for the accumulated distance d are calculated based on the drawn random number (Lines 3.1-15 to 3.1-24). The outermost loop continues until the index reaches (m, n), at which point the accumulated distance d is reported as the distance between the given time series instances.

Figure 3.1 shows an example of the DTW and RTW warping paths obtained for two time series instances. The DTW warping path is shown in white while the RTW warping path is shown in red. The background shows a heatmap based on the global distance matrix obtained from calculating the DTW distance measure. For a global distance matrix, each cell value $D_{i,j}$ is equal to the sum of the squared difference between the ith and jth data points of the two time series instances $(d_{i,j})$, and the smallest cell distance value from the three neighboring cells $(i-1, j)$, $(i-1, j-1)$, and $(i, j-1)$,

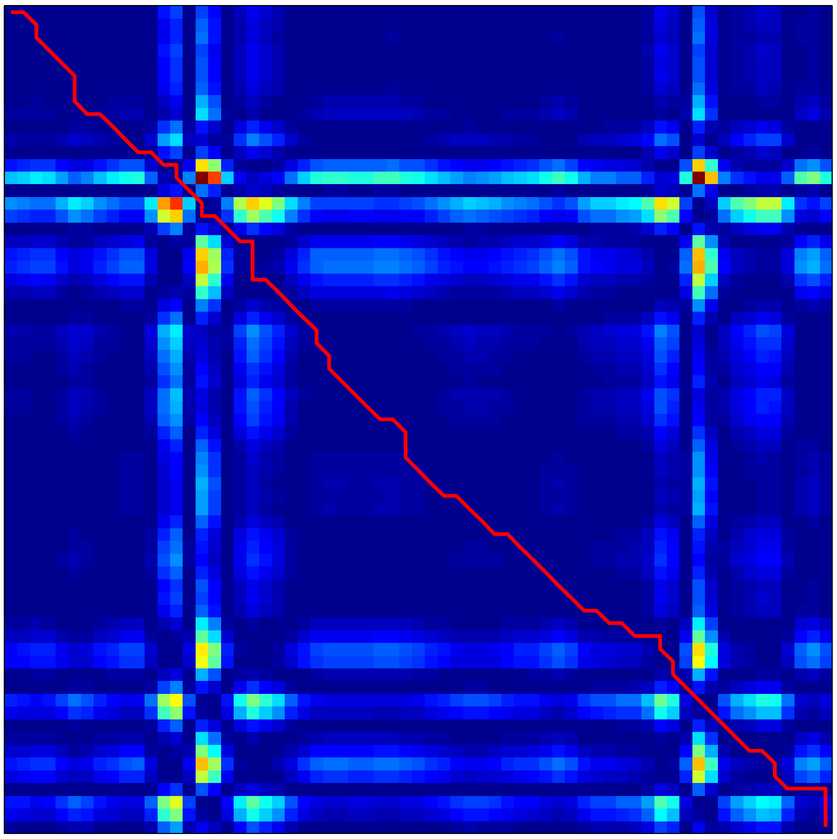


Fig. 3.2. An RTW warping path for two time series instances plotted on a heatmap of the local distance matrix.

i.e., $D_{i,j} = d_{i,j} + \mathbf{min}(D_{i-1,j}, D_{i-1,j-1}, D_{i,j-1})$. A numerical comparison of the distance values contained in the global distance matrix shows that (i) the warping path starts from a high distance value at cell (m, n) and goes to the smallest distance value at cell $(1, 1)$, (ii) the slope of the optimal warping path is always decreasing or zero, and (iii) the global distance values on either side of the optimal warping path tend to increase very fast and reach very high values on either extremes of the global distance matrix, i.e., at cells $(m, 1)$ and $(1, n)$.

In the case of a local distance matrix, each cell represents the squared difference between the ith and the jth data points for the two time series instances. The local distance matrix can be visualized as a highly irregular terrain. Regions of time series instances which have diverging trends introduce increasing distance values to the local distance matrix, while regions with converging trends introduce decreasing distance values. Time series data points which are at alternate extremes of each other will contribute the largest distance values. Figure 3.2 shows a heatmap generated from the local distance matrix of the time series instances used to determine the DTW and RTW warping paths in Figure 3.1. The RTW warping path can be traversed on the basis of the above stated properties of the local distance matrix. The local distance matrix based warping path traversal is done in a look-ahead manner, i.e., at each step of the process, the algorithm jumps to either the $right$, $diag$, or $down$ cell based on the scaled distance and a randomized decision. This causes the warping path to steer away from high distance cells. This can also be seen in Figure 3.2, as the RTW warping path avoids those cells which have larger values corresponding to the larger difference regions of the time series instances, traversing a low difference path towards the cell (m, n).

3.1.2 Design and Parameter Choices

The two main aspects of the RTW algorithm which can affect its performance are the scaling method and the random number generator. In this regard, the following sections present the design choices and their effects on the performance of RTW.

Scaling Method

The distance values between the data points of the time series instances include all non-negative real numbers ($\mathbb{R}_{\geq 0}$), therefore, we scale and normalize the distance values to restrict their range such that the sum of the scaled distance values is equal to one. There are two scaling methods that we have tested for RTW, namely, linear and exponential scaling. Linear scaling is an unbiased method which results in scaled values that are directly proportional to the original distance values. Using an unbiased scaling method allows higher exploration at the cost of longer and highly jagged warping paths. On the other hand, exponential scaling is a biased scaling method which scales the distance values such that the cell with the smallest distance from the current cell has the highest chance of selection. The γ parameter controls the bias of the exponential scaling. $\gamma = 0$ is a special case of the exponential

Algorithm 3.2 RankDistances (d_{right}, d_{diag}, d_{down}, $rankType$)

```
1:  $\gamma \leftarrow 2,\ c \leftarrow 0,\ I \leftarrow \{right, diag, down\}$
2:  if $rankType = Exponential$ then
3:      for all $i \in I$ do
4:          $d_i \leftarrow e^{-\gamma d_i}$
5:      end for
6:  end if
7:  $d_{sum} \leftarrow \sum_{i \in I} d_i$
8:  for all $i \in I$ do
9:      if $rankType = Exponential$ then
10:         $r_i \leftarrow (d_i / d_{sum}) + c$
11:     else
12:         $r_i \leftarrow ((d_{sum} - d_i)/(2 \times d_{sum})) + c$
13:     end if
14:     $c \leftarrow c + r_i$
15: end for
16: return $(r_{right}, r_{diag}, r_{down})$
```

scaling method that ignores the distance values entirely and assigns an equal value to each scaled distance, effectively making the decision of choosing the next cell completely random. As the value of γ increases, the inter-cell distances start taking effect and each segment of the scaling factor starts getting adjusted accordingly. Additionally, the biasing effect also starts to show up and smaller distances get a larger scaling. However, the γ parameter cannot be increased limitlessly because at larger values of γ, all scaled distances become very close to zero, effectively breaking the scaling method. Algorithm 3.2 details the scaling and normalizing procedure.

Random Numbers

RTW relies on random numbers to integrate an explorative aspect to the process of choosing the next cells of the warping path. The random numbers are drawn in the range $[0, 1)$, since the range of the scaled and normalized inter-cell distances is $[0, 1]$. We evaluated the performance of RTW using the Uniform $\mathcal{U}(a, b)$, Normal $\mathcal{N}(\mu, \sigma^2)$, and Skewed Normal $\mathcal{SN}(\xi, \omega^2, \alpha)$ distributions for drawing random numbers. Drawing random numbers from the Uniform distribution $\mathcal{U}(0, 1)$ provides equal probability of selection for each of the $right$, $diag$, and $down$ cells. Drawing random numbers from the Normal distribution $\mathcal{N}(0.5, 0.1)$ provides a higher probability of obtaining random numbers close to 0.5 and hence favoring the selection of the diagonal cell[1], because 68.27% of the numbers drawn will lie within one standard deviation of the mean.[2] Drawing the random numbers from the Skewed Normal distribution, we adjusted the parameters of the distribution in each iteration of the main RTW algorithm, so that a warping path close to the main diagonal would be selected. The skewness of the distribution was coupled with the number of cells that the warping path diverged from the main diagonal. Therefore, when the warping path shifted towards one side of the main diagonal, the random numbers generated would be in favor of moving in the other direction, which pulls the warping path back to the main diagonal of the cost matrix, basically restricting the amount of warping. However, it was observed that the use of a Skewed Normal distribution gave consistently similar results as the Normal distribution, but at twice the computational cost of using the Normal distribution. Therefore, the Skewed Normal distribution was not considered further for the empirical evaluation of the algorithm.

3.2 Empirical Evaluation

An extensive set of experiments was performed for the evaluation and comparison of RTW against Euclidean distance (ED), Dynamic Time Warping (DTW), and Lucky Time Warping (LTW). The different algorithms were implemented using a consistent program structure to avoid any implementation or optimization bias. The source code for the different algorithms and the

[1]The word diagonal should not be confused with the main diagonal of the local distance matrix, rather it is referring to any cell which is located diagonally from the current cell and can be reached as $(i, j) \to (i + 1, j + 1)$.

[2]The specific μ and σ^2 parameters for the Gaussian distribution are chosen to generate 99.7% of the random numbers between the allowed range of $[0, 1)$.

scripts for generating the different plots are available online.[3] The main goal of the experimental evaluation is to investigate the classification accuracy and computational cost of the proposed algorithm compared to ED, DTW, and LTW. Nearest neighbor coupled with the respective distance measures was used to determine the classification accuracy for each dataset. The runtime of each algorithm was monitored within the program.

The time series datasets available from the *UCR Archive (2014)* were used for the empirical evaluation [58]. We also present the results for the same training and testing splits provided by the *UCR Archive (2014)* for the sake of reproducibility.

We have intentionally avoided a learning strategy (e.g., parameter tuning by grid search and interval cross-validation) for finding the Sakoe-Chiba band for any warping technique. We have presented the results at specific window sizes for all the competing techniques. This way we can compare the classification accuracy of the different techniques using the same criteria.

Early abandoning allows to stop a distance calculation as soon as the accumulated distance value exceeds the current best so far value. Early abandoning ED, LTW, and RTW is trivial, however, DTW requires the introduction of lower bounding techniques before early abandoning can be incorporated. The inclusion of lower bounding for DTW can introduce a bias in the computational comparisons as well, therefore, we have refrained from investigating early abandoning in our experimentation.

Since RTW is a randomized algorithm, we evaluated each dataset ten times and compare the competing algorithms based on the average classification accuracy and runtime performance of the ten runs. The runtime of the algorithms was monitored using standard timing utilities provided by the implementation platform. The experiments were performed on a high performance cluster, so the variation in the runtime performance of the deterministic methods, i.e., DTW, LTW, and ED, is a result of the load factor of the processing node at different times. In the case of RTW, the variation is a combination of the variability introduced due to the shared execution platform as well as the randomized nature of the algorithm.

[3]RTW Repository: https://github.com/kramerlab/randomized-time-warping.

The empirical evaluation was carried out on the basis of the following experimental and parameter settings:

- The **warping window size** was set to 100, 20, 10, and 5 percent of the time series length for each of RTW, LTW, and DTW.
- **Random numbers** were generated based on the Uniform and Normal distributions. It was observed that the classification accuracy was always better when using the Normal distribution rather than the Uniform distribution, therefore, the following results are based on the Normally distributed random numbers.
- **Scaling** was performed using both exponential and linear methods. The γ parameter for exponential scaling can be tuned for individual datasets, however, we always kept it constant at 2. Overall, the exponential scaling method always resulted in a better classification accuracy, therefore, we only consider exponential scaling for the following discussion.

3.3 Results

Our empirical evaluation showed that using Normally distributed random numbers with exponential scaling provided the best classification accuracy. On the other hand, window size can have a significant effect on the overall accuracy, therefore, we only show the results when using the maximum window size to eliminate unintentional bias against any algorithm. The following discussion is based on the experimental evaluation using a warping window size of 100, using exponential scaling, while the random numbers are drawn from a Gaussian distribution with the mean and standard deviation set to the values already discussed in the previous section.

3.3.1 Classification Accuracy

Previous research has shown that 1NN is quite competitive and both the 1NN-ED and 1NN-DTW are hard to beat [96]. Since RTW has been proposed as an approximating distance measure, the main aspect that will be considered regarding classification accuracy is that RTW should be able to perform on par compared to ED and DTW. Since RTW is an explorative as well as an exploitive algorithm, it is expected that it will perform better compared to the LTW algorithm since LTW is only an exploitive algorithm.

Figures 3.3, 3.4, and 3.5 present a comparison of the classification accuracy obtained by RTW compared to ED, LTW, and DTW, respectively. The average classification accuracy obtained for each dataset after ten runs of RTW is plotted along the vertical axis while the classification accuracy for the other algorithm is plotted along the horizontal axis. Data points lying on the diagonal and plotted using blue "." markers are for those datasets for which both algorithms provided almost equal classification accuracy ($\pm 0.5\%$). The data points plotted using a green "+" sign indicate that RTW achieves higher classification accuracy. Finally, data points plotted with a red "$\times$" are for datasets where the other algorithm performs better than RTW.

Figures 3.3 and 3.4 show that more than half of the datasets in the *UCR Archive* get better classification accuracy when using RTW compared to ED or LTW. The classification accuracy of RTW is on par compared to ED, since the data points are clustered around the main diagonal. When comparing RTW against LTW, the results are in favor of RTW since the data points move away from the main diagonal in favor of RTW and the number of datasets with better classification accuracy when using RTW is also increased. When comparing RTW against DTW, Figure 3.5 shows that the majority of datasets get better classification accuracy using DTW, but the clustering of almost all datasets near the upper right corner and close to the main diagonal suggests that RTW performs very well in approximating the correct classification here as well. There is a group of four datasets which get less than 50% accuracy when using RTW but the classification accuracy with the competing algorithms is higher than 50%. The classification accuracy for this group of datasets is also quite consistent among the other algorithms, i.e, ED, LTW, and DTW. Two of these datasets are spectral analyses while the other two are image outline datasets. The severely degraded classification accuracy for these datasets does not, however, indicate that RTW is ill-suited for spectral analyses or image outline datasets because there are other datasets from these categories which get better results.

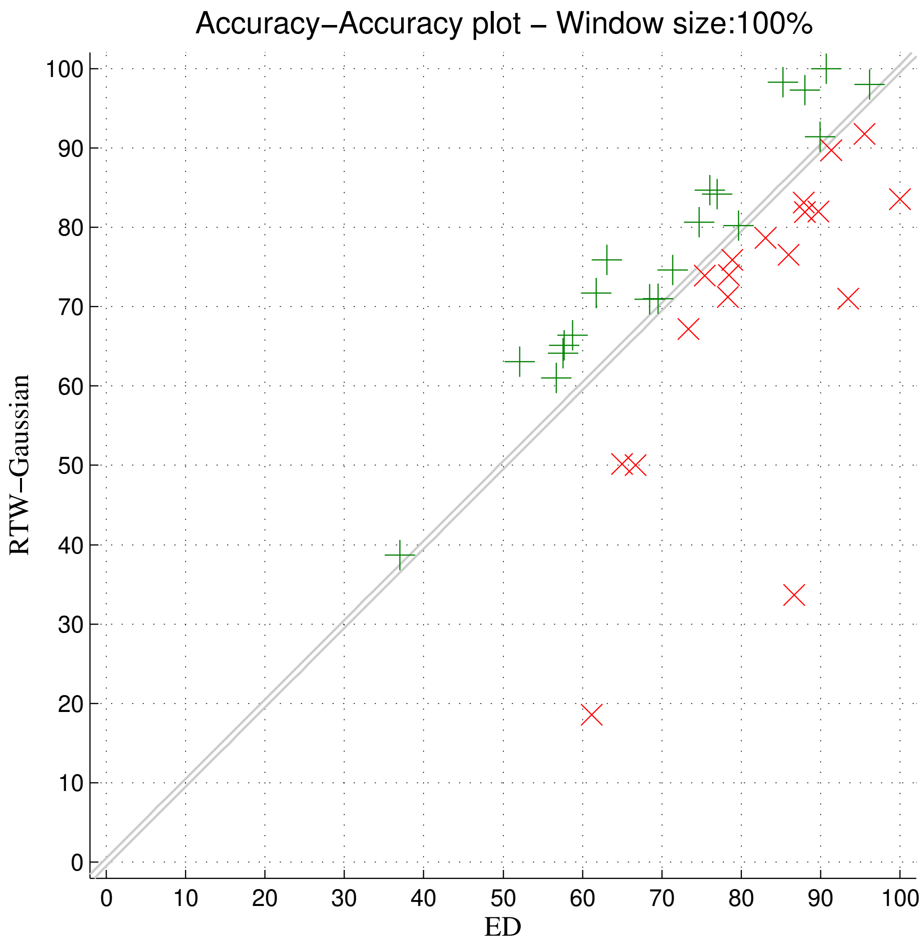


Fig. 3.3. Comparing the classification accuracy of RTW against ED obtained using a 100% window size. Each marker represents the classification accuracy obtained by the two algorithms for a single dataset.

3.3.2 Running Times

LTW and DTW are deterministic algorithms, so the time required to evaluate a given dataset is the same over multiple runs. On the other hand, RTW is a randomized algorithm, therefore, we can observe a slight variation in its runtime requirements over different runs while evaluating the same dataset. Therefore, we have reported an average runtime requirement for each dataset when evaluating with RTW. The runtime of the LTW algorithm is used as a baseline for comparing the runtime performance of the different algorithms. For each dataset, the time required by LTW is used to normalize the runtime requirements of the other algorithms, therefore the runtime requirements of RTW and DTW are reported as multiples of the runtime requirements for

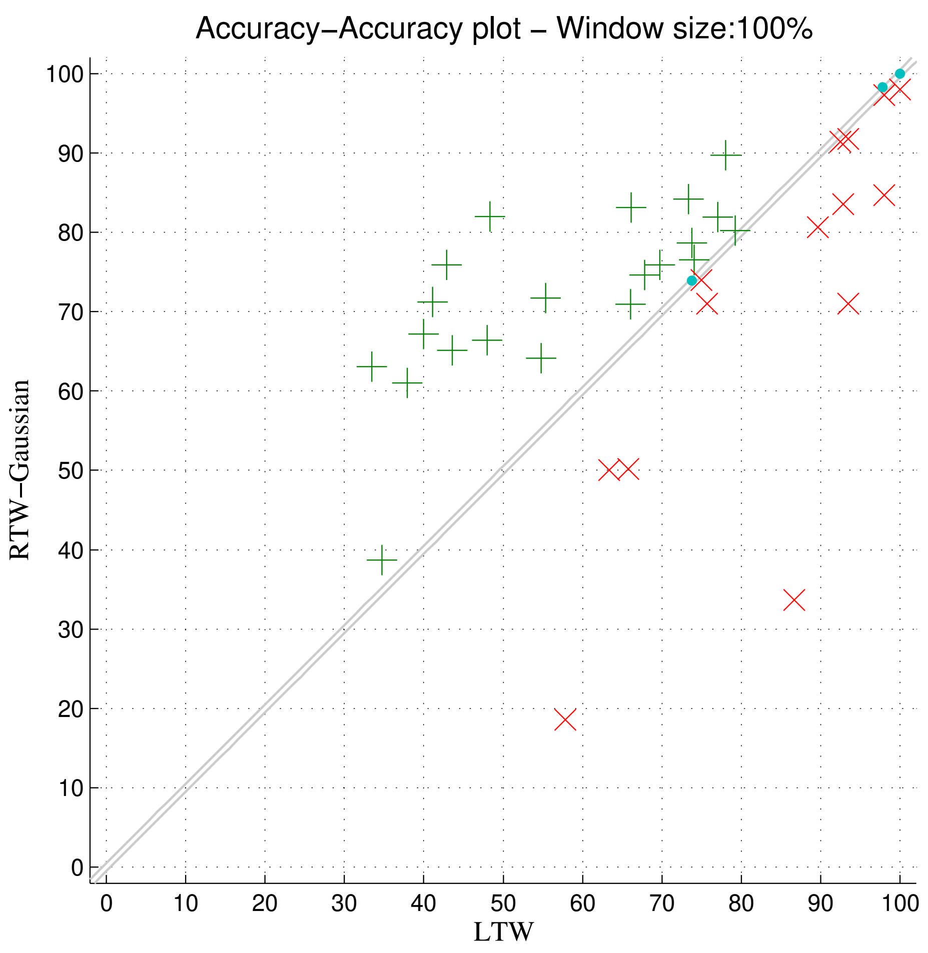


Fig. 3.4. Comparing the classification accuracy of RTW against LTW obtained using a 100% window size. Each marker represents the classification accuracy obtained by the two algorithms for a single dataset.

LTW. Figure 3.6 shows a comparison of the running times of DTW, LTW and RTW for the evaluated datasets. The LTW runtime for each dataset shows up as one due to the normalization. For almost all datasets, RTW is one to two times slower than LTW, however, a highly unexpected observation is that for some datasets RTW is faster than LTW. On the other hand, RTW is always faster than DTW except for one dataset. For smaller datasets, the computational advantage of RTW compared to DTW is minimal, but for larger datasets we can see that RTW performs much better in terms of the required computational effort and is two to eight times faster than DTW. The advantage of being less computationally intensive than DTW along with a better classification approximation than LTW allows RTW to perform much better than LTW as a greedy warping algorithm.

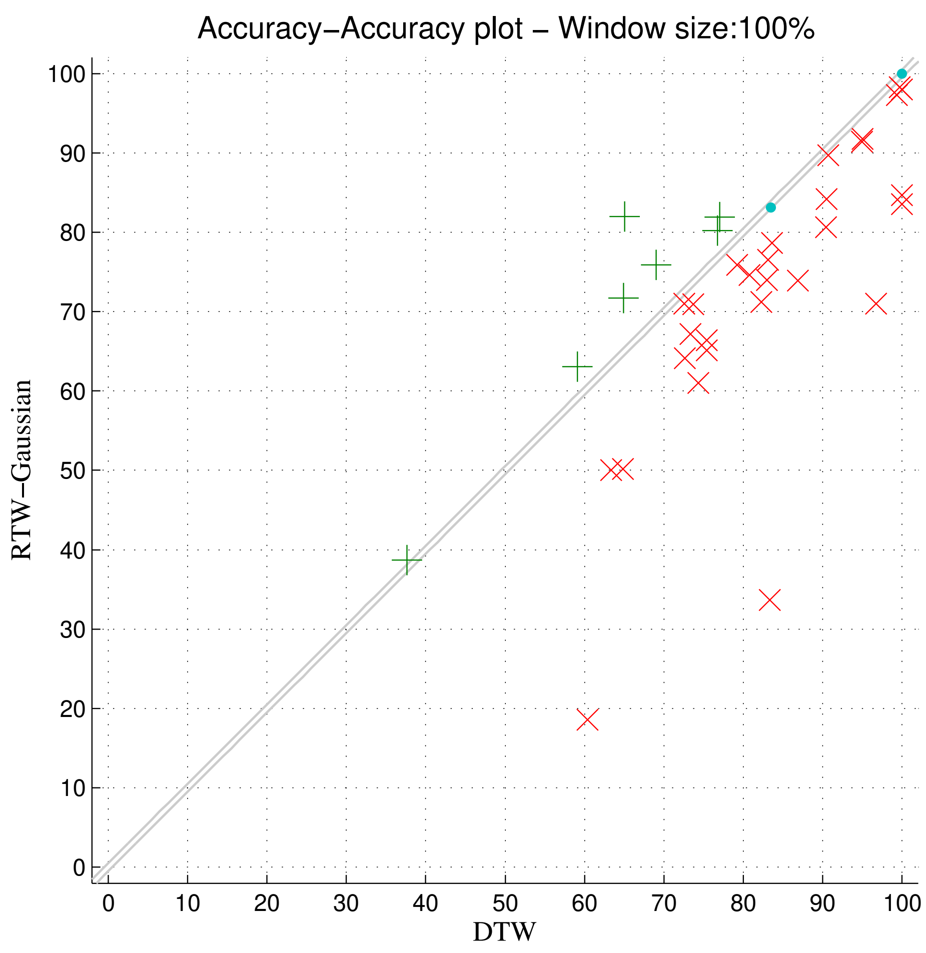


Fig. 3.5. Comparing the classification accuracy of RTW against DTW obtained using a 100% window size. Each marker represents the classification accuracy obtained by the two algorithms for a single dataset.

3.3.3 Unified Comparison of Accuracy and Runtime

In order to further illustrate the performance gains offered by RTW compared to the other algorithms, Figures 3.7, 3.8, and 3.9 present a unified comparison of RTW against ED, LTW, and DTW regarding classification accuracy and runtime. The figures show the comparison of the different algorithms based on the "scaled" ratios of classification accuracy (along the horizontal axis) and run-times (along the vertical axis) for all datasets. The value of "1" on the respective axes indicates that the accuracy or runtime of the competing algorithms is equal. A value greater than "1" for the ratio of classification accuracies means that RTW has a better classification accuracy than the other algorithm, while a value greater than "1" for the ratio of run-times means

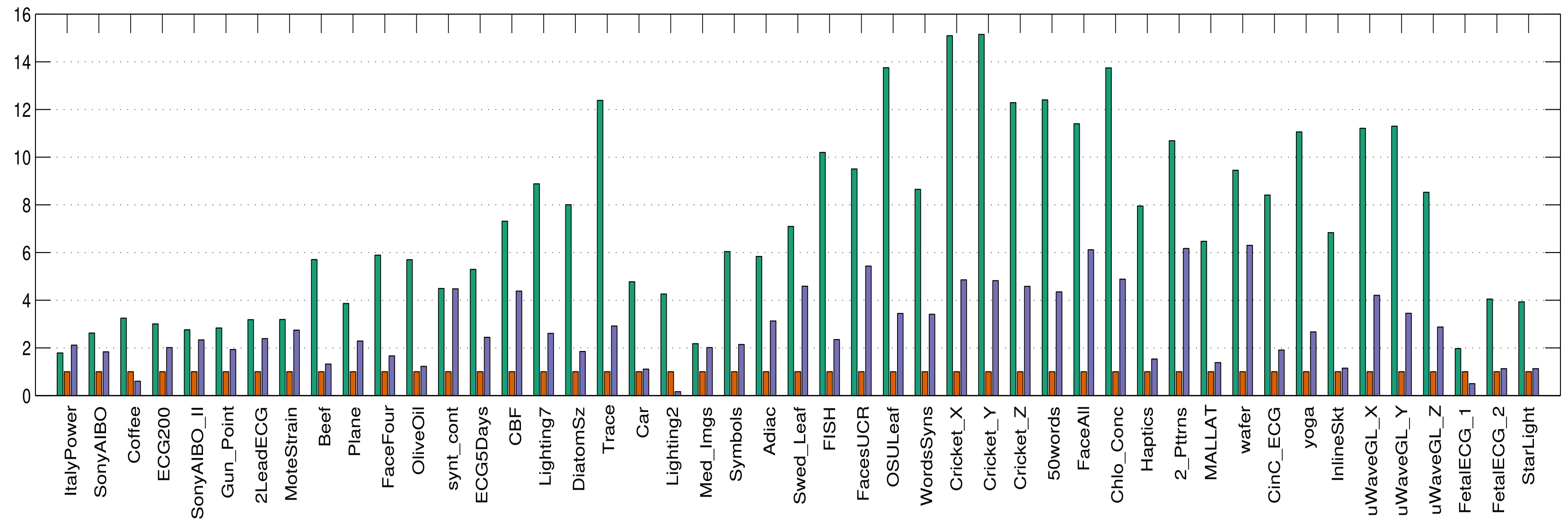


Fig. 3.6. Run times of DTW (green), LTW (orange) and RTW (violet) for all *UCR Archive (2014)* datasets. Run time of LTW is set as the baseline and DTW and RTW run times are reported as multiples of LTW run time so that if LTW takes "x" units of time to compute the warping, DTW and RTW take a multiple of that "x" value. Datasets are listed in the order of increasing size in terms of instances to be compared and the size of cost matrix to be computed.

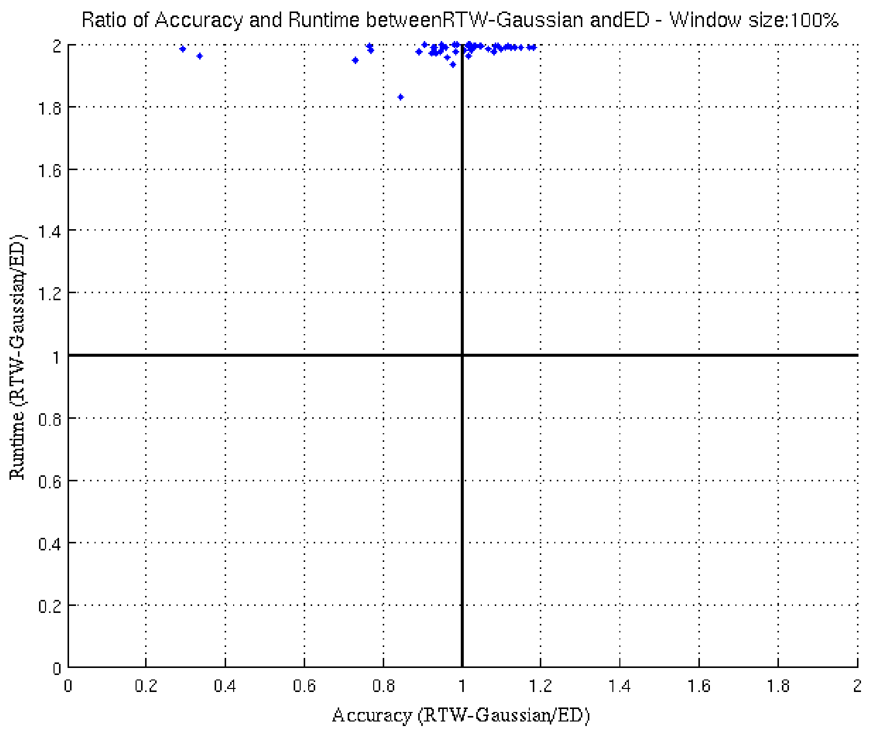


Fig. 3.7. Comparing the classification accuracy and runtime performance of of RTW against ED using scaled ratios of the two quantities. Each point represents the ratio of classification accuracy and runtime obtained by the two algorithms for a single dataset.

that RTW has a longer running time than the other algorithm. Figure 3.7 shows that RTW is always on par with ED regarding classification accuracy while its runtime is always greater than ED. Figure 3.8 shows the comparison of RTW and LTW, where the majority of data points lie in the $x > 1; y > 1$ region. This implies that RTW is computationally demanding than LTW. On the other hand, RTW also obtains better classification accuracy as compared to LTW. Figure 3.9 compares RTW against DTW, where we see that almost all data points lie around the $x = 1$ line, which means that the difference between the classification accuracy of RTW and DTW is relatively small even though RTW fails to obtain a better classification accuracy than DTW for most datasets. Regarding the computational requirements of RTW and DTW, almost all data points lie below the $y = 1$ line, proving that RTW takes less computation effort than DTW.

