Knowledge and Information Systems (2020) 62:1113–1141
https://doi.org/10.1007/s10115-019-01378-7

REGULAR PAPER

Accelerating pattern-based time series classification: a linear time and space string mining approach

Atif Raza[1] · Stefan Kramer[1]

Received: 24 August 2018 / Revised: 15 June 2019 / Accepted: 23 June 2019 / Published online: 12 July 2019

Abstract
Subsequences-based time series classification algorithms provide interpretable and generally more accurate classification models compared to the nearest neighbor approach, albeit at a considerably higher computational cost. A number of discretized time series-based algorithms have been proposed to reduce the computational complexity of these algorithms; however, the asymptotic time complexity of the proposed algorithms is also cubic or higher-order polynomial. We present a remarkably fast and resource-efficient time series classification approach which employs a linear time and space string mining algorithm for extracting frequent patterns from discretized time series data. Compared to other subsequence or pattern-based classification algorithms, the proposed approach only requires a few parameters, which can be chosen arbitrarily and do not require any fine-tuning for different datasets. The time series data are discretized using symbolic aggregate approximation, and frequent patterns are extracted using a string mining algorithm. An independence test is used to select the most discriminative frequent patterns, which are subsequently used to create a transformed version of the time series data. Finally, a classification model can be trained using any off-the-shelf algorithm. Extensive empirical evaluations demonstrate the competitive classification accuracy of our approach compared to other state-of-the-art approaches. The experiments also show that our approach is at least one to two orders of magnitude faster than the existing pattern-based methods due to the extremely fast frequent pattern extraction, which is the most computationally intensive process in pattern-based time series classification approaches.

Keywords Time series · Classification · String mining · Linear time and space

1 Introduction

Time series data mining has evolved into a significant research avenue over the last couple of decades. The continued interest in this field stems from the growing need to extract useful

✉ Atif Raza
raza@informatik.uni-mainz.de

Stefan Kramer
kramer@informatik.uni-mainz.de

[1] Institute of Computer Science, Johannes Gutenberg University Mainz, Staudingerweg 9, Mainz, Germany

bits of information from the vast data stores filling up with the ever-increasing generation and storage of time series data from diverse fields including natural processes, scientific research, business, finance, activity and/or interaction logs, etc. Time series data comprise sequences of real-valued measurements recorded over time. The inherent temporal order of these measurements characterizes the varying behavior of the recorded process and allows to find the patterns which can identify similar or anomalous behavior over time; therefore, it is vital to keep this order intact so as to infer information about time-dependent features, e.g., seasonality, trend, etc. Time series data mining algorithms are used primarily for extracting knowledge from data with a temporal order; however, the same algorithms can also be applied to data with any form of logical ordering, e.g., images converted to series using shape outlines, chemical composition data obtained from spectral analyses, etc. This effectively increases the scope and impact of time series data mining research and its applications.

Classification has been an important research topic in the time series data mining domain. Generally, time series classification problems can be divided into two categories. The first category encompasses those 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/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. 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) [4]. The second category covers those problems for whom the entire shape of the time series instances is irrelevant and only small subsequences are indicative of the class. 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. In such problems, identifying whether the particular subsequences are present or absent in the time series instances is a better suited classification approach. The *Shapelets*-based time series classification algorithm was proposed to handle problems from this category [17].

Effectively, shapelets are subsequences occurring frequently in a specific class of time series instances while being absent or infrequent in instances of the other classes. The distance between a shapelet and a time series is defined as the minimum observed distance between the shapelet and all shapelet-length subsequences of the time series instance. The presence or absence of a shapelet in a time series is determined based on whether the minimum distance between the shapelet and the time series is within a given threshold. Once the presence of a specific shapelet in a given time series is confirmed, the time series is classified as belonging to the class being represented by the particular shapelet. The original shapelets-based algorithm creates a classification model by embedding shapelets and their corresponding distance thresholds in the internal nodes of a decision tree. The classification of a given time series instance starts at the root node where the distance between the time series and the shapelet specific to the root node is calculated. The subsequent branch to be taken is determined based on whether the calculated distance is above or below the threshold for the root node. Subsequent nodes are traversed similarly until a leaf node is reached, at which point the classification decision is taken based on the leaf node's class label. For complex and/or multi-class problems, the induction of the classification model may lead to the identification of a number of shapelets, whose presence ultimately determines a specific class of instances. This also indicates that a class of instances can have more than one representative shapelet and the order of the discovered shapelets heavily depends on the initial conditions used to initiate the shapelet discovery process.

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 calculate a few distance values for the time series instance and the shapelets encountered in the nodes of the tree, whereas the 1-NN approach needs to compare the said time series instance with all available reference time series instances before finalizing the classification decision. In contrast, 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 model. At any node, the time required for shapelet discovery is on the order of $O(N^2n^4)$, where N represents the number of time series instances reaching the particular node, while n is the length of the time series instances. Hence, the overall time required for shapelet-based model induction can become unreasonably high.

The Shapelet Transform (ST) algorithm dissociates the shapelet discovery and model induction processes [9]. It reduces the computational requirements of the training process, because a single call to the $O(N^2n^4)$ shapelet discovery process is required, unlike the original shapelets-based algorithm. For ST, a single invocation of the shapelet discovery process extracts the k best shapelets, which are then used to transform the time series classification problem into a feature-based classification problem. The transformed dataset consists of N rows and $k + 1$ columns, where each row is associated with a specific time series instance, while each column is associated with a specific shapelet from among the k best shapelets and the last column is used for the class label. The cell corresponding to the ith row and jth column holds the minimum distance between the ith training instance and the jth shapelet. The transformed dataset provides a feature-based view of the time series classification problem; therefore, any off-the-shelf classification algorithm can be used for model induction. Although ST is faster than the original shapelet algorithm, its asymptotic time complexity can still make it extremely computation intensive.

To scale up pattern-based time series classification to massive amounts of data, however, the algorithmic complexity has to be reduced to linear time at most. One way of addressing the complexity issue of pattern-based time series classification 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 [11–15], the approaches are still suffering from high computational complexity: the complexity of the Fast Shapelets [12] approach is $O(Nn^2)$, the one of BoP (Bag of Patterns) [11] is $O(Nn^3)$, the one of SAX-VSM (Symbolic Aggregate approXimation–Vector Space Model) [15] is $O(Nn^3)$, the one of BOSS (Bag of SFA Symbols) [13] is $O(N^2n^2)$, and the complexity of BOSS VS (Bag of SFA Symbols in Vector Space) [14] is $O(Nn^{\frac{3}{2}})$. Overall, it appears 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. In other words, 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 et al. [5,6]. Further research has optimized their practical running times and theoretical properties [3].

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. Also, as 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.

The main contributions of this paper are as follows: first, we propose to build time series classification on the basis of a linear time string mining algorithm which enables pattern-based time series classification to scale up to massive datasets.[1] Whereas discretized time series have been used before, *explicitly basing time series classification on string mining algorithms* has not been considered yet. Second, the complexity advantage can be used to explore different parameterizations and still be much more time efficient than any super-linear time series classification scheme. Third and more specifically, we present a highly efficient pattern extraction method for time series which does not require any user provided limits for minimum/maximum pattern lengths. Fourth, the resulting classification scheme provides competitive accuracy compared to state-of-the-art symbolic representation-based approaches, and last, we show how the overall scheme can lead to extremely fast parameter optimization. In the following, our proposed approach will be referred to as *MiSTiCl* (Mining Strings for Time Series Classification).

The rest of the paper is organized as follows: Sect. 2 provides some background about SAX and the string mining algorithm. Section 3 presents the proposed algorithm. Sections 4 and 5 present the implementation details, experimental protocol, and results. Finally, Sect. 6 presents the conclusions.

2 Background

An ordered, real-valued sequence of n observations is called a *time series* and is denoted as $T = (t_1, t_2, \ldots, t_n)$. A time series belonging to a specific class is assigned a label $y \in C$, where C is the set of all possible class labels. A set of N labeled time series instances $\{(T_1, y_1), (T_2, y_2), \ldots, (T_N, y_N)\}$ forms a time series dataset D. Figure 1 shows an illustration of a time series dataset, where the instances belonging to the same class are plotted together.

Time series data are inherently high-dimensional, which makes it difficult to design efficient algorithms for time series mining. It is also highly susceptible to noise, which can adversely affect the accuracy of an algorithm. Therefore, time series discretization is often used to (i) reduce the dimensionality and cardinality of the data, which enables to employ/develop efficient algorithms, and (ii) reduce the effects of noise present in the raw data. Symbolic aggregate approximation (SAX) is a widely used time series discretization approach [10]. SAX 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 $\widehat{T} = (\widehat{t_1}, \widehat{t_2}, \ldots, \widehat{t_p})$ of length $p = \lfloor \frac{n}{w} \rceil$ such that $p \ll n$, where $w \in \mathbb{Z}_{\geq 1}$ represents the dimensionality reduction factor (or, the averaging window size).[2] First, T is converted to a reduced dimensionality version $\overline{T} = (\overline{t}_1, \overline{t}_2, \ldots, \overline{t}_p)$ of length p such that each nonover-

[1] Throughout the text, we refer to real-valued time series segments as "subsequences" and the discretized/symbolic segments as "patterns."

[2] The presented mathematical notation is for the simple case of integer values of p; later SAX refinements enable handling non-integer window sizes as well.