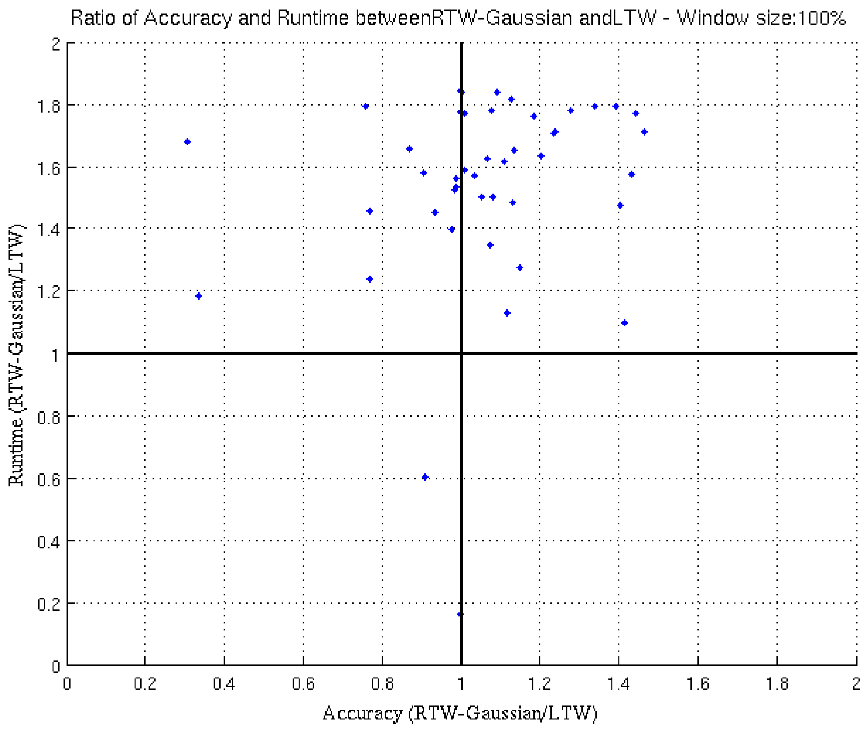


Fig. 3.8. Comparing the classification accuracy and runtime performance of of RTW against LTW using scaled ratios of the two quantities. Each point represents the ratio of classification accuracy and runtime obtained by the two algorithms for a single dataset.

3.4 Conclusion

We proposed Randomized Time Warping (RTW) for time series similarity search, which uses a randomized greedy approach and has a near-linear space and time complexity. The comparison of classification accuracy and computational complexity of RTW against DTW and LTW shows that RTW provides a very balanced trade-off between classification accuracy and computational complexity. It provides a reasonably good approximation of classification as compared to DTW, while its computation time is much less than that of DTW.

We proposed the Randomized Time Warping (RTW) distance for an efficient 1-NN time series classification approximation, which determines a warping path using a randomized greedy approach. In this work we have shown that RTW:

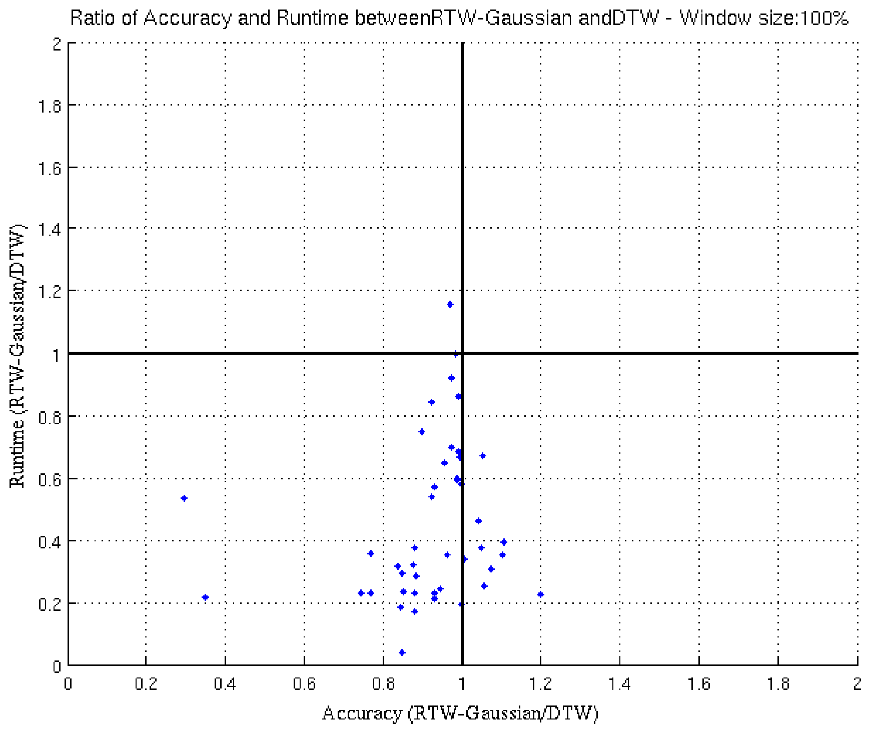


Fig. 3.9. Comparing the classification accuracy and runtime performance of of RTW against DTW using scaled ratios of the two quantities. Each point represents the ratio of classification accuracy and runtime obtained by the two algorithms for a single dataset.

- approximates time series distance at the cost of classification accuracy,
- can perform better classification than a simple greedy approach via stochastic search,
- is a simple, but effective algorithm with linear time and space requirements,
- has a less computationally intensive warping process than DTW,
- can trade off computational effort in favor of classification accuracy and vice versa, and
- is able to provide a consistent distance approximation at varying warping window sizes and hence, is unaffected by warping constraints.

The algorithms in this research work were tested in quite a “pure" setting, without specific optimizations, like for instance with respect to the window size parameter. Also, it should be noticed that for a subset of the UCR data sets, Euclidean distance cannot be beaten by warping distance measures. Those cases could easily be determined in practice by a simple cross-validation. In other words, it would be easy to filter those cases in which a trivial warping path (or, equivalent, minimal window size) gives the best results, and only use the more involved variants in the non-trivial cases. For the non-trivial cases, for which larger window sizes pay off, RTW could offer an intriguing possibility: DTW is often used with a parameter optimization step using leave-one-out cross-validation for finding the most suitable window. This incurs a hidden computation cost, which is mostly not mentioned. Since RTW is computationally less expensive, we could use RTW to create an irregular window for enhancing the accuracy of RTW. RTW can be used multiple times to figure out the most traversed paths for each class in the training set, and then these paths can be used to create irregular windows for each class's instances. Afterwards, RTW can run for each irregular window only once instead of evaluating the time series pair multiple times to figure out the best classification.

It would be tempting to measure success not by mere prediction accuracy (fraction of correctly predicted classes), but also taking into consideration the confidence in the predictions (for instance, the margin, the difference in the confidence in the most probable and the second most probable class). It is quite likely that some of the results for the variants of DTW will appear in a different light then.

We have discussed the use of different design parameters that affect the performance and/or accuracy of our algorithm including different probability distributions, using different ranking strategies and the effects of re-evaluations. There is potential for adopting early abandoning for speed improvements. Further improvements in classification accuracy may be achieved using weighting strategies for the choice of directing the warping path based on diverging or converging time series.

Random Shapelet Ensembles

4

4.1 Background and Motivation

Many time series classification problems are concerned with data for which the overall shape of the time series instances provides little to no information about the class, but small subsequences present in the data are indicative of the class of the time series instances. These subsequences are (i) much smaller than the overall length of the time series, (ii) phase independent, and (iii) can occur at any point in the time series instances. These problem characteristics imply that determining the presence or absence of discriminative subsequences in the time series instances is a better suited classification approach compared to whole-matching. Ye and Keogh proposed an algorithm to handle problems of this nature and referred to the discriminative subsequences as *Shapelets* [32]. The proposed algorithm is referred to as the YK-shapelets algorithm in this text. An introduction of the shapelets based classification approach as well as the YK-shapelets algorithm is provided in Section 2.4.2.

Shapelets based classification is much faster compared to the 1-NN approach because, for a given time series instance, shapelet based classification only needs to compute a few inexpensive distance values for the given time series instance and the shapelets encountered in the nodes of the decision tree, whereas the 1-NN approach needs to compare the given time series instance with all available reference time series instances before finalizing the classification decision. On the other hand, shapelet based model induction can become untenable for larger datasets because shapelet discovery is an exhaustive process of evaluating all possible candidate subsequences in the time series dataset to determine the best shapelet and its corresponding distance threshold at each node of the shapelet based decision tree model. At each level of the decision tree, the time required for shapelet discovery is on the order of $O(N^2n^4)$, where N represents the number of time series instances in the dataset, while n is the length of the time series instances. Hence, the overall time required for shapelet based model induction can become unreasonably high. A number of approaches were already proposed by Ye and Keogh to reduce the computational complexity of the YK-shapelets algorithm [32], however, these approaches only reduce the computational cost by a fraction of the total cost since the asymptotic cost of the algorithm remains the same. Further research efforts aimed at reducing the computational complexity include *Logical-Shapelets* [81], *Fast-Shapelets* [34], *Learning Shapelets* [97], etc.

4.2 Randomized Shapelet Ensembles

The YK-shapelets algorithm generates the shapelet candidates with a unit step size (see Algorithm 2.3-4), therefore, each candidate almost entirely overlaps its immediate neighboring candidates. The neighboring candidates have, therefore, been aptly named trivial matches in the research literature and it has been established that such overlap is undesirable for multiple reasons [98]. Since the overlapping subsequences are almost identical and have very similar distances to the time series instances, if a subsequence is pruned because it is not a useful candidate, the algorithm still evaluates its immediate neighbors, which leads to considerable wasted computational effort.

4.2.1 Random Shapelets

The Random-Shapelets algorithm aims to reduce the computational cost of the YK-shapelets algorithm by intentionally skipping shapelet candidates based on a uniform random sampling approach. Algorithm 4.1 shows the details of the randomized shapelet discovery procedure used in the Random-Shapelets algorithm. The overall shapelet discovery procedure of the Random-Shapelets and YK-shapelets algorithms is the same except for the introduction of a control flow block which skips the evaluation of a given candidate shapelet based on a uniform random number and a user provided sampling percentage (Lines 4.1-7 to 4.1-9).

Figure 4.1 illustrates the candidate generation approaches of the YK-shapelets and Random-Shapelets algorithms, respectively. It shows the first ten shapelet candidates generated for a time series instance using both the approaches. The candidates generated using the unit step size approach of YK-shapelets have a very high overlap and only cover a small section of the time series, while the candidates generated using a uniform random sampling approach of Random-Shapelets have lesser overlap and the same number of candidates also covers a longer section of the time series. Using an $m\%$ sampling, the number of possible shapelet candidates of length $l \in [min\,..\,max]$ is approximately equal to:

$$\frac{m}{100} \times N \times \sum_{l=min}^{max} (n - l + 1),$$

where N is the size of the dataset and n is the length of the time series instances. Therefore, the Random-Shapelets algorithm reduces the total number of evaluated shapelet candidates while also avoiding the significant candidate overlap problem.

The Random-Shapelets algorithm skips a large percentage of all possible shapelet candidates at each node of the decision tree. This drastically reduces the computational cost, but also leads to deeper decision tree models compared to the models induced by the YK-shapelets algorithm. The deeper decision tree models can often lead the Random-Shapelets based classification models to overfit the problem.

Ensemble methods are based on the idea of combining the opinions of different experts to obtain a decision about a given problem. Members of an ensemble can have a different view of the data, or they can use different features for making their decision, or they can be totally different algorithms. This provides diversity in the ensemble decisions, which often makes them more accurate than individual models because ensemble methods inherently tend to reduce variance and sometimes also bias. Ensembles of machine learning models have been shown to often outperform individual models, provided

Algorithm 4.1 FindShapelet (D, l_{min}, l_{max})

```
1: $IG_{best} \leftarrow 0$
2: for $l \in [l_{min} \mathinner{..} l_{max}]$ do
3:     for $i \in [1 \mathinner{..} |D|]$ do
4:         $n \leftarrow |D[i]|$
5:         for $p \in [1 \mathinner{..} (n - l + 1)]$ do
6:             $x \sim \mathcal{U}(0, 1)$
7:             if $m < x$ then                    ▷ $m$ is a globally defined variable
8:                 continue
9:             end if
10:            $S \leftarrow D[i]^l_p$
11:            $order\_line \leftarrow \varnothing$
12:            for $j \in [1 \mathinner{..} |D|]$ do
13:                $dist_j \leftarrow dist(S, D[j])$
14:                place $dist_j$ on $order\_line$
15:            end for
16:            $IG, \delta \leftarrow$ CheckCandidate$(order\_line)$
17:            if $IG > IG_{best}$ then
18:                $IG_{best} \leftarrow IG$
19:                $\delta_{best} \leftarrow \delta$
20:                $S_{best} \leftarrow S$
21:                $d_{best} \leftarrow order\_line$
22:            end if
23:        end for
24:    end for
25: end for
26: return $S_{best}$, $\delta_{best}$, $d_{best}$
```

the models constituting the ensemble are (i) accurate, i.e., provide better results than random guessing, and (ii) diverse, i.e., make different errors for an unseen problem [99]. The Random-Shapelets algorithm's approach of

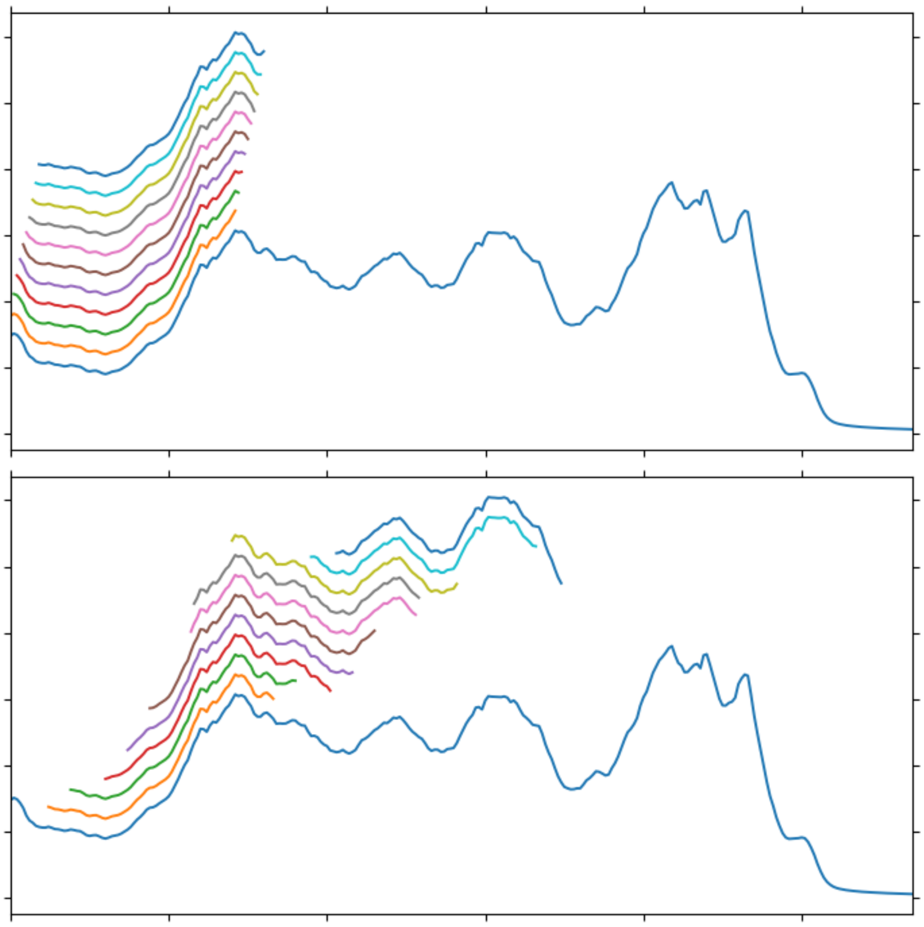


Fig. 4.1. The first ten shapelet candidates generated using the unit step approach of YK-shapelets (top), and the randomized approach of Random-Shapelets (bottom). The time series is also shown for reference while the candidates are shown with an offset for clarity.

creating good-enough but inexpensive models makes it a prime candidate for incorporating an ensemble of Random-Shapelets based models.

The various proposed combinations in the literature regarding ensembles make it possible to experiment with a number of options. A basic ensemble of classifiers can simply be obtained by combining multiple diverse models all trained using the same data. Another approach for constructing ensembles is that of Bagging, which trains Q models, each with a different bootstrap sample of data such that $|D|$ instances are sampled with replacement from the original dataset [100]. This introduces a diversification effect and a duplication of instances also allows individual models to be focused on the duplicated instances. Algorithm 4.2 provides the pseudo-code for bagging.

Algorithm 4.2 Creating Bagging Ensembles

```
1: $M \leftarrow \varnothing$
2: for $q \in [1 \,..\, Q]$ do
3:     $D_q \leftarrow$ sample $|D|$ instances from $D$ with replacement
4:     $M_q \leftarrow$ train classifier on $D_q$
5:     $M \leftarrow M \cup M_q$
6: end for
7: return $M$
```

Another approach for constructing ensembles is that of Boosting that relies on weighted instances [101]. All training instances are assigned equal weights so that the sum of instance weights equals one. A model is trained and a classification of training data using this model identifies the misclassified instances, whose weights are increased. Next, the weights of all instances are normalized to keep the sum of weights equal to one. This, in turn, decreases the weights of correctly classified instances and provides emphasis on the misclassified instances in the next iteration and in many cases leads to an improved overall accuracy of the ensemble. Algorithm 4.3 provides the pseudo-code for the AdaBoost.M1 algorithm.

4.2.2 Random Shapelet Ensembles

The Random-Shapelets algorithm is computationally less expensive than the YK-shapelets algorithm, however, the induced models have suboptimal results since the randomization can introduce variability into the models. The small computational cost and inherent randomization of the Random-Shapelets algorithm make it a well-suited candidate for incorporation in an ensemble as a base classifier. This combines the strengths of ensemble learning with the efficient but non-exact shapelet discovery process of Random-Shapelets to provide a cost effective alternative to the exact YK-shapelets approach. Another benefit of choosing Random-Shapelets as the base classifier is that it requires a single parameter, i.e. the sampling ratio, which allows to reduce the number of evaluated candidates and directly corresponds to the amount of computation we are willing to spend for finding the shapelets. Using a

Algorithm 4.3 Creating Boosted Ensembles

```
1: $M \leftarrow \varnothing$
2: $w_{1i} \leftarrow \frac{1}{|D|}, \forall x_i \in D$
3: **for** $q \in [1..Q]$ **do**
4:     $M_q \leftarrow$ train classifier on $D$ with $w_q$
5:     calculate weighted error $\epsilon_t$
6:     **if** $\epsilon_n \geq 0.5$ **then**
7:         $Q \leftarrow q - 1$
8:         $BREAK$
9:     **else**
10:        $\alpha_q \leftarrow \frac{1}{2} \times ln\frac{1-\epsilon_q}{\epsilon_q}$
11:        $w^i_{(q+1)} \leftarrow \frac{w^i_q}{2\epsilon_q}, \forall$ misclassified $x_i \in D$
12:        $w^j_{(q+1)} \leftarrow w^j_q, \forall$ correctly classified $x_j \in D$
13:        re-normalize $w_{(q+1)}$
14:    **end if**
15:    $M \leftarrow M \cup M_q$
16: **end for**
17: **return** $M(x) = \sum_{t=1}^{Q} \alpha_q M_q$
```

small value for the sampling ratio provides speed-up while a higher value provides results which are more consistent with those of the brute force YK-shapelets approach.

Our proposed method is referred to as the Random-Shapelets Ensembles (RSE). The model generation within the ensemble methods use the decision tree construction from Algorithm 2.2, which in turn uses the randomized shapelet discovery process detailed in Algorithm 4.1. The simple RSE creates an ensemble of Random-Shapelets based models using the original training set and only relies on the diversification provided by the Random-Shapelets classifier. The RSE-Bagging approach is based on the bootstrap aggregation algorithm detailed in Algorithm 4.2. RSE-Bagging employs different boots-traps of the training data to train the models that constitute the ensemble. This incorporates randomization at two levels in the overall process, i.e., the shapelet discovery process and the input data for each model. The bootstrapping process also provides instance duplication, which allows the shapelet discovery process to quickly identify and separate the duplicated instances that allows to efficiently perform the search for shapelets in the other instances. The RSE-Boosting approach is based on the boosting approach detailed in Algorithm 4.3. For RSE-Boosting, the training set instances are weighted and the weights of the instances are updated after each iteration so that the subsequent iteration trains a model that is focused on the misclassified instances from the previous iterations. This approach calculates the Information Gain using the instance weights instead of simply the class counts in the current split.

4.2.3 Algorithmic Optimizations

Algorithm 2.4 provides the details for calculating the distance between a shapelet candidate and a time series instance. This involves computing the distance between the shapelet candidate and all possible subsequences of the time series instance whose length is equal to the shapelet candidate. Normalizing the subsequences before calculating their distance from the shapelet candidate improves the similarity search process, however, this requires the calculation of mean and standard deviation values for each subsequence prior to the distance calculation. Calculating these properties for all possible subsequences becomes prohibitively expensive, therefore, subsequence normalization was not employed by YK-shapelets or Random-Shapelets. The RSE approach, however, employs an extremely efficient approach for subsequence normalization. This allows to improve the overall accuracy without any serious impact on the computational cost of the RSE approach.

Sakurai, Papadimitriou, and Faloutsos introduced a constant time method for calculating the mean and standard deviation values of arbitrarily long subsequences that have been extracted from a time series instance whose "sufficient statistics" are available [102]. For a time series X of length n, the sufficient statistics are the cumulative sum of the individual data points ΣX, and the cumulative sum of the individual data points squared ΣX^2. Both these vectors have the same length as the time series X, and they can be efficiently precomputed as part of the time series data loading procedure. Mathematically, the sufficient statistics for X can be defined as:

$$\Sigma\, X[i] = \sum_{k=1}^{i} x_k, \quad \text{for } i = 1, 2, \dots, n$$
$$\Sigma\, X^2[i] = \sum_{k=1}^{i} x_k^2, \quad \text{for } i = 1, 2, \dots, n$$

A subsequence S_p^l that is extracted from X can be z-normalized as follows:

$$S_i = \frac{S_i - \mu_S}{\sigma_S}, \quad \text{for } i = 1, 2, \dots, l.$$

where μ_S and σ_S are the mean and standard deviation of the subsequence, respectively, and are calculated as follows:

$$\mu_S = \frac{x_p + \Sigma\, X_{p+l-1} - \Sigma\, X_p}{l}$$

$$\sigma_S = \sqrt{\frac{x_p^2 + \Sigma\, X_{p+l-1}^2 - \Sigma\, X_p^2}{l} - \mu_S^2}$$

where l is the length of the subsequence while p denotes the starting point of the subsequence in the original time series instance such that $l \leq n$ and $1 \leq p \leq n - l + 1$. The above calculations only require a few lookups for precomputed values followed by a couple of mathematical operations, which makes the entire calculation constant time and independent of the length of the subsequence S_p^l.

4.3 Empirical Evaluation

An extensive set of experiments have been performed for an empirical evaluation of our proposed approach. Specifically, we want to evaluate whether the proposed approach provides on par classification accuracy compared to the other approaches. On the other hand, we also want to determine the required computational cost compared to the other algorithms, in particular the YK-shapelets algorithm. The YK-shapelets, Random-Shapelets, and the Random-Shapelets Ensembles variants are implemented using a consistent program structure to avoid any platform or implementation bias. The source code for our implementation of these algorithms is available online.[1] The Fast-Shapelets implementation was obtained from the *UEA TSML Repository*.

The different algorithms were evaluated using the original train and test splits of the 47 real-world and synthetic time series datasets provided by the 2014 version of the *UCR Archive* [58]. YK-shapelets is an exact algorithm and provides the same results given the same input parameters, therefore, the experiments for YK-shapelets were carried out only once to avoid the high computational costs for large datasets. For the randomized algorithms, each dataset was evaluated 100 times with the random number generator seed set to an integer corresponding to the evaluation number, and the mean and standard deviation of the classification accuracy and runtime are used for the comparison. The number of classification models per ensemble was set to ten for each ensemble variant and majority voting is used for classification. The ensembles are created using fully grown decision tree models without pruning, i.e., the decision trees are allowed to grow until the entropy of the incoming dataset split drops below a very small value or the incoming split has been reduced to a single time series instance.

The shapelet discovery process evaluates all possible shapelet candidates in the range l_{min} and l_{max}, therefore, setting these values to the extreme cases, i.e., $l_{min} = 1$ and $l_{max} = n$, makes the algorithm search over the entire candidate set, resulting in a huge computational cost. A reasonable alternative is to specify the range of shapelet candidate lengths, however, setting these parameters incorrectly can be detrimental to the shapelet discovery process. Setting the parameters to a very small window can cause the shapelet discovery process to miss important features because they are not covered by the window size, while setting the window to a very large size can incur huge computational costs. The best l_{min} and l_{max} parameter values could be determined via a parameter optimization step, however, this strategy can contribute a significant runtime overhead for the YK-shapelets algorithm. Therefore, instead of setting the parameter values to the extreme cases or making any assumptions about the possible shapelet lengths, or

[1] RSE repository: `https://github.com/kramerlab/random-shapelet-ensembles`.

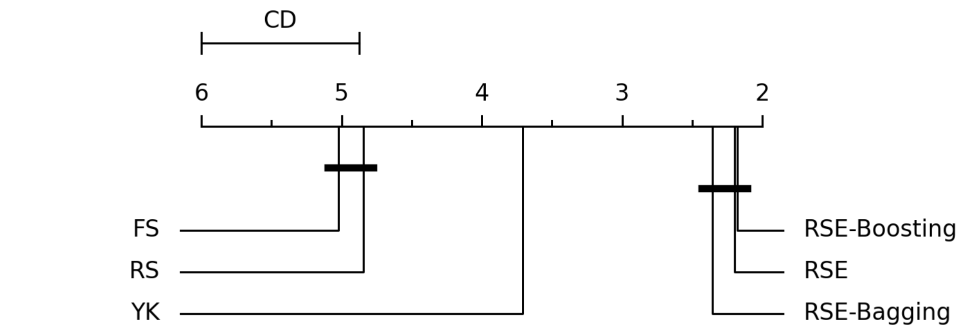
setting the parameter values after a parameter optimization step, we set the parameters to a constant fraction of the time series length for all datasets such that $l_{min} = \lfloor 0.25 \times n \rceil$ and $l_{max} = \lfloor 0.67 \times n \rceil$. This allows to cover all the candidates with lengths ranging from a quarter of the time series all the way up to two-thirds of the time series length.

One final aspect is that of the sampling ratio for the randomized shapelet discovery procedure. In this regard, all experiments based on the Random-Shapelets algorithm and the Random-Shapelets Ensembles and its variants were performed with a 1% sampling ratio.

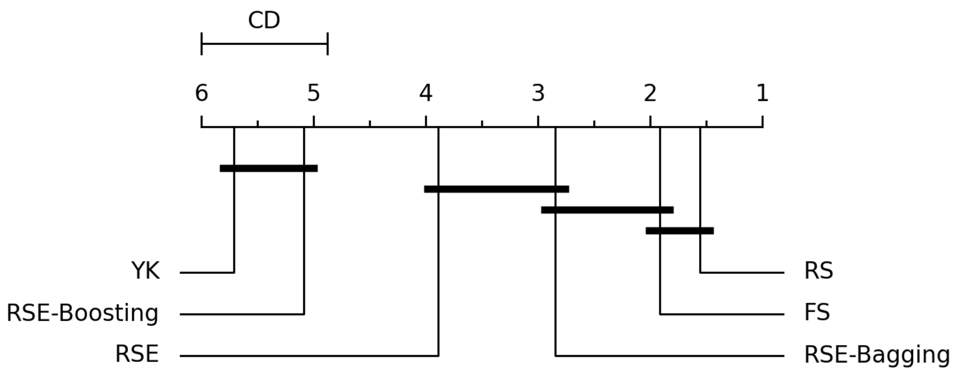
4.4 Results

Figures 4.2a and 4.2b show the critical differences diagrams [103] based on the classification accuracy and runtime performance of the different algorithms, respectively. The RSE variants gain a clear advantage regarding classification accuracy by significantly outperforming the other algorithms, although the individual models in the ensembles were created using only 1% of all the possible shapelet candidates. The average ranks of the RSE variants also indicate that, compared to each other, the classification accuracy provided by either ensemble approach does not differ by much. It was also observed that the classification accuracy improved by as much as 20% for a number of datasets when using the RSE variants compared to the classification accuracy obtained using the YK-shapelets algorithm. The YK-shapelets algorithm achieves the fourth position according to the classification accuracy-based average ranks, and is significantly different than either the best and the worst performing groups of algorithms. The Random-Shapelets algorithm was slightly better than the Fast-Shapelets algorithm, but both these algorithms were placed together in the group of the worst performing algorithms regarding classification accuracy.

In terms of the runtime performance, the Fast-Shapelets and Random-Shapelets algorithms are the fastest with the first and second average ranks, respectively, while the YK-shapelets algorithm is the slowest coming in at last place. RSE-Bagging, RSE, and RSE-boosting take the third, fourth, and fifth places, respectively. In this regard, there is no clear separation between



(a) Average ranks for different algorithms based on classification accuracy.



(b) Average ranks for different algorithms based on runtime performance.

Fig. 4.2. Average ranks for YK-shapelets (YK), Fast-Shapelets (FS), Random-Shapelets (RS), and variants of the Random-Shapelets Ensembles (Simple Combination: RSE, Bagging: RSE-Bagging, and Boosting: RSE-Boosting). Groups of classifiers not significantly different at $p = 0.05$ are connected. The critical difference is 1.124.

the different algorithms as was seen regarding the classification accuracy results. RSE-Bagging is the fastest RSE variant due to the duplicated time series instances in the bootstrap samples of the data that are created for each individual model in the ensemble. The duplicated instances can be separated very early because it is much easier for the shapelet discovery process to identify a shapelet specific to the duplicated instances. RSE-Bagging also manages to gain further speed-ups as a result of the candidate pruning strategy based on the optimistic Information Gain, etc. (see Section 2.4.2). Finally, the exclusion of multiple instances very early in the decision tree induction process also reduces the required computational costs. RSE-Boosting, however, is surprisingly not significantly faster than the YK-shapelets algorithm, which is shown by the clique connecting the average ranks of the two algorithms. The

RSE-Boosting approach performs slower because weighting the individual instances increases the computation required for performing data splits, and hence, candidate pruning. Since the candidate pruning strategy creates optimistic splits in each call, this becomes a limiting factor for the RSE-Boosting approach.

RSE provides almost the same average rank as RSE-Boosting and RSE-Bagging when comparing the different algorithms in terms of classification accuracy, while it also provides a balanced performance regarding the runtime requirements, therefore, the following comparison is based on RSE against the other algorithms. The aim of the following discussion is to provide a unified and head-to-head comparison of the classification accuracy and runtime performance of the different algorithms. Figures 4.3, 4.4, and 4.5 present the comparisons in terms of classification accuracy and runtime performance of Random-Shapelets Ensembles against YK-shapelets, Fast-Shapelets, and Random-Shapelets, respectively. Each evaluated dataset is represented by a marker. The relationship between the runtimes of the compared algorithms is shown using the position of the markers, while the color and shape of the markers show the relationship between the classification accuracy results. The x- and y-axes show the runtime required for training a classification model for RSE and the other algorithm, respectively. Both axes are based on a log scale to clearly establish the performance gains in terms of orders of magnitude. The figures also have a boxed background to emphasize the scale of difference between the runtimes of the compared algorithms. A dotted line, drawn for $y = x$, divides the plot area to indicate which algorithm requires less time for evaluating a specific dataset compared to the other. A marker on the dotted line indicates that both algorithms have the same runtime requirements, while a marker which resides off the dotted line indicates otherwise. Markers above the dotted line show that RSE is faster, while markers below the dotted line indicate otherwise. Upward facing green colored markers indicate that RSE provides better accuracy, whereas downward facing red colored markers indicate otherwise. Marker size corresponds to the absolute difference between the classification accuracy values of the compared algorithms, i.e., larger markers indicate a greater difference between the accuracy of the two algorithms. We can see that RSE provides an improvement of almost an order of magnitude with respect to runtime when compared against YK-shapelets, while it is almost an order of magnitude slower than Fast-Shapelets, and Random-Shapelets. Since the randomized shapelet discovery process evaluated 1% of all the shapelet candidates, while ten individual models are used for each ensemble, therefore, the overall performance gains compared to the YK-shapelets algorithm are consistent. When compared to Random-Shapelets, we see that RSE has to induce ten models similar to Random-Shapelets, therefore, RSE is an order of magnitude slower. In terms of classification accuracy, RSE is clearly dominating all the other algorithms.

4.5 Conclusion

The use of an inexpensive but reasonably accurate base learner for creating ensembles of shapelet based classifiers proved to be highly effective. The benefits of the proposed approach include better classification accuracy for almost all the evaluated datasets and reduced computational costs compared to the exact YK-shapelets algorithm. The proposed approach can also be used in a contractual setup where individual models can be added to the ensemble until the required classification accuracy is not achieved, or we do not run out of the computational quota.

There are a number of possible variations that can be employed for a Random-Shapelets based ensembling approach. The sampling ratio for individual ensemble members can be varied, or the individual models can be trained using different shapelet candidate lengths. The simplicity and added benefits of the approach make it very suitable for shapelet discovery and classification. Using RSE-Bagging can reduce the required computation, however, in some cases the classification accuracy of the obtained model is slightly worse than the RSE classifiers trained on the original training dataset, albeit not significantly. Ensembles of Random-Shapelets classifiers provide higher accuracy and therefore, can be used as an alternative to the YK-Shapelets algorithm to improve the overall classification accuracy.

Currently, the Random-Shapelets algorithm can only evaluate candidates with a sampling ratio set at the start of the process. The possibility of changing the fraction of evaluated candidates and use the results in an additive fashion to the already obtained results could prove beneficial. This would require

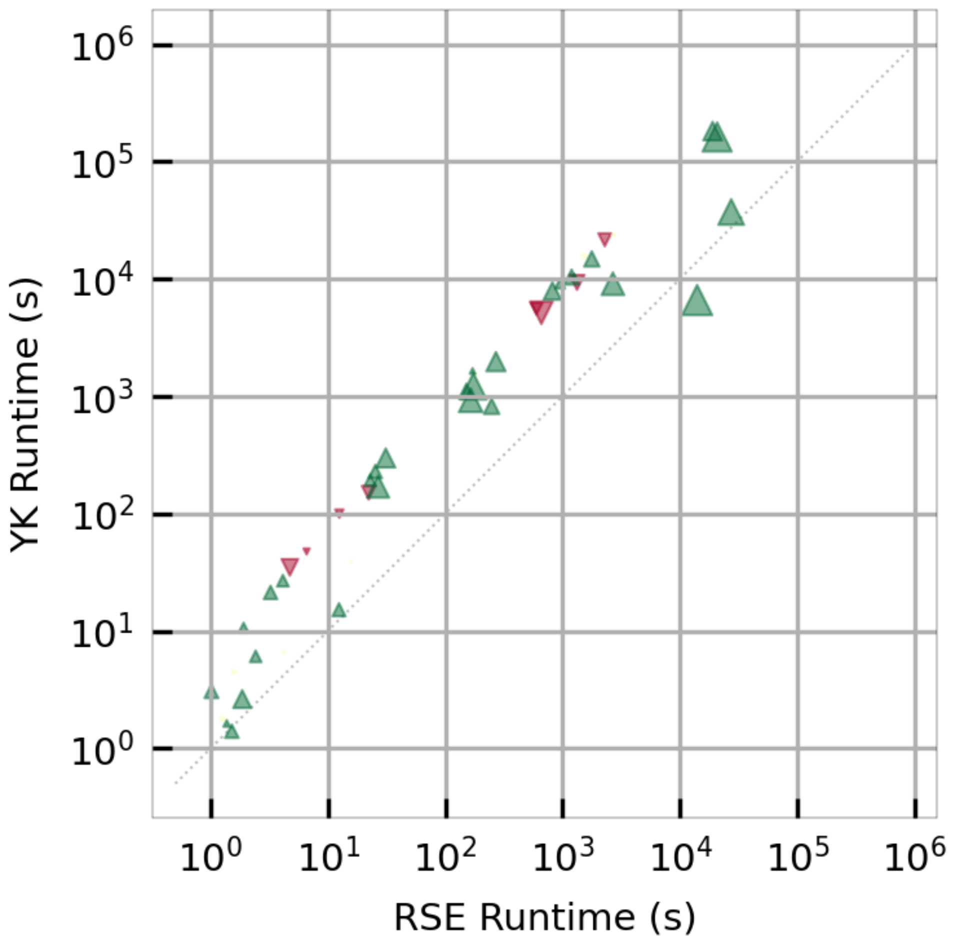


Fig. 4.3. Comparing Random-Shapelets Ensembles against YK-shapelets regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the runtime requirements (for training) in seconds. Markers above the dotted line indicate that RSE is faster than the other algorithm. A green marker indicates that RSE provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that the difference was greater than 2% in favor of RSE, while a downward facing triangle shows that the difference was greater than 2% in favor of the other algorithm. Bubbles indicate that the difference in classification accuracy was only 1%.

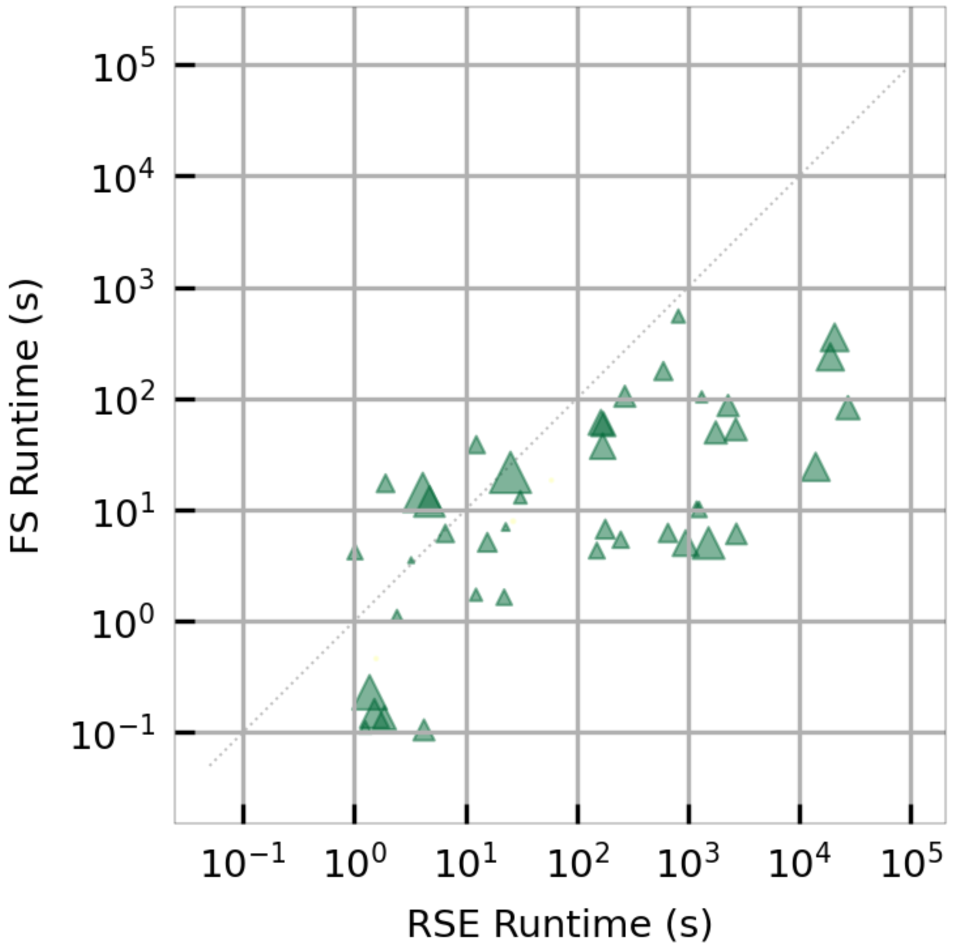


Fig. 4.4. Comparing Random-Shapelets Ensembles against Fast-Shapelets regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the runtime requirements (for training) in seconds. Markers above the dotted line indicate that RSE is faster than the other algorithm. A green marker indicates that RSE provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that the difference was greater than 2% in favor of RSE, while a downward facing triangle shows that the difference was greater than 2% in favor of the other algorithm. Bubbles indicate that the difference in classification accuracy was only 1%.

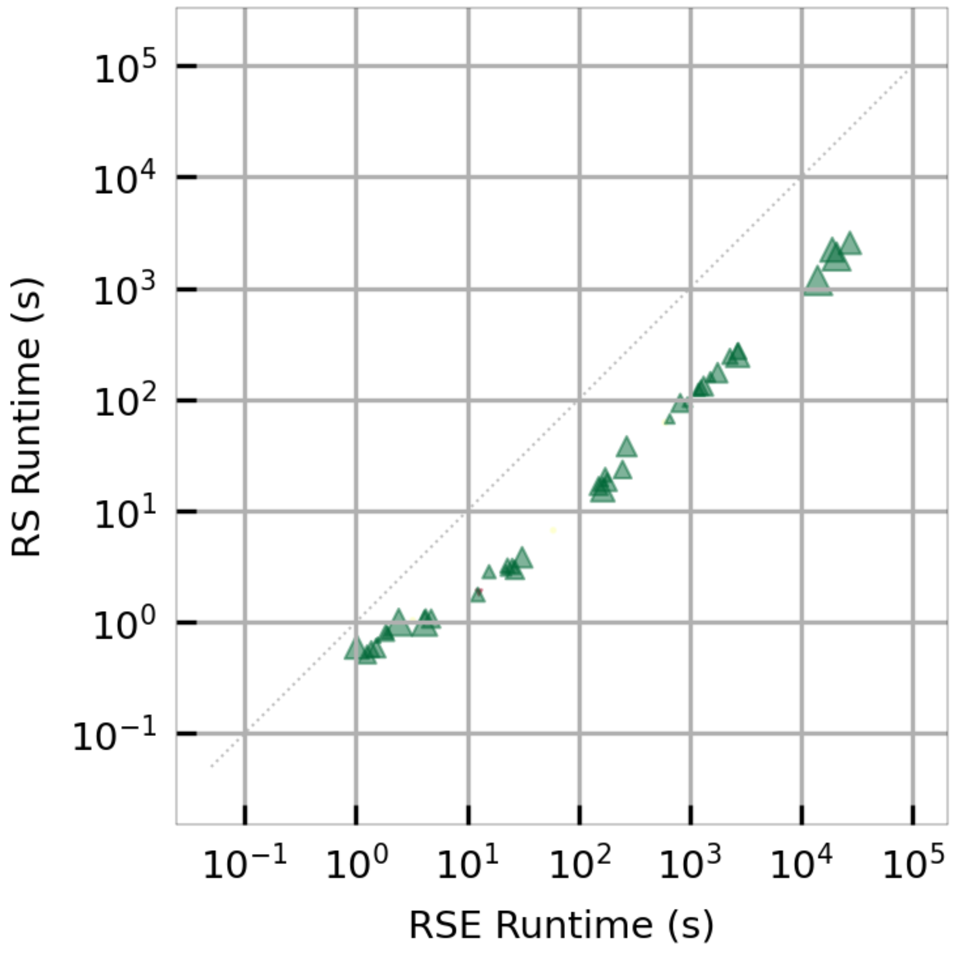


Fig. 4.5. Comparing Random-Shapelets Ensembles against Random-Shapelets regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the runtime requirements (for training) in seconds. Markers above the dotted line indicate that RSE is faster than the other algorithm. A green marker indicates that RSE provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that the difference was greater than 2% in favor of RSE, while a downward facing triangle shows that the difference was greater than 2% in favor of the other algorithm. Bubbles indicate that the difference in classification accuracy was only 1%.

some book keeping about the already evaluated candidates and the obtained results, but if the storage requirements can be kept low, this could turn out to be a refinement step for an approximate solution. Another future research avenue could be the use of Random-Shapelets based classification models trained using randomly chosen window length parameters and combining the models in an ensemble. This should, theoretically, enhance the diversity of the individual models and also remove the need to perform parameter tuning before model generation.

The Fast-Shapelets algorithm is also a heuristic method and can be used as a base learner in the ensemble learning approach, however, the Fast-Shapelets algorithm does not provide much diversity in the models, which makes its use in ensembles less effective than the Random-Shapelets algorithm.

5 Leveraging String Mining for Shapelet Discovery for Randomized Learning Schemes

5.1 Background and Motivation

In order to scale up pattern-based time series classification to massive amounts of data, the algorithmic complexity has to be reduced to linear time at most. One way of addressing this issue is by transforming the time series to symbolic (i.e. string) representations. Although the time series community has recognized and acknowledged the benefits of transforming time series data to strings by discretization and quantization for some time [34–38], the approaches are still suffering from high computational complexity: the complexity of the Fast Shapelets approach is $O(Nn^2)$ [34], the one of BoP (Bag of Patterns) is $O(Nn^3)$ [35], the one of SAX-VSM (Symbolic Aggregate approXimation - Vector Space Model) $O(Nn^3)$ [36], the one of BOSS (Bag of SFA Symbols) $O(N^2n^2)$ [37], and the complexity of BOSS VS (Bag of SFA Symbols in Vector Space) is $O(Nn^{\frac{3}{2}})$ [38]. Overall, it seems that the opportunities arising from the positive results from the string mining literature have not yet been fully realized. Notably, string mining was the first and still remains, the only area of pattern mining, where, in terms of time complexity, optimal results can be guaranteed. Therefore, modern string mining algorithms can extract almost arbitrary frequency-related string patterns in linear time. The first algorithms in this line of research were proposed by Fischer, Heun, and Kramer in 2005 and 2006 [49, 50]. Further research has optimized the practical running times and theoretical properties of the aforementioned linear time string mining algorithms [51].

Transforming numeric time series to symbolic sequences can, of course, lead to a loss of information, and it is evident that using one such transformation might lead to incorrect or suboptimal results due to badly chosen or just nearly missed interval boundaries. However, we argue that the advantage in terms of time complexity (linear vs. higher-order polynomials) enables exploring the space of transformations much more efficiently and effectively than with any other more costly transformation: It becomes feasible to explore a multitude of parameterizations, for instance, multiple alphabet sizes and discretization schemes, given that the basic underlying algorithm has linear time complexity. Moreover, since classification is statistical and not a "precise science" anyway, the imperfection of any well-chosen string transformation does not harm and may even improve the results in noisy application domains.

5.2 String Mining – A Primer

String mining is concerned with the discovery of statistically relevant substrings from data sequences, which are characteristic of the given string database [6]. For a binary class problem, the patterns of interest could be the ones which are frequent in one class and infrequent in the other, hence the extracted patterns could help in distinguishing between the two classes, or they could be patterns which are common in the two classes, in which case the patterns could identify similarities between the two classes, etc.

Given a pair of symbolic datasets $\mathcal{D}^+$ and $\mathcal{D}^-$, each representing a positive and a negative class respectively, extracting patterns which can discriminate between the two datasets is referred to as the Emerging Substrings Mining problem. Formally, the problem of emerging substrings mining is to report all patterns in $\mathcal{D}^+$ and $\mathcal{D}^-$ such that each reported pattern occurs in at least f^+ different strings in $\mathcal{D}^+$, but does not occur in more than f^- different strings in $\mathcal{D}^-$, where f^+ and f^- are the relative support values of the pattern in the respective datasets.

During the last couple of decades, concerted research efforts in the fields of bioinformatics and natural language processing have led to the development of several string mining algorithms for extracting frequent patterns from symbolic datasets. The first time-optimal algorithms for string mining were proposed by Fischer, Heun, and Kramer [49, 50] and required $O(m)$ time and $O(m\ log\ m)$ bits of space for extracting frequent patterns from symbolic datasets, where $m = \sum_{i=1}^{|\mathcal{M}|} |s_i|$ is the total length of all strings in the dataset concatenated and $|\mathcal{M}|$ is the number of strings in the database. The authors showed how suffix arrays and longest common prefix (lcp) tables could be used to efficiently solve string mining problems under frequency constraints. Later, an improved version of the algorithm was proposed by Dhaliwal, Puglisi, and Turpin, which constructs the internal data structures in fixed size blocks instead of constructing them all at once, which allows to reuse memory and reduce the overall space requirements to only $O(m\ log\ \alpha)$ bits, while increasing the worst case computational complexity to only $O(m\ log^2\ m)$, where α is the size of the alphabet used [51].

Let $x = aaba\#_1^1abaaab\#_{|\mathcal{D}^+|}^1bbabb\#_1^2abba\#_{|\mathcal{D}^-|}^2$ be a string of length q made up of all the strings in $\mathcal{D}^+ = \{aaba, abaaab\}$ and $\mathcal{D}^- = \{bbabb, abba\}$ concatenated. The $\#_s^r$ symbols mark the end of individual strings and have the lexicographic property that $\#_1^1 < \ldots < \#_{|\mathcal{D}^+|}^1 < \#_1^2 < \ldots < \#_{|\mathcal{D}^-|}^2$. A substring of x is denoted by $x[i, j]$, where i and j represent the starting and ending positions of the substring. A suffix array (SA) is an array of integers which is used to describe the lexicographic order of all suffixes of x, such that $x[SA[k], q] < x[SA[k + 1], q]$ for all $1 \leq k < q$. The lcp array (LCP) contains the lengths of the longest common prefixes of x's suffixes that are consecutive in lexicographic order, and is defined as $LCP[i] = lcp(x[SA[i], q], x[SA[i - 1], q])$ for all $1 < i \leq q$, and $LCP[1] = 0$. The algorithm starts with the construction of the suffix array and the lcp array. Both of these data structures can be constructed in time linear in the length of x. Once the suffix array and the lcp array have been created, the string mining algorithm processes the lcp array to answer any "range minimum queries" (RMQs) in constant time. Formally, for any two indices i and j, the query $RMQ_{LCP}(i, j)$ asks for the position of the minimum element in $LCP[i, j]$, i.e., $RMQ_{LCP}(i, j) := argmin_{k \in \{i,\ldots,j\}}\{LCP[k]\}$. If the minimum value is not unique, then the smallest index is returned. Finally, the algorithm calculates correction terms for establishing the occurrence frequency of the different substrings based on the lcp array values. Table 5.1 shows the suffix array and the lcp array for the string x being used as in our example. For further details of the algorithm, we refer the interested reader to the respective papers [49–51].

5.3 Design

Mining Strings for Time Series Classification (*MiSTiCl*) is a subsequences based time series classification algorithm which employs string mining for efficient extraction of discriminative subsequences from the time series data. The subsequences are used as features to create a transformed dataset similar to the shapelet transform approach. The main steps involved are: (i) time series discretization, (ii) frequent pattern extraction, (iii) determining independent and highly discriminative frequent patterns, (iv) creating a transformed dataset using the best *K* frequent patterns, and finally (v) model induction. Looking at steps (ii) and (iii) in a bit more detail, we aim for patterns that are frequent enough in the positive class and not too frequent in the negative class. From the patterns that pass this filter, we choose the most discriminative ones, and from those with the same discriminative power, we choose the most general ones. Step (ii) makes sure that the patterns are statistically meaningful in the first place. Step (iii) picks from the remaining those that are predictive individually. To reduce their number, the most general ones are actually used in case of equal discriminative power, to obtain high coverage on unseen cases.

Algorithms based on discretized time series data are faced with a significant challenge regarding the information and feature loss due to discretization, therefore, we would like to address a crucial design aspect before going into the details of our proposed algorithm. Although SAX can preserve the overall shape of the time series instances, using a predefined window size to reduce the dimensionality of data can still lead to undesired feature loss due to inadvertent splitting of important features. A possible approach to counter this problem is to use those values of the parameters α and ω which result in minimum information loss. Usually, a brute force parameter tuning approach

Tab. 5.1. The suffix array for x and its lcp table. For each index position i, the string $x[SA[i], q]$ is shown until reaching the first end-of-string marker. The example has been taken from a previous publication [50, p. 6].

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
x	a	a	b	a	$\#^1_1$	a	b	a	a	a	b	$\#^1_2$	b	b	a	b	b	$\#^2_1$	a	b	b	a	$\#^2_2$
SA	5	12	18	23	4	22	8	9	1	10	2	6	15	19	11	17	3	21	7	14	16	20	13
LCP	0	0	0	0	0	1	1	2	3	1	2	3	2	3	0	1	1	2	2	2	1	2	3
	$\#^1_1$	$\#^1_2$	$\#^2_1$	$\#^2_2$	a	a	a	a	a	a	a	a	a	a	b	b	b	b	b	b	b	b	b
					$\#^1_1$	$\#^2_2$	a	a	a	b	b	b	b	b	$\#^1_2$	$\#^2_1$	a	a	a	a	b	b	b
							a	b	b	$\#^1_2$	a	a	b	b			$\#^1_1$	$\#^2_2$	a	b	$\#^2_1$	a	a
							b	$\#^1_2$	a		$\#^1_1$	a	$\#^2_1$	a					a	b		$\#^2_2$	b
							$\#^1_2$		$\#^1_1$			a		$\#^2_2$					b	$\#^2_1$			b
												b							$\#^2_1$				$\#^1_2$
												$\#^1_2$											

is used to come up with such parameters. This can incur a significant up-front computational cost depending on the number of evaluated parameter combinations and the size of the dataset split used to perform the search, still, the research community has generally opted for this approach. There is, however, another approach, which can be beneficial in two ways, specifically (i) improved model accuracy, and (ii) reduced wasted computation. Instead of using a single discretized version of the data created with optimized parameters, creating multiple discretized versions of the data using a number of arbitrary α and ω parameters allows to extract discriminative patterns from a multi-resolution perspective and helps to capture different features at each resolution. Using this approach, a feature-based dataset can be created with multi-cardinality and multi-dimensionality properties, which allows to incorporate a diverse set of features in a single feature set that can be exploited by complex classification algorithms to provide better generalization and improved classification accuracy. Moreover, using a linear time string mining algorithm for feature extraction from multiple discretized versions of the data only increases the computational complexity by a constant factor. Therefore, we have opted for the latter approach to tackle the challenge mentioned at the beginning of this paragraph.

5.3.1 Overview

MiSTiCl is a powerful and extremely efficient time series classification algorithm which brings together highly effective approaches from the string mining and time series mining domains. This subsection provides a general overview of the algorithm, while the following subsections provide details of the individual steps. Algorithm 5.1 lists the steps involved in the creation of feature based dataset splits using MiSTiCl. The input parameters include the real-valued training and testing splits (D_{train} and D_{test}), a set of alphabet and window sizes (A and Ω), the minimum positive and maximum negative frequency for the extracted frequent patterns (f^+ and f^-), and a parameter K to limit the number of features used per class. The procedure starts with the initialization of associative arrays for the standalone feature sets which will be merged later to form the multi-resolution feature sets (Line 5.1-1). Next, the class labels are enumerated from the training split (Line 5.1-2). Next,

Algorithm 5.1 MiSTiCl (D_{train}, D_{test}, A, Ω, f^+, f^-, K)

```
1:  $FS_{train} \leftarrow \{\}$, $FS_{test} \leftarrow \{\}$        ▷ Init. map objects for single-resolution feature sets
2:  $C \leftarrow$ EXTRACTCLASSLABELS($D_{train}$)
3:  for all $(\alpha, \omega) \in A \times \Omega$ do
4:      $\hat{D}_{train}, \hat{D}_{test} \leftarrow$ DISCRETIZE($D_{train}$, $D_{test}$, $\alpha$, $\omega$)
5:      $FP \leftarrow$ GETDISCRIMINATIVEFREQUENTPATTERNS($\hat{D}_{train}$, $f^+$, $f^-$, $C$)
6:      $FS^{\alpha,\omega}_{train}, FS^{\alpha,\omega}_{test} \leftarrow$ CREATEFEATURESETS($D_{train}$, $D_{test}$, $\hat{D}_{train}$, $FP$, $K$)
7:  end for
8:  $ParamCombo \leftarrow$ FINDBESTPARAMCOMBO($FS_{train}$, $FS_{test}$, $A$, $\Omega$, $|D_{train}|$)

9:  $MultiResFS_{train} \leftarrow \phi$, $MultiResFS_{test} \leftarrow \phi$        ▷ Initialize multi-resolution feature sets
10: for all $(\alpha, \omega) \in ParamCombo$ do
11:     $MultiResFS_{train} \leftarrow MultiResFS_{train} \; || \; FS^{\alpha,\omega}_{train}$
12:     $MultiResFS_{test} \leftarrow MultiResFS_{test} \; || \; FS^{\alpha,\omega}_{test}$
13: end for
14: return $MultiResFS_{train}, MultiResFS_{test}$
```

feature sets corresponding to each $(\alpha, \omega) \in A \times \Omega$ parameter combination are created (Lines 5.1-3 to 5.1-7). The real-valued training and testing splits are discretized corresponding to the current α and ω parameters (Line 5.1-4). Next, frequent patterns are extracted from the discretized training split (Line 5.1-5). Now, training and testing feature sets are created using the top K frequent patterns for the current α and ω parameters (Line 5.1-6). Once the individual feature sets have been created for all $(\alpha, \omega) \in A \times \Omega$, the procedure to determine the best parameter combination is called, which returns a list of (α, ω) parameters (Line 5.1-8). Finally, the training and testing feature sets corresponding to the (α, ω) pairs in the list of best parameter combination are concatenated horizontally to create multi-resolution training and testing feature sets, respectively (Lines 5.1-10 to 5.1-13). The multi-resolution feature sets transform the time series classification problem into a feature-based classification problem, which allows to employ any off-the-shelf classification algorithm for model induction.