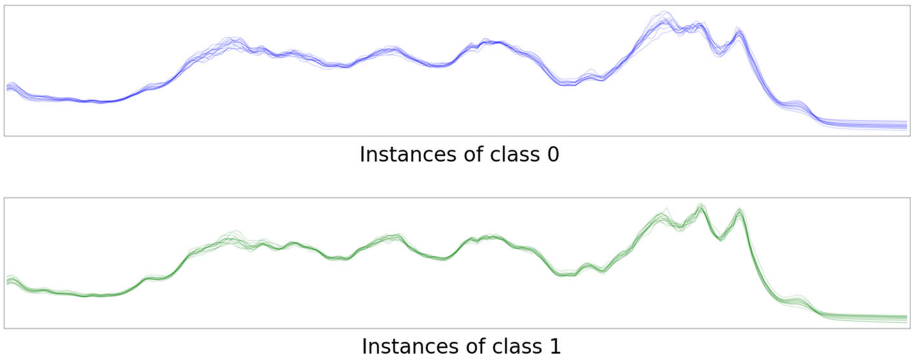


Fig. 1 Illustration of a real-valued binary class time series dataset. Each instance is 286 time points long. Instances of the same class are shown together (color figure online)

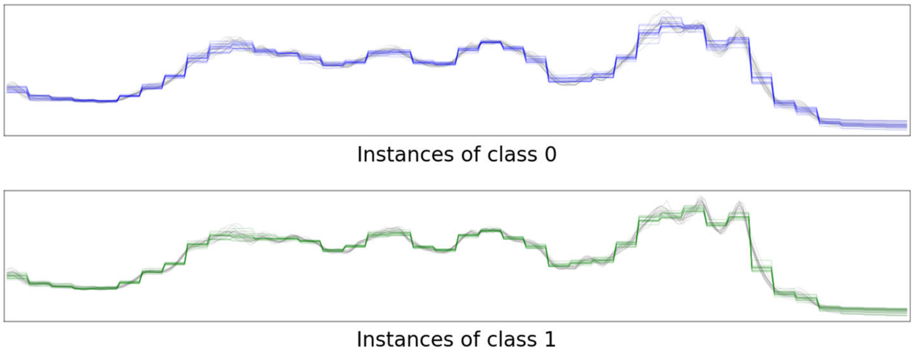


Fig. 2 Illustration of the PAA version of time series instances superimposed on their real-valued counterparts. The 286 time points long time series have been reduced to 40 time points based on a dimensionality reduction factor $w = 7$. The PAA versions have been stretched (along the x-axis) to emphasize the retention of the overall shape of time series instances (color figure online)

lapping sequence of w observations of T is averaged to provide one observation. $\overline{T}$ is also referred to as a Piecewise Aggregate Approximation (PAA). Mathematically, we can get each $\bar{t}_i$ using the equation:

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

Figure 2 shows the reduced dimensionality version of time series instances superimposed on their real-valued counterparts.

For cardinality reduction, each observation $\bar{t}_i \in \overline{T}$ 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 remaining breakpoints are chosen such that area under the $N(\mu = 0, \sigma = 1)$ Gaussian curve from β_i to β_{i+1} equals $\frac{1}{\alpha}$. This ensures an equiprobable selection of every alphabet. A Gaussian curve with $\mu = 0$ and $\sigma = 1$ is used because z-normalized time series instances have

Table 1 Breakpoints lookup table for dividing the $N(\mu = 0, \sigma = 1)$ Gaussian curve into equiprobable regions for $\alpha = 3$ to 8. β_0 and β_α are $-\infty$ and ∞, respectively

	Cardinality level, α					
	3	4	5	6	7	8
β_1	− 0.43	− 0.67	− 0.84	− 0.97	− 1.07	− 1.15
β_2	0.43	0	− 0.25	− 0.43	− 0.57	− 0.67
β_3		0.67	0.25	0	− 0.18	− 0.32
β_4			0.84	0.43	0.18	0
β_5				0.97	0.57	0.32
β_6					1.07	0.67
β_7						1.15

Table 2 Symbolic representation of time series instances obtained after mapping each time point of the PAA versions to the corresponding characters in an alphabet based on 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`

the same mean and standard deviation, and they also tend to follow a Gaussian distribution.[3] Table 1 shows the breakpoints for different values of α. All observations $\overline{t}_i \in \overline{T}$, which have their values in the range $[\beta_0, \beta_1)$, are mapped to the first character in the alphabet. Similarly, all observations $\overline{t}_i \in \overline{T}$ 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 $\widehat{t_i}$ as follows:

$$\widehat{t_i} = alpha_j, \quad \text{iff} \quad \beta_{j-1} \leq \overline{t}_i < \beta_j$$

Illustrations of real-valued time series instances and their PAA versions are shown in Figs. 1 and 2, respectively. The symbolic representations of these example time series instances are shown in Table 2. For a detailed analysis of the SAX algorithm and a review of its diverse applications, we refer the reader to the corresponding article [10].

String mining is concerned with the discovery of patterns (substrings) which are characteristic of a string dataset. 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 lead 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 [5,6] and required

[3] Research suggests that a large number of time series datasets follow the Gaussian distribution. For the minority of datasets which do not follow this assumption, selecting the breakpoints using the Gaussian curve can deteriorate the efficiency of SAX; however, the "correctness of the algorithm is unaffected" [10].

$O(m)$ 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. 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 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 [3].

For completeness, a brief summary of the string mining algorithm is presented below. Let $\mathcal{D}^+ = \{aaba, abaaab\}$ and $\mathcal{D}^- = \{bbabb, abba\}$ be two string datasets. Let x denote a string of length q consisting of all the strings in $\mathcal{D}^+$ and $\mathcal{D}^-$ concatenated, i.e., $x = aaba\#_1^1abaaab\#_{|\mathcal{D}^+|}^1bbabb\#_1^2abba\#_{|\mathcal{D}^-|}^2$, where each $\#_l^m$ is a unique symbol not occurring in any dataset and is used for marking the end of strings. A substring of x starting at position i and ending at position j is denoted by $x[i, j]$, which is a concatenation of the symbols $x[i]x[i+1]\ldots x[j]$. A suffix array is an array of integers which is used to describe the lexicographic order of all suffixes of x, s.t. $x[SA[k], q] < x[SA[k+1], q]$ for all $1 \leq k < q$. The lcp array 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 (SA) and the lcp array (LCP). 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 $RMC_{LCP}(i, j)$ 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. Table 3 shows the string x, the suffix array SA, and the lcp array LCP. Finally, the algorithm calculates so-called correction terms for establishing the occurrence frequency of the different substrings based on the lcp array values. For further details of the algorithm, we refer the interested reader to the respective papers [3,5,6].

3 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.

Table 3 The suffix array for x and its lcp table

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	$\#_2^1$	b	b	a	b	b	$\#_1^2$	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
	$\#_1^1$	$\#_2^1$	$\#_1^2$	$\#_2^2$	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	$\#_2^1$	$\#_1^2$	a	a	a	a	b	b	b
							a	b	b	$\#_2^1$	a	a	b	b			$\#_1^1$	$\#_2^2$	a	b	$\#_1^2$	a	a
							b	$\#_2^1$	a		$\#_1^1$	a	$\#_1^2$	a					a	b		$\#_2^2$	b
							$\#_2^1$		$\#_1^1$			a		$\#_2^2$					b	$\#_1^2$			b
												b							$\#_1^2$				$\#_2^1$
												$\#_2^1$											

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 [6]

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 very important 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 w which result in minimum information loss. Usually, a brute force parameter tuning approach is used to come up with such parameters. This can incur a significant upfront 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 w parameters allows to extract discriminative patterns from some form of multi-view perspective and helps to capture different features at each view.[4] 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.

3.1 Main algorithm

MiSTiCl is a powerful yet extremely efficient time series classification algorithm which brings together a number of highly effective approaches from the time series and string mining domains. This subsection provides a general overview of the algorithm, while the following subsections provide details of the individual steps. Algorithm 1 lists the steps involved in the creation of feature-based dataset splits using MiSTiCl. Transforming the real-valued time series training and testing data into a feature-based representation requires a set of values for the alphabet (A) and window sizes (W) each, the minimum and maximum frequency values for frequent pattern extraction, and a parameter K limiting the number of used frequent patterns. The first step is the initialization of map data structures for single-view feature sets (Line 1). The next step is the extraction of class labels which are subsequently used for frequent pattern extraction (Line 2). Next, single-view feature sets are created corresponding to each $(\alpha, w) \in A \times W$ parameter combination (Lines 3–7). The real-valued time series splits are discretized to create symbolic splits corresponding to the current α and w parameters (Line 4). Next, frequent patterns are extracted from the symbolic training split (Line 5). Now, single-view training and testing feature sets are created using the top K frequent patterns (Lines 6). Finally, the single-view feature sets are combined to create multi-view feature sets (Line 8). The multi-view feature sets can be used for model induction using any off-the-shelf algorithm.