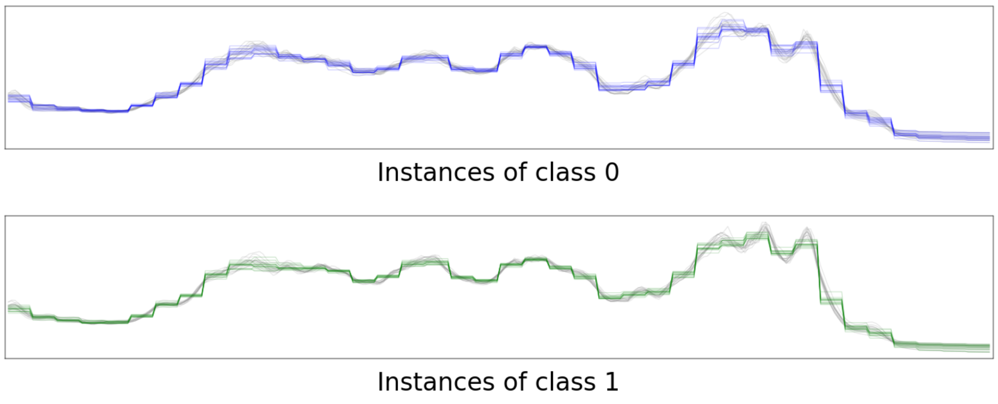


Fig. 5.1. PAA version of time series instances superimposed on their real-valued counterparts. The original time series (shown in light grey) are 286 time points long and have been reduced to 40 time points based on a dimensionality reduction factor $\omega = 7$. The PAA versions have been stretched (along the x-axis) to emphasize the retention of the overall shape of time series instances.

5.3.2 Time Series Discretization

Symbolic aggregate approximation (SAX) is a widely used time series discretization algorithm [9, 10]. It combines multiple time series observations into single averaged values to reduce the dimensionality and then maps these averaged values to characters from an alphabet to reduce the cardinality. A time series T of length n can be converted to a symbolic string $\hat{T} = \left(\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_p\right)$ of length $p = \left\lfloor \frac{n}{\omega} \right\rfloor$ such that $p \ll n$, where $\omega \in \mathbb{Z}_{\geq 1}$ represents the dimensionality reduction factor (or the averaging window size). The presented mathematical notation is for the simple case of integer values of ω, later SAX refinements can use non-integer window sizes as well. First, T is converted to a reduced dimensionality version $\bar{T} = \left(\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_p\right)$ of length p such that each non overlapping sequence of ω observations of T is averaged to provide one observation. $\bar{T}$ is also referred to as a Piecewise Aggregate Approximation (PAA). Mathematically, we can get each $\bar{t}_i$ using the equation:

$$\bar{t}_i = \frac{\omega}{n} \sum_{j=\frac{n}{\omega}(i-1)+1}^{\frac{n}{\omega}i} t_j$$

Figure 5.1 shows the reduced dimensionality version of time series instances superimposed on their real-valued counterparts.

For cardinality reduction, each observation $\bar{t}_i \in \bar{T}$ is mapped to a character from an alphabet of size $\alpha \in \mathbb{Z}_{\geq 2}$. A small value of α leads to large quantization blocks and vice versa. The quantization blocks for each character in the alphabet are chosen based on breakpoints $B = (\beta_0, \beta_1, \ldots, \beta_\alpha)$, where β_0 and β_α are defined as $-\infty$ and ∞, respectively, while the other breakpoints are chosen such that area under the $N(\mu = 0, \sigma^2 = 1)$ curve from β_i to β_{i+1} equals $\frac{1}{\alpha}$ ensuring an equiprobable selection of every character in the alphabet. Table 5.2 shows the breakpoints for different values of α. A Gaussian curve with $\mu = 0$ and $\sigma^2 = 1$ is used because z-normalized time series instances have the same mean and standard deviation [10, p. 8]. For the minority of datasets which are not Normally distributed, selecting the breakpoints using the Gaussian curve can deteriorate the efficiency of SAX, however, the "correctness of the algorithm is unaffected" [10]. All observations $\bar{t}_i \in \bar{T}$, which have their values in the range $[\beta_0, \beta_1)$, are mapped to the first character in the alphabet. Similarly, all observations $\bar{t}_i \in \bar{T}$ with values in the range $[\beta_1, \beta_2)$ are mapped to the second character in the alphabet, and so on. Mathematically, we get each $\hat{t}_i$ as follows:

$$\hat{t}_i = alpha_j, \quad \text{iff} \quad \beta_{j-1} \leq \bar{t}_i < \beta_j$$

Continuing with the example time series instances shown in Figure 5.1 for illustrating the real-valued time series and their PAA version, Table 5.3 shows the symbolic representations obtained by mapping the PAA versions to the corresponding characters in the alphabet based on a cardinality reduction factor $\alpha = 6$. For a detailed analysis of the SAX algorithm and a review of its diverse applications, we refer the reader to the corresponding article [10].

5.3.3 Frequent Patterns Extraction

The next step in the MiSTiCl algorithm is to extract the frequent patterns from the discretized training split as detailed in Algorithm 5.2. The input parameters of the procedure include the discretized training split ($\hat{D}$), the minimum positive and maximum negative frequency for accepting candidate frequent patterns (f^+ and f^-), and the set of class labels in the training dataset split (C). The procedure returns an associative array (FP) indexed with the class labels and containing the most discriminative frequent patterns

Tab. 5.2. Breakpoints look-up table for dividing the $N(\mu = 0, \sigma^2 = 1)$ Gaussian curve into equiprobable regions for $\alpha = 2$ to 6.

	Cardinality level, α				
	2	3	4	5	6
β_0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
β_1	0.00	−0.43	−0.67	−0.84	−0.97
β_2	∞	0.43	0.00	−0.25	−0.43
β_3		∞	0.67	0.25	0.00
β_4			∞	0.84	0.43
β_5				∞	0.97
β_6					∞

Tab. 5.3. Symbolic representation of time series instances (shown in Figure 5.1) obtained using a dimensionality reduction factor $\omega = 7$ and a cardinality reduction factor $\alpha = 6$.

b b a a a b b c e e e e e e d d e e d d e f e e c c c e f f f f f c a a a a a a
b b b a a b b c d e e e e e d d e e d d e f e e c c c e f f f f f c a a a a a a
. . .
b b a a a b b c e e e e e e d e e e d d e e e d c c c e f f f f f c a a a a a a
b a a a a b b c e e e e e e d d e e d d e e e d c c c e f f f f f c a a a a a a
. . .

per class. The string mining algorithm extracts frequent patterns from binary class datasets, therefore, multi-class problems are transformed into multiple binary problems using a one-vs-all approach. The procedure initializes an empty associative array for the frequent patterns extracted per class (Line 5.2-1). For each class $c \in C$, a pair of datasets is created such that all instances with class label c are assigned to the positive class dataset $\hat{P}$, while all remaining instances are assigned to the negative class dataset $\hat{N}$ (Lines 5.2-3 and 5.2-4). The string mining algorithm extracts all those patterns from $\hat{P}$ and $\hat{N}$ which satisfy the f^+ and f^- frequency constraints, and provides a list of frequent patterns along with actual occurrence frequencies of each pattern in the $\hat{P}$ and $\hat{N}$ splits (Line 5.2-5). Next, the discriminative power of the frequent patterns for the current class c is determined, and the frequent patterns are placed on the map FP along with their discriminative power (Line 5.2-6). Once all the one-vs-all data splits have been processed, the map containing the discriminative frequent patterns is returned.

Algorithm 5.2 GetDiscriminativeFrequentPatterns ($\hat{D}$, f^+, f^-, C)

```
1: $FP \leftarrow \{\}$
2: for all $c \in C$ do
3:     $\hat{P} \leftarrow \{\hat{T} \mid \forall\, \hat{T} \in \hat{D},\ \hat{T}.y = c\}$
4:     $\hat{N} \leftarrow \{\hat{T} \mid \forall\, \hat{T} \in \hat{D},\ \hat{T}.y \neq c\}$
5:     $Z \leftarrow$ExtractAllFrequentPatterns($\hat{P}$, $\hat{N}$, $f^+$, $f^-$)
6:     $FP[c] \leftarrow$ SelectDiscriminativePatterns($Z$, $|\hat{P}|$, $|\hat{N}|$)
7: end for
8: return $FP$
```

5.3.4 Selecting Discriminative Patterns

The discriminative power of a given pattern regarding correctly identifying a specific class of instances can be assessed using the χ^2 independence test, or Information Gain among other methods. The occurrence frequency of a pattern in the positive and negative class datasets can be used to create a confusion matrix for calculating either the χ^2 independence test or the Information Gain value. The χ^2 test assesses whether the observations, expressed as a contingency table, are statistically independent of each other. A higher value of the test statistic indicates a greater level of independence and vice versa. Information Gain measures the difference between two probability distributions and can be used to assess the association between features. For a binary problem, the Information Gain value of one indicates perfect class purity, while a value of zero indicates the opposite.

Setting the frequency constraints such that $0 \lessapprox f^- < f^+ \leq 1$ allows to extract frequent patterns which are maximally representative of the positive class. The strictness of frequency constraints directly affects the number of frequent patterns extracted by the string mining algorithm. A very small value for the f^+ constraint can result in the extraction of a huge number of frequent patterns, whereas a large value can result in no frequent patterns being extracted at all. The huge number of frequent patterns extracted with a lower f^+ constraint is attributed to the presence of various prefix and/or suffix based variants of base patterns (see Table 5.4 for examples). The occurrence frequency of a base pattern is always greater than or equal to the

Tab. 5.4. Examples of base and variant patterns.

Base	**Variants**
dccb	dccb*b*
	*a*dccb
bdcd	*a*bdcd*cc*
	*a*bdcd*bd*

occurrence frequency of its variants. In case a variant pattern has the same occurrence frequency as its base pattern, the independence tests rank the two patterns equally, however, duplicate detection can be used to discard the variant pattern. Hence, using reasonably lenient frequency constraints and ranking the extracted patterns using an independence test followed by filtering for duplicates can provide independent, highly discriminative, and diverse frequent patterns.[1][2] Algorithm 5.3 lists the pseudocode for selecting such frequent patterns. The procedure receives a list containing all the extracted frequent patterns along with their corresponding occurrence frequencies (Z), and the instance counts for the positive and negative dataset splits ($N_{\hat{P}}$ and $N_{\hat{N}}$). The output of the procedure is an ordered associative array of lists containing the filtered frequent patterns. The associative array is ordered in decreasing order of the independence test statistic, where each slot in the associative array contains a list of all the frequent patterns which have the same value of the test statistic. After initializing the array, the procedure iterates over all the patterns provided as input (Lines 5.3-2 to 5.3-8). An independence test is used to obtain a value for the test statistic for the current pattern (Line 5.3-3).[3] Next, the list of patterns corresponding to the current test statistic is retrieved from the array OL, and if no such list exists, one is initialized and placed in the location pointed to by the test statistic (Line 5.3-5). Next, the current pattern f is compared against all the patterns in $listOfFPs$ to see whether the list already contains its base pattern

[1]This criterion and procedure is not to be confused with *closed* or *open/free* patterns [104].

[2]Note that there can be two patterns p and q, with one pattern p being more general than the other, $p \prec q$, both having the same value of χ^2 ($\chi^2(p) = \chi^2(q)$), but yet occurring in different sets of positive and negative examples. However, this should be expected to be a rather infrequent case. The overall filtering procedure of patterns just makes sure that the patterns are frequent enough in the positives, infrequent enough in the negatives, highly discriminative and, given the same discriminative power, as general as possible.

[3]A discussion about calculation of the χ^2 test statistic and the Information Gain is provided in Appendix A.

Algorithm 5.3 SelectDiscriminativePatterns (Z, $N_{\hat{P}}$, $N_{\hat{N}}$)

```
1: Initialize $OL \leftarrow \{\}$                    ▷ Initialize an ordered associative array
2: for all $z \in Z$ do
3:     $indVal \leftarrow$ GetIndependenceValue($z$, $N_{\hat{P}}$, $N_{\hat{N}}$)
4:     $f \leftarrow z.pattern$      ▷ $z$ contains the frequent pattern $z.pattern$, and its
                                     ▷ occurrence frequencies $z.f_{\hat{P}}$ and $z.f_{\hat{N}}$ in $\hat{P}$ and $\hat{N}$.
5:     Retrieve $listOfFPs$ corresponding to $indVal$,
           if none exists, add a new list to $OL$ at location $indVal$
6:     for all $p$ in $listOfFPs$,
           if $p$ is a substring of $f$ continue outer loop
7:     add $f$ to $listOfFPs$      ▷ Since $f$ has been determined to be unique
8: end for
9: return $OL$
```

(Line 5.3-6). If a base pattern is found, f is discarded, otherwise it is inserted in the list (Line 5.3-7). Once all the patterns have been evaluated, the ordered array of lists containing the filtered frequent patterns is returned.

5.3.5 Creating Feature Sets

Once the discriminative frequent patterns have been identified, the next step is the creation of feature-based datasets. One approach is to create real-valued datasets, such that real-valued subsequences corresponding to the symbolic frequent patterns are used as features in the transformed datasets, while the distance values between the said subsequences and the real-valued time series instances are used as feature values. Alternatively, binary-valued datasets can be created using the extracted frequent patterns as features and zeros/ones as feature values representing the absence/presence of a feature in the discretized instances. The latter approach is straightforward and highly efficient because the discretized time series data is already at hand and searching for the presence of frequent patterns in discretized time series instances also requires very little overhead. On the downside, preliminary experiments showed that classification models based on the binary-valued transformation almost always performed inferior to their real-valued counterparts. Algorithm 5.4 lists the pseudocode for creating real-valued feature sets. The input parameters include the real-valued training and testing splits D_{train} and D_{test}, the discretized training split $\hat{D}_{train}$, the set of class labels C, the parameter K for maximum number of patterns to use per class, and the associative array FP containing the discriminative frequent patterns filtered for duplicates and sorted in descending order of discriminative power. The procedure starts with initializing the training and testing feature based splits (Line 5.4-1). Next, the procedure iterates over all class labels $c \in C$ to add the top K features for each class (Lines 5.4-2 to 5.4-12). The loops in Lines 5.4-4 and 5.4-5 iterate over the frequent patterns for the current class c (in order of their independence test ranks). For each frequent pattern f, a reverse lookup is performed to extract the best corresponding subsequence from the real-valued training split (Line 5.4-6). Next, the procedure iterates over all instances of the training and testing splits to populate the respective feature set columns. Once the top K features have been added for the current class, the procedure breaks out of the feature adding loop and the process repeats itself for the next class.

5.3.6 Merging Feature Sets

MiSTiCl aims to mitigate the problem of feature loss by combining multiple single-resolution feature sets, each created with its own α and ω parameter, thus creating a multi-resolution feature set. A multi-resolution feature set is simply a horizontal concatenation of multiple single-resolution feature sets and requires zero computational overhead. Finding the best multi-resolution feature set, however, involves (i) creating all possible multi-resolution feature sets, (ii) model induction, and finally (iii) performance testing for each candidate feature set, all of which can incur a significant computational cost. A naïve approach would be to combine all available single-resolution feature sets, however, this could potentially introduce redundant features in the multi-resolution feature set. This presents an optimization problem where a minimum number of single-resolution feature sets should be combined to form the multi-resolution feature set which provides maximum classification accuracy. For a cardinality reduction factor set A and a dimensionality

Algorithm 5.4 CreateFeatureSets (D_{train}, D_{test}, $\hat{D}_{train}$, FP, C, K)

```
1:  $FS_{train} \leftarrow TABLE\ (|D_{train}|,\ K \times |C|)$,
    $FS_{test} \leftarrow TABLE\ (|D_{test}|,\ K \times |C|)$
2:  for all $c \in C$ do
3:      $k \leftarrow 0$
4:      for all $listOfFPs \in FP_c$ do                          ▷ Outer loop
5:          for all $f \in listOfFPs$ do                         ▷ Inner loop
6:              $s \leftarrow$PERFORMREVERSELOOKUP($D_{train}$, $\hat{D}_{train}$, $f$)
7:              For each $T \in D_{train}$
                    populate the respective row and column of $FS_{train}$
                    with the distance value between $T$ and $s$
8:              For each $T \in D_{test}$
                    populate the respective row and column of $FS_{test}$
                    with the distance value between $T$ and $s$
9:              Increment $k$ and if $k$ equals $K$ break Outer loop
10:         end for
11:     end for
12: end for
13: return $FS_{train}, FS_{test}$
```

reduction factor set Ω, the total number of single-resolution feature sets created by MiSTiCl equals $|A| \times |\Omega|$. Using the brute force approach to find the best multi-resolution feature set is the most computation intensive option because it would require creating and evaluating a staggering $\left(2^{|A| \times |\Omega|}\right) - 1$ candidate multi-resolution feature sets. We can think of our optimization problem as a set cover problem since we want to cover all training set instances (the universe, $\mathcal{U}$) with the minimum number of single-resolution feature sets (subset of a collection, S), where each single-resolution feature set covers some arbitrary training set instances. Formally, the set cover problem aims to identify the smallest subset of a collection S whose union equals the universe $\mathcal{U}$. It is a classical question in combinatorics, however, it is an NP-complete problem, which implies that we need a heuristic approach. The steps involved in finding the best combination of feature sets to combine are listed in Algorithm 5.5. First, the procedure creates a set of integers $\{1, 2, \ldots, N\}$, where each element of the set corresponds to the indices of the instances in the actual training set split. This set of indices is split into two disjoint sets (Trn and Val), which will be used as the training and validation sets during the parameter optimization process (Line 1). Next, the procedure initializes a list $ParamsList$, and a multimap (dual-key associative array) S. The $ParamsList$ is used to keep track of the number of instances correctly classified by a feature set along with the α and ω parameters while the multimap S is used to keep track of the specific instances correctly classified by a feature set based on the α and ω parameters. The procedure iterates over all the $(\alpha, \omega) \in A \times \Omega$ parameter combinations (Lines 5.5-3 to 5.5-8) and performs the following steps for each single-resolution feature set $FS^{\alpha,\omega}_{train}$.

- Train a classification model M using those instances from $FS^{\alpha,\omega}_{train}$ whose indices occur in the Trn set.
- Test the classification model M using those instances of $FS^{\alpha,\omega}_{train}$ whose indices occur in the Val set.
- Save the indices of all correctly classified instances in the Val set into the multimap S using α and ω.
- Add the number of correctly classified Val instances and the current α and ω parameters to the $ParamsList$.

Next, an empty set is initialized for the best found parameters $BestParams$. The procedure iterates over the associative array containing the $ParamsList$ (Lines 5.5-11 to 5.5-19). If the difference between the Cal set and the set $S^{\alpha,\omega}$ (correctly classified Val instances by the $FS^{\alpha,\omega}_{train}$) is not empty, then (i) the current (α, ω) pair is added to the set of best parameters, and (ii) the Val set is updated with the still uncovered indices. When the Val set gets empty, i.e., all validation instances have been covered, or the number of parameter pairs in the $BestParams$ set is equal to some preset limit, the loop is terminated. Finally, the $BestParams$ set is returned.

5.3.7 Complexity Analysis

The asymptotic time complexity of MiSTiCl can be calculated by aggregating the time complexities of the individual steps. MiSTiCl creates multiple single-resolution feature sets and the creation of each feature set contributes equally

Algorithm 5.5 FindBestParamCombo ($FS_{train},\ FS_{test},\ A,\ \Omega,\ N$)

```
1:  $Trn, Val \leftarrow$ RandomSplit($\{1, 2, \dots, N\}$)
2:  $ParamsList \leftarrow [\,]$,
    $S \leftarrow TABLE\ (|A|,\ |\Omega|)$
3:  for all $(\alpha, \omega) \in A \times \Omega$ do
4:      Train $M$ using $\{in_j | j \in Trn \wedge in_j \in FS^{\alpha,\omega}_{train}\}$
5:      Test $M$ using $\{in_k | k \in Val \wedge in_k \in FS^{\alpha,\omega}_{train}\}$
6:      Save indices of correctly classified validation instances in $S^{\alpha,\omega}$
7:      Add $(|S^{\alpha,\omega}|, (\alpha, \omega))$ to $ParamsList$
8:  end for
9:  Sort $ParamsList$ in descending order
10: $BestParams \leftarrow \{\}$
11: for all $(|S^{\alpha,\omega}|, (\alpha, \omega)) \in ParamsList$ do
12:     if $Val \setminus S^{\alpha,\omega} \neq \emptyset$ then
13:         $BestParams \leftarrow BestParams \cup \{(\alpha, \omega)\}$
14:         $Val \leftarrow Val \setminus S^{\alpha,\omega}$
15:         if $Val = \emptyset$ or $|BestParams| \geq \frac{|A| \times |\Omega|}{3}$ then
16:             break
17:         end if
18:     end if
19: end for
20: return $BestParams$
```

towards the overall complexity, therefore, we shall first consider a single iteration for an arbitrary combination of α and ω. The time taken by SAX to discretize a dataset is on the order of $O(Nn)$. The time taken by the string mining algorithm for frequent pattern extraction is on the order of $O(m)$. Rewriting m in terms of N and n gives $m = N\frac{n}{\omega}$, therefore, the time complexity of string mining is on the order of $O(N\frac{n}{\omega})$. To calculate the time complexity of filtering the frequent patterns for obtaining highly independent patterns, we first need to approximate the number of extracted frequent patterns. For a given α and ω, the number of all possible frequent patterns of all lengths is approximately equal to $\alpha^2 + \alpha^3 + \cdots + \alpha^p$, which can be simplified using a geometric progression as $\frac{1-\alpha^{(p+1)}}{1-\alpha}$, where $p = \left\lfloor \frac{n}{\omega} \right\rfloor$. Since all the terms in this expression are constant, we can approximate the time complexity of filtering the frequent patterns to be constant. Creating a feature set requires adding K features for each class $c \in C$. For each feature, N feature values have to be calculated, where each feature value calculation takes $O(ns)$, where s is the length of a subsequence and $s \ll n$. Since K and $|C|$ are constant and much smaller than N and n, the time required for creating a feature set is on the order of $O(Nns)$. Aggregating the time complexity of the individual steps, the overall time complexity of creating a feature set for a given α and ω parameter combination is on the order of $O(Nn) + O(N\frac{n}{\omega}) + O(1) + O(Nns) \approx O(Nns)$. Creating $|A| \times |\Omega|$ many feature sets multiplies the complexity by a factor of $|A| \times |\Omega|$, therefore, the asymptotic time complexity of the MiSTiCl algorithm is on the order of $O(Nns)$. Comparing the time complexity of MiSTiCl with the time complexity of some other state-of-the-art symbolic time series algorithms indicates that MiSTiCl is the fastest symbolic representation based time series classification algorithm. The reported time complexity for the BoP, SAX-VSM, BOSS and BOSS VS algorithms is on the order of $O(Nn^3)$, $O(Nn^3)$, $O(N^2n^2)$, and $O(Nn^{\frac{3}{2}})$, respectively [38].

5.4 Empirical Evaluation

An extensive set of experiments was carried out to compare different symbolic representation-based time series classification algorithms, namely, BoP, BOSS, SAX-VSM, and MiSTiCl. The main goal of our experimental evaluation is twofold. First, we want to ascertain whether the classification accuracy of MiSTiCl is on par with that of state-of-the-art symbolic representation-based approaches for time series classification, and second, whether the theoretical computational gains can be achieved in practice as well.

The *UEA TSML Repository* provides a Weka based framework for the development of time series data mining algorithms, as well as providing implementations of a number of time series mining algorithms including BoP, BOSS, and SAX-VSM. MiSTiCl has been implemented using the same framework so that any platform or implementation bias can be minimized. The complete source code is available online.[4] The different algorithms were evaluated using 85 real-world and synthetic time series datasets provided by the 2015 expansion of the *UCR Archive* [56]. For each dataset, 100 random shuffles were created such that the class distribution and the total number of instances in the training/testing splits of each shuffle were kept the same as in the original splits. The random number seeds used to create the different shuffles were kept the same when creating the shuffles for different algorithms. Each classification algorithm was then used to evaluate all 100 shuffles of each dataset to obtain a comprehensive set of measurements regarding classification accuracy and runtime requirements.

Parameter settings play a critical role in getting optimal results for almost all machine learning algorithms. Therefore, each algorithm was provided a predefined set of starting parameters, while the algorithms performed parameter optimization to choose the best parameters for each run, i.e., each shuffle of each dataset was evaluated with optimized parameters. This also allows to account for the time required for parameter optimization. We have also taken great care towards using a consistent set of parameters, wherever possible. In this regard, the SAX based MiSTiCl, BoP, and SAX-VSM algorithms were provided default cardinality levels of $\{2, 3, \ldots, 8\}$ and dimensionality reduction levels of $\{2, 3, \ldots, 6\}$. MiSTiCl also requires minimum positive and maximum negative frequency values (f^+ and f^-), and a parameter K for limiting the per-class frequent pattern count. In all the MiSTiCl experiments, both f^+ and f^- were kept fixed at 0.2 and 0.1, respectively, while the parameter K was set using optimization from $\{1, 2, 4\}$. The BoP and SAX-VSM implementations provided by the UEA Repository require an interval count per window in addition to the cardinality and dimensionality level parameters, which is also determined during parameter optimization. The BOSS algorithm is based on the Symbolic Fourier Approximation (SFA) [11]. We used the parameters already specified in the UEA implementation assuming the best set of possible parameters is provided. The alphabet size (or cardinality level) is fixed at four characters, while the word lengths are selected using parameter optimization from the possible values of $\{8, 10, 12, 14, 16\}$. Compared to the 35 parameter combinations evaluated for BoP, SAX-VSM, and MiSTiCl, the BOSS algorithm is only evaluated using five combinations, therefore it already has an advantage in terms of the required runtime.

[4]MiSTiCl repository: `https://github.com/kramerlab/MiSTiCl`.

MiSTiCl transforms a time series classification problem into a feature-based classification problem incorporating a number of diverse time series features, therefore, any off-the-shelf classification algorithm can be used for model induction. Ensemble techniques can be extremely effective for creating accurate classification models given numerous diverse features, so we employed Random Forests (RF) [105], Extremely Randomized Trees (ET) [106], and AdaBoost.M1 (AB) [101] for creating classification models using a maximum of 1000 trees per ensemble, while keeping all other parameters unchanged.

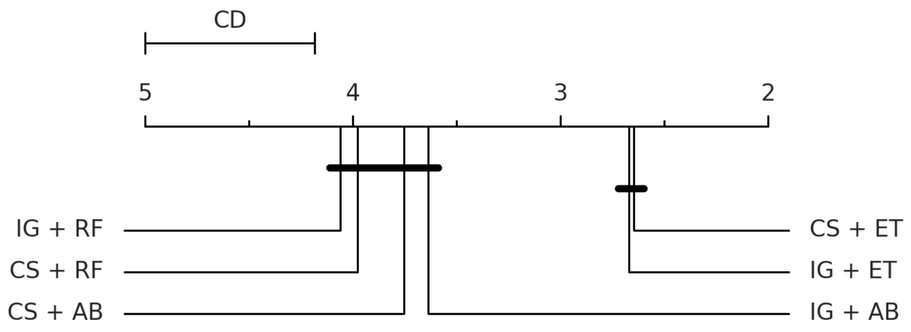
5.5 Results

This section provides an empirical analysis of the classification accuracy and runtime performance of MiSTiCl, BoP, BOSS, and SAX-VSM based on averaged values of the statistics collected for each dataset. We have also included the classification accuracy results for the Shapelet Transform (ST) algorithm [33, 107] to establish a baseline comparison with a real-valued shapelet based classification algorithm.[5] For statistical comparison of different algorithms, we employ the Friedman test followed by Nemenyi post-hoc test based on the average ranks attained by the different algorithms and show the comparisons as critical difference (CD) diagrams [103].

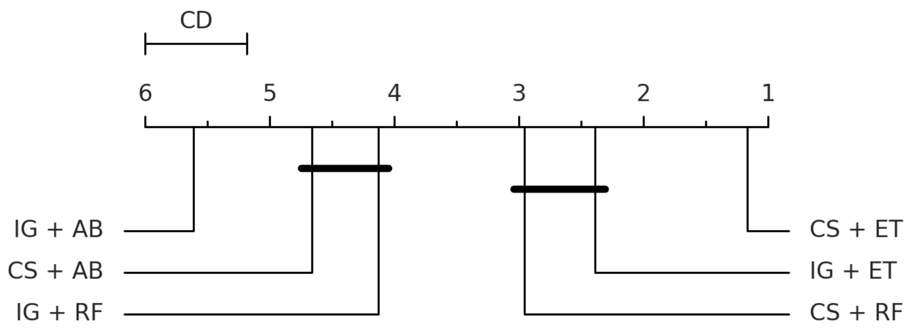
MiSTiCl can employ either the χ^2 independence test or the Information Gain for selecting independent and discriminative frequent patterns, therefore, we carried out all the MiSTiCl experiments using both these methods. In addition, we used Random Forests (RF), Extremely Randomized Trees (ET), and AdaBoost.M1 (AB) for model induction. Figures 5.2a and 5.2b show the critical difference diagrams for different MiSTiCl variants based on classification accuracy and total runtime, respectively. MiSTiCl variants using the χ^2 independence test yield better overall results both in terms of classification accuracy and runtime, therefore, the following analysis will be based on the results obtained using the χ^2 independence test.

Figures 5.3a and 5.3b show the average ranks of the different algorithms based on classification accuracy and runtime, respectively. Regarding the

[5]The results for ST have been taken from the *UEA TSML Repository* (https://www.timeseriesclassification.com/results.php).



(a) Average ranks for different MiSTiCl variants based on classification accuracy.

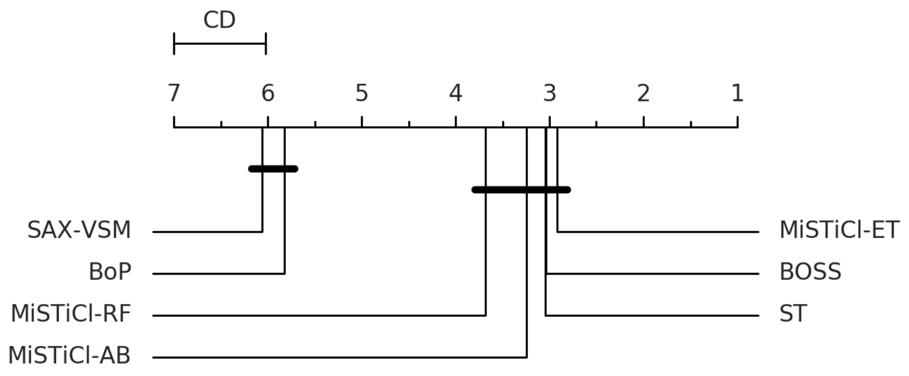


(b) Average ranks for different MiSTiCl variants based on total runtime.

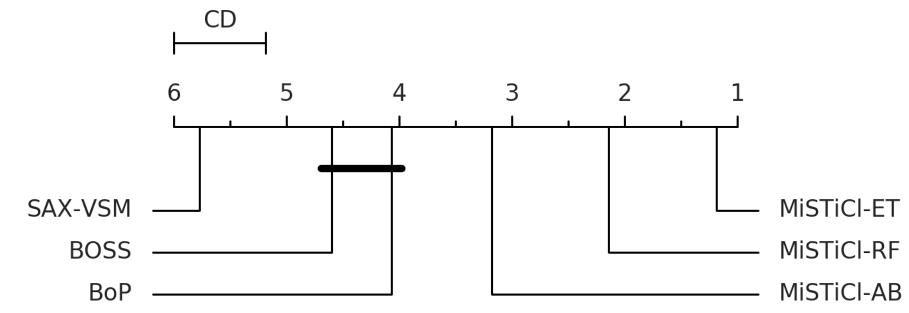
Fig. 5.2. Average ranks for different MiSTiCl variants based on (a) classification accuracy, and (b) runtime. **CS** and **IG** represent χ^2 independence test and Information Gain based pattern selection, respectively. **ET**, **AB**, and **RF** represent the three ensemble classifiers. MiSTiCl variants which are not significantly different at $p = 0.05$ are connected. The critical difference (CD) for significantly different algorithms is 0.81.

classification accuracy, MiSTiCl is placed on par with ST and BOSS with average ranks of 2.92, 3.25, and 3.68 for the ET, AB and RF based variants, respectively, while SAX-VSM and BoP are significantly different compared to the other algorithms. Regarding the runtime, the MiSTiCl variants easily achieve the top three spots with average ranks of 1.33, 2.13 and 3.09 for ET, RF, and AB, respectively, making it significantly faster than any of its competitors.

Figures 5.4, 5.5, and 5.6 present the comparisons in terms of classification accuracy and runtime performance of MiSTiCl against BoP, BOSS, and SAX-VSM, respectively. Each dataset is represented by a marker. The marker positions show the relationship between the runtimes of the compared algorithms,



(a) Average ranks of the different algorithms based on classification accuracy. The critical difference (CD) for significantly different algorithms is 0.97.



(b) Average ranks of the different algorithms based on total runtime. The critical difference (CD) for significantly different algorithms is 0.81.

Fig. 5.3. Average ranks for different algorithms based on (a) classification accuracy, and (b) runtime. Algorithms which are not significantly different at $p = 0.05$ are connected.

while the color and shape of the markers show the relationship between the classification accuracy results. The x- and y-axes show the runtime required to train and test a classification model using MiSTiCl and the other algorithm, respectively. A dotted line, drawn for $y = x$, divides the plot area to indicate which algorithm requires less time for evaluating a specific dataset compared to the other. A marker on the dotted line indicates that both algorithms have the same runtime requirements, while a marker which resides off the dotted line indicates otherwise. Markers above the dotted line show that MiSTiCl is faster, while markers below the dotted line indicate otherwise. The figures also have a boxed background to emphasize the scale of difference between the runtimes of the compared algorithms. Green colored markers indicate that MiSTiCl provides better accuracy, whereas red colored markers indicate

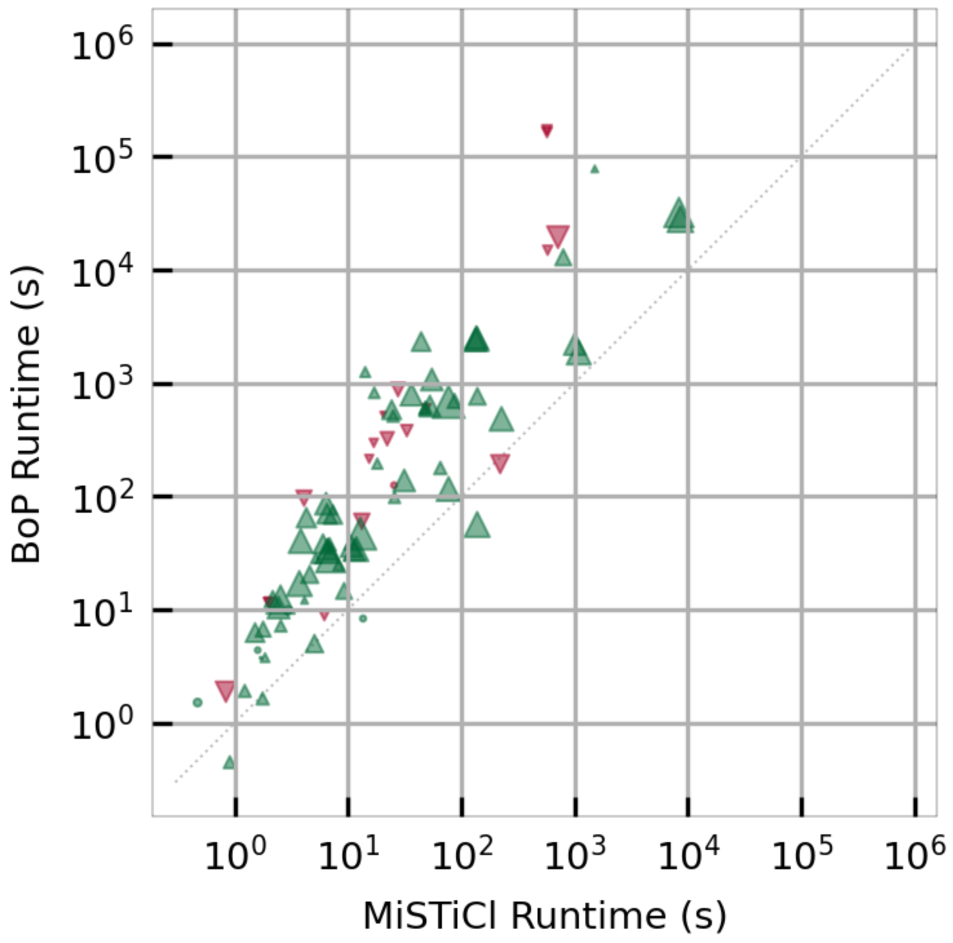


Fig. 5.4. Comparing MiSTiCl against BoP regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the total runtime (training + testing) in seconds. Markers above the dotted line indicate that MiSTiCl is faster than the other algorithm. A green marker indicates that MiSTiCl provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that a significant difference was found in favor of MiSTiCl, while a downward facing triangle shows a significant difference in favor of the other algorithm. Bubbles indicate there was no significance determined.

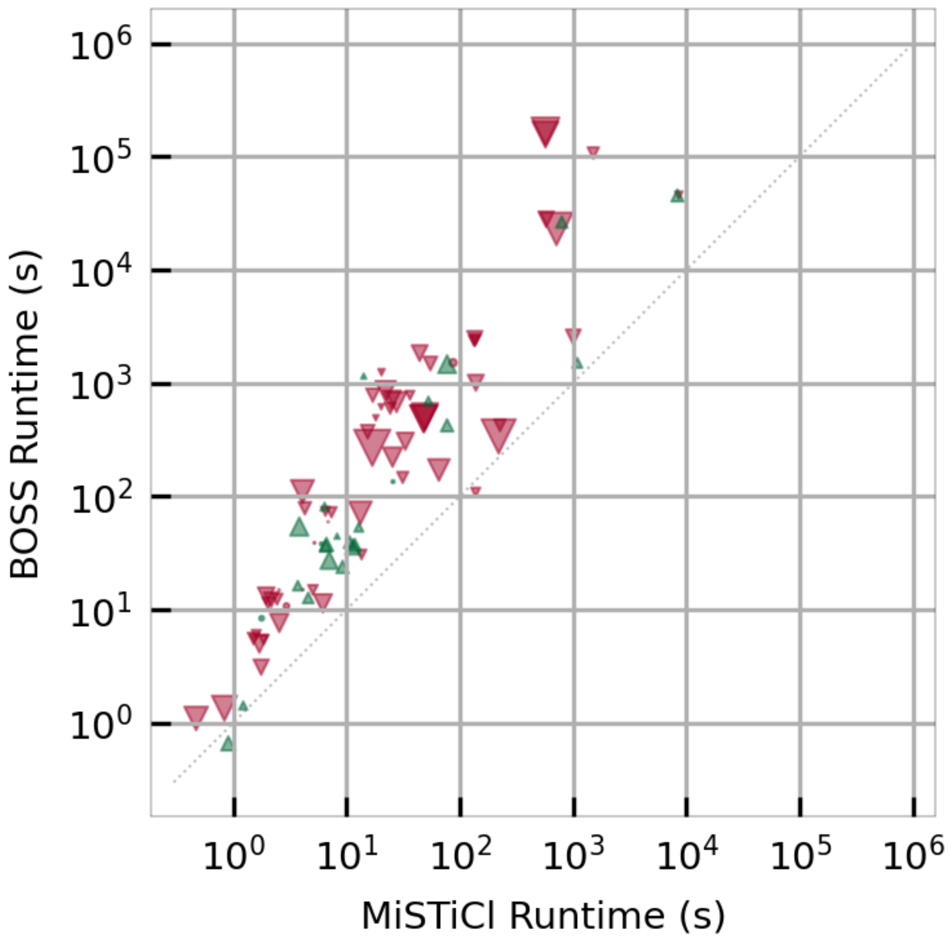


Fig. 5.5. Comparing MiSTiCl against BOSS regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the total runtime (training + testing) in seconds. Markers above the dotted line indicate that MiSTiCl is faster than the other algorithm. A green marker indicates that MiSTiCl provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that a significant difference was found in favor of MiSTiCl, while a downward facing triangle shows a significant difference in favor of the other algorithm. Bubbles indicate there was no significance determined.

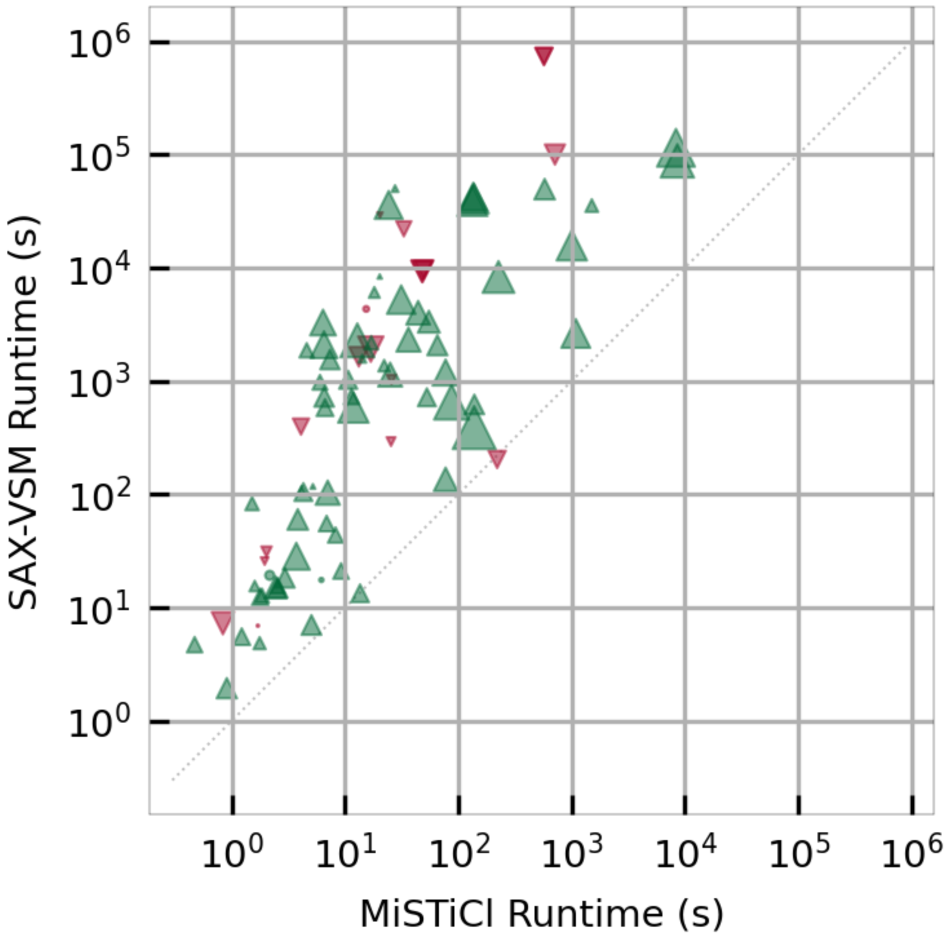


Fig. 5.6. Comparing MiSTiCl against SAX-VSM regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the total runtime (training + testing) in seconds. Markers above the dotted line indicate that MiSTiCl is faster than the other algorithm. A green marker indicates that MiSTiCl provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that a significant difference was found in favor of MiSTiCl, while a downward facing triangle shows a significant difference in favor of the other algorithm. Bubbles indicate there was no significance determined.

Tab. 5.5. Pairwise win/tie/loss comparison of different algorithms using 85 datasets. The margin for wins/losses was set to ±2%.

	ST	BoP	BOSS	SAX-VSM
MiSTiCl-ET	30/29/26	69/4/12	29/22/34	67/10/8
MiSTiCl-RF	28/36/21	64/6/15	28/32/25	63/11/11
MiSTiCl-AB	28/32/25	65/6/14	30/28/27	65/10/10

otherwise. Marker size corresponds to the absolute difference between the classification accuracy values of the compared algorithms, i.e., larger markers indicate a greater difference between the accuracy of the two algorithms. For each dataset, we also performed Wilcoxon's signed-ranks test to establish whether one algorithm performs significantly better or worse compared to the other. In this regard, an upward facing triangle indicates that MiSTiCl is significantly better, a downward facing triangle indicates that the other algorithm is significantly better, while a circle indicates that the difference between the classification accuracy obtained for the two algorithms was insignificant. We can see that MiSTiCl provides an improvement of at least an order of magnitude with respect to runtime, while it is on par with BOSS and dominates BoP and SAX-VSM regarding classification accuracy.

A pairwise win/tie/loss analysis was also performed for a head-to-head comparison between the different MiSTiCl variants and other algorithms. A win was registered if the difference between the classification accuracy achieved by MiSTiCl and the competing algorithm was greater than +2%, a loss was registered if the difference was less than -2%, while a tie was registered otherwise. The numbers of wins/ties/losses are reported in Table 5.5.

5.5.1 Runtime Breakdown

A breakdown of the time required for training and testing a classification model using MiSTiCl is provided in Figure 5.7. This includes the time required for (i) discretizing the training and testing sets, (ii) extracting frequent patterns from the discretized training data, (iii) creating transformed datasets, (iv) parameter optimization, and (v) model training and testing. The time required for each phase of the algorithm is converted to a percentage of the

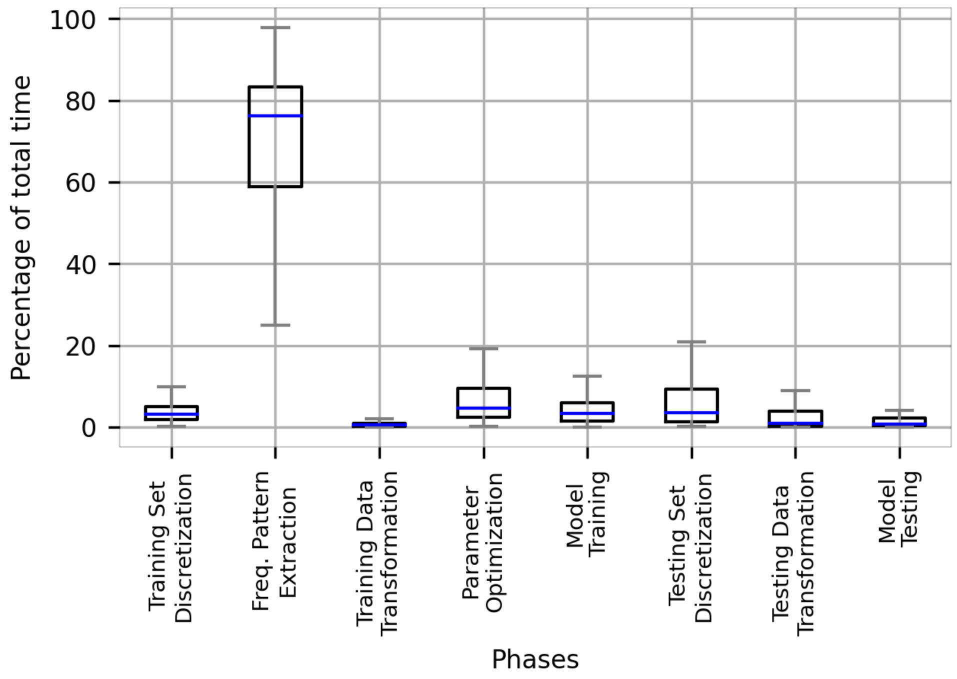


Fig. 5.7. A breakdown of the time spent for individual phases as a percentage of total runtime. The box plots show the aggregated data for all the evaluated datasets when using MiSTiCl (CS + ET).

total time required and box plots are created using the data for all evaluated datasets. We can see that the median time required for data discretization, data transformation, parameter optimization, and model training/testing is less than 10% of the total time required to process a given dataset, while the median time required for pattern extraction is still almost 80%, despite using a string mining algorithm with a linear time complexity. Therefore, the pattern extraction phase still overshadows the other phases in terms of runtime requirements. This breakdown analysis was carried out purely for the MiSTiCl algorithm, and a similar analysis for the other algorithms was not possible due to a monolithic implementation structure.

5.5.2 Results using a conservative parameter set

We also carried out experiments for evaluating the classification and runtime performance of the MiSTiCl algorithm using a conservative parameter set. In these experiments, we provided MiSTiCl with cardinality and dimensionality reduction levels of only $\{3,4,5,6\}$ and $\{2,3,4,5\}$, respectively. This reduces

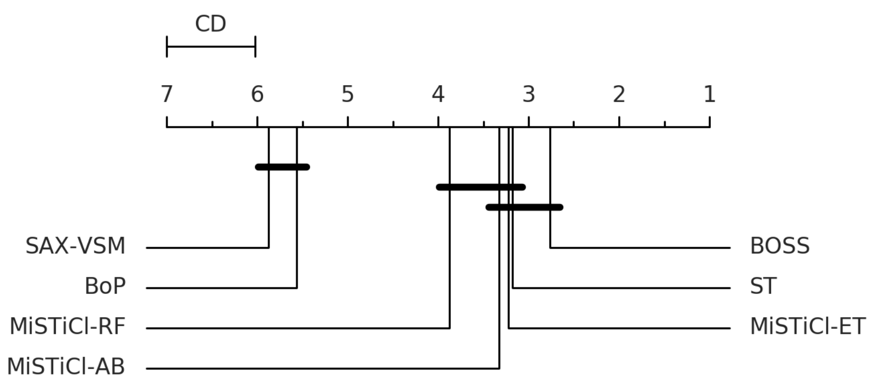


Fig. 5.8. Average ranks of the different algorithms based on classification accuracy. The critical difference (CD) for significantly different algorithms is 0.97. Algorithms which are not significantly different (at $p = 0.05$) are connected.

the number of evaluated parameter combinations to only 16 as compared to the 35 parameter combinations evaluated in the full set of experiments. This results in a further reduction in the total runtime for MiSTiCl as well as a slight reduction in the classification accuracy, however, the overall performance is generally the same. Figure 5.8 shows the average ranks of the MiSTiCl variants, BoP, BOSS, and SAX-VSM based on classification accuracy, when using a conservative set of parameters for the MiSTiCl algorithm.

5.6 Conclusion

MiSTiCl is a time series classification algorithm using a linear time string mining algorithm for feature extraction. Using the string mining algorithm for feature extraction allows to drastically reduce the runtime of the whole classification task compared to many other state-of-the-art approaches. On average, MiSTiCl achieved an order of magnitude speed-up over the most accurate state-of-the-art algorithm, BOSS, while still being robust enough to provide a classification accuracy that is statistically not discernible from it. We used a multi-cardinality and multi-dimensionality approach to incorporate a multi-resolution aspect to the discretized time series classification algorithm. One alternative to using this multi-cardinality and multi-dimensionality approach is to use a form of stacking (or meta-ensemble approach), where each individual base model constituting the ensemble is induced using a single-resolution feature set. However, stacking usually leads to a more complex classification model.

MiSTiCl can also be extended in a couple of ways: Instead of creating the feature based dataset using only the SAX based frequent patterns, we can incorporate features extracted from different time series discretization algorithms. One such algorithm is the Symbolic Fourier Approximation (SFA) used in BOSS and BOSS VS. Incorporating a frequency-based symbolic representation could potentially provide a dual feature based dataset. The local shape-based features can be extracted from SAX representations, while the frequency components dominating the whole shape of the time series can be obtained from a frequency decomposition-based representation. This could provide a local and global view to a single classification algorithm, and the efficiency of the string mining based feature extraction process ensures that the approach could scale well for a variety of problems with large and complex data.

Pattern Sampling as an Alternative to Pattern Extraction

6

6.1 Motivation and Background

Over the last decade, the time series data mining community started developing algorithms that utilized discretized time series data and relied on already established text mining approaches [35–38, 83, 84]. These approaches were mostly based on creating bag-of-words representations of the time series data such that each word was, in essence, a discretized time series subsequence extracted via a sliding window approach. The asymptotic time complexity of these algorithms is cubic or higher-order polynomial, which is only a marginal improvement compared to the $O(N^2n^4)$ computational complexity of real-valued shapelet-based algorithms.

On the other hand, the asymptotic time complexity of MiSTiCl is only $O(Nns)$ (see Section 5.3.7), which makes it one to two orders of magnitude faster than BoP, BOSS, and SAX-VSM (see Section 5.5). Section 5.5.1 provides a breakdown analysis of the time required for model induction using MiSTiCl. It was observed that the pattern extraction phase for MiSTiCl consumes approximately 80% of the total time required for model induction, although the string mining algorithm used as the pattern extractor has a linear time complexity in the length of all discretized time series instances concatenated. This can be attributed to the pattern explosion problem when searching for frequent patterns, since the number of possible subsequences in an m character long string based on an alphabet size α is equal to $\frac{1-\alpha^{m+1}}{1-\alpha}$. Similar arguments can be made regarding the other algorithms since they also rely on the enumeration of vast quantities of patterns. In fact, the sliding window approach adopted by BoP, BOSS, SAX-VSM, and Mr-SEQL implies that the pattern explosion problem haunts these algorithms even more so than it does MiSTiCl.

6.1.1 Pattern Explosion Problem

Pattern mining methods rely on enumerating all possible combinations of a set of base items to find interesting patterns or itemsets. As the number of base items increases, the number of possible pattern combinations increases exponentially, which is commonly referred to as the infamous pattern explosion problem. The exponential rise in the number of candidate patterns consequently leads to a proportional increase in the time required to evaluate all the candidate patterns. Various approaches have been proposed to address this phenomenon, however, these approaches have their own associated drawbacks. *Condensed representations* can be mined efficiently but may result in a large number of patterns nonetheless [108]. *Top-k mining* is efficient, but results in strongly related and redundant patterns lacking diversity [109]. *Constrained mining* can result in either too few or too many patterns depending on the user-provided constraints [110]. *Pattern set mining* considers the intra-pattern relationships and provides a small solution set, but the approach is computationally expensive [111].

6.1.2 Pattern Sampling

Pattern sampling has been proposed as an alternative to the exhaustive pattern enumeration approach [112]. It allows to sample patterns one by one based on a probability distribution that is proportional to a given quality measure. The benefits of using this approach are: (i) flexibility of using a broad range of quality measures and constraints, (ii) 'anytime' data exploration that allows to generate a representative set of patterns which can be further grown and inspected at any time, (iii) diversity of generated sets of patterns due to sampling from different regions in the solution space. Pattern samplers also have to provide some theoretical guarantees regarding sampling accuracy to be reliable, i.e., there should be an estimate of how accurate the empirical probability of sampling a pattern will be compared to the actual (generally unknown) target probability.

Formally, the pattern sampling problem is defined as follows: given a dataset $\mathcal{D}$, a pattern language $\mathcal{L}$, a set of constraints $\mathcal{C}$, and a quality measure $\varphi : \mathcal{L} \rightarrow \mathbb{R}^+$, generate random patterns that satisfy constraints in $\mathcal{C}$ with probability proportional to their qualities:

$$P_\varphi(p) = \begin{cases} \frac{\varphi(p)}{Z_\varphi}, & \text{if } p \in \mathcal{L} \text{ satisfies } \mathcal{C} \\ 0, & \text{otherwise} \end{cases}$$

where Z_φ is an (often unknown) normalization constant. A quality measure quantifies the domain-specific interestingness of a pattern. The choice of a quality measure and constraints allows to express specific analysis requirements. The sampling procedure meets these requirements by satisfying the constraints and generating high-quality patterns more frequently. Thus, sampled patterns are a representative subset of all interesting regularities in the dataset.

The benefits offered by pattern sampling make it an enticing option to reduce the computational requirements of the pattern extraction procedure for pattern based time series classification. A rudimentary pattern sampler could be grown based on a limited number of instances of the discretized time series data or a limited set of pattern combinations to kick off the process, allowing a continued improvement of the sampler. This also allows to stop investing any more resources into improving the sampler once an acceptable level of sampling accuracy has been achieved.

6.2 Design

Pattern Sampling for Shapelet-based Time Series Classification (*PS2C*) employs pattern sampling instead of enumerating all candidate patterns to determine the most discriminative patterns for a given time series dataset. The basic structure of the PS2C algorithm is similar to other feature/pattern based time series classification algorithms, e.g., ST [107], MiSTiCl, etc. The main steps of the algorithm include: (i) time series discretization, (ii) pattern sampler creation, (iii) feature set creation using K sampled patterns, (iv) creating a transformed dataset, and finally (v) model induction. Steps (ii) and (iii) leverage the power of pattern sampling to generate highly discriminative and diverse patterns. The quality measure for pattern acceptance ensures that only highly discriminative patterns are ingested in the sampler, which enables quick convergence of the sampler and better sampling accuracy.

The PS2C algorithm discretizes the data using the SAX algorithm, therefore, the details regarding the time series discretization are the same as described in Section 5.3.2. SAX preserves the overall shape of the time series instances, however, it can also lead to loss of features. This is an artifact of inadvertent feature splitting due to the use of an arbitrary window size ω. One effective way of dealing with this problem is to initiate multiple independent feature extraction pipelines, each based on a different combination of the α and ω parameters, and merging the results of each individual feature extraction problem into one aggregate feature-based dataset. This results in a diverse feature set that leads to better overall accuracy when using an ensemble classifier for model induction, because ensemble methods inherently tend to reduce variance and sometimes also bias. MiSTiCl was modeled to create such a multi-resolution feature set to identify features at various dimensionality and cardinality levels. This approach provides a significant boost to the overall accuracy, therefore, PS2C is also modeled similarly to create a multi-resolution feature set.

6.2.1 Overview

Algorithm 6.1 lists the main PS2C algorithm. The input parameters include the real-valued training and testing splits (D_{train} and D_{test}), a set of alphabet and window sizes (A and Ω), the maximum allowed length of the patterns used for generating the pattern sampler (l_{max}), the minimum allowed quality measure value per pattern used for generating the pattern sampler (s_{min}), a scaling factor τ for scaling the quality measure values, and a parameter K for patterns sampled per parameter combination. The procedure starts with the initialization of associative arrays for the standalone feature sets which will be merged later to form the multi-resolution feature sets (Line 6.1-1).