[4] Usually, multi-view learning refers to learning with different sets of features of vectorial data; however, here we use the term for multiple representations of a time series data originating from different parameterizations.

```
Algorithm 1 MiSTiCl ($D_{train}$, $D_{test}$, $A$, $W$, $f^+$, $f^-$, $K$)
Parameters: $D_{train}$, $D_{test}$: real-valued training and testing splits, $A$: set of alphabet sizes, $W$: set of window sizes, $f^+$: minimum occurrence frequency of a pattern in positive class, $f^-$: maximum occurrence frequency of a pattern in negative class, $K$: number of patterns to be used per class for feature set creation
Returns: $FS_{train}$, $FS_{test}$: training and testing feature sets
1: $SV_{train} \leftarrow \{\}$, $SV_{test} \leftarrow \{\}$        ▷ Initialize associative arrays for single-view feature sets
2: $C \leftarrow$ExtractClassLabels($D_{train}$)
3: for all $(\alpha, w) \in A \times W$ do
4:     $\widehat{D}_{train}$, $\widehat{D}_{test} \leftarrow$Discretize($D_{train}$, $D_{test}$, $\alpha$, $w$)
5:     $FP \leftarrow$GetDiscriminativeFrequentPatterns($\widehat{D}_{train}$, $f^+$, $f^-$, $C$)
6:     $SV^{\alpha,w}_{train}$, $SV^{\alpha,w}_{test} \leftarrow$CreateFeatureSets($D_{train}$, $D_{test}$, $\widehat{D}_{train}$, $FP$, $C$, $K$)
7: end for
8: $FS_{train}$, $FS_{test} \leftarrow$ CombineFeatureSets($SV_{train}$, $SV_{test}$, $A$, $W$, $|D_{train}|$)
9: return $FS_{train}$, $FS_{test}$
```

```
Algorithm 2 GetDiscriminativeFrequentPatterns ($\widehat{D}$, $f^+$, $f^-$, $C$)
Parameters: $\widehat{D}$: symbolic dataset, $f^+$: minimum frequency of the pattern in positive class, $f^-$: maximum frequency of the pattern in negative class, $C$: set of class labels
Returns: $FP$: an associative array containing the most discriminative frequent patterns for each class
1: $FP \leftarrow \{\}$        ▷ Initialize an associative array for filtered frequent patterns
2: for all $c \in C$ do
3:     $\widehat{P} \leftarrow \{\widehat{T} \mid \forall \widehat{T} \in \widehat{D},\ \widehat{T}.y = c\}$
4:     $\widehat{N} \leftarrow \{\widehat{T} \mid \forall \widehat{T} \in \widehat{D},\ \widehat{T}.y \neq c\}$
5:     $Z \leftarrow$ExtractAllFrequentPatterns($\widehat{P}$, $\widehat{N}$, $f^+$, $f^-$)
6:     $FP[c] \leftarrow$ SelectDiscriminativePatterns($Z$, $|\widehat{P}|$, $|\widehat{N}|$)
7: end for
8: return $FP$
```

3.2 Extracting frequent patterns

Algorithm 2 lists the steps involved in extracting frequent patterns from a symbolic dataset. The string mining algorithm extracts frequent patterns from binary problems; therefore, a multi-class problem is transformed into multiple binary problems using a one-vs-all approach. 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 $\widehat{P}$, while all remaining instances are assigned to the negative class dataset $\widehat{N}$ (Lines 3 and 4). The string mining algorithm extracts all the patterns from $\widehat{P}$ and $\widehat{N}$ 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 positive and negative class datasets (Line 5). Next, the most discriminative frequent patterns for the current class c are selected from the extracted frequent patterns and placed on the map FP (Line 6). Finally, the map containing discriminative frequent patterns for all the classes is returned.

3.3 Selecting discriminative patterns

The discriminative power of a given frequent pattern regarding correctly identifying a specific class of instances can be assessed using different measures, e.g., the χ^2 independence test, information gain, etc. The occurrence frequencies 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

Table 4 A few examples of base and variant patterns

Pattern	Base (b)/Variant (v)
dccb	b
dccb*b*	v
*a*dccb	v
abdcdcc	b

statistic indicates 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 4 for examples). The occurrence frequency of a base pattern is always greater than or equal to the 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.[5] Algorithm 3 lists the pseudo-code for selecting such frequent patterns. The procedure receives a list of all the frequent patterns along with their corresponding occurrence frequencies, and the instance counts for the positive and negative datasets. The filtered frequent patterns are returned in an ordered associative array of lists, where each slot in the associative array holds all the frequent patterns with the same test statistic and the array is sorted in decreasing order of the independence test statistic. After initializing the array, the procedure iterates over all the patterns provided as input (Lines 2–8). An independence test is used to obtain a value for the test statistic for the current pattern (Line 3).[6] 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). Next, the current pattern f is compared against all the patterns in $fpList$ to see whether the list already contains its base pattern (Lines 6). If a base pattern is found, f is discarded, otherwise it is inserted in the list (Line 7). Once all the patterns have been evaluated, the ordered array of lists containing the filtered frequent patterns is returned.

[5] 1. This criterion and procedure are not to be confused with *closed* or *open/free* patterns [16]. 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.

[6] A discussion about calculation of the χ^2 test statistic and the information gain is provided in "Appendix."

```
Algorithm 3 SelectDiscriminativePatterns ($Z$, $N_{\widehat{P}}$, $N_{\widehat{N}}$)
Parameters: $Z$: list of all extracted frequent patterns with their respective occurrence frequencies, $N_{\widehat{P}}$, $N_{\widehat{N}}$:
    number of instances in the positive and negative class splits
Returns: $OL$: an ordered associative array of lists containing frequent patterns; sorted in decreasing order of
    independence value
1: Initialize $OL \leftarrow \{\}$                              ▷ Initialize an ordered associative array
2: for all $z \in Z$ do
3:     $indVal \leftarrow$ GETINDEPENDENCEVALUE($z$, $N_{\widehat{P}}$, $N_{\widehat{N}}$)
4:     $f \leftarrow z.pattern$     ▷ $z$ contains the frequent pattern $z.pattern$, along with its occurrence frequencies
    $z.freq_{\widehat{P}}$ and $z.freq_{\widehat{N}}$ in $\widehat{P}$ and $\widehat{N}$.
5:     Retrieve $fpList$ corresponding to $indVal$, if none exists, add a new list to $OL$
           at location $indVal$
6:     for all $p$ in $fpList$, check if $p$ is a sub-string of $f$
7:     if $f$ is a unique pattern, add to $fpList$
8: end for
9: return $OL$
```

3.4 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 said subsequences and the real-valued time series instances are used as feature values. Alternatively, binary-valued datasets can be created using the 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 are 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 4 lists the pseudo-code for creating real-valued feature-based datasets. After initializing the training and testing feature-based splits (Line 1), the procedure iterates over all class labels $c \in C$ to add the top K features for each class (Lines 2–13). The loops in Lines 5 and 6 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 7). 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.

3.5 Creating the multi-view feature sets

MiSTiCl aims to mitigate the problem of feature loss by combining multiple single-view feature sets, each created with its own α and w parameter, thus creating a multi-view feature set. A candidate multi-view feature set is created by simply joining multiple single-view feature sets and requires zero computational overhead. Finding the best multi-view feature set, however, involves (i) creating all possible multi-view 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

Algorithm 4 CreateFeatureSets (D_{train}, D_{test}, $\widehat{D}_{train}$, FP, C, K)

```
Parameters: $D_{train}$, $D_{test}$: real-valued time series splits, $\widehat{D}_{train}$: discretized time series training split, $FP$:
    ordered associative array of lists containing frequent patterns; sorted in decreasing order of independence,
    $C$: set of class labels, $K$: number of patterns to be used per class
Returns: $SV_{train}$, $SV_{test}$: single-view transformed training and testing splits
 1: $SV_{train} \leftarrow MATRIX(|D_{train}|, K \times |C|)$, $SV_{test} \leftarrow MATRIX(|D_{test}|, K \times |C|)$
 2: for all $c \in C$ do
 3:    $k \leftarrow 0$                    ▷ feature count variable, incremented after adding each feature
 4:    $FPCurrClass \leftarrow FP[c]$
 5:    for all $fpList \leftarrow FPCurrClass$ do                                            ▷ Outer loop
 6:       for all $f \leftarrow fpList$ do                                                    ▷ Inner loop
 7:          $s \leftarrow$ PerformReverseLookup($D_{train}$, $\widehat{D}_{train}$, $f$)
 8:          For each $T \in D_{train}$
               populate the respective row and column of $SV_{train}$
               with the distance value between $T$ and $s$
 9:          For each $T \in D_{test}$
               populate the respective row and column of $SV_{test}$
               with the distance value between $T$ and $s$
10:          Increment $k$ and if $k$ equals $K$ break Outer loop
11:       end for
12:    end for
13: end for
14: return $SV_{train}$, $SV_{test}$
```

single-view feature sets; however, this could potentially introduce redundant features in the multi-view feature set. This presents an optimization problem where a minimum number of single-view feature sets should be used to create a multi-view feature set which provides maximum classification accuracy. For a cardinality reduction factor set A and a dimensionality reduction factor set W, the total number of single-view feature sets created by MiSTiCl equals $|A| \times |W|$. Using the brute force approach to find the best multi-view feature set would require creating and evaluating $2^{|A|\times|W|} - 1$ candidate feature sets and is obviously the most computation intensive option. The set cover problem is a classical question in combinatorics which aims to identify the smallest subset of a collection S whose union equals the universe $\mathcal{U}$. Finding an exact solution to the set cover problem can also be computationally very demanding; however, using a heuristic approach can significantly reduce the computational impact. Algorithm 5 lists the steps involved in creating the best multi-view feature set splits using a heuristic set cover approach. First, the procedure initializes a descending order associative array $Map\text{-}CoveredInsts\text{-}Params$, and a table (dual-key associative array) S. Next, the set $\{1, \ldots, N\}$ is randomly split into two disjoint sets, $All\text{-}Train\text{-}Indices$ and $All\text{-}Val\text{-}Indices$ (Line 2). Each element of the set $\{1, \ldots, N\}$ corresponds to an instance in the original training set split, while the two disjoint sets, $All\text{-}Train\text{-}Indices$ and $All\text{-}Val\text{-}Indices$, are used to create subsets of the single-view training feature sets for model training and validation, respectively. The procedure iterates over all the $(\alpha, w) \in A \times W$ parameter combinations (Lines 3–8) and performs the following steps for each single-view training feature set $SV^{\alpha,w}_{train}$.