Algorithm 6.1 PS2C (D_{train}, D_{test}, A, Ω, l_{max}, s_{min}, τ, K)

```
1: $SV_{train} \leftarrow \{\}$, $SV_{test} \leftarrow \{\}$   ▷ Initialize associative arrays for feature sets
2: for all $(\alpha, \omega) \in A \times \Omega$ do
3:     $\widehat{D}_{train}, \widehat{D}_{test} \leftarrow$DISCRETIZE($D_{train}$, $D_{test}$, $\alpha$, $\omega$)
4:     $\widehat{D'}_{train} \leftarrow$CREATESUFFIXTREES($\widehat{D}_{train}$)
5:     $PS \leftarrow$FITPATTERNSAMPLER($\widehat{D}_{train}$, $\widehat{D'}_{train}$, $l_{max}$, $s_{min}$, $\tau$)
6:     $SV^{\alpha,\omega}_{train}, SV^{\alpha,\omega}_{test} \leftarrow$CREATEFEATURESET($D_{train}$, $D_{test}$, $\widehat{D}_{train}$, $PS$, $K$)
7: end for
8: $MultiResFS_{train} \leftarrow \phi$,   ▷ Initialize multi-resolution feature sets
   $MultiResFS_{test} \leftarrow \phi$
9: for all $(\alpha, \omega) \in A \times \Omega$ do
10:    $MultiResFS_{train} \leftarrow MultiResFS_{train} \mid\mid FS^{\alpha,\omega}_{train}$
11:    $MultiResFS_{test} \leftarrow MultiResFS_{test} \mid\mid FS^{\alpha,\omega}_{test}$
12: end for
13: return $MultiResFS_{train}, MultiResFS_{test}$
```

The first step is the discretization of the train and test splits corresponding to the current (α, ω) parameter combination (Line 6.1-3). Next, the discretized training set instances are used to create suffix tree representations for efficient substring search (Line 6.1-4), however, the choice of suffix trees is rather superficial, since there are a number of other data structures that can be utilized to efficiently search a given string for the presence of query substrings. Next, a probabilistic pattern sampler is induced based on the discretized training data (Line 6.1-5), followed by the creation of transformed training and testing feature sets based on the sampled patterns (Line 6.1-6). Finally, the train and test feature sets corresponding to all the $(\alpha, \omega) \in A \times \Omega$ pairs are concatenated horizontally to create the multi-resolution training and testing feature sets, respectively (Lines 6.1-9 to 6.1-12).

6.2.2 Creating the Pattern Sampler

A pattern sampler can be modeled in several ways, e.g., graphs (MCMC), trees, XOR constraints, etc. [112]. In the case of PS2C, the pattern sampler has been envisaged as a trie with weighted edges, since this allows incorporating constraints into the sampler generation and pattern sampling processes as well as allowing fast, iterative updates to the sampler. A trie is a data structure used to store strings in order to support fast pattern matching. Formally, if S is a set of s strings from an alphabet Σ, then a standard trie for S is an ordered tree with the following properties:

- Each node of a trie, except the root, is labeled with a character of Σ.
- The children of an internal node of the trie have distinct labels.
- The trie has s leaves, each associated with a string of S, such that the concatenation of the labels of the nodes on the path from the root to a leaf v of the trie yields the string of S associated with v.

Thus, a trie represents the strings of S with paths from the root to the leaves. Strings with a common prefix share the edges for the common prefix, while a split is created when the characters in the strings differ. The edges are augmented with weights such that inserting a string in the trie also associates a corresponding weight to all the inserted edges. An edge shared between multiple strings has a weight equal to the aggregate of the weights associated with all the strings that share the particular edge.

Edge weights are calculated on the basis of the discriminative capability of the inserted patterns. The χ^2 statistic can be used to determine whether there is a statistically significant difference between the expected and observed counts for a given contingency table consisting of two or more categories. In case of a symbolic time series dataset, the categories are the different classes, while the counts are the numbers of instances belonging to each class in which the given pattern is present or absent. The range of values for the χ^2 statistic is $(0, |D_{train}|]$. For a binary class problem, if a pattern occurs in all instances of one class, whereas it is absent in all instances of the other class, then the χ^2 statistic will be maximized, whereas if the pattern is present/absent in most of the instances, then the χ^2 statistic will be close to 0. In order to simplify subsequent calculations, the χ^2 statistic is normalized with $|D_{train}|$ so that the effective range becomes $(0, 1]$, where the value of 1 indicates that the given pattern is a perfect discriminator, while a value close to 0 indicates otherwise.

The normalized χ^2 statistics can be directly used as weights for the edges, however, we can introduce a bias towards highly discriminative patterns using temperature scaling. The scaled edge weights are calculated as $q^{(1/\tau)}$, where q is the normalized χ^2 statistic and τ is the temperature scaling factor. During pattern sampling, the probability of selecting an edge is given as $f_\tau(q)_i = \frac{q_i^{(1/\tau)}}{\sum_j q_j^{(1/\tau)}}$, where $\sum_j q_j^{(1/\tau)}$ is the sum of all edge weights originating from the node. When $\tau = 1$, the edge weights are linearly proportional to the normalized χ^2 statistics. As τ decreases, the bias towards patterns with higher normalized χ^2 statistics increases, e.g., a quadratic scaling is applied to the values for $\tau = 0.5$. As $\tau \to 0$, the function turns into an $\mathbf{argmax}$ function. Therefore, a very small value of τ will almost always lead to the selection of the most discriminative patterns in the trie, while a value close to one will lead to a fair selection. Figure 6.1 shows an example trie created from a set of words extracted from a discretized dataset. The figure is based on a binary class dataset with 14 instances in each class. The scaling factor τ is set to 0.33, the alphabet size α is set to 6, and the dimensionality reduction factor ω is set to 4. The pattern `ffe` occurs in all instances of one class but is absent in all instances of the other class, therefore, its normalized χ^2 statistic is equal to 1.0 that translates into a scaled weight of 1.0 as well. The pattern `ffc` occurs in 13 of the 14 instances of one class but is absent in all instances of the other class, therefore, its normalized χ^2 statistic is equal to 0.867 and the scaled weight is equal to 0.65. Inserting the first pattern into the trie creates the required edges and associates the corresponding scaled weight to all the created edges. When the second pattern is inserted, the edges that correspond to the substring `ff` have their weight updated to be the sum of the previously assigned weight and the weight associated with the current pattern, while a new edge is inserted for the suffix `c` with the respective weight for the pattern. The remaining patterns are also inserted similarly.

Algorithm 6.2 lists the steps involved in the creation of a weighted trie based on patterns up to a user-specified maximum length. The procedure iterates over the allowed pattern lengths to extract and evaluate patterns of a given length for subsequent insertion in the trie (Lines 6.2-2 to 6.2-10). First, all patterns of a given length are extracted using the suffix trees created during the preprocessing phase (Line 6.2-3). Next, each candidate pattern from the list of patterns extracted in the previous step is evaluated to determine its discriminative capability using the χ^2 test statistic (Line 6.2-5). The number

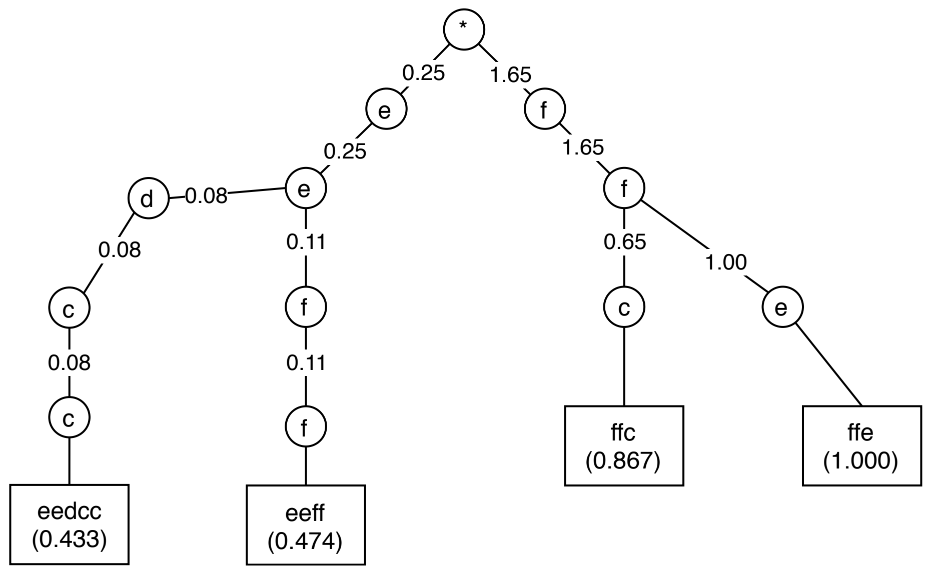


Fig. 6.1. An illustration of a weighted trie with shared edges between multiple strings. The edges are weighted using the scaled quality measures of the strings. The weight of a shared edge is equal to the aggregated weights contributed by all strings sharing that edge. The leaf nodes point to the patterns added to the trie along with their normalized χ^2 test statistic.

of evaluated candidate patterns can be reduced by randomly sampling the list of candidate patterns. In case the candidate patterns are randomly sampled, the procedure can be initiated multiple times to update the pattern sampler. In this case, the procedure can be altered to keep track of the already evaluated patterns so that subsequent calls do not waste computational resources on re-evaluating the same candidate patterns repeatedly. The candidate pattern is inserted into the trie if its normalized χ^2 statistic is greater than or equal to the minimum allowed discriminative value (Lines 6.2-6 to 6.2-8). Starting from the root node, the insertion procedure checks if an edge corresponding to the first character in the candidate pattern is present or not. If the edge is absent, the procedure creates the corresponding edge and sets its weight equal to the scaled quality measure of the current pattern. If an edge corresponding to the character is already present, then the edge weight is updated by adding the scaled quality measure of the current pattern to the existing edge weight. The node weight is also set to the updated aggregate value of the edge weights. The procedure then traverses down the edge corresponding to the first character and the same procedure is repeated for the second character in the pattern and so on, until all the characters have been inserted.

Algorithm 6.2 FitPatternSampler ($\widehat{D}_{train}$, $\widehat{D'}_{train}$, l_{max}, s_{min}, τ)

```
 1: $trie \leftarrow \phi$                                   ▷ Initialize an empty trie
 2: for $l \leftarrow 2$ to $l_{max}$ do
 3:     $S \leftarrow$ FindAllPatternsWithLength($l, \widehat{D'}_{train}$)
 4:     for $s \in S$ do
 5:         $q \leftarrow$ CalculateChiSqStatistic($s, \widehat{D'}_{train}$)
 6:         if $q \geq s_{min}$ then
 7:             Insert($s, q, \tau, trie$)
 8:         end if
 9:     end for
10: end for
11: return $trie$
```

6.2.3 Creating Feature Sets

Algorithm 6.3 lists the pseudocode for creating the real-valued feature datasets from K sampled patterns. After initialization of the necessary data structures, K patterns are sampled from the pattern sampler and their corresponding subsequences are used to generate the feature sets (Lines 6.3-2 to 6.3-7). Sampling a pattern involves traversing the weighted trie from the root node to a leaf node using the fitness proportionate (roulette wheel) selection method. At any node, the probability of selecting the ith edge is calculated by dividing the edge weight $q_i^{1/\tau}$ by the sum of all edge weights for the current node $\sum_j q_j^{1/\tau}$. First, a uniformly distributed random number r is drawn in the range $\left[0, \sum_j q_j^{1/\tau}\right)$. Next, all the edge weights are compared with the random number r based on the lexical order of the edges. For an edge i, if the random number r is less than the edge weight $q_i^{1/\tau}$ then the ith edge is selected as the next edge, otherwise $q_i^{1/\tau}$ is subtracted from r and the next edge weight is compared. If a node has child nodes as well as being a leaf node, the decision to return the string terminating at the current node or to traverse the trie further is also based on a random number. In this case, a uniform random number is drawn in the range $[0, 1)$ and the procedure continues trie traversal if the random number is less than 0.5. For each sampled pattern, a reverse lookup is performed to extract the

Algorithm 6.3 CreateFeatureSets (D_{train}, D_{test}, $\widehat{D}_{train}$, $Sampler$, K)

```
1: $SV_{train} \leftarrow MATRIX(|D_{train}|, K)$, $SV_{test} \leftarrow MATRIX(|D_{test}|, K)$
2: for $k \leftarrow 1$ to $K$ do
3:     $f \leftarrow$SamplePattern($Sampler$)
4:     $s \leftarrow$PerformReverseLookup($D_{train}$, $\widehat{D}_{train}$, $f$)
5:     For each $T \in D_{train}$
           populate the respective row and column of $SV_{train}$
           with the distance value between $T$ and $s$
6:     For each $T \in D_{test}$
           populate the respective row and column of $SV_{test}$
           with the distance value between $T$ and $s$
7: end for
8: return $SV_{train}, SV_{test}$
```

real-valued subsequence from the original time series data corresponding to the symbolic pattern (Line 6.3-4). Next, the kth column of the feature sets SV_{train} and SV_{test} are populated with the distance values between the time series instances and the shapelet discovered after the reverse-lookup procedure (Lines 6.3-5 and 6.3-6).

6.2.4 Merging Feature Sets

Discretization-based time series mining algorithms can often have deteriorated performance due to information and feature loss. A multi-resolution feature set based on feature sets created using different discretization and quantization parameter combinations can mitigate the feature loss problem. It was shown in the previous chapter that concatenating various feature sets that have been created using different $(\alpha, \omega) \in A \times \Omega$ parameter combinations can solve the feature loss problem, as well as improve the overall accuracy due to the inclusion of features obtained from different discretization and quantization levels.

In case of MiSTiCl, at most $\frac{|A| \times |\Omega|}{3}$ feature sets are used to create a multi-resolution feature set. A greedy set cover approach is used to combine the feature sets so that the resulting multi-resolution feature set will correctly classify as many training instances as possible (see Section 5.3.6). This approach provides better overall results as well as keeping the number of features in the multi-resolution feature set to a minimum, however, it also requires significant computational resources. MiSTiCl creates feature sets using class-correlated frequent patterns, therefore, each feature set can have K features based on as many frequent patterns specific to a given class, where K is a user-provided parameter. This approach can sometimes result in expansive feature sets when the number of classes in the dataset is large, and K is also set to a large value. Therefore, combining a small number of feature sets allows to restrict the overall number of features in the multi-resolution feature sets.

In case of the PS2C algorithm, a different merging strategy has been adopted. Instead of concatenating a few feature sets, PS2C concatenates all the feature sets created for each $(\alpha, \omega) \in A \times \Omega$ parameter combination to form the multi-resolution training and testing feature sets. It was observed that merging all the feature sets results in a multi-resolution feature set that allows to create classification models that are as accurate as the classification models created using the multi-resolution feature set created after the greedy set cover optimization. The pattern sampling approach employed for PS2C is not restricted to providing patterns for each class individually, therefore, only a few patterns can be used to create the individual feature sets. This also allows to concatenate a larger number of feature sets together without running into the problem of a multi-resolution feature set with a very large number of features. This insight also allows PS2C to forgo the optimization step based on the expensive greedy set cover approach.

6.2.5 Complexity Analysis

The overall computational complexity of the PS2C algorithm can be determined by investigating the creation of an individual feature set and then aggregating the impact of all pipelines since the creation of feature sets (based on individual (α, ω) parameter combinations) contributes equally towards the overall complexity. SAX requires $O(Nn)$ operations to discretize a dataset with N instances each having length n. The time taken for creating a pattern sampler depends on: (i) the time taken to extract candidate patterns, and (ii) the time taken in finding the candidate pattern in each discretized instance of the training set. The suffix trees allow to search the candidate patterns in time linear to the length of the discretized time series, while extracting the patterns from N instances results in the time required for both these steps to be $O(Nm)$, where m is the length of the discretized time series. Sampling a pattern using the trie is proportional to the maximum pattern length in the trie $O(l_{max})$. For a feature corresponding to a symbolic pattern, N feature values have to be calculated, where each feature value calculation takes $O(ns)$ time, where s is the length of a subsequence and $s \ll n$. Since K is a constant and much smaller than N and n, the time required for creating a complete feature set is on the order of $O(Nns)$. The overall time complexity of creating a feature set for a given α and ω parameter combination is on the order of $O(Nn) + O(N\frac{n}{w}) + O(1) + O(Nns) \approx O(Nns)$. Since the quantity $|A| \times |\Omega|$ is a constant, the asymptotic time complexity of the algorithm is on the order of $O(Nns)$. PS2C and MiSTiCl have the same asymptotic time complexity, which makes these algorithms the fastest symbolic representation based time series classification algorithms.

6.3 Empirical Evaluation

An extensive set of experiments was conducted to evaluate the performance of our proposed algorithm compared to other symbolic representation based time series classification algorithms. The empirical evaluation is based on the same strategy as employed for MiSTiCl (see Section 5.4). The different algorithms were evaluated using 85 real-world and synthetic time series datasets provided by the 2015 expansion of the *UCR Archive* [56]. For each dataset, 100 random shuffles were created such that the class distribution and the total number of instances in the training and testing splits of each shuffle were kept the same as in the original splits. The random number seeds used to create the different shuffles were kept the same when creating the shuffles for different algorithms. Each classification algorithm was then used to evaluate all 100 shuffles of each dataset to obtain a comprehensive set of measurements regarding classification accuracy and runtime requirements. In order to compare PS2C against other well-known algorithms regarding classification accuracy, we have used the results provided by the *UEA TSML*

Repository and the Mr-SEQL repository. The runtime requirements of the different algorithms, however, are not provided by the *UEA TSML Repository*, therefore, the runtime requirements for MiSTiCl, BOSS, BoP, and SAX-VSM were taken from the evaluation carried out for MiSTiCl and a comparison of the PS2C algorithm against these four algorithms was carried out regarding runtime requirements. The Mr-SEQL implementation is based on a different programming language and framework, therefore, we have not compared the runtime requirements directly and only restrict the evaluation to a theoretical comparison of the asymptotic time complexity requirements. The PS2C algorithm has also been implemented using the same framework used for implementing MiSTiCl, and the complete source code is available online.[1]

All experiments were performed with a fixed set of parameters for all datasets. The cardinality levels A and the dimensionality reduction factors Ω were set to the same combination as used for the evaluation of MiSTiCl. The A and Ω parameters were set to $\{2, 3, \ldots, 8\}$ and $\{2, 3, \ldots, 6\}$, respectively. The seed for the random number generator was explicitly set for each experiment. The maximum allowed pattern length l_{max} was set to 20, the minimum acceptable discriminative power s_{min} (normalized χ^2 statistic) was set to be 0.05, the scaling factor τ was set to 0.5, and maximum patterns K per (α, ω) parameter combination was set to 4. PS2C transforms a time series classification problem into a feature-based classification problem incorporating a diverse set of time series features, therefore, any off-the-shelf classification algorithm can be used for model induction. Ensemble techniques can be extremely effective for creating accurate classification models given numerous diverse features, and we already employed Random Forests (RF) [105], Extremely Randomized Trees (ET) [106], and AdaBoost.M1 (AB) [101] for creating classification models for the empirical evaluation of the MiSTiCl algorithm. Since the results reported for MiSTiCl are based on the Extremely Randomized Trees (ET) [106] ensemble creation approach, the PS2C algorithm was also evaluated using the same ensemble technique. In this regard, a maximum of 1000 trees per ensemble were used, while keeping all other parameters unchanged. For statistical comparison of different algorithms, we employ the Friedman test followed by Nemenyi post-hoc test based on the average ranks attained by the different algorithms, and show the comparisons as critical difference (CD) diagrams [103].

[1]PS2C repository: `https://github.com/kramerlab/ps2c`.

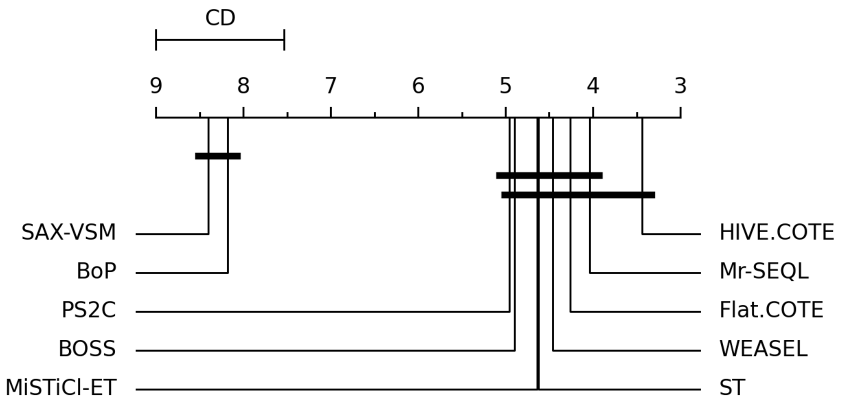
6.4 Results

In terms of classification accuracy, the PS2C algorithm performs on par with other algorithms for datasets with two to six classes, however, the classification accuracy deteriorates as the number of classes in a dataset goes beyond eight. This behavior is due to the fact that a single pattern sampler is created and there is no provision for sampling class-correlated patterns. An obvious alternative is to create samplers for each class individually in a one-vs-all fashion, however, another alternative is to incorporate additional information into the samplers along with the patterns, which could enable class-correlated pattern sampling. The minimum acceptable discriminative power s_{min} for a candidate pattern allows to adjust the acceptance threshold for candidate patterns. A high value allows to accept only the very best patterns, while a value close to zero allows to accept almost all patterns. Accepting a large number of patterns can lead to a densely populated trie, but a stringent scaling factor τ can help deal in this case by heavily weighting the useful patterns and diminishing the chances of sampling less useful patterns. Therefore, s_{min} and τ are complementary parameters and a slight adjustment is all that is needed to get the best results. The maximum allowed pattern length l_{max} is used to limit the number of patterns inserted into the trie. In most cases, the discriminative patterns are much shorter than the length of the discretized time series instances, however, many discriminative patterns can have a huge number of variants with either a prefix or a suffix. The l_{max} parameter allows to restrict the inclusion of too many variant patterns in the trie, and in doing so, helps to keep the trie balanced since the inclusion of too many variants with the same discriminative power would cause the sampling procedure to return related and/or redundant patterns.

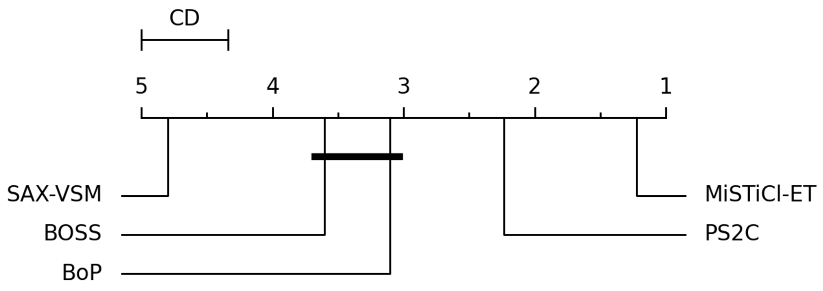
Figure 6.2a shows a critical differences diagram for different time series classification algorithms regarding classification accuracy. Overall, PS2C performs impressively and is on par with algorithms like ST and Flat.COTE. HIVE.COTE and Flat.COTE are two hybrid ensemble classifiers which base their classification on the basis of various types of classifiers, including ST, BOSS, etc. Both these algorithms have an extremely high computational cost due to their dependence on training several different types of classification algorithms. Among the pattern-based time series classification algorithms, SAX-VSM and BoP perform worse, while MiSTiCl, Mr-SEQL, BOSS and PS2C perform similarly and are not significantly different from each other. PS2C is not significantly different from ST or Flat.COTE, however, its average rank is slightly worse than BOSS which causes PS2C to also miss the group that forms the cohort of the best performing time series algorithms in this comparison.

Figure 6.2b shows the critical differences diagram for pattern-based time series classification algorithms regarding runtimes. MiSTiCl was a clear winner and PS2C was the second fastest, while BoP/BOSS and SAX-VSM were significantly slower than either of the two algorithms. Overall, PS2C was 1.1 to 1.3 times slower than MiSTiCl on average, however, since MiSTiCl was shown to be significantly faster than the other algorithms, we can infer that PS2C is also substantially faster than the remaining algorithms. This is backed up by complexity considerations (see Section 6.2.5).

Figures 6.3, 6.4, 6.5, and 6.6 present the comparisons in terms of classification accuracy and runtime performance of PS2C against BoP, BOSS, SAX-VSM, and MiSTiCl, respectively. The figures have been created using the same approach as used for creating the figures for comparing MiSTiCl against the other algorithms. Each dataset is represented by a marker. The marker positions show the relationship between the runtimes of the compared algorithms, while the color and shape of the markers show the relationship between the classification accuracy results. The x- and y-axes show the runtime required to train and test a classification model using PS2C and the other algorithm, respectively. A dotted line, drawn for $y = x$, divides the plot area to indicate which algorithm requires less time for evaluating a specific dataset compared to the other. A marker on the dotted line indicates that both algorithms have the same runtime requirements, while a marker which resides off the dotted line indicates otherwise. Markers above the dotted line show that PS2C is faster, while markers below the dotted line indicate otherwise. The figures also have a boxed background to emphasize the scale of difference between the runtimes of the compared algorithms. Green colored markers indicate that PS2C provides better accuracy, whereas red colored markers indicate otherwise. Marker size corresponds to the absolute difference between the classification accuracy values of the compared algorithms, i.e., larger markers indicate a greater difference between the accuracy of the two algorithms. For



(a) Average ranks based on classification accuracy for different time series classification algorithms. The critical difference (CD) for significantly different algorithms is 1.46.



(b) Average ranks based on runtime performance for PS2C, MiSTiCl, BoP, BOSS, and SAX-VSM. The critical difference (CD) for significantly different algorithms is 0.6.

Fig. 6.2. Average ranks for comparing PS2C against different algorithms based on classification accuracy and runtime. Algorithms which are not significantly different at $p = 0.05$ are connected.

each dataset, we also performed Wilcoxon's signed-ranks test to establish whether one algorithm performs significantly better or worse compared to the other. In this regard, an upward facing triangle indicates that PS2C is significantly better, a downward facing triangle indicates that the other algorithm is significantly better, while a circle indicates that the difference between the classification accuracy obtained for the two algorithms was insignificant. We can see that PS2C performs similar to MiSTiCl when comparing BoP, BOSS, and SAX-VSM. Figure 6.6 shows a comparison between PS2C and MiSTiCl. Regarding the classification accuracy, almost all datasets have a very small difference. The number of datasets with significantly different classification accuracy are also quite few. Regarding the runtime performance, we can see that there are two groups of datasets, one group of datasets requires

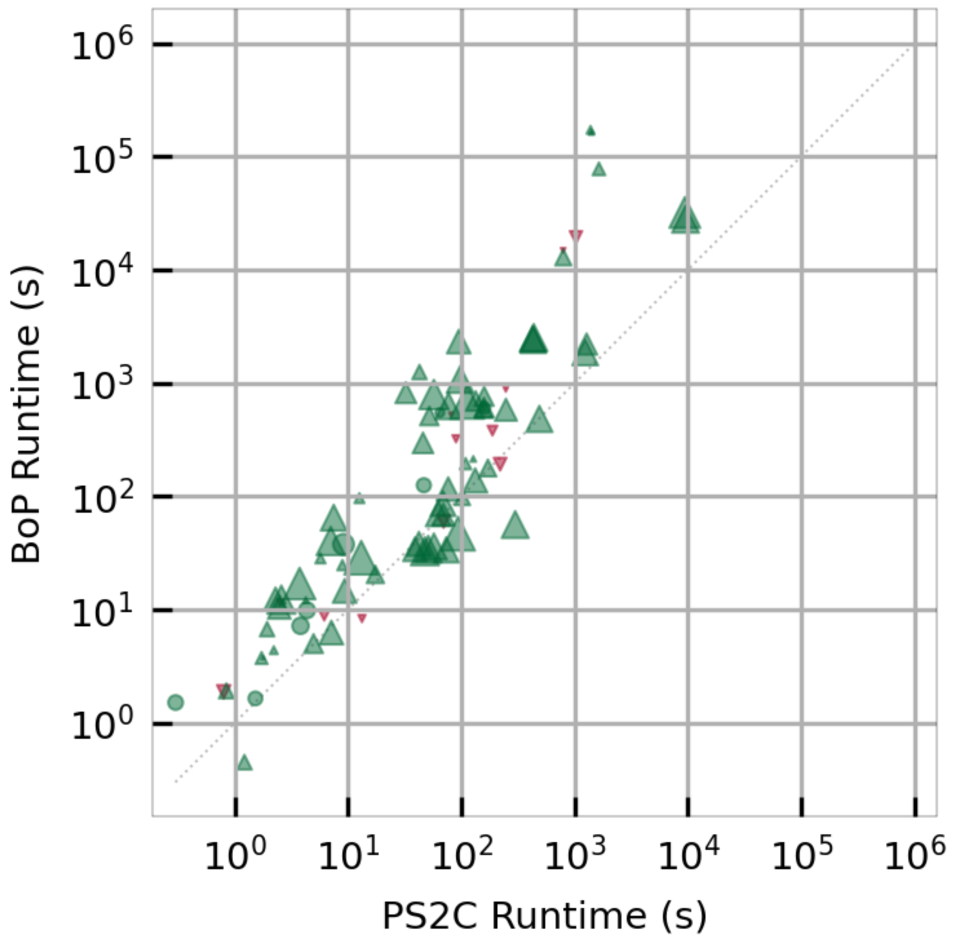


Fig. 6.3. Comparing PS2C against BoP regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the total runtime (training + testing) in seconds. Markers above the dotted line indicate that PS2C is faster than the other algorithm. A green marker indicates that PS2C provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that a significant difference was found in favor of PS2C, while a downward facing triangle shows a significant difference in favor of the other algorithm. Bubbles indicate there was no significance determined.