- Train a classification model CM using only those instances from $SV^{\alpha,w}_{train}$ whose indices occur in the $All\text{-}Train\text{-}Indices$ set
- Test the classification model CM using only those instances of $SV^{\alpha,w}_{train}$ whose indices occur in the $All\text{-}Valid\text{-}Indices$ set and save the indices of correctly classified instances in the table S using α and w as index values.
- Add the current (α, w) pair in the associative array using the size of correctly classified validation instances as key

Algorithm 5 CombineFeatureSets (SV_{train}, SV_{test}, A, W, N)

```
Parameters: $SV_{train}$, $SV_{test}$: single-view feature sets for training and testing splits, $A$: set of alphabet sizes,
    $W$: set of window sizes, $N$: number of training set instances, $MaxNumberParams$: maximum number
    of selected parameters (default: 5)
Returns: $MV_{train}$, $MV_{test}$: multi-view training and testing feature set splits
 1: Initialize $S$ as a table, $Map\text{-}CoveredInsts\text{-}Params$ as a descending order associative array
 2: ($All\text{-}Train\text{-}Indices$, $All\text{-}Val\text{-}Indices$) ←RANDOMSPLIT($\{1, 2, \ldots, N\}$)
 3: for all $(\alpha, w) \in A \times W$ do
 4:     Train model $CM$ using $\{Inst_i | i \in All\text{-}Train\text{-}Indices \wedge Inst_i \in SV_{train}^{\alpha,w}\}$
 5:     Test model $CM$ using $\{Inst_i | i \in All\text{-}Val\text{-}Indices \wedge Inst_i \in SV_{train}^{\alpha,w}\}$
 6:     Save indices of validation instances correctly classified by $CM$ in $S^{\alpha,w}$
 7:     Add $(|S^{\alpha,w}|, (\alpha, w))$ to $Map\text{-}CoveredInsts\text{-}Params$
 8: end for
 9: $BestParams \leftarrow \{\}$
10: $Covered\text{-}Val\text{-}Indices \leftarrow \{\}$
11: for all $(|S^{\alpha,w}|, (\alpha, w)) \in Map\text{-}CoveredInsts\text{-}Params$ do
12:     if $(S^{\alpha,w} \setminus Covered\text{-}Val\text{-}Indices) \neq \emptyset$ then
13:         $BestParams \leftarrow BestParams \cup \{(\alpha, w)\}$
14:         $Covered\text{-}Val\text{-}Indices \leftarrow Covered\text{-}Val\text{-}Indices \cup S^{\alpha,w}$
15:         if $(All\text{-}Val\text{-}Indices \setminus Covered\text{-}Val\text{-}Indices) = \emptyset$ or
                $|BestParams| \geq MaxNumberParams$ then
16:             break
17:         end if
18:     end if
19: end for
20: Initialize $MV_{train}$, $MV_{test}$
21: for all $(\alpha, w) \in BestParams$ do
22:     $MV_{train} \leftarrow$ Join $MV_{train}$ and $SV_{train}^{\alpha,w}$
23:     $MV_{test} \leftarrow$ Join $MV_{test}$ and $SV_{test}^{\alpha,w}$
24: end for
25: return $MV_{train}$, $MV_{test}$
```

Next, two empty sets are initialized, one for keeping track of all the covered validation set instances $Covered\text{-}Val\text{-}Indices$ and the other for the best found parameters $BestParams$. Next, the procedure iterates over the associative array containing the (α, w) pairs (Lines 11–19). If the difference between the set $S^{\alpha,w}$ and all covered instances $Covered\text{-}Val\text{-}Indices$ is not empty, then (i) the current (α, w) pair is added to the set of best parameters, and (ii) the set of covered indices is updated with the union of $S^{\alpha,w}$ and $Covered\text{-}Val\text{-}Indices$. If the difference between the sets $All\text{-}Val\text{-}Indices$ and $Covered\text{-}Val\text{-}Indices$ is empty, i.e., all validation instances have been covered, or the number of parameter pairs in the $BestParams$ set is equal to the maximum allowed parameter pairs, then the loop is terminated. Finally, the single-view training and testing feature sets corresponding to each (α, w) pair in the set of best parameters $BestParams$ are joined together to form the multi-view training and testing feature sets, respectively (Lines 21–24).

3.6 Complexity analysis

The asymptotic time complexity of the MiSTiCl algorithm can be calculated by aggregating the time complexities of the individual steps. MiSTiCl creates multiple single-view feature sets, and the creation of each feature set contributes equally toward the overall complexity; therefore, we shall first consider a single iteration for an arbitrary combination of α and w. 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 $m = N\frac{n}{w}$; therefore, the time complexity of string mining is on the order of $O(N\frac{n}{w})$. 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 w, 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 $p = \lfloor \frac{n}{w} \rceil$. 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 $O(ns)$ time, 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 complexities of the individual steps, the overall time complexity of creating a feature set for a given α and w parameter combination is on the order of $O(Nn) + O(N\frac{n}{w}) + O(1) + O(Nns) \approx O(Nns)$. Creating $|A| \times |W|$ many feature sets increases the complexity by a factor of $|A| \times |W|$; 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 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 [14].

4 Experiments

We have carried out an extensive set of experiments to compare different symbolic representation-based time series classification algorithms namely BoP, SAX-VSM, BOSS, 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 and the algorithm can perform at speeds which allows to handle very large scale time series classification problems.

The UEA Time Series Repository provides Weka-based implementations of a number of time series algorithms including BoP, SAX-VSM, and BOSS.[7] We have also implemented the MiSTiCl algorithm using the framework provided by the UEA Time Series Repository so that any platform or implementation bias can be minimized.[8] The different algorithms were evaluated using 85 real-world and synthetic time series datasets provided by the UCR Time Series Archive.[9] For each dataset, 100 random shuffles were created such that the class distribution and 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.

[7] http://www.timeseriesclassification.com/.

[8] Our implementation is available from https://github.com/atifraza/MiSTiCl.

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

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 and 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 toward 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 $\{2, 3, \ldots, 6\}$.[10] MiSTiCl also requires a minimum positive frequency f^+, a maximum negative frequency 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 was also determined during parameter optimization. The BOSS algorithm is based on a different discretization technique called Symbolic Fourier Approximation (SFA). 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.

MiSTiCl transforms a time series dataset into a feature-based representation which creates a generic classification problem and allows to utilize any off-the-shelf classification algorithm. Since the transformed dataset incorporates a number of diverse features, ensemble techniques can be extremely effective for creating accurate classification models. Therefore, we employed random forests (RF) [1], extremely randomized trees (ET) [8], and AdaBoost.M1 (AB) [7] for creating classification models using a maximum of 1000 trees per ensemble, while keeping all other parameters unchanged.

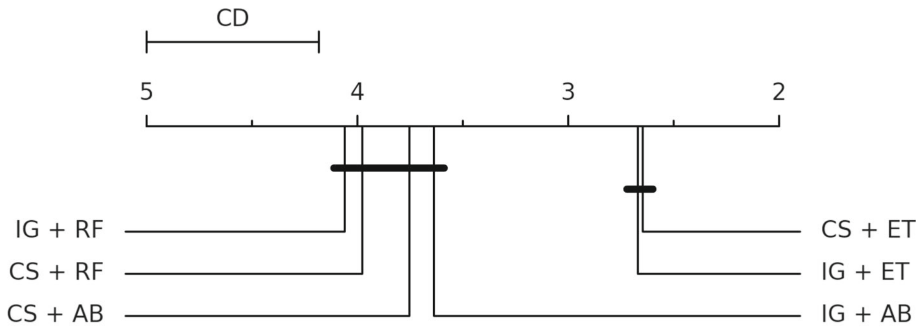
5 Results

This section provides an empirical analysis of the classification accuracy and runtime performance of MiSTiCl, BoP, SAX-VSM, and BOSS based on averaged values of the statistics collected for each dataset. We have also included the classification accuracy results for ST to establish a baseline comparison with a real-valued shapelet-based classification algorithm.[11] For statistical comparison of different algorithms, we employ the Friedman test followed by Nemenyi post hoc test based on average ranks attained by the different algorithms and show the comparisons as critical difference (CD) diagrams [2].

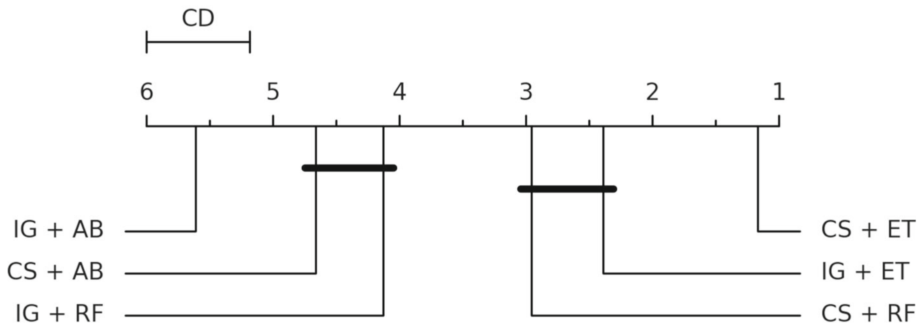
MiSTiCl can employ either the χ^2 independence test or the information gain for selecting independent and discriminative frequent patterns; therefore, we carried out all the MiSTiCl experiments using both these methods. Figure 3a, b shows the critical difference diagrams for the different MiSTiCl variants based on classification accuracy and total runtime, respectively. We can see that using the χ^2 test to determine pattern independence yields better overall

[10] Following the parameter settings provided by the UEA Time Series Repository.

[11] The results for ST have been taken from the UEA Time Series Repository.



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



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

Fig. 3 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, while **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

results in terms of both classification accuracy and runtime; therefore, the following analysis will be based on the results obtained using the χ^2 independence test.

Figure 4 presents a comparison of MiSTiCl against BoP, SAX-VSM, and BOSS, regarding classification accuracy and runtime. The x- and y-axes show the total time required for training and testing a classification model using MiSTiCl and the compared algorithm, respectively. Each evaluated dataset is represented by a marker. Green markers indicate that MiSTiCl provides better accuracy, whereas red markers indicate otherwise. The marker size corresponds to the absolute difference between classification accuracies 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 was not significant. We can see that MiSTiCl consistently 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.

Figure 5a, b shows the average ranks of the different algorithms based on classification accuracy and runtime, respectively. For classification accuracy, MiSTiCl is placed on par with

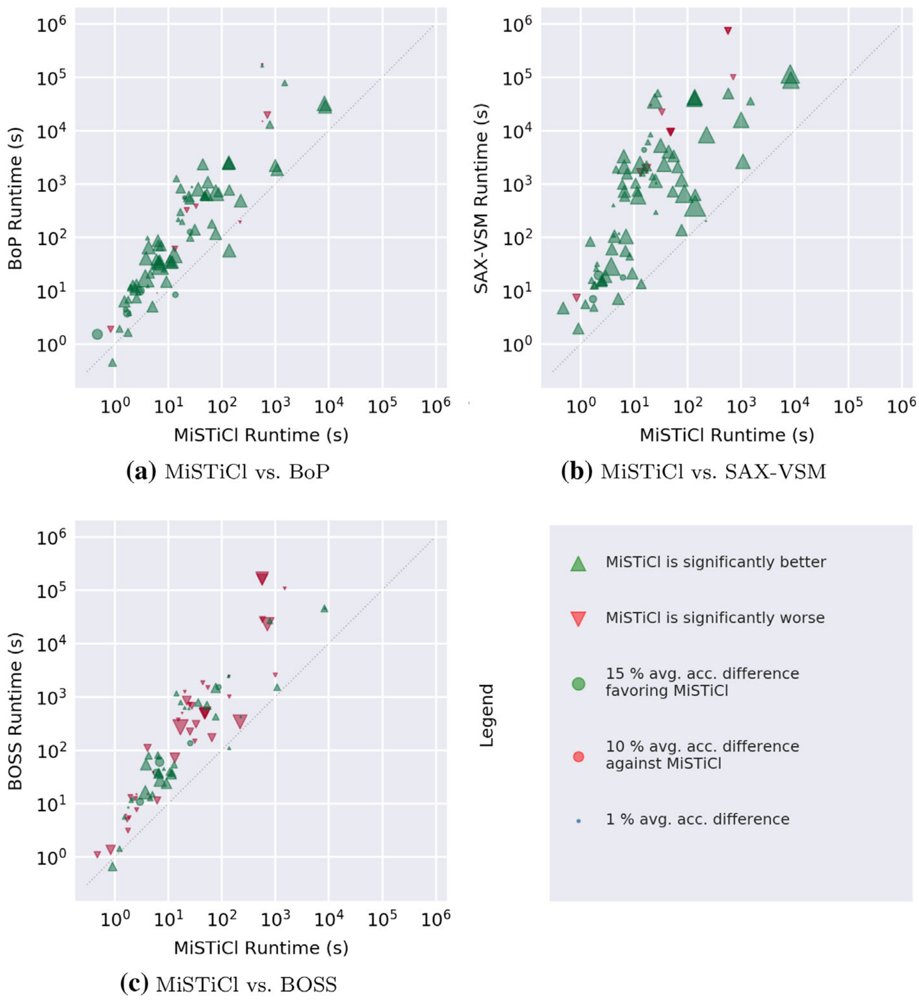
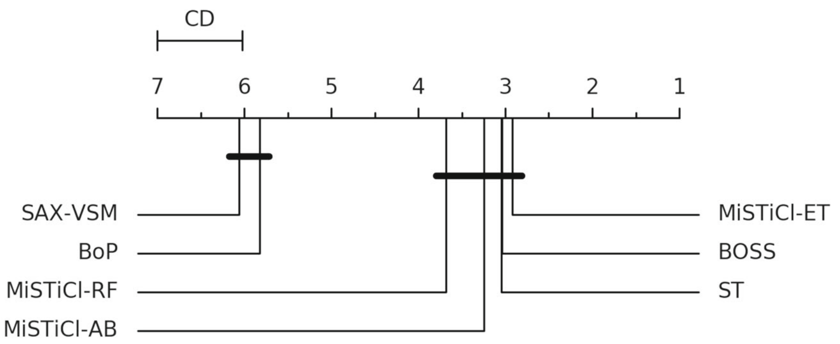


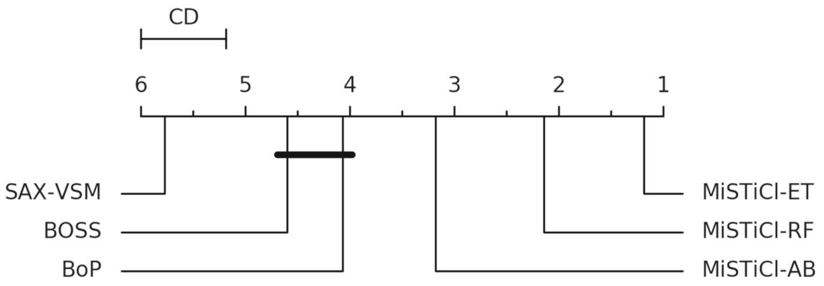
Fig. 4 Comparing MiSTiCl against BoP, SAX-VSM, and BOSS regarding runtime and classification accuracy. Each marker represents one dataset. The *x*- and *y*-axes show the runtime (training + testing) in seconds. Markers below the dotted line indicate that MiSTiCl is slower than the other algorithm. A 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 (color figure online)

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. For the runtime, MiSTiCl easily achieves the top three spots with average ranks of 1.33, 2.13, and 3.09 for ET-, RF-, and AB-based variants, respectively, making it significantly faster than any of its competitors.

A pairwise win/tie/loss analysis was also performed for a head-to-head comparison between MiSTiCl and the other algorithms. A win was registered if the difference between



(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 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

Table 5 Pairwise win/tie/loss comparison of different algorithms using 85 datasets

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

The margin for wins/ties/losses was set to 2%

the classification accuracy achieved by MiSTiCl and the competing algorithm was greater than $+2\%$, while 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. When comparing the different MiSTiCl variants with each other, we see that MiSTiCl-ET provides better classification accuracy for most of the datasets.

A breakdown of the time required for training and testing a classification model using MiSTiCl-ET is provided in Fig. 6. 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 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 and transformation, parameter

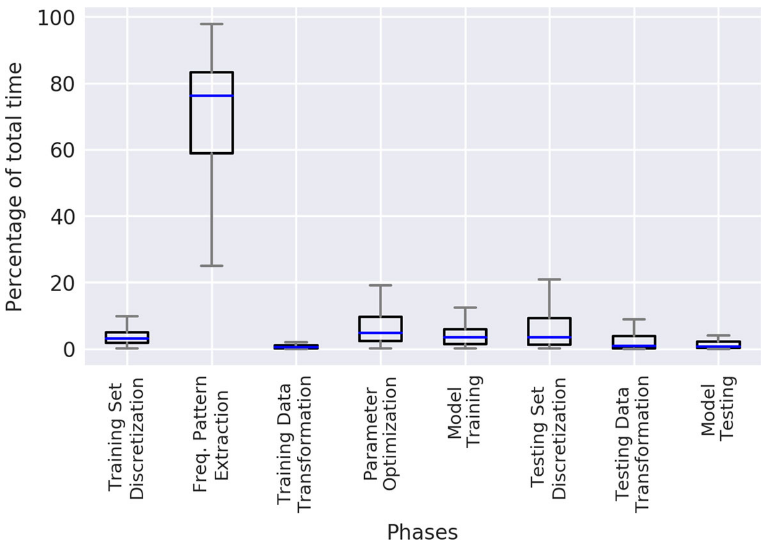


Fig. 6 A breakdown of the time spent for individual phases as a percentage of total runtime. The box plots show the aggregated data for all the evaluated datasets when using MiSTiCl (CS + ET)

optimization, and model training/testing is less than 10% of the total time required to process a given dataset.