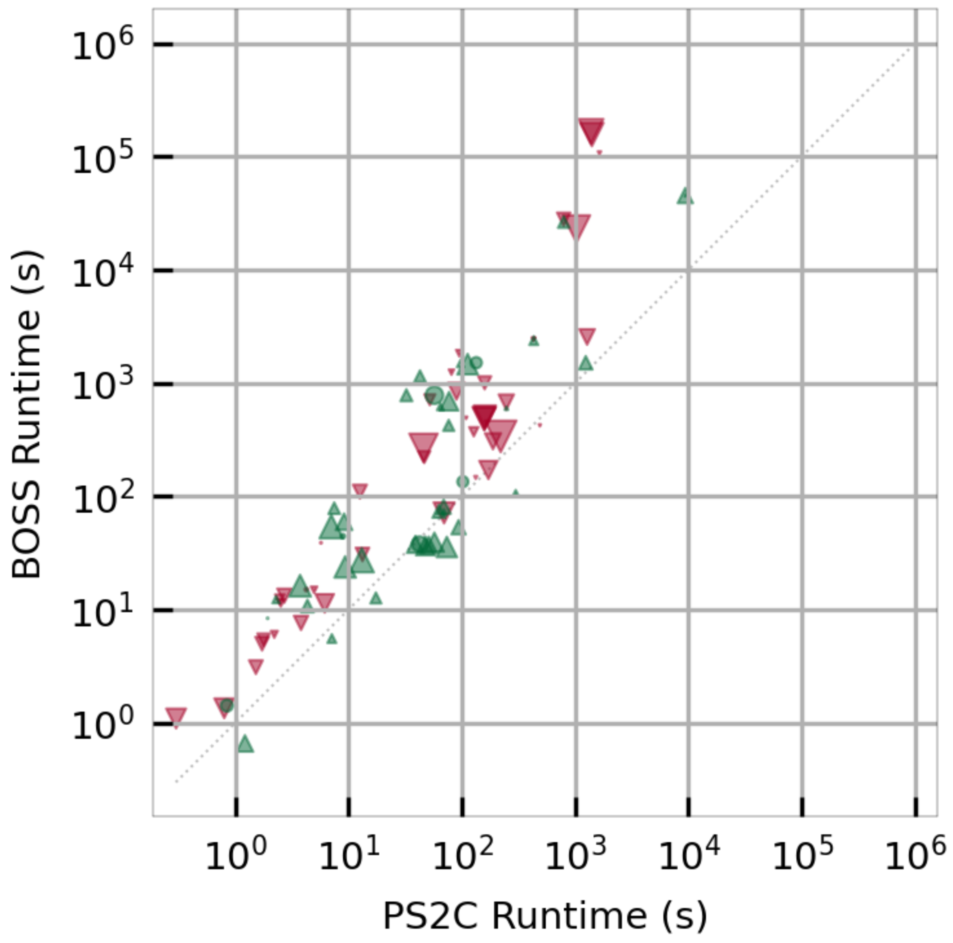


Fig. 6.4. Comparing PS2C against BOSS regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the total runtime (training + testing) in seconds. Markers above the dotted line indicate that PS2C is faster than the other algorithm. A green marker indicates that PS2C provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that a significant difference was found in favor of PS2C, while a downward facing triangle shows a significant difference in favor of the other algorithm. Bubbles indicate there was no significance determined.

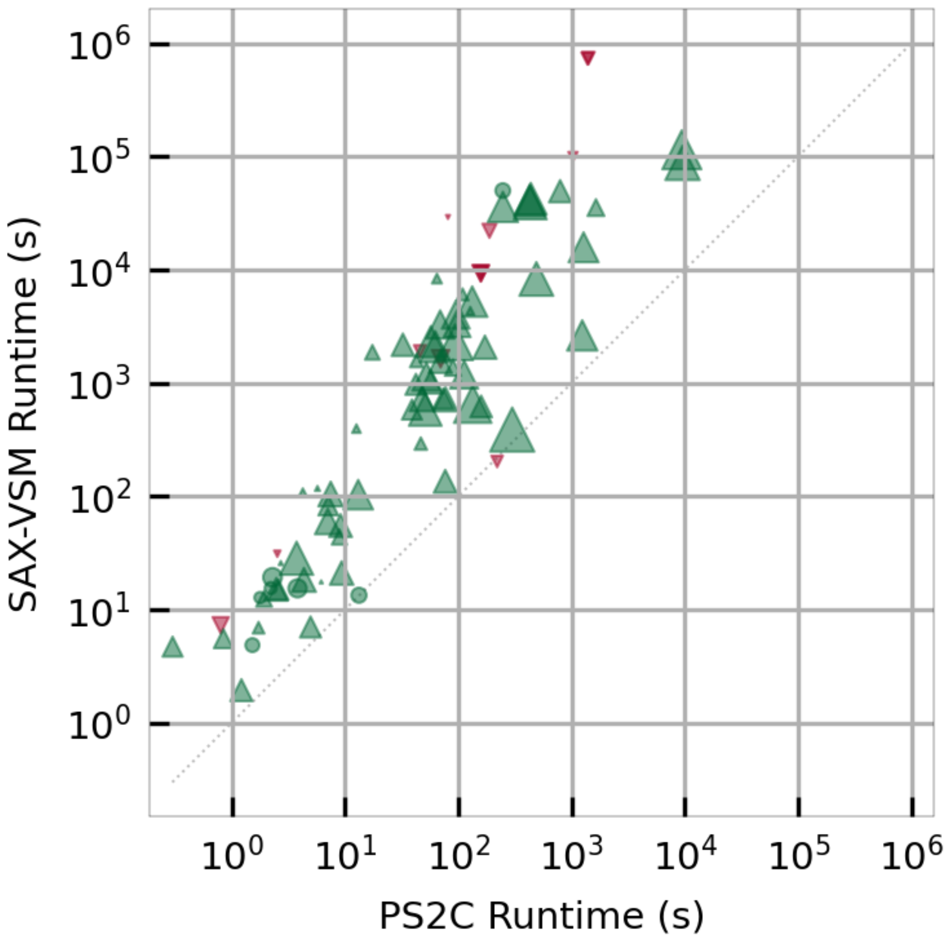


Fig. 6.5. Comparing PS2C against SAX-VSM regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the total runtime (training + testing) in seconds. Markers above the dotted line indicate that PS2C is faster than the other algorithm. A green marker indicates that PS2C provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that a significant difference was found in favor of PS2C, while a downward facing triangle shows a significant difference in favor of the other algorithm. Bubbles indicate there was no significance determined.

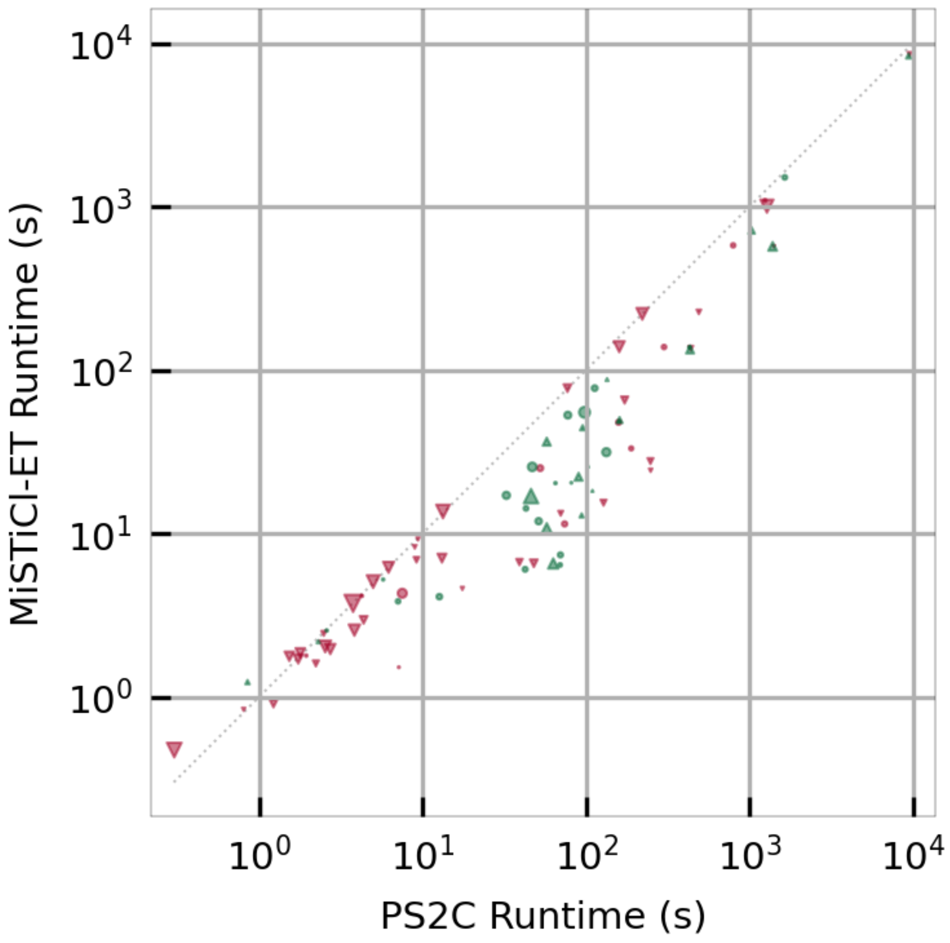


Fig. 6.6. Comparing PS2C against MiSTiCl regarding classification accuracy and runtime. Each marker represents one dataset. The x- and y-axis show the total runtime (training + testing) in seconds. Markers above the dotted line indicate that PS2C is faster than the other algorithm. A green marker indicates that PS2C provides better classification accuracy while red indicates otherwise. Marker sizes correspond to the absolute difference between the mean classification accuracy provided by the competing algorithms for a particular dataset, i.e., larger markers indicate greater difference in classification accuracy of the two algorithms. An upward facing triangle indicates that a significant difference was found in favor of PS2C, while a downward facing triangle shows a significant difference in favor of the other algorithm. Bubbles indicate there was no significance determined.

the same computational effort with either of the two datasets, while the other group requires less computational cost when evaluated with MiSTiCl although the difference is less than an order of magnitude. Overall, we see that PS2C also provides an improvement of at least an order of magnitude with respect to runtime, while it is on par with BOSS and dominates BoP and SAX-VSM regarding classification accuracy similar to MiSTiCl.

6.5 Conclusion

We have introduced a pattern sampling algorithm for time series classification. This is, to the best of our knowledge, the first algorithm for time series classification to be based on pattern sampling. The pattern sampler is used in a shapelet based classification algorithm. It was demonstrated that pattern sampling can be an effective alternative to the exhaustive shapelet/pattern discovery processes, since it enables to extract frequent patterns based on a quality measure to counteract the pattern explosion phenomenon. We used a multi-resolution feature set creation approach in our experiments, since it is proven to be highly effective. Our pattern sampling based algorithm was mostly on par with other similarly structured algorithms regarding classification accuracy. In terms of computational cost, our approach is slightly slower than MiSTiCl, however, the complexity analysis indicates the asymptotic complexity for our approach is similar to that of MiSTiCl, implying that the proposed method is also faster than the other algorithms.

Shapelet based time series classification gives rise to explainable classifications by construction. Therefore, the proposed pattern sampler is another option for constructing interpretable feature sets for time series. Interesting combinations with deep neural networks, especially for smaller sized datasets, remain a topic for future research [113].

There are a few optimizations that have been identified as further future avenues to be explored. We need to explore class-correlated pattern sampling in order to improve the accuracy in cases where the pattern sampler keeps providing patterns for one or a few classes rather than for the majority of classes. We can also experiment with fuzzy pattern sampling to diversify the identified feature set per pattern sampler.

7 Conclusion

Time series data is ubiquitous, and the digital age has enabled us to record gargantuan amounts of this data. The readily available time series data underscores the development of fast and efficient yet widely applicable data mining algorithms. Most research work in time series mining overwhelmingly emphasizes accuracy metrics but fails to report the (usually huge) runtime requirements. The development of efficient algorithms will enable the deployment of knowledge discovery infrastructures on a large scale.

This thesis is on the premise that metaheuristics and randomized approaches can provide the underpinnings for efficient yet accurate data mining algorithms for time series data. We introduced four such algorithms targeting time series classification, with a specific aim to reduce computational costs while being on par with previously developed methods. The proposed methods were evaluated on the datasets constituting the *UCR Archive* since the archive has attained the de facto status of a "touchstone" for time series classification research. The empirical evaluations showed that our proposed algorithms are one to two orders of magnitude faster than the previous state-of-the-art methods while providing on par or better classification accuracy.

7.1 Contributions

7.1.1 Randomized Time Warping

Dynamic Time Warping (DTW) is an exact distance measure for time series data. Its quadratic computational cost can become a limiting factor in deploying it in resource-limited scenarios. Many applications require reasonably accurate but fast time series classification, therefore, a quick and approximate distance measure is more useful in these cases than a slow and exact one. Examples of such scenarios include time series classification on low power edge computing devices, e.g., smart watches, heart rate monitors, etc.

Chapter 3 introduced Randomized Time Warping (RTW) for calculating an approximate distance between time series instances. RTW employs a greedy look-ahead approach to calculate the warping distance between two time series instances. Compared to the pure greedy approach of Lucky Time Warping (LTW), RTW provides much better accuracy without a considerable increase in the runtime requirements. On the other hand, RTW provides on par classification accuracy compared to DTW as well as being two to eight times faster than the exact algorithm. Therefore, it can act as a perfect drop-in replacement for DTW in applications that require fast and reasonably accurate results.

7.1.2 Randomized Shapelet Ensembles

The YK-shapelets algorithm is the seminal method for subsequences-based time series classification. It provides interpretable classification models that have better classification accuracy compared to the 1NN algorithm coupled with ED or DTW. The only downside of the YK-shapelets algorithm is the $O(N^2 n^4)$ training time complexity, where N is the number of time series training instances and n is the length of each time series.

Chapter 4 introduced Random-Shapelets Ensembles (RSE) to mitigate the huge computational complexity of the YK-shapelets algorithm. RSE employs a randomized sampling approach for evaluating candidate subsequences. A sampling ratio controls the amount of afforded computational resources, however, exceedingly small sampling ratios can lead to deteriorated classification accuracy. Therefore, RSE creates an ensemble of multiple randomized shapelet-based classifiers to offset the accuracy loss. An incremental approach towards the creation of the ensemble also allows to employ the RSE algorithm in a contractual manner, i.e., create the ensemble until achieving an acceptable level of accuracy or reaching the imposed limits on the computational cost. Compared to the YK-shapelets algorithm, RSE is one to two orders of magnitude faster and provides on par classification accuracy as well.

7.1.3 Mining Strings for Time Series Classification

Early shapelet-based classification algorithms classified time series instances by determining the presence or absence of shapelets in a time series instance. Shapelet Transform (ST) introduced the idea of using shapelets to transform a time series classification problem into a feature-based classification problem. This novel idea allowed to create classification models with any off-the-shelf algorithm. Since ST relied on the same shapelet discovery procedure as the YK-shapelets algorithm, its asymptotic time complexity was also $O(N^2 n^4)$.

In order to scale up shapelet-based time series classification to massive amounts of data, the algorithmic complexity of the shapelet discovery process has to be reduced drastically. The time series mining community has been exploring text mining algorithms for time series classification using discretized time series data in the hopes of reducing the overall computational costs. The dictionary-based approaches have been of little help though, as the asymptotic complexity of the proposed methods is still on the order of a cubic or higher-order polynomial.

Chapter 5 introduced the MiSTiCl algorithm (short for Mining Strings for Time Series Classification) that also transforms the time series classification problem into a feature-based classification problem. MiSTiCl extracts frequent patterns from discretized time series data using a linear time and space string mining algorithm. The frequent patterns are further filtered on the basis of their discriminative power, and the most discriminative patterns end up as features in the transformed dataset. The linear time complexity of the underlying pattern miner allowed us to incorporate a multi-resolution approach into the MiSTiCl algorithm without any noticeable impact on its runtime performance. MiSTiCl is one to two orders of magnitude faster than any other state-of-the-art pattern-based time series classifier while its classification accuracy is on par or better than the competition.

7.1.4 Pattern Sampling for Time Series Classification

Pattern-based time series classification algorithms provide accurate and interpretable classification models, but training these models is extremely computation intensive. MiSTiCl has been shown to be the fastest pattern-based time series classification algorithm, however, the pattern extraction phase of MiSTiCl can also become a bottleneck for larger datasets because the string mining algorithm underlying MiSTiCl performs a simple enumeration of all the possible patterns in the discretized dataset. The exponential increase in the possible patterns that have to be enumerated is called the pattern explosion problem.

Pattern sampling is an effective technique for mitigating the pattern explosion problem. It allows to sample patterns one by one based on a probability distribution that is proportional to a given quality measure. A pattern sampler can be grown incrementally as well, therefore, it allows to sample patterns while it learns from more data.

Chapter 6 introduced the PS2C algorithm (short for Pattern Sampling for Shapelet-based Time Series Classification) which is based on a pattern sampling approach for feature extraction from discretized time series data. PS2C is, to the best of our knowledge, the first time series algorithm that has a pattern sampler at its core for feature extraction. The pattern sampler is used to sample patterns that end up as features in the transformed dataset. The PS2C algorithm is also a multi-resolution approach. PS2C is one to two orders of magnitude faster than previous state-of-the-art pattern-based time series classifiers while its classification accuracy is on par with the other algorithms. Compared to MiSTiCl, PS2C provides on par classification accuracy using roughly the same computational resources.

7.2 Outlook

This thesis is concerned with the development of fast and efficient time series classification algorithms using metaheuristics and randomized approaches. In this regard, we developed several methods targeting univariate time series data and showed that metaheuristics and randomized approaches can indeed, be used to design efficient algorithms for time series data. Time series mining is still a growing field, although the first research regarding time series data appeared almost three decades ago. This leaves a lot of room for future work as well.

A logical continuation of this research work is the adaptation of the developed methods for multivariate time series data. The bulk of previous research work in the field has targeted univariate time series data, however, multivariate time series data is getting considerable attention now, and rightly so, because developing data mining algorithms aimed at this data type is not trivial. Multivariate time series data usually has an intricate set of interactions within the different data channels. The current time series mining algorithms aimed at multivariate time series data handle the different channels independently of each other, however, this ignores the intra-channel interactions. Future algorithms would have to figure out ways to incorporate the intra-channel interactions in the knowledge discovery process.

The emergence of newer technologies, e.g., the Internet of Things and sensor networks, introduces even further challenging tasks from a data mining perspective. Knowledge discovery from streaming time series data is one such challenge. Streaming data requires real-time analysis and involves messy, large-scale, intermittent, and/or volatile data. Handling these aspects and any other challenges, will require the use of ideas from various domains of data mining research. Matrix Profile and its various application-specific variants have gained a lot of attention since the introduction of the algorithm in 2016 [88]. The versatility of the Matrix Profile algorithm, coupled with its ability to be employed in a parameter-free setting, allows to evaluate any given time series to gain insights and extract motifs and discords, etc. Evaluating the streaming time series data using Matrix Profile can help to create systems that rely on edge computing to extract knowledge without requiring a large-scale central computing infrastructure.

Time series mining using deep learning strategies has also started gaining traction. The dominance of deep learning in other application domains has also lured in the time series mining community. A review of the deep learning architectures for time series mining was provided by Ismail Fawaz *et al.* in 2019 [85]. In 2020, two deep learning-based time series classification algorithms, namely *ROCKET* and *InceptionTime*, were proposed independently, and mark the state-of-the-art in deep learning-based time series classification algorithms [86, 87]. Overall, the deep learning-based schemes have caught up with previously developed approaches in terms of classification accuracy, and it is safe to assume that a lot of future research regarding time series mining will employ deep learning architectures. Shapelet-based time series classifiers provide simple and easily interpretable results, while neural network-based methods lack this ability altogether. The output of neural networks has to be presented and interpreted separately using various techniques specifically developed for this task. The current NN-based models do not offer simple explanations for classifying certain instances. On the other hand, interpreting and explaining shapelet-based classifiers is trivial. For a given time series, the presence or absence of one or more shapelets is enough to classify the instance. Domain experts can dissect entire classification models to verify the significance of shapelets used for model induction. In this regard, shapelet-based time series classifiers will continue to hold relevance until NN-based models can directly and intuitively explain their results.

A Calculating Independence Test Statistics

The discriminative power of a given pattern can be determined using a number of tests including, but not limited to, the χ^2 independence test, Information Gain values, etc. The discriminative power of a pattern tells us how effectively it can identify the instances of a given class. This section elaborates the procedure of calculating these statistics based on the occurrence frequency of a pattern in the positive and negative class dataset splits.

Let us consider a binary class problem where the positive and negative class dataset splits are represented by $\hat{P}$ and $\hat{N}$, respectively. Let p be a candidate pattern which occurs in $\hat{P}$ and $\hat{N}$ with relative frequencies $f_{\hat{P}}$ and $f_{\hat{N}}$, respectively.

A.1 Calculating the χ^2 Test Statistic

The χ^2 test statistic is calculated based on observed (O_{ij}) and expected (E_{ij}) values for the given categorical variables. The formula for calculating the χ^2 statistic is given below.

$$\chi^2 = \sum \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

Observed values (O_{ij}) correspond to the number of instances observed as belonging to a certain categorical variable. In our case, the categorical variable is the presence or absence of the candidate pattern p and the observed values are the number of instances in $\hat{P}$ and $\hat{N}$ containing and not containing the candidate pattern p. These values can be determined using the instance counts of $\hat{P}$ and $\hat{N}$ and occurrence frequency values of the given pattern in the respective dataset splits. Based on these values, a contingency table can be created as follows.

	Dataset split	
	Positive, $\hat{P}$	Negative, $\hat{N}$
$With(p)$	$O_{11} = \lfloor f_{\hat{P}} \times \lvert\hat{P}\rvert \rfloor$	$O_{12} = \lfloor f_{\hat{N}} \times \lvert\hat{N}\rvert \rfloor$
$WithOut(p)$	$O_{21} = \lvert\hat{P}\rvert - O_{11}$	$O_{22} = \lvert\hat{N}\rvert - O_{12}$

The expected values (E_{ij}) are calculated as follows:

$$E_{ij} = \frac{\sum_{i*} \times \sum_{*j}}{n},$$

where $\sum_{i*} = O_{i1} + O_{i2}$ represents the sum of the cells in row i and $\sum_{*j} = O_{1j} + O_{2j}$ represents the sum of cells in column j.

The combined total of row and column sums equals the total number of instances in the positive and negative dataset splits. The χ^2 test statistic determines if any relationship between the positive and negative dataset splits exists given the frequent pattern. If the pattern occurs in both datasets, then the χ^2 value will be close to zero which signifies a relationship exists between the two dataset splits. We can order the frequent patterns based on their χ^2 statistic and select only those for which the dataset splits do not exhibit any mutual relationship.

A.2 Calculating the Information Gain value

Entropy (H) is a measure for establishing whether a dataset has a uniform or varying distribution in terms of the different classes of instances. Given a dataset with positive and negative class instances, the entropy of the dataset can be calculated using the following formula.

$$H = -\left(\frac{|\hat{P}|}{|\hat{P}| + |\hat{N}|} \times log_2 \frac{|\hat{P}|}{|\hat{P}| + |\hat{N}|}\right) - \left(\frac{|\hat{N}|}{|\hat{P}| + |\hat{N}|} \times log_2 \frac{|\hat{N}|}{|\hat{P}| + |\hat{N}|}\right)$$

If a pattern p occurs frequently in either class of instances in the dataset, we can create positive and negative class subsets based on the presence or absence of this pattern in each of the instances. The entropy of these subsets can then be calculated using the following equations.

$$\begin{aligned} H_{\hat{P}} = &-\left(\frac{f_{\hat{P}} \times |\hat{P}|}{f_{\hat{P}} \times |\hat{P}| + f_{\hat{N}} \times |\hat{N}|} \times log_2 \frac{f_{\hat{P}} \times |\hat{P}|}{f_{\hat{P}} \times |\hat{P}| + f_{\hat{N}} \times |\hat{N}|}\right) \\ &-\left(\frac{f_{\hat{N}} \times |\hat{N}|}{f_{\hat{P}} \times |\hat{P}| + f_{\hat{N}} \times |\hat{N}|} \times log_2 \frac{f_{\hat{N}} \times |\hat{N}|}{f_{\hat{P}} \times |\hat{P}| + f_{\hat{N}} \times |\hat{N}|}\right) \end{aligned}$$

$$\begin{aligned} H_{\hat{N}} = &-\left(\frac{f_{\hat{N}} \times |\hat{P}|}{f_{\hat{N}} \times |\hat{P}| + f_{\hat{P}} \times |\hat{N}|} \times log_2 \frac{f_{\hat{N}} \times |\hat{P}|}{f_{\hat{N}} \times |\hat{P}| + f_{\hat{P}} \times |\hat{N}|}\right) \\ &-\left(\frac{f_{\hat{P}} \times |\hat{N}|}{f_{\hat{N}} \times |\hat{P}| + f_{\hat{P}} \times |\hat{N}|} \times log_2 \frac{f_{\hat{P}} \times |\hat{N}|}{f_{\hat{N}} \times |\hat{P}| + f_{\hat{P}} \times |\hat{N}|}\right) \end{aligned}$$

Using the entropy values of the source dataset and the positive and negative subsets, we can calculate the Information Gain value using the following formula.

$$IG = H - \left(\frac{f_{\hat{P}} \times |\hat{P}| + f_{\hat{N}} \times |\hat{N}|}{|\hat{P}| + |\hat{N}|} \times H_{\hat{P}} + \frac{f_{\hat{N}} \times |\hat{P}| + f_{\hat{P}} \times |\hat{N}|}{|\hat{P}| + |\hat{N}|} \times H_{\hat{N}}\right)$$

If the frequent pattern effectively distinguishes between the two classes, the positive and negative class subsets will have very few or no instances of the other class, resulting in a smaller value of entropy for the two subsets. This in turn will cause a higher Information Gain value, indicating that the pattern is a good candidate for distinguishing between the two classes of instances. If, however, the converse is true, then the pattern is not a good candidate. This way, the candidates can be selected on the basis of their discriminative power.

Bibliography

[1] Zaraza Friedman. “Nilometer”. In: *Encyclopaedia of the History of Science, Technology, and Medicine in Non-Western Cultures*. Ed. by Helaine Selin. Dordrecht: Springer Netherlands, 2016, pp. 3386–3404. ISBN: 978-94-007-7747-7. DOI: 10.1007/978-94-007-7747-7_9644 (cit. on p. 1).

[2] Chotirat Ann Ratanamahatana *et al.* “Mining Time Series Data”. In: *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2009, pp. 1049–1077. ISBN: 978-0-387-09823-4. DOI: 10.1007/978-0-387-09823-4_56 (cit. on pp. 2, 3).

[3] Tak-chung Fu. “A Review on Time Series Data Mining”. In: *Engineering Applications of Artificial Intelligence* 24.1 (Feb. 2011), pp. 164–181. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2010.09.007 (cit. on pp. 2, 3, 14).

[4] Philippe Esling and Carlos Agon. “Time-Series Data Mining”. In: *ACM Computing Surveys*. CSUR 45.1 (Nov. 2012), p. 34. ISSN: 0360-0300. DOI: 10.1145/2379776.2379788 (cit. on pp. 2, 3).

[5] Qiang Yang and Xindong Wu. “10 CHALLENGING PROBLEMS IN DATA MINING RESEARCH”. In: *International Journal of Information Technology & Decision Making* 5.4 (2006), pp. 597–604. ISSN: 1793-6845. DOI: 10.1142/S0219622006002258 (cit. on p. 3).

[6] Nizar R. Mabroukeh and C. I. Ezeife. “A Taxonomy of Sequential Pattern Mining Algorithms”. In: *ACM Computing Surveys* 43.1 (Nov. 2010), pp. 1–41. ISSN: 0360-0300. DOI: 10.1145/1824795.1824798 (cit. on pp. 3, 74).

[7] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. “Efficient Similarity Search in Sequence Databases”. In: *Foundations of Data Organization and Algorithms, FODO ’93*. Ed. by D.B. Lomet. Vol. 730. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, Oct. 1993, pp. 69–84. ISBN: 978-3-540-48047-1. DOI: 10.1007/3-540-57301-1_5 (cit. on pp. 4, 13, 19).

[8] Eamonn Keogh *et al.* “Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases”. In: *Knowledge and Information Systems*. KAIS 3.3 (Aug. 2001), pp. 263–286. ISSN: 0219-3116. DOI: 10.1007/PL00011669 (cit. on p. 4).

[9] Eamonn Keogh, Jessica Lin, and Ada Fu. “HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence”. In: *Fifth IEEE International Conference on Data Mining (ICDM’05)*. ICDM. Houston, TX: IEEE, Nov. 2005, pp. 226–233. ISBN: 0-7695-2278-5. DOI: 10.1109/ICDM.2005.79 (cit. on pp. 4, 80).

[10] Jessica Lin *et al.* “Experiencing SAX: A Novel Symbolic Representation of Time Series”. In: *Data Mining and Knowledge Discovery*. DMKD 15.2 (Oct. 2007), pp. 107–144. ISSN: 1573-756X. DOI: 10.1007/s10618-007-0064-z (cit. on pp. 4, 31, 80, 81).

[11] 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*. EDBT ’12. ACM, Mar. 2012, pp. 516–527. ISBN: 978-1-4503-0790-1. DOI: 10.1145/2247596.2247656 (cit. on pp. 4, 32, 91).

[12] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. “Fast Subsequence Matching in Time-Series Databases”. In: *SIGMOD Records*. SIGMOD 23.2 (June 1994), pp. 419–429. ISSN: 0163-5808. DOI: 10.1145/191843.191925 (cit. on pp. 4, 11, 19).