All the above observations regarding classification accuracy and runtime confirm that MiSTiCl is a faster algorithm for providing classification accuracy which is on par with existing algorithms, while incorporating extremely fast parameter optimization. Tables 6 and 7 provide the results used for the analysis performed for the classification accuracy and runtime, respectively.

5.1 Results using a conservative parameter set

We also carried out experiments for evaluating the classification and runtime performance of 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 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 with a slight reduction in the classification accuracy as well; however, the overall performance is generally the same. Figure 7 shows the average ranks of the different algorithms based on classification accuracy when using a conservative set of parameters for the MiSTiCl algorithm.

6 Conclusion

In this paper, we proposed MiSTiCl, a time series classification approach using a linear time string mining algorithm for feature extraction. Using the string mining algorithm for feature extraction allowed us 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

Table 6 Average classification accuracy obtained for 100 runs using shuffled splits

	ST	ET	RF	AB	BOSS	BoP	SAX-VSM
Adiac	**76.84**	75.86	75.15	75.41	74.96	59.41	32.65
ArrowHead	85.11	85.93	82.43	83.10	**87.51**	77.12	75.85
Beef	73.57	**78.27**	74.64	71.77	61.47	49.70	47.50
BeetleFly	87.45	95.79	95.91	**96.01**	94.85	84.25	88.35
BirdChicken	92.70	97.00	96.68	96.47	**98.40**	85.10	85.05
CBF	90.18	98.45	97.53	96.49	**99.81**	96.28	93.74
Car	**98.56**	92.14	90.08	91.45	85.47	81.93	80.02
ChlorineConcentration	**68.21**	64.87	62.84	64.09	65.95	63.38	62.63
CinCECGtorso	**91.83**	71.34	69.15	68.29	90.11	71.84	71.18
Coffee	**99.50**	99.04	98.84	98.90	98.86	94.29	93.43
Computers	78.46	77.62	76.43	76.84	**79.47**	69.21	59.50
CricketX	**77.71**	65.67	63.82	65.77	76.35	57.42	71.33
CricketY	**76.22**	63.58	62.94	64.35	74.93	61.42	67.14
CricketZ	**79.78**	66.10	64.13	65.55	77.57	58.75	71.63
DiatomSizeReduction	91.12	**96.65**	93.17	93.05	93.94	85.13	83.32
DistalPhalanxOutlineAgeGroup	82.93	85.99	85.95	**86.54**	81.48	73.68	76.84
DistalPhalanxOutlineCorrect	81.94	85.71	85.71	**86.03**	81.58	72.49	68.42
DistalPhalanxTW	69.04	72.72	74.04	**75.35**	67.31	60.50	63.05
ECG200	73.73	90.80	**91.59**	90.73	89.04	78.27	82.07
ECG5000	84.02	93.82	93.90	93.91	**94.05**	91.02	90.78
ECGFiveDays	94.34	96.23	94.97	94.31	**98.33**	91.09	91.34
Earthquakes	**95.50**	77.05	76.29	77.54	74.60	72.91	73.80
ElectricDevices	**89.54**	83.06	84.69	83.06	80.11	77.29	DNF
FaceAll	96.76	92.15	90.79	91.76	**97.42**	93.91	96.07
FaceFour	79.40	94.74	95.90	92.07	**99.56**	94.75	91.86
FacesUCR	90.93	85.98	84.62	85.46	**95.06**	89.10	92.22
FiftyWords	**71.30**	70.57	67.28	70.11	70.22	54.96	45.07
Fish	**97.42**	92.13	92.08	93.78	96.87	89.39	90.38
FordA	**96.54**	80.14	79.80	80.02	91.95	79.43	83.50
FordB	**91.51**	76.02	76.61	77.68	91.11	76.33	80.90
GunPoint	**99.87**	99.14	97.09	98.35	99.41	96.97	95.16
Ham	80.84	78.65	77.49	79.34	**83.59**	77.14	77.74
HandOutlines	**92.39**	89.67	89.36	90.16	90.41	86.59	83.98
Haptics	51.19	52.48	51.45	**53.54**	45.90	38.20	40.67
Herring	65.34	72.24	**73.81**	72.69	60.55	54.92	58.25
InlineSkate	39.30	49.22	44.92	46.57	**50.29**	38.50	36.26
InsectWingbeatSound	**61.65**	53.56	54.35	53.69	51.02	48.41	53.19
ItalyPowerDemand	**95.31**	93.79	93.21	92.58	86.61	87.81	81.94
LargeKitchenAppliances	93.25	92.74	**93.54**	93.47	84.10	68.17	76.08
Lightning2	65.89	**85.71**	83.99	83.93	80.97	69.20	70.47
Lightning7	72.44	**80.51**	77.94	78.80	65.87	53.63	59.57
Mallat	97.23	**98.88**	98.40	97.57	94.86	85.52	85.99

Table 6 continued

	ST	ET	RF	AB	BOSS	BoP	SAX-VSM
Meat	96.57	98.78	**98.84**	98.48	98.03	96.17	93.47
MedicalImages	69.11	75.59	73.47	**75.97**	71.48	50.78	47.98
MiddlePhalanxOutlineAgeGroup	**81.51**	75.25	75.85	73.98	66.18	56.54	60.65
MiddlePhalanxOutlineCorrect	69.39	82.41	82.44	**84.70**	81.05	70.81	63.43
MiddlePhalanxTW	57.93	63.21	**63.77**	62.63	53.44	48.58	53.32
MoteStrain	**88.23**	87.20	87.17	87.60	84.39	82.57	79.61
NonInvasiveFetalECGThorax1	**94.68**	88.61	87.47	89.18	84.09	68.19	55.76
NonInvasiveFetalECGThorax2	73.86	**90.84**	89.49	90.50	90.36	74.57	64.65
OSULeaf	88.07	74.57	72.35	79.13	**96.73**	69.71	82.30
OliveOil	93.41	**97.34**	97.10	97.10	87.00	84.67	84.93
PhalangesOutlinesCorrect	79.45	**82.60**	81.68	82.34	82.09	71.59	62.16
Phoneme	**32.91**	29.82	29.42	29.63	25.61	15.13	9.42
Plane	**99.96**	99.64	99.64	99.54	99.79	98.72	98.14
ProximalPhalanxOutlineAgeGroup	88.09	89.19	**89.68**	88.29	81.90	75.95	79.44
ProximalPhalanxOutlineCorrect	84.09	86.83	86.88	**87.41**	86.64	76.84	75.93
ProximalPhalanxTW	80.28	82.18	**83.70**	83.10	76.97	67.52	59.72
RefrigerationDevices	76.08	77.34	74.81	77.92	**78.86**	65.55	60.95
ScreenType	**67.61**	57.24	59.17	59.82	58.62	44.01	43.94
ShapeletSim	93.36	99.87	99.96	**100.00**	**100.00**	82.35	91.19
ShapesAll	85.42	89.30	86.98	85.53	**90.87**	75.65	65.86
SmallKitchenAppliances	80.25	79.07	**80.37**	79.26	76.45	70.77	48.38
SonyAIBORobotSurface1	88.76	85.94	81.85	81.20	**89.78**	76.41	72.44
SonyAIBORobotSurface2	**92.44**	80.39	80.33	80.16	88.66	84.15	85.98
StarlightCurves	97.74	94.30	93.50	93.87	**97.75**	94.35	83.03
Strawberry	96.84	97.95	97.61	**98.09**	97.03	96.39	95.95
SwedishLeaf	**93.85**	90.11	89.75	91.58	91.77	78.68	71.15
Symbols	86.16	96.16	**96.25**	92.10	96.10	93.26	87.11
SyntheticControl	98.69	**99.67**	99.58	99.44	96.79	92.74	94.68
ToeSegmentation1	**95.40**	94.72	93.23	91.64	92.88	92.55	92.54
ToeSegmentation2	94.72	93.88	91.57	91.58	**95.98**	91.30	91.22
Trace	**99.99**	99.81	99.70	99.93	**99.99**	97.65	99.39
TwoLeadECG	98.44	96.25	95.92	94.32	**98.45**	90.11	90.08
TwoPatterns	95.17	94.77	93.82	94.00	**99.12**	94.47	89.27
UWaveGestureLibraryAll	80.59	77.22	75.86	78.11	**94.45**	81.48	79.99
UWaveGestureLibraryX	73.70	**75.72**	74.66	75.22	75.32	59.00	53.28
UWaveGestureLibraryY	**74.68**	65.71	65.11	65.84	66.10	51.24	44.34
UWaveGestureLibraryZ	**94.21**	69.86	69.62	71.23	69.50	56.61	47.54
Wafer	**99.98**	99.19	99.03	99.27	99.90	99.62	99.61
Wine	92.61	96.02	**96.51**	94.71	91.17	89.00	82.09
WordSynonyms	58.24	59.68	56.40	58.58	**65.88**	52.24	46.69
Worms	71.95	78.85	77.69	**80.24**	73.51	61.16	58.95

Table 6 continued

	ST	ET	RF	AB	BOSS	BoP	SAX-VSM
WormsTwoClass	77.87	82.97	**85.62**	84.12	80.97	74.49	73.07
Yoga	82.25	83.39	81.57	84.75	**91.00**	85.92	78.91
Average ranks	3.05	2.92	3.68	3.25	3.04	5.82	6.06

The best average accuracy for each dataset is represented in bold

Table 7 Average runtime (in seconds) for each algorithm obtained over 100 runs

	ET	RF	AB	BOSS	BoP	SAX-VSM
Adiac	138	156	300	109	**56.5**	389
ArrowHead	**2.56**	2.64	3.82	7.65	7.21	15.4
Beef	**3.73**	3.86	3.75	16.3	17	28.5
BeetleFly	**2.17**	**2.17**	2.3	12.3	12.4	19.3
BirdChicken	2.44	**2.44**	2.48	12.4	10.5	15.1
CBF	1.86	1.89	**1.78**	5.44	3.77	12.8
Car	**6.9**	7.16	9.16	59.8	37.8	55.3
ChlorineConcentration	**15.4**	17.5	128	372	214	4.33e+03
CinCECGtorso	221	221	221	339	**193**	203
Coffee	**1.79**	1.83	1.94	8.41	6.74	12.5
Computers	**25.1**	26.8	52.3	713	513	1.14e+03
CricketX	**48.5**	58.2	159	484	610	9.31e+03
CricketY	**49.4**	58.5	160	509	602	9.59e+03
CricketZ	**47.8**	57.6	158	485	588	9.32e+03
DiatomSizeReduction	5.07	5.13	**4.98**	15	5.03	7.04
DistalPhalanxOutlineAgeGroup	**6.03**	8.63	42.4	38	38.1	976
DistalPhalanxOutlineCorrect	**6.42**	10.8	69.3	81.2	86.3	3.29e+03
DistalPhalanxTW	**10.7**	14.5	57.4	39.5	36.2	1.03e+03
ECG200	**1.52**	1.85	7.15	5.57	6.28	83.1
ECG5000	**18.2**	21.7	109	494	195	6.07e+03
ECGFiveDays	**1.71**	1.76	1.73	5.01	3.75	6.94
Earthquakes	**14.2**	16.4	42.9	1.15e+03	1.25e+03	1.58e+03
ElectricDevices	**729**	979	796	2.64e+04	1.29e+04	*DNF*
FaceAll	**33.1**	49.1	189	307	381	2.21e+04
FaceFour	6.2	6.25	**6.18**	11.4	8.98	17.6
FacesUCR	**13.3**	17	69.9	71.5	60.2	1.64e+03
FiftyWords	**227**	256	490	424	484	8.31e+03
Fish	**25.5**	27.5	46.8	222	126	292
FordA	**572**	632	1.41e+03	1.56e+05	1.63e+05	7.38e+05
FordB	**571**	632	1.38e+03	1.67e+05	1.72e+05	7.31e+05
GunPoint	**1.6**	1.68	2.22	6.06	4.4	15.5
Ham	**4.1**	4.56	12.6	110	96.5	397

Table 7 continued

	ET	RF	AB	BOSS	BoP	SAX-VSM
HandOutlines	**1.51e+03**	1.52e+03	1.64e+03	1.07e+05	7.78e+04	3.54e+04
Haptics	**52.9**	54.8	77.3	692	628	720
Herring	**3.84**	4.03	7.07	54.2	40.4	60.2
InlineSkate	**139**	140	159	1.01e+03	765	623
InsectWingbeatSound	**25.8**	28.7	102	135	97.9	1.03e+03
ItalyPowerDemand	0.908	0.697	1.22	0.66	**0.451**	1.95
LargeKitchenAppliances	**77.4**	80.1	113	1.47e+03	684	1.19e+03
Lightning2	**4.3**	4.44	7.5	78.7	64.9	105
Lightning7	**7.06**	7.61	13.1	27.3	28.8	104
Mallat	77.4	77.3	**76.9**	424	116	136
Meat	**8.28**	8.41	8.95	44.5	24.7	43.8
MedicalImages	**12.9**	19.3	93.9	53.5	46.9	2.29e+03
MiddlePhalanxOutlineAgeGroup	**6.57**	9.2	47.9	36.5	32.8	726
MiddlePhalanxOutlineCorrect	**6.54**	10.8	62.9	73.7	70.5	2.1e+03
MiddlePhalanxTW	**11.4**	16.2	73.9	35.8	33.8	730
MoteStrain	1.23	1.01	**0.848**	1.43	1.92	5.55
NonInvasiveFetalECGThorax1	**8.35e+03**	8.47e+03	9.41e+03	4.53e+04	3.21e+04	1.15e+05
NonInvasiveFetalECGThorax2	**8.59e+03**	8.71e+03	9.54e+03	4.55e+04	2.79e+04	8.79e+04
OSULeaf	**16.9**	19.1	46	268	297	1.91e+03
OliveOil	**9.26**	9.32	9.37	23.9	14.7	21.1
PhalangesOutlinesCorrect	**24.4**	41.3	248	600	582	3.59e+04
Phoneme	**1.09e+03**	1.1e+03	1.24e+03	1.51e+03	1.86e+03	2.64e+03
Plane	**4.14**	4.77	4.26	15.1	12.1	109
ProximalPhalanxOutlineAgeGroup	**6.68**	9.03	39.1	37.7	34.1	584
ProximalPhalanxOutlineCorrect	**7.39**	11.9	69.6	72.5	69.1	1.55e+03
ProximalPhalanxTW	**11.9**	15.3	51.2	37.3	34.3	585
RefrigerationDevices	**44.3**	47.6	95.2	1.84e+03	2.33e+03	4.01e+03
ScreenType	**55.1**	58.9	97.8	1.49e+03	1.08e+03	3.35e+03
ShapeletSim	2.55	**2.54**	2.59	14.9	12.3	15
ShapesAll	**1.01e+03**	1.05e+03	1.28e+03	2.56e+03	2.21e+03	1.58e+04
SmallKitchenAppliances	**87.4**	90.3	134	1.51e+03	695	653
SonyAIBORobotSurface1	0.472	0.382	**0.302**	1.1	1.52	4.72
SonyAIBORobotSurface2	0.839	**0.604**	0.802	1.35	1.89	7.3
StarlightCurves	**579**	580	794	2.79e+04	1.49e+04	4.94e+04
Strawberry	**20.3**	24.3	64.9	619	550	8.38e+03
SwedishLeaf	**31.4**	44.1	133	147	138	5.24e+03
Symbols	13.6	13.7	**13.3**	30.7	8.36	13.4
SyntheticControl	**4.61**	7.17	17.5	12.7	20.5	1.87e+03
ToeSegmentation1	**2.04**	2.07	2.53	11.5	11.6	31.2
ToeSegmentation2	**1.96**	2	2.72	13.3	11.7	25.8
Trace	**5.22**	5.48	5.73	39	28.4	118

Table 7 continued

	ET	RF	AB	BOSS	BoP	SAX-VSM
TwoLeadECG	1.77	1.54	**1.53**	3.11	1.65	4.87
TwoPatterns	**27.7**	38.7	248	680	879	5e+04
UWaveGestureLibraryAll	**714**	737	1.03e+03	2.3e+04	1.94e+04	9.98e+04
UWaveGestureLibraryX	**136**	156	437	2.47e+03	2.53e+03	4.25e+04
UWaveGestureLibraryY	**138**	156	428	2.44e+03	2.46e+03	4.09e+04
UWaveGestureLibraryZ	**133**	154	433	2.37e+03	2.39e+03	3.89e+04
Wafer	**20.4**	24.9	81.2	1.25e+03	530	2.92e+04
Wine	**2.96**	3.08	4.35	10.8	9.89	18.3
WordSynonyms	**65.5**	73.6	172	171	178	2.07e+03
Worms	**36.4**	38	57.4	782	793	2.33e+03
WormsTwoClass	**17.1**	18.1	32.5	782	823	2.18e+03
Yoga	**22.2**	23.1	90	848	323	1.37e+03
Average ranks	1.329	2.129	3.094	4.576	4.047	5.774

The time for the fastest algorithm is represented in bold

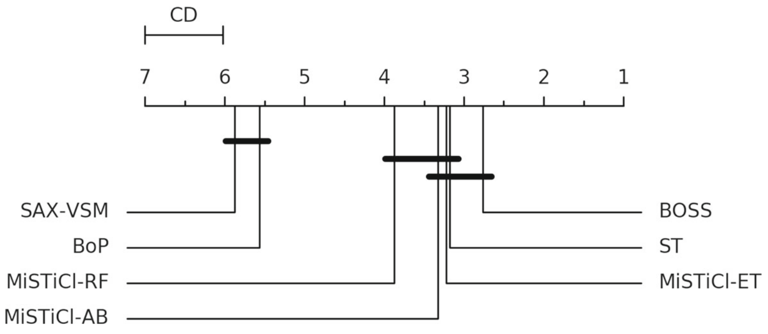


Fig. 7 Average ranks of the different algorithms based on classification accuracy. The critical difference (CD) for significantly different algorithms is 0.97. Algorithms which are not significantly different (at $p = 0.05$) are connected

of magnitude speedup 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-view 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) such that an ensemble of base models is created where each single-view feature set is used to create the base models. 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 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 would scale very well for a variety of problems with large and complex data. In the future, we intend to adapt our algorithm for multi-variate time series problems. Another avenue to explore would be the application to online or streaming time series problems. A multi-variate streaming time series classification algorithm based on string mining could potentially have a huge impact on time series classification for sensor data from the Internet of things (IoT).