[13] T. Kahveci and A. Singh. “Variable Length Queries for Time Series Data”. In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2001, p. 0273. DOI: 10.1109/ICDE.2001.914838 (cit. on pp. 4, 11, 19).

[14] Kaushik Chakrabarti *et al.* “Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases”. In: *ACM Transactions on Database Systems* 27.2 (June 1, 2002), pp. 188–228. ISSN: 0362-5915. DOI: 10.1145/568518.568520 (cit. on pp. 4, 11, 17).

[15] I. Popivanov and R. J. Miller. “Similarity Search over Time-Series Data Using Wavelets”. In: *Proceedings 18th International Conference on Data Engineering*. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2002, p. 0212. DOI: 10.1109/ICDE.2002.994711 (cit. on pp. 4, 11, 19).

[16] Donald J. Berndt and James Clifford. “Using Dynamic Time Warping to Find Patterns in Time Series”. In: *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. AAAIWS’94. Seattle, WA: AAAI Press, July 1994, pp. 359–370. DOI: 10.5555/3000850.3000887 (cit. on pp. 4, 7, 19).

[17] Eamonn Keogh and Michael J Pazzani. “An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback”. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. KDD-98. New York, NY: AAAI Press, 1998, pp. 239–243. DOI: 10.5555/3000292.3000335 (cit. on pp. 4, 11, 12).

[18] Pierre Geurts. “Pattern Extraction for Time Series Classification”. In: *Principles of Data Mining and Knowledge Discovery*. Ed. by Luc De Raedt and Arno Siebes. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pp. 115–127. ISBN: 978-3-540-44794-8. DOI: 10.1007/3-540-44794-6_10 (cit. on pp. 4, 11).

[19] Anne Debregeas and Georges Hebrail. “Interactive Interpretation of Kohonen Maps Applied to Curves”. In: (1998), p. 5 (cit. on pp. 4, 11).

[20] John Aach and George M. Church. “Aligning Gene Expression Time Series with Time Warping Algorithms”. In: *Bioinformatics* 17.6 (June 1, 2001), pp. 495–508. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/17.6.495 (cit. on pp. 4, 11).

[21] K. Kalpakis, D. Gada, and V. Puttagunta. “Distance Measures for Effective Clustering of ARIMA Time-Series”. In: *Proceedings 2001 IEEE International Conference on Data Mining*. Nov. 2001, pp. 273–280. DOI: 10.1109/ICDM.2001.989529 (cit. on pp. 4, 11).

[22] Valery Guralnik and Jaideep Srivastava. “Event Detection from Time Series Data”. In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’99. New York, NY, USA: Association for Computing Machinery, Aug. 1, 1999, pp. 33–42. ISBN: 978-1-58113-143-7. DOI: 10.1145/312129.312190 (cit. on pp. 4, 12).

[23] C. Shahabi, X. Tian, and W. Zhao. “TSA-Tree: A Wavelet-Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries on Time-Series Data”. In: *Proceedings. 12th International Conference on Scientific and Statistica Database Management*. July 2000, pp. 55–68. DOI: 10.1109/SSDM.2000.869778 (cit. on pp. 4, 12, 19).

[24] Eamonn Keogh, Stefano Lonardi, and Bill ’Yuan-chi’ Chiu. “Finding Surprising Patterns in a Time Series Database in Linear Time and Space”. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’02. New York, NY, USA: Association for Computing Machinery, July 23, 2002, pp. 550–556. ISBN: 978-1-58113-567-1. DOI: 10.1145/775047.775128 (cit. on pp. 4, 12).

[25] J. J. Van Wijk and E. R. Van Selow. “Cluster and Calendar Based Visualization of Time Series Data”. In: *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis’99)*. Oct. 1999, pp. 4–9. DOI: 10.1109/INFVIS.1999.801851 (cit. on pp. 4, 11).

[26] Piotr Indyk, Nick Koudas, and S. Muthukrishnan. “Identifying representative trends in massive time series data sets using sketches”. In: *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB’00*. Proceedings of the 26th International Conference on Very Large Data Bases, VLDB’00. 2000, pp. 363–372. ISBN: 1-55860-715-3 (cit. on pp. 4, 11).

[27] H. Sakoe and S. Chiba. “Dynamic Programming Algorithm Optimization for Spoken Word Recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (Feb. 1978), pp. 43–49. ISSN: 0096-3518. DOI: 10.1109/TASSP.1978.1163055 (cit. on pp. 5, 21, 35).

[28] F. Itakura. “Minimum Prediction Residual Principle Applied to Speech Recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (Feb. 1975), pp. 67–72. ISSN: 0096-3518. DOI: 10.1109/TASSP.1975.1162641 (cit. on pp. 5, 21, 35).

[29] Byoung-Kee Yi, H.V. Jagadish, and Christos Faloutsos. “Efficient Retrieval of Similar Time Sequences under Time Warping”. In: *Proceedings 14th International Conference on Data Engineering*. ICDE ’98. IEEE Comput. Soc, Feb. 1998, pp. 201–208. ISBN: 0-8186-8289-2. DOI: 10.1109/ICDE.1998.655778 (cit. on pp. 5, 21, 35).

[30] Sang-Wook Kim, Sanghyun Park, and Wesley W. Chu. “An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases”. In: *Proceedings 17th International Conference on Data Engineering*. ICDE ’01. IEEE Comput. Soc, Apr. 2001, pp. 607–614. ISBN: 0-7695-1001-9. DOI: 10.1109/ICDE.2001.914875 (cit. on pp. 5, 21, 35).

[31] Eamonn Keogh. “Exact Indexing of Dynamic Time Warping”. In: *Proceedings of the 28th International Conference on Very Large Data Bases*. VLDB ’02. Aug. 2002, pp. 406–417 (cit. on pp. 5, 21).

[32] Lexiang Ye and Eamonn Keogh. “Time Series Shapelets: A New Primitive for Data Mining”. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’09. ACM Press, June 2009, pp. 947–956. ISBN: 978-1-60558-495-9. DOI: 10.1145/1557019.1557122 (cit. on pp. 5, 7, 24, 25, 29, 55, 56).

[33] Jason Lines *et al.* “A Shapelet Transform for Time Series Classification”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’12. ACM, Aug. 2012, pp. 289–297. ISBN: 978-1-4503-1462-6. DOI: 10.1145/2339530.2339579 (cit. on pp. 5, 25, 92).

[34] 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*. SDM. Society for Industrial and Applied Mathematics, May 2013, pp. 668–676. ISBN: 978-1-61197-283-2. DOI: 10.1137/1.9781611972832.74 (cit. on pp. 5, 31, 56, 73).

[35] Jessica Lin, Rohan Khade, and Yuan Li. “Rotation-Invariant Similarity in Time Series Using Bag-of-Patterns Representation”. In: *Journal of Intelligent Information Systems* 39.2 (Oct. 2012), pp. 287–315. ISSN: 1573-7675. DOI: 10.1007/s10844-012-0196-5 (cit. on pp. 5, 32, 73, 103).

[36] Pavel Senin and Sergey Malinchik. “SAX-VSM: Interpretable Time Series Classification Using SAX and Vector Space Model”. In: *13th International Conference on Data Mining*. ICDM ’13. IEEE, Dec. 2013, pp. 1175–1180. ISBN: 978-0-7695-5108-1. DOI: 10.1109/ICDM.2013.52 (cit. on pp. 5, 32, 73, 103).

[37] Patrick Schäfer. “The BOSS Is Concerned with Time Series Classification in the Presence of Noise”. In: *Data Mining and Knowledge Discovery*. DMKD 29.6 (Nov. 2015), pp. 1505–1530. ISSN: 1573-756X. DOI: 10.1007/s10618-014-0377-7 (cit. on pp. 5, 32, 73, 103).

[38] Patrick Schäfer. “Scalable Time Series Classification”. In: *Data Mining and Knowledge Discovery*. DMKD 30.5 (Sept. 2016), pp. 1273–1298. ISSN: 1573-756X. DOI: 10.1007/s10618-015-0441-y (cit. on pp. 5, 32, 73, 90, 103).

[39] Patrick Schäfer and Ulf Leser. “Fast and Accurate Time Series Classification with WEASEL”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM ’17*. New York, New York, USA: ACM Press, 2017, pp. 637–646. ISBN: 978-1-4503-4918-5. DOI: 10.1145/3132847.3132980 (cit. on pp. 5, 32).

[40] Christian Blum and Andrea Roli. “Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison”. In: *ACM Computing Surveys*. CSUR 35.3 (Sept. 2003), pp. 268–308. ISSN: 0360-0300. DOI: 10.1145/937503.937505 (cit. on p. 6).

[41] Leonora Bianchi *et al.* “A Survey on Metaheuristics for Stochastic Combinatorial Optimization”. In: *Natural Computing* 8.2 (June 1, 2009), pp. 239–287. ISSN: 1572-9796. DOI: 10.1007/s11047-008-9098-4 (cit. on p. 6).

[42] Franz Rothlauf. *Design of Modern Heuristics*. Natural Computing Series. Springer-Verlag Berlin Heidelberg, July 15, 2011. ISBN: 978-3-540-72962-4. DOI: 10.1007/978-3-540-72962-4 (cit. on p. 6).

[43] Sean Luke. *Essentials of Metaheuristics*. 2nd ed. 2013. ISBN: 978-1-300-54962-8 (cit. on p. 6).

[44] Zhigang Zheng *et al.* “An Efficient GA-Based Algorithm for Mining Negative Sequential Patterns”. In: *Advances in Knowledge Discovery and Data Mining, PAKDD 2010*. Vol. 6118. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, June 2010, pp. 262–273. ISBN: 978-3-642-13657-3. DOI: 10.1007/978-3-642-13657-3_30 (cit. on p. 6).

[45] Debahuti Mishra *et al.* “Genetic Algorithm Based Fuzzy Frequent Pattern Mining from Gene Expression Data”. In: *Soft Computing Techniques in Vision Science*. Vol. 395. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2012, pp. 1–14. ISBN: 978-3-642-25507-6. DOI: 10.1007/978-3-642-25507-6_1 (cit. on p. 6).

[46] Zakaria Suliman Zubi and Marim Aboajela Emsaed. “Sequence Mining in DNA Chips Data for Diagnosing Cancer Patients”. In: *Proceedings of the 10th International Conference on Applied Computer Science*. ACS ’10. 2010, pp. 139–151. ISBN: 978-960-474-231-8 (cit. on p. 6).

[47] M. Martínez-Ballesteros *et al.* “An Evolutionary Algorithm to Discover Quantitative Association Rules in Multidimensional Time Series”. In: *Soft Computing* 15.10 (Oct. 2011), pp. 2065–2084. ISSN: 1433-7479. DOI: 10.1007/s00500-011-0705-4 (cit. on p. 6).

[48] Meinard Müller. “Dynamic Time Warping”. In: *Information Retrieval for Music and Motion*. Springer Berlin Heidelberg, 2007, pp. 69–84. ISBN: 978-3-540-74048-3. DOI: 10.1007/978-3-540-74048-3_4 (cit. on p. 7).

[49] Johannes Fischer, Volker Heun, and Stefan Kramer. “Fast Frequent String Mining Using Suffix Arrays”. In: *5th International Conference on Data Mining*. ICDM ’05. IEEE, Nov. 2005, pp. 609–612. ISBN: 0-7695-2278-5. DOI: 10.1109/ICDM.2005.62 (cit. on pp. 8, 73, 75, 76).

[50] Johannes Fischer, Volker Heun, and Stefan Kramer. “Optimal String Mining Under Frequency Constraints”. In: *Knowledge Discovery in Databases, PKDD 2006*. Vol. 4213. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Sept. 2006, pp. 139–150. ISBN: 978-3-540-46048-0. DOI: 10.1007/11871637_17 (cit. on pp. 8, 73, 75–77).

[51] Jasbir Dhaliwal, Simon J. Puglisi, and Andrew Turpin. “Practical Efficient String Mining”. In: *IEEE Transactions on Knowledge and Data Engineering*. TKDE 24.4 (Apr. 2012), pp. 735–744. ISSN: 1041-4347. DOI: 10.1109/TKDE.2010.242 (cit. on pp. 8, 74–76).

[52] Romain Briandet, E. Katherine Kemsley, and Reginald H. Wilson. “Discrimination of Arabica and Robusta in Instant Coffee by Fourier Transform Infrared Spectroscopy and Chemometrics”. In: *Journal of Agricultural and Food Chemistry* 44.1 (Jan. 18, 1996), pp. 170–174. ISSN: 0021-8561. DOI: 10.1021/jf950305a (cit. on p. 13).

[53] Eamonn Keogh and Shruti Kasetty. “On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration”. In: *Data Mining and Knowledge Discovery*. DMKD 7.4 (Oct. 2003), pp. 349–371. ISSN: 1573-756X. DOI: 10.1023/A:1024988512476 (cit. on p. 13).

[54] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml (cit. on p. 13).

[55] Hoang Anh Dau *et al. UCR Time Series Classification Archive*. Oct. 2018. URL: https://www.cs.ucr.edu/~eamonn/time_series_data_2018/ (cit. on pp. 13, 14, 23, 46, 64, 91, 114, 125).

[56] Hoang Anh Dau *et al. The UCR Time Series Archive*. Sept. 8, 2019. arXiv: 1810.07758 [cs, stat]. URL: http://arxiv.org/abs/1810.07758 (visited on 09/27/2020) (cit. on pp. 13, 91, 114).

[57] Anthony Bagnall *et al. UEA Time Series Repository*. URL: https://github.com/uea-machine-learning/tsml (cit. on pp. 13, 14, 64, 90, 92, 114, 115).

[58] Eamonn Keogh *et al. UCR Time Series Classification Archive*. 2014. URL: http://www.cs.ucr.edu/~eamonn/time_series_data/ (cit. on pp. 14, 36, 44, 50, 64).

[59] Yanping Chen *et al. UCR Time Series Classification Archive*. July 2015. URL: http://www.cs.ucr.edu/~eamonn/time_series_data/ (cit. on p. 14).

[60] Mark Hall *et al.* “The WEKA Data Mining Software”. In: *SIGKDD Explorations* 11.1 (2009), pp. 10–18 (cit. on p. 14).

[61] T. Warren Liao. “Clustering of Time Series Data—a Survey”. In: *Pattern Recognition*. PatCog 38.11 (Nov. 2005), pp. 1857–1874. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2005.01.025 (cit. on p. 14).

[62] Xiaoyue Wang *et al.* “Experimental Comparison of Representation Methods and Distance Measures for Time Series Data”. In: *Data Mining and Knowledge Discovery*. DMKD 26.2 (Mar. 2013), pp. 275–309. ISSN: 1573-756X. DOI: 10.1007/s10618-012-0250-5 (cit. on pp. 14, 19).

[63] Joan Serrà and Josep Lluis Arcos. “An Empirical Evaluation of Similarity Measures for Time Series Classification”. In: *Knowledge-Based Systems*. KnoSys 67 (Sept. 2014), pp. 305–314. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2014.04.035 (cit. on p. 14).

[64] Dina Q. Goldin and Paris C. Kanellakis. “On Similarity Queries for Time-Series Data: Constraint Specification and Implementation”. In: *Principles and Practice of Constraint Programming — CP ’95*. Ed. by Ugo Montanari and Francesca Rossi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 137–153. ISBN: 978-3-540-44788-7. DOI: 10.1007/3-540-60299-2_9 (cit. on p. 16).

[65] Rakesh Agrawal *et al.* “Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases”. In: *Proceedings of the 21th International Conference on Very Large Data Bases*. VLDB ’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Sept. 11, 1995, pp. 490–501. ISBN: 978-1-55860-379-0 (cit. on p. 16).

[66] Kin-Pong Chan and Ada Wai-Chee Fu. “Efficient Time Series Matching by Wavelets”. In: *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*. Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337). Mar. 1999, pp. 126–133. DOI: 10.1109/ICDE.1999.754915 (cit. on p. 19).

[67] Changzhou Wang and X. Sean Wang. “Supporting Content-Based Searches on Time Series via Approximation”. In: *Proceedings. 12th International Conference on Scientific and Statistica Database Management*. Proceedings. 12th International Conference on Scientific and Statistica Database Management. July 2000, pp. 69–81. DOI: 10.1109/SSDM.2000.869779 (cit. on p. 19).

[68] Yi-Leh Wu, Divyakant Agrawal, and Amr El Abbadi. “A Comparison of DFT and DWT Based Similarity Search in Time-Series Databases”. In: *Proceedings of the Ninth International Conference on Information and Knowledge Management*. CIKM ’00. New York, NY, USA: Association for Computing Machinery, Nov. 6, 2000, pp. 488–495. ISBN: 978-1-58113-320-2. DOI: 10.1145/354756.354857 (cit. on p. 19).

[69] Yuhan Cai and Raymond Ng. “Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials”. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 599–610. ISBN: 1-58113-859-8. DOI: 10.1145/1007568.1007636 (cit. on p. 19).

[70] P. Marteau. “Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.2 (Feb. 2009), pp. 306–318. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2008.76 (cit. on p. 19).

[71] Lei Chen and Raymond Ng. “On the Marriage of Lp-Norms and Edit Distance”. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. VLDB ’04. Toronto, Canada: VLDB Endowment, Aug. 31, 2004, pp. 792–803. ISBN: 978-0-12-088469-8 (cit. on p. 19).

[72] Joan Serrà and Josep Lluís Arcos. “A Competitive Measure to Assess the Similarity between Two Time Series”. In: *Case-Based Reasoning Research and Development*. Ed. by Belén Díaz Agudo and Ian Watson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 414–427. ISBN: 978-3-642-32986-9. DOI: 10.1007/978-3-642-32986-9_31 (cit. on p. 19).

[73] Stephan Spiegel, Brijnesh-Johannes Jain, and Sahin Albayrak. “Fast Time Series Classification Under Lucky Time Warping Distance”. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. SAC ’14. ACM, Mar. 2014, pp. 71–78. ISBN: 978-1-4503-2469-4. DOI: 10.1145/2554850.2554885 (cit. on pp. 19, 21, 35, 36).

[74] Christian Hundt *et al.* “GEM: An Elastic and Translation-Invariant Similarity Measure with Automatic Trend Adjustment”. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. SAC ’14. ACM, Mar. 2014, pp. 105–112. ISBN: 978-1-4503-2469-4. DOI: 10.1145/2554850.2555041 (cit. on pp. 19, 22, 23).

[75] Eamonn Keogh and Chotirat Ann Ratanamahatana. “Exact Indexing of Dynamic Time Warping”. In: *Knowledge and Information Systems*. KAIS 7.3 (Mar. 2005), pp. 358–386. ISSN: 0219-3116. DOI: 10.1007/s10115-004-0154-9 (cit. on pp. 21, 35).

[76] Eamonn Keogh and Michael J. Pazzani. “Derivative Dynamic Time Warping”. In: *Proceedings of the 2001 SIAM International Conference on Data Mining*. SDM. Society for Industrial and Applied Mathematics, Apr. 2001, pp. 1–11. ISBN: 978-1-61197-271-9. DOI: 10.1137/1.9781611972719.1 (cit. on p. 21).

[77] Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. “Weighted Dynamic Time Warping for Time Series Classification”. In: *Pattern Recognition*. PatCog 44.9 (Sept. 2011), pp. 2231–2240. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2010.09.022 (cit. on p. 21).

[78] Abdullah Mueen *et al.* “AWarp: Fast Warping Distance for Sparse Time Series”. In: *16th International Conference on Data Mining*. ICDM ’16. IEEE, Dec. 2016, pp. 350–359. ISBN: 978-1-5090-5473-2. DOI: 10.1109/ICDM.2016.0046 (cit. on p. 21).

[79] Stan Salvador and Philip Chan. “Toward Accurate Dynamic Time Warping in Linear Time and Space”. In: *Intelligent Data Analysis* 11.5 (Oct. 2007), pp. 561–580. ISSN: 1088-467X. DOI: 10.3233/IDA-2007-11508 (cit. on pp. 21, 35).

[80] Hui Ding *et al.* “Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures”. In: *Proceedings of the VLDB Endowment* 1.2 (Aug. 2008), pp. 1542–1552. ISSN: 2150-8097. DOI: 10.14778/1454159.1454226 (cit. on pp. 24, 35).

[81] 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*. KDD ’11. ACM Press, Aug. 2011, pp. 1154–1162. ISBN: 978-1-4503-0813-7. DOI: 10.1145/2020408.2020587 (cit. on pp. 31, 56).

[82] Xavier Renard *et al.* “Random-Shapelet: An Algorithm for Fast Shapelet Discovery”. In: *2015 IEEE International Conference on Data Science and Advanced Analytics*. DSAA ’15. IEEE, Oct. 2015, pp. 1–10. ISBN: 978-1-4673-8273-1. DOI: 10.1109/DSAA.2015.7344782 (cit. on p. 31).

[83] Thach Le Nguyen, Severin Gsponer, and Georgiana Ifrim. “Time Series Classification by Sequence Learning in All-Subsequence Space”. In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, Apr. 2017, pp. 947–958. ISBN: 978-1-5090-6543-1. DOI: 10.1109/ICDE.2017.142 (cit. on pp. 32, 103).

[84] Thach Le Nguyen *et al.* “Interpretable Time Series Classification Using Linear Models and Multi-Resolution Multi-Domain Symbolic Representations”. In: *Data Mining and Knowledge Discovery* 33.4 (May 21, 2019), pp. 1183–1222. ISSN: 1573-756X. DOI: 10.1007/s10618-019-00633-3 (cit. on pp. 32, 103).

[85] Hassan Ismail Fawaz *et al.* “Deep Learning for Time Series Classification: A Review”. In: *Data Mining and Knowledge Discovery* 33.4 (July 1, 2019), pp. 917–963. ISSN: 1573756X. DOI: 10.1007/s10618-019-00619-1. arXiv: 1809.04356 (cit. on pp. 33, 129).

[86] Angus Dempster, François Petitjean, and Geoffrey I. Webb. “ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels”. In: *Data Mining and Knowledge Discovery* (July 13, 2020), pp. 1–42. ISSN: 1573756X. DOI: 10.1007/s10618-020-00701-z. arXiv: 1910.13051 (cit. on pp. 33, 130).

[87] Hassan Ismail Fawaz *et al.* “InceptionTime: Finding AlexNet for Time Series Classification”. In: *Data Mining and Knowledge Discovery* 34.6 (Nov. 1, 2020), pp. 1936–1962. ISSN: 1573-756X. DOI: 10.1007/s10618-020-00710-y (cit. on pp. 33, 130).

[88] Chin-chia Michael Yeh *et al.* “Matrix Profile 1: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets”. In: *16th International Conference on Data Mining*. ICDM ’16. IEEE, Dec. 2016, pp. 1317–1322. ISBN: 978-1-5090-5473-2. DOI: 10.1109/ICDM.2016.0179 (cit. on pp. 34, 129).

[89] Chin-chia Michael Yeh *et al.* “Time Series Joins, Motifs, Discords and Shapelets: A Unifying View That Exploits the Matrix Profile”. In: *Data Mining and Knowledge Discovery* 32.1 (Jan. 24, 2018), pp. 83–123. ISSN: 1384-5810. DOI: 10.1007/s10618-017-0519-9 (cit. on p. 34).

[90] Anthony Bagnall and Jason Lines. “An Experimental Evaluation of Nearest Neighbour Time Series Classification”. In: *CoRR* abs/1406.4 (June 2014), p. 7. arXiv: 1406.4757 (cit. on p. 35).

[91] Chotirat Ann Ratanamahatana and Eamonn Keogh. “Making Time-Series Classification More Accurate Using Learned Constraints”. In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. SDM. Society for Industrial and Applied Mathematics, Apr. 2004, pp. 11–22. ISBN: 978-1-61197-274-0. DOI: 10.1137/1.9781611972740.2 (cit. on p. 35).

[92] Xiaopeng Xi *et al.* “Fast Time Series Classification Using Numerosity Reduction”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML ’06. ACM Press, June 2006, pp. 1033–1040. ISBN: 1-59593-383-2. DOI: 10.1145/1143844.1143974 (cit. on p. 35).

[93] Krisztian Buza, Alexandros Nanopoulos, and Lars Schmidt-Thieme. “INSIGHT: Efficient and Effective Instance Selection for Time-Series Classification”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Joshua Zhexue Huang, Longbing Cao, and Jaideep Srivastava. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 149–160. ISBN: 978-3-642-20847-8. DOI: 10.1007/978-3-642-20847-8_13 (cit. on p. 35).

[94] Zhengzheng Xing, Jian Pei, and Philip S. Yu. “Early Prediction on Time Series: A Nearest Neighbor Approach”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. IJCAI ’09. July 2009, pp. 1297–1302 (cit. on p. 36).

[95] Scott MacLean and George Labahn. “Elastic Matching in Linear Time and Constant Space”. In: International Workshop on Document Analysis Systems. 2010, p. 4 (cit. on p. 36).

[96] Gustavo E.A.P.A. Batista, Xiaoyue Wang, and Eamonn Keogh. “A Complexity-Invariant Distance Measure for Time Series”. In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. SDM. Society for Industrial and Applied Mathematics, Apr. 2011, pp. 699–710. ISBN: 978-1-61197-281-8. DOI: 10.1137/1.9781611972818.60 (cit. on p. 45).

[97] Josif Grabocka *et al.* “Learning Time-Series Shapelets”. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’14. ACM, Aug. 2014, pp. 392–401. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623613 (cit. on p. 56).

[98] Abdullah Mueen *et al.* “Exact Discovery of Time Series Motifs”. In: *Proceedings of the 2009 SIAM International Conference on Data Mining*. SDM. Society for Industrial and Applied Mathematics, Apr. 2009, pp. 473–484. ISBN: 978-1-61197-279-5. DOI: 10.1137/1.9781611972795.41 (cit. on p. 56).

[99] L. K. Hansen and P. Salamon. “Neural Network Ensembles”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (Oct. 1, 1990), pp. 993–1001. ISSN: 0162-8828. DOI: 10.1109/34.58871 (cit. on p. 58).

[100] Leo Breiman. “Bagging Predictors”. In: *Machine Learning*. ML 24.2 (Aug. 1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1007/BF00058655 (cit. on p. 59).

[101] Yoav Freund. “Boosting a Weak Learning Algorithm by Majority”. In: *Information and Computation* 121.2 (Sept. 1995), pp. 256–285. ISSN: 0890-5401. DOI: 10.1006/inco.1995.1136 (cit. on pp. 60, 92, 115).

[102] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. “BRAID: Stream Mining through Group Lag Correlations”. In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’05. New York, NY, USA: Association for Computing Machinery, June 14, 2005, pp. 599–610. ISBN: 978-1-59593-060-6. DOI: 10.1145/1066157.1066226 (cit. on p. 62).

[103] J Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *Journal of Machine Learning Research*. JMLR 7 (Dec. 2006), pp. 1–30. ISSN: 1533-7928 (cit. on pp. 65, 92, 115).

[104] Hannu Toivonen. “Frequent Pattern”. In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2017, pp. 524–529. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_318 (cit. on p. 84).

[105] Leo Breiman. “Random Forests”. In: *Machine Learning*. ML 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324 (cit. on pp. 92, 115).

[106] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely Randomized Trees”. In: *Machine Learning* 63.1 (Apr. 2006), pp. 3–42. ISSN: 1573-0565. DOI: 10.1007/s10994-006-6226-1 (cit. on pp. 92, 115).

[107] Jon Hills *et al.* “Classification of Time Series by Shapelet Transformation”. In: *Data Mining and Knowledge Discovery*. DMKD 28.4 (July 2014), pp. 851–881. ISSN: 1573-756X. DOI: 10.1007/s10618-013-0322-1 (cit. on pp. 92, 105).

[108] Toon Calders, Christophe Rigotti, and Jean-François Boulicaut. “A Survey on Condensed Representations for Frequent Sets”. In: *Constraint-Based Mining and Inductive Databases*. Ed. by Jean-François Boulicaut, Luc De Raedt, and Heikki Mannila. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 64–80. ISBN: 978-3-540-31351-9 (cit. on p. 104).

[109] Albrecht Zimmermann and Siegfried Nijssen. “Supervised Pattern Mining and Applications to Classification”. In: *Frequent Pattern Mining*. Ed. by Charu C. Aggarwal and Jiawei Han. Cham: Springer International Publishing, 2014, pp. 425–442. ISBN: 978-3-319-07821-2. DOI: 10.1007/978-3-319-07821-2_17 (cit. on p. 104).

[110] Siegfried Nijssen and Albrecht Zimmermann. “Constraint-Based Pattern Mining”. In: *Frequent Pattern Mining*. Ed. by Charu C. Aggarwal and Jiawei Han. Cham: Springer International Publishing, 2014, pp. 147–163. ISBN: 978-3-319-07821-2. DOI: 10.1007/978-3-319-07821-2_7 (cit. on p. 104).

[111] Björn Bringmann *et al.* "Mining Sets of Patterns". In: *Tutorial at ECMLPKDD* (2010) (cit. on p. 104).

[112] Vladimir Dzyuba, Matthijs van Leeuwen, and Luc De Raedt. "Flexible Constrained Sampling with Guarantees for Pattern Mining". In: *Data Mining and Knowledge Discovery* 31.5 (Sept. 2017), pp. 1266–1293. ISSN: 1573-756X. DOI: 10.1007/s10618-017-0501-6. arXiv: 1610.09263 (cit. on pp. 104, 107).

[113] Stefan Kramer. "A Brief History of Learning Symbolic Higher-Level Representations from Data (And a Curious Look Forward)". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by Christian Bessiere. International Joint Conferences on Artificial Intelligence Organization, July 2020, pp. 4868–4876. DOI: 10.24963/ijcai.2020/678 (cit. on p. 123).