Acknowledgements We are grateful to the reviewers for their comments and suggestions which helped in improving the quality of this paper. The first author was supported by a scholarship from the Higher Education Commission (HEC), Pakistan, and the German Academic Exchange Service (DAAD), Germany.

Appendix: Calculating independence test statistics

Section 3.3 provides the algorithmic details for selecting frequent patterns based on their discriminative power. χ^2 independence test or information gain values can be used to determine the discriminative power of a given pattern and find out how effectively it can identify the instances of a given class. This section explains the procedure of calculating these statistics based on the occurrence frequency of a pattern in the positive and negative class dataset splits. In this regard, the notation used for the following discussion is given below.

Symbol	Representation
$\widehat{P}$	Positive class dataset
$\widehat{N}$	Negative class dataset
$N_{\widehat{P}}$	Number of instances in $\widehat{P}$
$N_{\widehat{N}}$	Number of instances in $\widehat{N}$
p	Frequent pattern
$f_{\widehat{P}}$	Occurrence frequency of p in $\widehat{P}$
$f_{\widehat{N}}$	Occurrence frequency of p in $\widehat{N}$

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, it is the number of instances labeled as belonging to the positive or negative class given a particular frequent pattern. This can be determined using the instance counts of the positive and negative class datasets 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 splits		
	Positive, $\widehat{P}$	Negative, $\widehat{N}$	
With(p)	$O_{11} = \lfloor f_{\widehat{P}} \times N_{\widehat{P}} \rceil$	$O_{12} = \lfloor f_{\widehat{N}} \times N_{\widehat{N}} \rceil$	$RSum_{1.} = O_{11} + O_{12}$
WithOut(p)	$O_{21} = N_{\widehat{P}} - O_{11}$	$O_{22} = N_{\widehat{N}} - O_{12}$	$RSum_{2.} = O_{21} + O_{22}$
	$CSum_{.1} = O_{11} + O_{21}$	$CSum_{.2} = O_{12} + O_{22}$	$n = \sum_{i,j} O_{ij}$

The rows of this contingency table correspond to the number of instances containing or not containing the given pattern p, while the columns correspond to the positive and negative dataset splits, respectively. The combined total of row and column sums equals the total number of instances in the positive and negative dataset splits. Finally, the expected values (E_{ij}) are calculated using the following formula.

$$E_{ij} = \frac{RSum_{i.} \times CSum_{.j}}{n}$$

The χ^2 test statistic determines whether 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 the ones for which the dataset splits do not exhibit any mutual relationship.

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.

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

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.

$$\begin{aligned} H_{\widehat{P}} = &-\left(\frac{f_{\widehat{P}} \times N_{\widehat{P}}}{f_{\widehat{P}} \times N_{\widehat{P}} + f_{\widehat{N}} \times N_{\widehat{N}}} \times \log_2 \frac{f_{\widehat{P}} \times N_{\widehat{P}}}{f_{\widehat{P}} \times N_{\widehat{P}} + f_{\widehat{N}} \times N_{\widehat{N}}}\right) \\ &-\left(\frac{f_{\widehat{N}} \times N_{\widehat{N}}}{f_{\widehat{P}} \times N_{\widehat{P}} + f_{\widehat{N}} \times N_{\widehat{N}}} \times \log_2 \frac{f_{\widehat{N}} \times N_{\widehat{N}}}{f_{\widehat{P}} \times N_{\widehat{P}} + f_{\widehat{N}} \times N_{\widehat{N}}}\right) \\ H_{\widehat{N}} = &-\left(\frac{(1 - f_{\widehat{P}}) \times N_{\widehat{P}}}{(1 - f_{\widehat{P}}) \times N_{\widehat{P}} + (1 - f_{\widehat{N}}) \times N_{\widehat{N}}} \times \log_2 \frac{(1 - f_{\widehat{P}}) \times N_{\widehat{P}}}{(1 - f_{\widehat{P}}) \times N_{\widehat{P}} + (1 - f_{\widehat{N}}) \times N_{\widehat{N}}}\right) \\ &-\left(\frac{(1 - f_{\widehat{N}}) \times N_{\widehat{N}}}{(1 - f_{\widehat{P}}) \times N_{\widehat{P}} + (1 - f_{\widehat{N}}) \times N_{\widehat{N}}} \times \log_2 \frac{(1 - f_{\widehat{N}}) \times N_{\widehat{N}}}{(1 - f_{\widehat{P}}) \times N_{\widehat{P}} + (1 - f_{\widehat{N}}) \times N_{\widehat{N}}}\right) \end{aligned}$$

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.

$$IG = H - \left(\frac{f_{\widehat{P}} \times N_{\widehat{P}} + f_{\widehat{N}} \times N_{\widehat{N}}}{N_{\widehat{P}} + N_{\widehat{N}}} \times H_{\widehat{P}} + \frac{(1 - f_{\widehat{P}}) \times N_{\widehat{P}} + (1 - f_{\widehat{N}}) \times N_{\widehat{N}}}{N_{\widehat{P}} + N_{\widehat{N}}} \times H_{\widehat{N}} \right)$$

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.

References

1. Breiman L (2001) Random forests. Mach Learn 45(1):5–32. https://doi.org/10.1023/A:1010933404324
2. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
3. Dhaliwal J, Puglisi SJ, Turpin A (2012) Practical efficient string mining. IEEE Trans Knowl Data Eng 24(4):735–744. https://doi.org/10.1109/TKDE.2010.242
4. Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data. Proc VLDB Endow 1(2):1542–1552
5. Fischer J, Heun V, Kramer S (2005) Fast frequent string mining using suffix arrays. In: 5th International conference on data mining, IEEE, ICDM '05, pp 609–612. https://doi.org/10.1109/ICDM.2005.62
6. Fischer J, Heun V, Kramer S (2006) Optimal string mining under frequency constraints. In: Knowledge discovery in databases, PKDD 2006, lecture notes in computer science, vol 4213. Springer, Berlin, pp 139–150. https://doi.org/10.1007/11871637_17
7. Freund Y (1995) Boosting a Weak Learning Algorithm by Majority. Inf Comput 121(2):256–285. https://doi.org/10.1006/inco.1995.1136
8. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42. https://doi.org/10.1007/s10994-006-6226-1
9. Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014) Classification of time series by shapelet transformation. Data Min Knowl Discov 28(4):851–881. https://doi.org/10.1007/s10618-013-0322-1
10. Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing SAX: a novel symbolic representation of time series. Data Min Knowl Discov 15(2):107–144. https://doi.org/10.1007/s10618-007-0064-z
11. Lin J, Khade R, Li Y (2012) Rotation-invariant similarity in time series using bag-of-patterns representation. J Intell Inf Syst 39(2):287–315. https://doi.org/10.1007/s10844-012-0196-5
12. Rakthanmanon T, Keogh E (2013) Fast shapelets: a scalable algorithm for discovering time series shapelets. In: Proceedings of the 2013 SIAM international conference on data mining, SDM, Society for Industrial and Applied Mathematics, pp 668–676. https://doi.org/10.1137/1.9781611972832.74
13. Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Min Knowl Discov 29(6):1505–1530. https://doi.org/10.1007/s10618-014-0377-7
14. Schäfer P (2016) Scalable time series classification. Data Min Knowl Discov 30(5):1273–1298. https://doi.org/10.1007/s10618-015-0441-y
15. Senin P, Malinchik S (2013) SAX-VSM: interpretable time series classification using SAX and vector space model. In: 13th International conference on data mining, IEEE, ICDM '13, pp 1175–1180. https://doi.org/10.1109/ICDM.2013.52
16. Toivonen H (2017) Frequent pattern. In: Sammut C, Webb GI (eds) Encyclopedia of machine learning and data mining. Springer, Boston, pp 524–529. https://doi.org/10.1007/978-1-4899-7687-1_318
17. Ye L, Keogh E (2011) Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min Knowl Discov 22(1):149–182. https://doi.org/10.1007/s10618-010-0179-5

Atif Raza is a Ph.D. candidate at the Institute of Computer Science, Johannes Gutenberg University (JGU) Mainz, Germany. He held a Ph.D. scholarship from 2014–2018 by the Higher Education Commission (HEC), Pakistan. He received his B.S. and M.S. degrees from Pakistan Institute of Engineering and Applied Sciences, Islamabad, Pakistan in 2004 and 2008, respectively. His research interests include metaheuristics, time-series and sequence mining using efficient greedy algorithms, and data mining.

Stefan Kramer is full professor at the Institute of Computer Science of Johannes Gutenberg University (JGU) Mainz and Honorary Professor of the University of Waikato. Before his appointment at JGU, he was associate professor at the Computer Science Department of Technische Universität München (2003–2011). He has been active in the field of data mining since the first conference worldwide in 1995 and is author of award-winning papers at ICDM, KDD and ILP. He was vice-chair of ICDM 2013 and is regularly area chair of conferences like ECML/PKDD. His research interests include mining structured data, stream mining, process mining and clustering.